Bioinformatics Programming 2013

Problem 1

Installing Python

Why Python?

Rosalind problems can be solved using any programming language. Our language of choice is **Python**. Why? Because it's simple, powerful, and even funny. You'll see what we mean.

> If you don't already have Python software, please download and install the appropriate version for your platform (Windows, Linux or Mac OS X). Please install Python of version 2.x (not 3.x) — it has more libraries support and many well-written guides.
>
> After completing installation, launch **IDLE** (default Python development environment; it's usually installed with Python, however you may need to install it separately on Linux).
>
> You'll see a window containing three arrows, like so:
>
> ```
> >>>
> ```
>
> The three arrows are Python's way of saying that it is ready to serve your every need. You are in interactive mode, meaning that any command you type will run immediately. Try typing `1+1` and see what happens.
>
> Of course, to become a Rosalind pro, you will need to write programs having more than one line. So select **File** → **New Window** from the IDLE menu. You can now type code as you would into a text editor. For example, type the following:
>
> ```
> print "Hello, World!"
> ```
>
> Select **File** → **Save** to save your creation with an appropriate name (e.g., `hello.py`).
>
> To run your program, select **Run** → **Run Module**. You'll see the result in the interactive mode window (Python Shell).
>
> Congratulations! You just ran your first program in Python!

Problem

After downloading and installing Python, type `import this` into the Python command line and see what happens. Then, click the "Download dataset" button below and copy the Zen of Python into the space provided.

Problem 2

Variables and Some Arithmetic

Variables and Some Arithmetic

One of the most important features of any programming language is its ability to manipulate variables. A variable is just a name that refers to a value; you can think of a variable as a box that stores a piece of data.

In Python, the basic data types are strings and numbers. There are two types of numbers: integers (both positive and negative) and floats (fractional numbers with a decimal point). You can assign numbers to variables very easily. Try running the following program:

```
a = 324
b = 24
c = a - b
print 'a - b is', c
```

In the above code, a, b, and c are all integers, and 'a - b is' is a string. The result of this program is to print:

```
a - b is 300
```

You can now use all common arithmetic operations involving numbers:

- Addition: `2 + 3 == 5`
- Subtraction: `5 - 2 == 3`
- Multiplication: `3 * 4 == 12`
- Division: `15 / 3 == 5`
- Division remainder: `18 % 5 == 3`
- Exponentiation: `2 ** 3 == 8`

It is important to note that if you try to divide two integers, Python always rounds *down* the result (so `18/5 == 3`).

To obtain a precise result for this division, you need to indicate floating point division; either of the following expressions results in a "float" data type: `18.0/5 == 3.6` or `float(18)/5 == 3.6`

In Python, the single equals sign (`=`) means "assign a value to a variable". For example, `a = 3` assigns 3 to the integer a. In order to denote equality, Python uses the double equals sign (`==`).

In Python, a string is an ordered sequence of letters, numbers and other characters. You can create string variables just like you did with :

```
a = "Hello"
b = "World"
```

Notice that the string must be surrounded by " or ' (but not a mix of both). You can use quotes inside the string, as long as you use the opposite type of quotes to surround the string, e.g., `a = "Monty Python's Flying Circus"` or `b = 'Project "Rosalind"'`.

String operations differ slightly from operations on numbers:

```
a = 'Rosalind'
b = 'Franklin'
```

```
c = '!'
print a + ' ' + b + c*3
```

Output:

```
Rosalind Franklin!!!
```

Problem

Given: Two positive integers a and b, each less than 1000.

Return: The integer corresponding to the square of the hypotenuse of the right triangle whose legs have lengths a and b.

Notes:

1. The dataset changes every time you click "Download dataset".
2. We check only your final answer to the *downloaded* dataset in the box below, not your code itself. If you would like to provide your code as well, you may use the upload tool. Please also note that the correct answer to this problem will not in general be 34; it is simply an example of what you should return in the specific case that the legs of the triangle have length 3 and 5.

Sample Dataset

```
3 5
```

Sample Output

```
34
```

Problem 3

Strings and Lists

Strings and lists

We've already seen numbers and strings, but Python also has variable types that can hold more than one piece of data at a time. The simplest such variable is a list.

You can assign data to a list in the following way: `list_name = [item_1, item_2, ..., item_n]`. The items of the list can be of any other type: integer, float, string. You even explore your inner Zen and make lists of lists!

Any item in a list can be accessed by its index, or the number that indicates its place in the list. For example, try running the following code:

```
tea_party = ['March Hare', 'Hatter', 'Dormouse', 'Alice']
print tea_party[2]
```

Your output should be:

```
Dormouse
```

Note that the output was *not* `Hatter`, as you might have guessed. This is because in Python, indexing begins with 0, not 1. This property is called 0-based numbering, and it's shared by many programming languages.

You can easily change existing list items by reassigning them. Try running the following:

```
tea_party[1] = 'Cheshire Cat'
print tea_party
```

This code should output the list with "Hatter" replaced by "Cheshire Cat":

```
March Hare, Cheshire Cat, Dormouse, Alice
```

You can also add items to the end of an existing list by using the function `append()`:

```
tea_party.append('Jabberwocky')
print tea_party
```

This code outputs the following:

```
March Hare, Cheshire Cat, Dormouse, Alice, Jabberwocky
```

If you need to obtain only some of a list, you can use the notation `list_name[a:b]` to get only those from index `a` up to but *not* including index `b`. For example, `tea_party[1:3]` returns `Cheshire Cat, Dormouse`, not `Cheshire Cat, Dormouse, Alice`. This process is called "list slicing".

If the first index of the slice is unspecified, then Python assumes that the slice begins with the beginning of the list (i.e., index 0); if the second index of the slice is unspecified, then you will obtain the items at the end of the list. For example, `tea_party[:2]` returns `March Hare, Cheshire Cat` and `tea_party[3:]` returns `Alice, Jabberwocky`.

You can also use negative indices to count items backtracking from the end of the list. So `tea_party[-2:]` returns the same output as `tea_party[3:]`: `Alice, Jabberwocky`.

Finally, Python equips you with the magic ability to slice strings the same way that you slice lists. A string can be considered as a list of characters, each of which having its own index starting from 0. For example, try running the following code:

```
a = 'flimsy'
b = 'miserable'
c = b[0:1] + a[2:]
print c
```

This code will output the string formed by the first character of `miserable` and the last four characters of `flimsy`:

```
mimsy
```

Problem

Given: A string s of length at most 200 letters and four integers a, b, c and d.

Return: The slice of this string from indices a through b and c through d (with space in between), *inclusively*.

Sample Dataset

```
HumptyDumptysatonawallHumptyDumptyhadagreatfallAlltheKingshorsesandallt
heKingsmenCouldntputHumptyDumptyinhisplaceagain.
22 27 97 102
```

Sample Output

```
Humpty Dumpty
```

Problem 4

Conditions and Loops

Conditions and Loops

If you need Python to choose between two actions, then you can use an `if`/`else` statement. Try running this example code:

```
a = 42
if a < 10:
  print 'the number is less than 10'
else:
  print 'the number is greater or equal to 10'
```

Note the indentation and punctuation (especially the colons), because they are important.

If we leave out an `else`, then the program continues on. Try running this program with different initial values of `a` and `b`:

```
if a + b == 4:
  print 'printed when a + b equals four'
print 'always printed'
```

If you want to repeat an action several times, you can use a while loop. The following program

prints `Hello` once, then adds 1 to the `greetings` counter. It then prints `Hello` twice because `greetings` is equal to 2, then adds 1 to `greetings`. After printing `Hello` three times, `greetings` becomes 4, and the `while` condition of `greetings <= 3` is no longer satisfied, so the program would continue past the `while` loop.

```
greetings = 1
while greetings <= 3:
  print 'Hello! ' * greetings
  greetings = greetings + 1
```

Be careful! If you accidentally create an infinite loop, your program will freeze and you will have to abort it. Here's an example of an infinite loop. Make sure you see why it will never exit the `while` loop:

```
greetings = 1
while greetings <= 3:
  print 'Hello! ' * greetings
  greetings = greetings + 0 # Bug here
```

If you want to carry out some action on every element of a list, the `for` loop will be handy

```
names = ['Alice', 'Bob', 'Charley']
for name in names:
  print 'Hello, ' + name
```

And if you want to repeat an action exactly n times, you can use the following template:

```
n = 10
for i in range(n):
  print i
```

In the above code, `range` is a function that creates a list of integers between 0 and n, where n is not included.

Finally, try seeing what the following code prints when you run it:

```
print range(5, 12)
```

More information about loops and conditions can be found in the Python documentation.

Problem

Given: Two positive integers a and b ($a < b < 10000$).

Return: The sum of all odd integers from a through b, inclusively.

Sample Dataset

```
100 200
```

Sample Output

```
7500
```

Hint

You can use `a % 2 == 1` to test if a is odd.

Problem 5

Working with Files

Reading and Writing

In Rosalind, sample datasets are given as files. Python has a lot of functions for reading and writing information in files.

To access a file, you must first open it. To do so, you can use the `open()` function, which takes two parameters: the name of the target file and the mode. Three modes are available:

- `r` - read mode (the file is opened for reading)
- `w` - write mode (the file is opened for writing, and if a file having the same name exists, it will be erased)
- `a` - append mode (the file is opened for appending, which means that data is only to be added to the existing data in the file)

```
f = open('input.txt', 'r')
```

This code told Python to open the file `input.txt` in `r` mode and store the result of this operation in a file object called `f`.

To obtain data from the file object you created, you can apply the following methods:

The command `f.read(n)` returns n bytes of data from the file as a string. When the size parameter is omitted, the entire contents of the file will be read and returned.

The command `f.readline()` takes a single line from the file. Every line (except the last line of file) terminates in a newline character (`\n`). To remove this character from the end of a line you have read, use the `.strip()` method. Note that every time you call `.readline()` it takes the next line in the file.

The command `f.readlines()` returns a list containing every line in the file. If you need to obtain a particular line, you can use a list item index, e.g., `f.readlines()[2]` returns the third line of the file object `f` (don't forget that Python utilizes 0-based numbering!)

An alternative way to read lines is to loop over the file object.

```
for line in f:
  print line
```

Using this loop, you can do anything you need with every line in the file.

If the data in the file are not separated by new lines but rather by whitespace, commas, or any other delimeter, then all three commands above will return the data only in the form of lines. As a workaround, you can use the command `line.split()`. It uses whitespace in addition to `\n` as delimeters by default, and runs of the same delimiter are regarded as a single separating space. For example,

`'Beautiful is better than ugly.\n'.split()` returns `['Beautiful', 'is', 'better', 'than', 'ugly.']`

You can even specify the delimiter as a parameter of `line.split()`:

`'Explicit, is better, than implicit.'.split(",")` returns `['Explicit', ' is better', ' than implicit.']`

Another convenient command for file parsing is `.splitlines()`. It returns a list of the lines in the string, breaking at line boundaries. Line breaks are not included.

`'Simple is\nbetter than\ncomplex.\n'.splitlines()` returns `['Simple is', 'better than', 'complex.']`

When you at last complete all your calculations and obtain a result, you need to store it somewhere. To save a file, output the desired file in write mode (if there is no such file, it will be created automatically):

```
f = open('output.txt', 'w')
```

You can then write your data using `.write()` method.

```
f.write('Any data you want to write into file')
```

The command `f.write(string)` writes the contents of `string` to file `f`. If you want to write something other than a string (an integer say), you must first convert it to a string by using the function `str()`.

```
inscription = ['Rosalind Elsie Franklin ', 1920, 1958]
s = str(inscription)
f.write(s)
```

You also can write list items into a file one at a time by using a `for` loop:

```
for i in inscription:
  output.write(str(i) + '\n')
```

Adding `\n` to `str(i)` means that every item will start with a new line.

When you are finished writing file, don't forget that you must close it using the command `f.close()`. It's a good habit to get into.

Problem

Given: A file containing at most 1000 lines.

Return: A file containing all the even-numbered lines from the original file. Assume 1-based numbering of lines.

Sample Dataset

```
`Bravely bold Sir Robin rode forth from Camelot
Yes, brave Sir Robin turned about
He was not afraid to die, O brave Sir Robin
And gallantly he chickened out
He was not at all afraid to be killed in nasty ways
Bravely talking to his feet
Brave, brave, brave, brave Sir Robin
He beat a very brave retreat
```

Sample Output

```
Yes, brave Sir Robin turned about
And gallantly he chickened out
Bravely talking to his feet
He beat a very brave retreat
```

Problem 6

Dictionaries

Intro to Python dictionary

We've already used lists and strings to store and process bunch of data. Python also has a variable type to matching one items (i.e. keys) to others (i.e. values) called dictionary. Dictionary is similar to list but instead of automatic index you provide your own index called key. You can assign data to a dictionary as follows: `phones = {'Zoe':'232-43-58', 'Alice':'165-88-56'}`. As with lists for a value could be used any type: string, number, float even dict or list. For keys you can use only strings, numbers, floats and other immutable types. Accessing values also similar to lists:

```
phones = {'Zoe':'232-43-58', 'Alice':'165-88-56'}
print phones['Zoe']
```

Output should be:

```
232-43-58
```

Adding new value to dictionary or assigning an existent can be done the same way as you do it with variable

```
phones['Zoe'] = '658-99-55'
phones['Bill'] = '342-18-25'
print phones
```

You should see the following:

```
{'Bill': '342-18-25', 'Zoe': '658-99-55', 'Alice': '165-88-56'}
```

Note that new `'Bill'` appeared in the beginning not in the end as you might expected. The thing is that dictionary basically does not have any ordering. New value appear in random place.

Remember that the dictionary is case-sensitive if you are using strings as keys. Keep in mind that 'key' and 'Key' are different keys:

```
d = {}
d['key'] = 1
d['Key'] = 2
d['KEY'] = 3
print d
```

Output:

```
{'KEY': 3, 'Key': 2, 'key': 1}
```

Note how we created an empty dictionary with `d = {}`. This could be useful in case you need to add values to dictionary dynamically (for example, when reading a file). If you need to check is there a key in dictionary you can use `key in d` syntax:

```
if 'Peter' in phones:
    print "We know Peter's phone"
else:
    print "We don't know Peter's phone"
```

Output:

```
We don't know Peter's phone
```

In case you need to delete a value from a dictionary please use `del`:

```
phones = {'Zoe':'232-43-58', 'Alice':'165-88-56'}
del phones['Zoe']
print phones
```

Output:

```
{'Alice': '165-88-56'}
```

Problem

Given: A string s of length at most 10000 letters.

Return: How many times any word occurred in string. Each letter case (upper or lower) in word matters. Lines in output can be in any order.

Sample Dataset

```
We tried list and we tried dicts also we tried Zen
```

Sample Output

```
and 1
We 1
tried 3
dicts 1
list 1
we 2
also 1
Zen 1
```

Hints

To iterate over words in string you can split it by space:

```
for word in str.split(' '):
    print word
```

To have nice output of dictionary you can use `.items()` method:

```
for key, value in dict.items():
    print key
    print value
```

Problem 7

Counting DNA Nucleotides

A Rapid Introduction to Molecular Biology

Making up all living material, the **cell** is considered to be the building block of life. The **nucleus**, a component of most **eukaryotic** cells, was identified as the hub of cellular activity 150 years ago. Viewed under a light microscope, the nucleus appears only as a darker region of the cell, but as we increase magnification, we find that the nucleus is densely filled with a stew of macromolecules called **chromatin**. During **mitosis** (eukaryotic cell division), most of the chromatin condenses into long, thin strings called **chromosomes**. See **Figure 1**

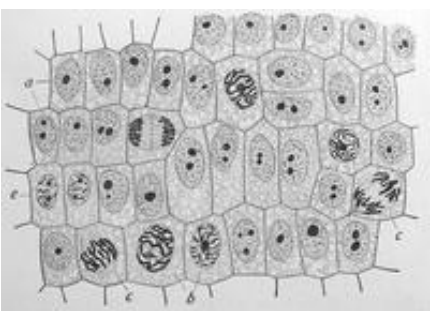


Figure 1. A 1900 drawing by Edmund Wilson of onion cells at different stages of mitosis. The sample has been dyed, causing

for a figure of cells in different stages of mitosis.

chromatin in the cells (which soaks up the dye) to appear in greater contrast to the rest of the cell.

One class of the macromolecules contained in chromatin are called **nucleic acids**. Early 20th century research into the chemical identity of nucleic acids culminated with the conclusion that nucleic acids are **polymers**, or repeating chains of smaller, similarly structured molecules known as **monomers**. Because of their tendency to be long and thin, nucleic acid polymers are commonly called **strands**.

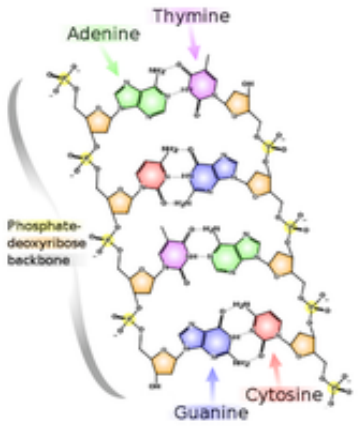


Figure 2. A sketch of DNA's primary structure.

The nucleic acid monomer is called a **nucleotide** and is used as a unit of strand length (abbreviated to nt). Each nucleotide is formed of three parts: a **sugar** molecule, a negatively charged **ion** called a **phosphate**, and a compound called a **nucleobase** ("base" for short). Polymerization is achieved as the sugar of one nucleotide bonds to the phosphate of the next nucleotide in the chain, which forms a **sugar-phosphate backbone** for the nucleic acid strand. A key point is that the nucleotides of a specific type of nucleic acid always contain the same sugar and phosphate molecules, and they differ only in their choice of base. Thus, one strand of a nucleic acid can be differentiated from another based solely on the *order* of its bases; this ordering of bases defines a nucleic acid's **primary structure**.

For example, **Figure 2** shows a strand of **deoxyribose nucleic acid** (DNA), in which the sugar is called **deoxyribose**, and the only four choices for nucleobases are molecules called **adenine** (A), **cytosine** (C), **guanine** (G), and **thymine** (T).

For reasons we will soon see, DNA is found in all living organisms on Earth, including bacteria; it is even found in many viruses (which are often considered to be nonliving). Because of its importance, we reserve the term **genome** to refer to the sum total of the DNA contained in an organism's chromosomes.

Problem

A **string** is simply an ordered collection of symbols selected from some **alphabet** and formed into a word; the **length** of a string is the number of symbols that it contains.

An example of a length 21 **DNA string** (whose alphabet contains the symbols 'A', 'C', 'G', and 'T') is "ATGCTTCAGAAAGGTCTTACG."

Given: A DNA string s of length at most 1000 nt.

Return: Four integers (separated by spaces) counting the respective number of times that the symbols 'A', 'C', 'G', and 'T' occur in s.

Sample Dataset

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
```

Sample Output

```
20 12 17 21
```

Problem 8

Transcribing DNA into RNA

The Second Nucleic Acid

In "Counting DNA Nucleotides", we described the primary structure of a nucleic acid as a polymer of nucleotide units, and we mentioned that the omnipresent nucleic acid DNA is composed of a varied sequence of four bases.

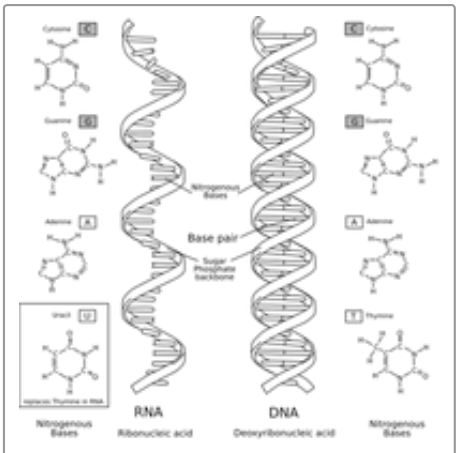


Figure 1. Structural differences between RNA and DNA

Yet a second nucleic acid exists alongside DNA in the chromatin; this molecule, which possesses a different sugar called **ribose**, came to be known as **ribose nucleic acid**, or RNA. RNA differs further from DNA in that it contains a base called **uracil** in place of thymine; structural differences between DNA and RNA are shown in **Figure 1**. Biologists initially believed that RNA was only contained in plant cells, whereas DNA was restricted to animal cells. However, this hypothesis dissipated as improved chemical methods discovered both nucleic acids in the cells of all life forms on Earth.

The primary structure of DNA and RNA is so similar because the former serves as a blueprint for the creation of a special kind of RNA molecule called **messenger RNA**, or mRNA. mRNA is created during **RNA transcription**, during which a strand of DNA is used as a template for constructing a strand of RNA by copying nucleotides one at a time, where uracil is used in place of thymine.

In eukaryotes, DNA remains in the nucleus, while RNA can enter the far reaches of the cell to carry out DNA's instructions. In future problems, we will examine the process and ramifications of RNA transcription in more detail.

Problem

An **RNA string** is a string formed from the alphabet containing 'A', 'C', 'G', and 'U'.

Given a DNA string t corresponding to a coding strand, its transcribed RNA string u is formed by replacing all occurrences of 'T' in t with 'U' in u.

Given: A DNA string t having length at most 1000 nt.

Return: The transcribed RNA string of t.

Sample Dataset

```
GATGGAACTTGACTACGTAAATT
```

Sample Output

```
GAUGGAACUUGACUACGUAAAUU
```

Problem 9

Complementing a Strand of DNA

The Secondary and Tertiary Structures of DNA

In "Counting DNA Nucleotides", we introduced nucleic acids, and we saw that the primary structure of a nucleic acid is determined by the ordering of its nucleobases along the sugar-phosphate backbone that constitutes the bonds of the nucleic acid polymer. Yet primary structure tells us nothing about the larger, 3-dimensional shape of the molecule, which is vital for a complete understanding of nucleic acids.

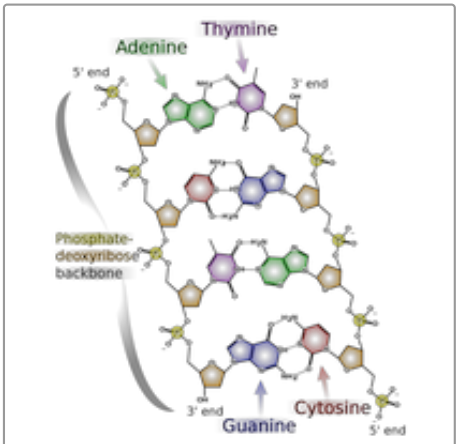


Figure 1. Base pairing across the two strands of DNA.

The search for a complete chemical structure of nucleic acids was central to molecular biology research in the mid-20th Century, culminating in 1953 with a publication in *Nature* of fewer than 800 words by James Watson and Francis Crick. Consolidating a high resolution X-ray image created by Rosalind Franklin and Raymond Gosling with a number of established chemical results, Watson and Crick proposed the following structure for DNA:

1. The DNA molecule is made up of two strands, running in opposite directions.
2. Each base bonds to a base in the opposite strand. Adenine always bonds with thymine, and cytosine always bonds with guanine; the **complement** of a base is the base to which it always bonds; see **Figure 1**.
3. The two strands are twisted together into a long spiral staircase structure called a **double helix**; see **Figure 2**.

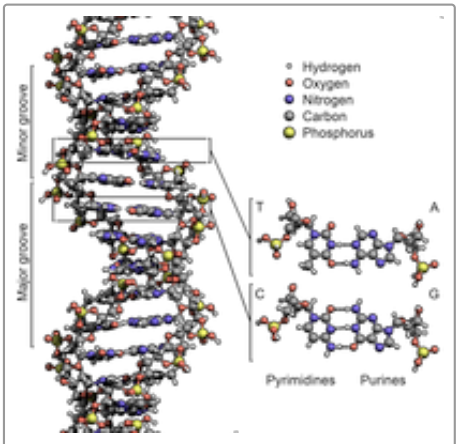


Figure 2. The double helix of DNA on the molecular scale.

Because they dictate how bases from different strands interact with each other, (1) and (2) above compose the **secondary structure** of DNA. (3) describes the 3-dimensional shape of the DNA molecule, or its **tertiary structure**.

In light of Watson and Crick's model, the bonding of two complementary bases is called a **base pair** (bp). Therefore, the length of a DNA molecule will commonly be given in bp instead of nt. By complementarity, once we know the order of bases on one strand, we can immediately deduce the sequence of bases in the complementary strand. These bases will run in the opposite order to match the fact that the two strands of DNA run in opposite directions.

Problem

In DNA strings, symbols 'A' and 'T' are complements of each other, as are 'C' and 'G'.

The **reverse complement** of a DNA string s is the string s^c formed by reversing the symbols of s, then taking the complement of each symbol (e.g., the reverse complement of "GTCA" is "TGAC").

Given: A DNA string s of length at most 1000 bp.

Return: The reverse complement s^c of s.

Sample Dataset

```
AAAACCCGGT
```

Sample Output

```
ACCGGGTTTT
```

Problem 10

Rabbits and Recurrence Relations

Wascally Wabbits

In 1202, Leonardo of Pisa (commonly known as Fibonacci) considered a mathematical exercise regarding the reproduction of a population of rabbits. He made the following simplifying assumptions about the population:

1. The population begins in the first month with a pair of newborn rabbits.
2. Rabbits reach reproductive age after one month.
3. In any given month, every rabbit of reproductive age mates with another rabbit of reproductive age.
4. Exactly one month after two rabbits mate, they produce one male and one female rabbit.
5. Rabbits never die or stop reproducing.

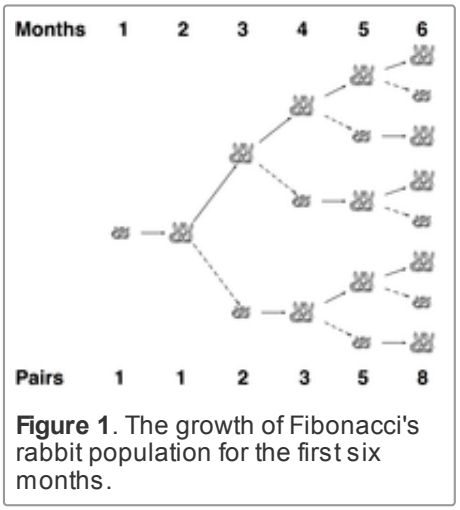


Figure 1. The growth of Fibonacci's rabbit population for the first six months.

Fibonacci's exercise was to calculate how many pairs of rabbits would remain in one year. We can see that in the second month, the first pair of rabbits reach reproductive age and mate. In the third month, another pair of rabbits is born, and we have two rabbit pairs; our first pair of rabbits mates again. In the fourth month, another pair of rabbits is born to the original pair, while the second pair reach maturity and mate (with three total pairs). The dynamics of the rabbit population are illustrated in **Figure 1**. After a year, the rabbit population boasts 144 pairs.

> Although Fibonacci's assumption of the rabbits' immortality may seem a bit farfetched, his model was not unrealistic for reproduction in a predator-free environment: European rabbits were introduced to Australia in the mid 19th Century, a place with no real indigenous predators for them. Within 50 years, the rabbits had already eradicated many plant species across the continent, leading to irreversible changes in the Australian ecosystem and turning much of its grasslands into eroded, practically uninhabitable parts of the modern Outback (see **Figure 2**). In this problem, we will use the simple idea of counting rabbits to introduce a new computational topic, which involves building up large solutions from smaller ones.

Figure 2. Erosion at Lake Mungo in New South Wales, which was initiated by European rabbits in the 19th Century. Courtesy Pierre Pouliquin.

Problem

A **sequence** is an ordered collection of objects (usually numbers), which are allowed to repeat. Sequences can be finite or infinite. Two examples are the finite sequence $(\pi, -\sqrt{2}, 0, \pi)$ and the infinite sequence of odd numbers $(1, 3, 5, 7, 9, \ldots)$. We use the notation a_n to represent the n-th term of a sequence.

A **recurrence relation** is a way of defining the terms of a sequence with respect to the values of previous terms. In the case of Fibonacci's rabbits from the introduction, any given month will contain the rabbits that were alive the previous month, plus any new offspring. A key observation is that the number of offspring in any month is equal to the number of rabbits that were alive two months prior. As a result, if F_n represents the number of rabbit pairs alive after the n-th month, then we obtain the **Fibonacci sequence** having terms F_n that are defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ (with $F_1 = F_2 = 1$ to initiate the sequence). Although the sequence bears Fibonacci's name, it was known to Indian mathematicians over two millennia ago.

When finding the n-th term of a sequence defined by a recurrence relation, we can simply use the recurrence relation to generate terms for progressively larger values of n. This problem introduces us to the computational technique of **dynamic programming**, which successively builds up solutions by using the answers to smaller cases.

Given: Positive integers $n \leq 40$ and $k \leq 5$.

Return: The total number of rabbit pairs that will be present after n months if we begin with 1 pair and in each generation, every pair of reproduction-age rabbits produces a litter of k rabbit pairs (instead of only 1 pair).

Sample Dataset

```
5 3
```

Sample Output

```
19
```

Problem 11

Computing GC Content

Identifying Unknown DNA Quickly

A quick method used by early computer software to determine the language of a given piece of text was to analyze the frequency with which each letter appeared in the text. This strategy was used because each language tends to exhibit its own letter frequencies, and as long as the text under consideration is long enough, software will correctly recognize the language quickly and with a very low error rate. See **Figure 1** for a table compiling English letter frequencies.

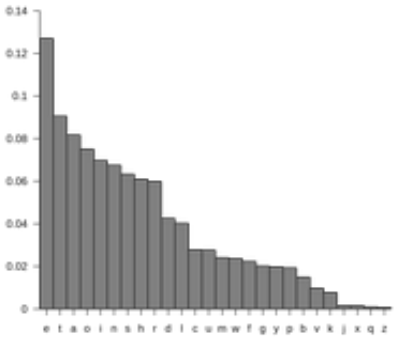


Figure 1. The table above was computed from a large number of English words and shows for any letter the frequency with which it appears in those words. These frequencies can be used to reliably identify a piece of English text and differentiate it from that of another language. Taken from http://en.wikipedia.org/wiki/File:Englis

You may ask: what in the world does this linguistic problem have to do with biology? Although two members of the same species will have different genomes, they still share the vast percentage of their DNA; notably, 99.9% of the 3.2 billion base pairs in a human genome are common to almost all humans (i.e., excluding people having major genetic defects). For this reason, biologists will speak of *the* human genome, meaning an average-case genome derived from a collection of individuals. Such an average case genome can be assembled for any species, a challenge that we will soon discuss.

The biological analog of identifying unknown text arises when researchers encounter a molecule of DNA deriving from an unknown species. Because of the base pairing relations of the two DNA strands, cytosine and guanine will always appear in equal amounts in a double-stranded DNA molecule. Thus, to analyze the symbol frequencies of DNA for comparison against a database, we compute the molecule's **GC-content**, or the percentage of its bases that are *either* cytosine or guanine.

In practice, the GC-content of most eukaryotic genomes hovers around 50%. However, because genomes are so long, we may be able to distinguish species based on very small discrepancies in GC-content; furthermore, most prokaryotes have a GC-content significantly higher than 50%, so that GC-content can be used to quickly differentiate many prokaryotes and eukaryotes by using relatively small DNA samples.

Problem

The GC-content of a DNA string is given by the percentage of symbols in the string that are 'C' or 'G'. For example, the GC-content of "AGCTATAG" is 37.5%. Note that the reverse complement of any DNA string has the same GC-content.

DNA strings must be labeled when they are consolidated into a database. A commonly used method of string labeling is called **FASTA format**. In this format, the string is introduced by a line that begins with '>', followed by some labeling information. Subsequent lines contain the string itself; the first line to begin with '>' indicates the label of the next string.

In Rosalind's implementation, a string in FASTA format will be labeled by the ID "Rosalind_xxxx", where "xxxx" denotes a four-digit code between 0000 and 9999.

Given: At most 10 DNA strings in FASTA format (of length at most 1 kbp each).

Return: The ID of the string having the highest GC-content, followed by the GC-content of that string. Rosalind allows for a default error of 0.001 in all decimal answers unless otherwise stated; please see the note on **absolute error** below.

Sample Dataset

```
>Rosalind_6404
CCTGCGGAAGATCGGCACTAGAATAGCCAGAACCGTTTCTCTGAGGCTTCCGGCCTTCCC
TCCCACTAATAATTCTGAGG
>Rosalind_5959
CCATCGGTAGCGCATCCTTAGTCCAATTAAGTCCCTATCCAGGCGCTCCGCCGAAGGTCT
ATATCCATTTGTCAGCAGACACGC
>Rosalind_0808
CCACCCTCGTGGTATGGCTAGGCATTCAGGAACCGGAGAACGCTTCAGACCAGCCCGGAC
TGGGAACCTGCGGGCAGTAGGTGGAAT
```

Sample Output

```
Rosalind_0808
60.919540
```

> ### Note on Absolute Error
>
> We say that a number x is within an absolute error of y to a correct solution if x is within y of the correct solution. For example, if an exact solution is 6.157892, then for x to be within an absolute error of 0.001, we must have that $|x - 6.157892| < 0.001$, or $6.156892 < x < 6.158892$.
>
> Error bounding is a vital practical tool because of the inherent round-off error in representing decimals in a computer, where only a finite number of decimal places are allotted to any number. After being compounded over a number of operations, this round-off error can become evident. As a result, rather than testing whether two numbers are equal with $x = z$, you may wish to simply verify that $|x - z|$ is very small.
>
> The mathematical field of **numerical analysis** is devoted to rigorously studying the nature of computational approximation.

Problem 12

Counting Point Mutations

> ### Evolution as a Sequence of Mistakes

A **mutation** is simply a mistake that occurs during the creation or copying of a nucleic acid, in particular DNA. Because nucleic acids are vital to cellular functions, mutations tend to cause a ripple effect throughout the cell. Although mutations are technically mistakes, a very rare mutation may equip the cell with a beneficial attribute. In fact, the macro effects of evolution are attributable by the accumulated result of beneficial microscopic mutations over many generations.

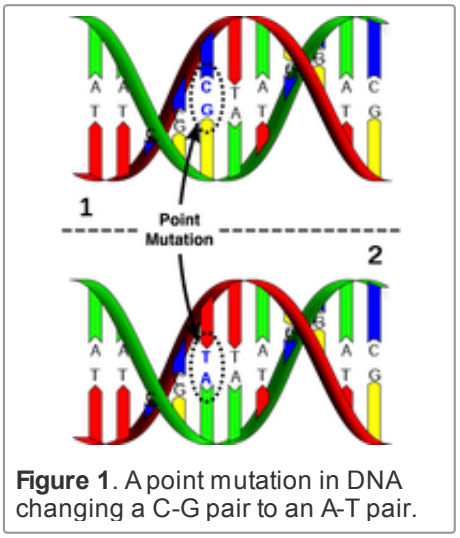


Figure 1. A point mutation in DNA changing a C-G pair to an A-T pair.

The simplest and most common type of nucleic acid mutation is a **point mutation**, which replaces one base with another at a single nucleotide. In the case of DNA, a point mutation must change the complementary base accordingly; see **Figure 1**.

Two DNA strands taken from different organism or species genomes are **homologous** if they share a recent ancestor; thus, counting the number of bases at which homologous strands differ provides us with the minimum number of point mutations that could have occurred on the evolutionary path between the two strands.

We are interested in minimizing the number of (point) mutations separating two species because of the biological principle of **parsimony**, which demands that evolutionary histories should be as simply explained as possible.

Problem

Given two strings s and t of equal length, the **Hamming distance** between s and t, denoted $d_{\mathrm{H}}(s, t)$, is the number of corresponding symbols that differ in s and t. See **Figure 2**.

Given: Two DNA strings s and t of equal length (not exceeding 1 kbp).

Return: The Hamming distance $d_{\mathrm{H}}(s, t)$.

GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT

Figure 2. The Hamming distance between these two strings is 7. Mismatched symbols are colored red.

Sample Dataset

```
GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT
```

Sample Output

```
7
```

Problem 13

Mendel's First Law

Introduction to Mendelian Inheritance

Modern laws of inheritance were first described by Gregor Mendel (an Augustinian Friar) in 1865. The contemporary hereditary model, called **blending inheritance**, stated that an organism must exhibit a blend of its parent's traits. This rule is obviously violated both empirically (consider the huge number of people who are taller than both their parents) and statistically (over time, blended traits would simply blend into the average, severely limiting variation).

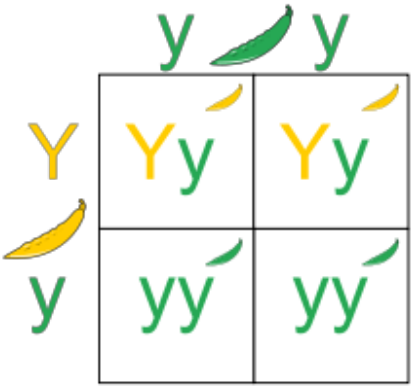


Figure 1. A Punnett square representing the possible outcomes of crossing a heterozygous organism (Yy) with a homozygous recessive organism (yy); here, the dominant allele Y corresponds to yellow pea pods, and the recessive allele y corresponds to green pea pods.

Mendel, working with thousands of pea plants, believed that rather than viewing traits as continuous processes, they should instead be divided into discrete building blocks called **factors**. Furthermore, he proposed that every factor possesses distinct forms, called **alleles**.

In what has come to be known as his **first law** (also known as the law of segregation), Mendel stated that every organism possesses a pair of alleles for a given factor. If an individual's two alleles for a given factor are the same, then it is **homozygous** for the factor; if the alleles differ, then the individual is **heterozygous**. The first law concludes that for any factor, an organism randomly passes one of its two alleles to each offspring, so that an individual receives one allele from each parent.

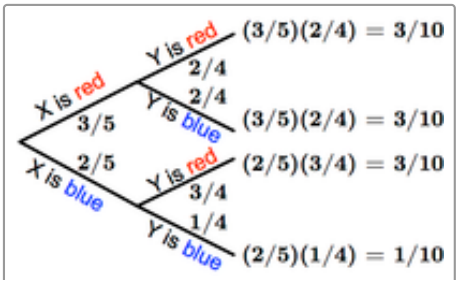
Mendel also believed that any factor corresponds to only two possible alleles, the **dominant** and **recessive** alleles. An organism only needs to possess one copy of the dominant allele to display the trait represented by the dominant allele. In other words, the only way that an organism can display a trait encoded by a recessive allele is if the individual is homozygous recessive for that factor.

We may encode the dominant allele of a factor by a capital letter (e.g., A) and the recessive allele by a lower case letter (e.g., a). Because a heterozygous organism can possess a recessive allele without displaying the recessive form of the trait, we henceforth define an organism's **genotype** to be its precise genetic makeup and its **phenotype** as the physical manifestation of its underlying traits.

The different possibilities describing an individual's inheritance of two alleles from its parents can be represented by a **Punnett square**; see **Figure 1** for an example.

Problem

Probability is the mathematical study of randomly occurring phenomena. We will model such a phenomenon with a **random variable**, which is simply a variable that can take a number of different distinct **outcomes** depending on the result of an underlying random process.



For example, say that we have a bag containing 3 red balls

and 2 blue balls. If we let X represent the random variable corresponding to the color of a drawn ball, then the **probability** of each of the two outcomes is given by $\Pr(X = \text{red}) = \frac{3}{5}$ and $\Pr(X = \text{blue}) = \frac{2}{5}$.

Figure 2. The probability of any outcome (leaf) in a probability tree diagram is given by the product of probabilities from the start of the tree to the outcome. For example, the probability that X is blue and Y is blue is equal to (2/5)(1/4), or 1/10.

Random variables can be combined to yield new random variables. Returning to the ball example, let Y model the color of a second ball drawn from the bag (without replacing the first ball). The probability of Y being red depends on whether the first ball was red or blue. To represent all outcomes of X and Y, we therefore use a **probability tree diagram**. This branching diagram represents all possible individual probabilities for X and Y, with outcomes at the endpoints ("leaves") of the tree. The probability of any outcome is given by the product of probabilities along the path from the beginning of the tree; see **Figure 2** for an illustrative example.

An **event** is simply a collection of outcomes. Because outcomes are distinct, the probability of an event can be written as the sum of the probabilities of its constituent outcomes. For our colored ball example, let A be the event "Y is blue." $\Pr(A)$ is equal to the sum of the probabilities of two different outcomes: $\Pr(X = \text{blue and } Y = \text{blue}) + \Pr(X = \text{red and } Y = \text{blue})$, or $\frac{3}{10} + \frac{1}{10} = \frac{2}{5}$ (see Figure 2 above).

Given: Three positive integers k, m, and n, representing a population containing $k + m + n$ organisms: k individuals are homozygous dominant for a factor, m are heterozygous, and n are homozygous recessive.

Return: The probability that two randomly selected mating organisms will produce an individual possessing a dominant allele (and thus displaying the dominant phenotype). Assume that any two organisms can mate.

Sample Dataset

```
2 2 2
```

Sample Output

```
0.78333
```

> ### Hint
>
> Consider simulating inheritance on a number of small test cases in order to check your solution.

Problem 14

Translating RNA into Protein

> ### The Genetic Code

Just as nucleic acids are polymers of nucleotides, **proteins** are chains of smaller molecules called **amino acids**; 20 amino acids commonly appear in every species. Just as the primary structure of a nucleic acid is given by the order of its nucleotides, the **primary structure** of a protein is the order of its amino acids. Some proteins are composed of several subchains called **polypeptides**, while others are formed of a single polypeptide; see **Figure 1**.

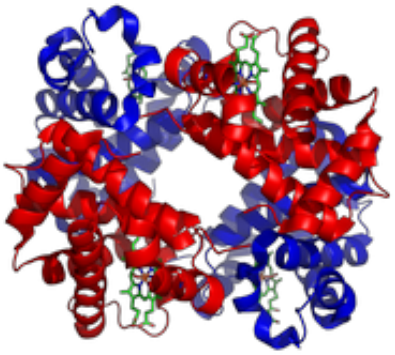


Figure 1. The human hemoglobin molecule consists of 4 polypeptide chains; α subunits are shown in red and β subunits are shown in blue

Proteins power every practical function carried out by the cell, and so presumably, the key to understanding life lies in interpreting the relationship between a chain of amino acids and the function of the protein that this chain of amino acids eventually constructs. **Proteomics** is the field devoted to the study of proteins.

How are proteins created? The **genetic code**, discovered throughout the course of a number of ingenious experiments in the late 1950s, details the **translation** of an RNA molecule called **messenger RNA** (mRNA) into amino acids for protein creation. The apparent difficulty in translation is that somehow 4 RNA bases must be translated into a language of 20 amino acids; in order for every possible amino acid to be created, we must translate 3-nucleobase strings (called **codons**) into amino acids. Note that there are $4^3 = 64$ possible codons, so that multiple codons may encode the same amino acid. Two special types of codons are the **start codon** (AUG), which codes for the amino acid methionine always indicates the start of translation, and the three **stop codons** (UAA, UAG, UGA), which do not code for an amino acid and cause translation to end.

The notion that protein is always created from RNA, which in turn is always created from DNA, forms the **central dogma of molecular biology**. Like all dogmas, it does not always hold; however, it offers an excellent approximation of the truth.

A eukaryotic organelle called a **ribosome** creates peptides by using a helper molecule called **transfer RNA** (tRNA). A single tRNA molecule possesses a string of three RNA nucleotides on one end (called an **anticodon**) and an amino acid at the other end. The ribosome takes an RNA molecule transcribed from DNA (see "Transcribing DNA into RNA"), called **messenger RNA** (mRNA), and examines it one codon at a time. At each step, the tRNA possessing the complementary anticodon bonds to the mRNA at this location, and the amino acid found on the opposite end of the tRNA is added to the growing peptide chain before the remaining part of the tRNA is ejected into the cell, and the ribosome looks for the next tRNA molecule.

Not every RNA base eventually becomes translated into a protein, and so an interval of RNA (or an interval of DNA translated into RNA) that does code for a protein is of great biological interest; such an interval of DNA or RNA is called a **gene**. Because protein creation drives cellular processes, genes differentiate organisms and serve as a basis for **heredity**, or the process by which traits are inherited.

Problem

The 20 commonly occurring amino acids are abbreviated by using 20 letters from the English alphabet (all letters except for B, J, O, U, X, and Z). **Protein strings** are constructed from these 20 symbols. Henceforth, the term **genetic string** will incorporate protein strings along with DNA strings and RNA strings.

The **RNA codon table** dictates the details regarding the encoding of specific codons into the amino

acid alphabet.

Given: An RNA string s corresponding to a strand of mRNA (of length at most 10 kbp).

Return: The protein string encoded by s.

Sample Dataset

```
AUGGCCAUGGCGCCCAGAACUGAGAUCAAUAGUACCCGUAUUAACGGGUGA
```

Sample Output

```
MAMAPRTEINSTRING
```

Problem 15

Finding a Motif in DNA

Combing Through the Haystack

Finding the same interval of DNA in the genomes of two different organisms (often taken from different species) is highly suggestive that the interval has the same function in both organisms.

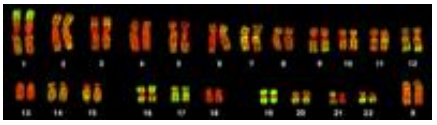


Figure 1. The human chromosomes stained with a probe for Alu elements, shown in green.

We define a **motif** as such a commonly shared interval of DNA. A common task in molecular biology is to search an organism's genome for a known motif.

The situation is complicated by the fact that genomes are riddled with intervals of DNA that occur multiple times (possibly with slight modifications), called **repeats**. These repeats occur far more often than would be dictated by random chance, indicating that genomes are anything but random and in fact illustrate that the language of DNA must be very powerful (compare with the frequent reuse of common words in any human language).

The most common repeat in humans is the **Alu repeat**, which is approximately 300 bp long and recurs around a million times throughout every human genome (see **Figure 1**). However, Alu has not been found to serve a positive purpose, and appears in fact to be parasitic: when a new Alu repeat is inserted into a genome, it frequently causes genetic disorders.

Problem

Given two strings s and t, t is a **substring** of s if t is contained as a contiguous collection of symbols in s (as a result, t must be no longer than s).

The **position** of a symbol in a string is the total number of symbols found to its left, including itself (e.g., the positions of all occurrences of 'U' in "AUGCUUCAGAAAGGUCUUACG" are 2, 5, 6, 15, 17,

and 18). The symbol at position i of s is denoted by $s[i]$.

A substring of s can be represented as $s[j:k]$, where j and k represent the starting and ending positions of the substring in s; for example, if s = "AUGCUUCAGAAAGGUCUUACG", then $s[2:5]$ = "UGCU".

The **location** of a substring $s[j:k]$ is its beginning position j; note that t will have multiple locations in s if it occurs more than once as a substring of s (see the Sample below).

Given: Two DNA strings s and t (each of length at most 1 kbp).

Return: All locations of t as a substring of s.

Sample Dataset

```
GATATATGCATATACTT
ATAT
```

Sample Output

```
2 4 10
```

> ### Note
>
> Different programming languages use different notations for positions of symbols in strings. Above, we use **1-based numbering**, as opposed to **0-based numbering**, which is used in Python. For s = "AUGCUUCAGAAAGGUCUUACG", 1-based numbering would state that $s[1]$ = 'A' is the first symbol of the string, whereas this symbol is represented by $s[0]$ in 0-based numbering. The idea of 0-based numbering propagates to substring indexing, so that $s[2:5]$ becomes "GCUU" instead of "UGCU".
>
> Note that in some programming languages, such as Python, s[j:k] returns only fragment from index j up to but *not* including index k, so that s[2:5] actually becomes "UGC", not "UGCU".

Problem 16

Consensus and Profile

Finding a Most Likely Common Ancestor

In "Counting Point Mutations", we calculated the minimum number of symbol mismatches between two strings of equal length to model the problem of finding the minimum number of point mutations occurring on the evolutionary path between two homologous strands of DNA. If we instead have several homologous strands that we wish to analyze simultaneously, then the natural problem is to find an average-case strand to represent the most likely common ancestor of the given strands.

Problem

A **matrix** is a rectangular table of values divided into rows and columns. An $m \times n$ matrix has m rows and n columns. Given a matrix A, we write $A_{i,j}$ to indicate the value found at the intersection of row i and column j.

Say that we have a collection of DNA strings, all having the same length n. Their **profile matrix** is a $4 \times n$ matrix P in which $P_{1,j}$ represents the number of times that 'A' occurs in the jth position of one of the strings, $P_{2,j}$ represents the number of times that C occurs in the jth position, and so on (see below).

A **consensus string** c is a string of length n formed from our collection by taking the most common symbol at each position; the jth symbol of c therefore corresponds to the symbol having the maximum value in the j-th column of the profile matrix. Of course, there may be more than one most common symbol, leading to multiple possible consensus strings.

		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
DNA Strings		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
	A	5	1	0	0	5	5	0	0
Profile	**C**	0	0	1	4	2	0	6	1
	G	1	1	6	3	0	1	0	0
	T	1	5	0	0	0	1	1	6
Consensus		A	T	G	C	A	A	C	T

Given: A collection of at most 10 DNA strings of equal length (at most 1 kbp) in FASTA format.

Return: A consensus string and profile matrix for the collection. (If several possible consensus strings exist, then you may return any one of them.)

Sample Dataset

```
>Rosalind_1
ATCCAGCT
>Rosalind_2
GGGCAACT
>Rosalind_3
ATGGATCT
>Rosalind_4
AAGCAACC
>Rosalind_5
TTGGAACT
>Rosalind_6
ATGCCATT
>Rosalind_7
```

```
ATGGCACT
```

Sample Output

```
ATGCAACT
A: 5 1 0 0 5 5 0 0
C: 0 0 1 4 2 0 6 1
G: 1 1 6 3 0 1 0 0
T: 1 5 0 0 0 1 1 6
```

Problem 17

Mortal Fibonacci Rabbits

Wabbit Season

In "Rabbits and Recurrence Relations", we mentioned the disaster caused by introducing European rabbits into Australia. By the turn of the 20th Century, the situation was so out of control that the creatures could not be killed fast enough to slow their spread (see **Figure 1**).

Figure 1. A c.1905 photo from Australia of a cart loaded to the hilt with rabbit skins.

The conclusion? Build a fence! The fence, intended to preserve the sanctity of Western Australia, was completed in 1907 after undergoing revisions to push it back as the bunnies pushed their frontier ever westward (see **Figure 2**). If it sounds like a crazy plan, the Australians at the time seem to have concurred, as shown by the cartoon in **Figure 3**.

Figure 2. Western Australia's rabbit fence is actually not the longest fence in the world as the sign claims. That honor goes to a 3,500 mile fence in southeastern Australia built to keep out dingoes. Courtesy Matt Pounsett.

By 1950, Australian rabbits numbered 600 million, causing the government to decide to release a virus (called myxoma) into the wild, which cut down the rabbits until they acquired resistance. In a final Hollywood twist, another experimental rabbit virus escaped in 1991, and some resistance has already been observed.

The bunnies will not be stopped, but they don't live forever, and so in this problem, our aim is to expand Fibonacci's rabbit population model to allow for mortal rabbits.

Problem

Recall the definition of the Fibonacci numbers from "Rabbits and Recurrence Relations", which followed the recurrence relation $F_n = F_{n-1} + F_{n-2}$ and assumed that each pair of rabbits reaches maturity

in one month and produces a single pair of offspring (one male, one female) each subsequent month.

Our aim is to somehow modify this recurrence relation to achieve a dynamic programming solution in the case that all rabbits die out after a fixed number of months. See **Figure 4** for a depiction of a rabbit tree in which rabbits live for three months (meaning that they reproduce only twice before dying).

Given: Positive integers $n \leq 100$ and $m \leq 20$.

Return: The total number of pairs of rabbits that will remain after the n-th month if all rabbits live for m months.

Sample Dataset

```
6 3
```

Sample Output

```
4
```

Figure 3. An 1884 cartoon from the Queensland Figaro proposing how the rabbits viewed their fence.

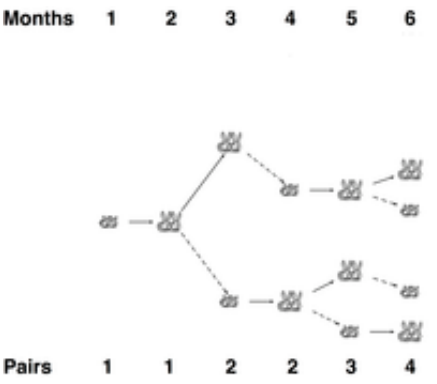


Figure 4. A figure illustrating the propagation of Fibonacci's rabbits if they die after three months.

Problem 18

Overlap Graphs

A Brief Introduction to Graph Theory

Networks arise everywhere in the practical world, especially in biology. Networks are prevalent in popular applications such as modeling the spread of disease, but the extent of network applications spreads far beyond popular science. Our first question asks how to computationally model a network without actually needing to render a picture of the network.

First, some terminology: **graph** is the technical term for a network; a graph is made up of hubs called **nodes** (or vertices), pairs of which are connected via segments/curves called **edges**. If an edge connects nodes v and w, then it is denoted by v, w (or equivalently w, v).

- an edge v, w is **incident** to nodes v and w; we say that v and w are **adjacent** to each other;
- the **degree** of v is the number of edges incident to it;
- a **walk** is an ordered collection of edges for which the ending node of one edge is the starting node of the next (e.g., $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\{v_3, v_4\}$, etc.);
- a **path** is a walk in which every node appears in at most two edges;
- **path length** is the number of edges in the path;
- a **cycle** is a path whose final node is equal to its first node (so that every node is incident to exactly two edges in the cycle); and

- the **distance** between two vertices is the length of the shortest path connecting them.

Graph theory is the abstract mathematical study of graphs and their properties.

Problem

A graph whose nodes have all been labeled can be represented by an **adjacency list**, in which each row of the list contains the two node labels corresponding to a unique edge.

A **directed graph** (or digraph) is a graph containing **directed edges**, each of which has an orientation. That is, a directed edge is represented by an arrow instead of a line segment; the starting and ending nodes of an edge form its **tail** and **head**, respectively. The directed edge with tail v and head w is represented by (v, w) (but *not* by (w, v)). A **directed loop** is a directed edge of the form (v, v).

For a collection of strings and a positive integer k, the **overlap graph** for the strings is a directed graph O_k in which each string is represented by a node, and string s is connected to string t with a directed edge when there is a length k suffix of s that matches a length k prefix of t, as long as $s \neq t$; we demand $s \neq t$ to prevent directed loops in the overlap graph (although directed cycles may be present).

Given: A collection of DNA strings in FASTA format having total length at most 10 kbp.

Return: The adjacency list corresponding to O_3. You may return edges in any order.

Sample Dataset

```
>Rosalind_0498
AAATAAA
>Rosalind_2391
AAATTTT
>Rosalind_2323
TTTTCCC
>Rosalind_0442
AAATCCC
>Rosalind_5013
GGGTGGG
```

Sample Output

```
Rosalind_0498 Rosalind_2391
Rosalind_0498 Rosalind_0442
Rosalind_2391 Rosalind_2323
```

Note on Visualizing Graphs

If you are looking for a way to actually visualize graphs as you are working through the Rosalind site, then you may like to consider **Graphviz** (link here), a cross-platform application for rendering graphs.

Problem 19

Calculating Expected Offspring

The Need for Averages

Averages arise everywhere. In sports, we want to project the average number of games that a team is expected to win; in gambling, we want to project the average losses incurred playing blackjack; in business, companies want to calculate their average expected sales for the next quarter.

> Molecular biology is not immune from the need for averages. Researchers need to predict the expected number of antibiotic-resistant pathogenic bacteria in a future outbreak, estimate the predicted number of locations in the genome that will match a given motif, and study the distribution of alleles throughout an evolving population. In this problem, we will begin discussing the third issue; first, we need to have a better understanding of what it means to average a random process.

Problem

For a random variable X taking integer values between 1 and n, the **expected value** of X is $\mathrm{E}(X) = \sum_{k=1}^{n} k \times \Pr(X = k)$. The expected value offers us a way of taking the long-term average of a random variable over a large number of trials.

As a motivating example, let X be the number on a six-sided die. Over a large number of rolls, we should expect to obtain an average of 3.5 on the die (even though it's not possible to roll a 3.5). The formula for expected value confirms that $\mathrm{E}(X) = \sum_{k=1}^{6} k \times \Pr(X = k) = 3.5$.

More generally, a random variable for which every one of a number of equally spaced outcomes has the same probability is called a **uniform random variable** (in the die example, this "equal spacing" is equal to 1). We can generalize our die example to find that if X is a uniform random variable with minimum possible value a and maximum possible value b, then $\mathrm{E}(X) = \frac{a+b}{2}$. You may also wish to verify that for the dice example, if Y is the random variable associated with the outcome of a second die roll, then $\mathrm{E}(X + Y) = 7$.

Given: Six positive integers, each of which does not exceed 20,000. The integers correspond to the number of couples in a population possessing each genotype pairing for a given factor. In order, the six given integers represent the number of couples having the following genotypes:

1. AA-AA
2. AA-Aa
3. AA-aa
4. Aa-Aa
5. Aa-aa
6. aa-aa

Return: The expected number of offspring displaying the dominant phenotype in the next generation, under the assumption that every couple has exactly two offspring.

Sample Dataset

```
1 0 0 1 0 1
```

Sample Output

```
3.5
```

Problem 20

Finding a Shared Motif

Searching Through the Haystack

In "Finding a Motif in DNA", we searched a given genetic string for a motif; however, this problem assumed that we know the motif in advance. In practice, biologists often do not know exactly what they are looking for. Rather, they must hunt through several different genomes at the same time to identify regions of similarity that may indicate genes shared by different organisms or species.

> The simplest such region of similarity is a motif occurring without mutation in every one of a collection of genetic strings taken from a database; such a motif corresponds to a substring shared by all the strings. We want to search for long shared substrings, as a longer motif will likely indicate a greater shared function.

Problem

A **common substring** of a collection of strings is a substring of every member of the collection. We say that a common substring is a **longest common substring** if there does not exist a longer common substring. For example, "CG" is a common substring of "A**CG**TACGT" and "AAC**CG**GTATA", but it is not as long as possible; in this case, "GTA" is a longest common substring of "AC**GTA**CGT" and "AACC**GTA**TA".

Note that the longest common substring is not necessarily unique; for a simple example, "AA" and "CC" are both longest common substrings of "AACC" and "CCAA".

Given: A collection of k ($k \leq 100$) DNA strings of length at most 1 kbp each in FASTA format.

Return: A longest common substring of the collection. (If multiple solutions exist, you may return any single solution.)

Sample Dataset

```
>Rosalind_1
GATTACA
>Rosalind_2
TAGACCA
>Rosalind_3
```

```
ATACA
```

Sample Output

```
AC
```

Problem 21

Independent Alleles

Mendel's Second Law

Recall that Mendel's first law states that for any factor, an individual randomly assigns one of its two alleles to its offspring. Yet this law does not state anything regarding the relationship with which alleles for different factors will be inherited.

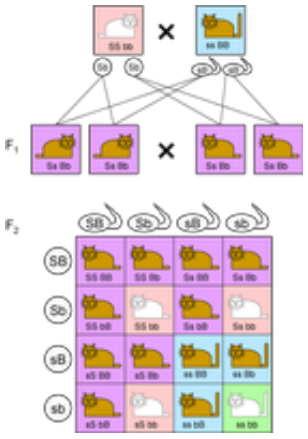


Figure 1. Mendel's second law dictates that every one of the 16 possible assignments of parental alleles is equally likely. The Punnett square for two factors therefore places each of these assignments in a cell of a 4 X 4 table. The probability of an offspring's genome is equal to the number of times it appears in the table, divided by 16.

After recording the results of crossing thousands of pea plants for *seven years*, Mendel surmised that alleles for different factors are inherited with no dependence on each other. This statement has become his **second law**, also known as the law of independent assortment.

What does it mean for factors to be "assorted independently?" If we cross two organisms, then a shortened form of independent assortment states that if we look only at organisms having the same alleles for one factor, then the inheritance of another factor should not change.

For example, Mendel's first law states that if we cross two Aa organisms, then 1/4 of their offspring will be aa, 1/4 will be AA, and 1/2 will be Aa. Now, say that we cross plants that are both heterozygous for two factors, so that both of their genotypes may be written as $\mathrm{Aa\ Bb}$. Next, examine only Bb offspring: Mendel's second law states that the same proportions of AA, Aa, and aa individuals will be observed in these offspring. The same fact holds for BB and bb offspring.

As a result, independence will allow us to say that the probability of an $\mathrm{aa\ BB}$ offspring is simply equal to the probability of an aa offspring times the probability of a BB organism, i.e., 1/16.

Because of independence, we can also extend the idea of Punnett squares to multiple factors, as shown in **Figure 1**. We now wish to quantify Mendel's notion of independence using probability.

Problem

Two events A and B are **independent** if $\Pr(A \text{ and } B)$ is equal to $\Pr(A) \times \Pr(B)$. In other words,

the events do not influence each other, so that we may simply calculate each of the individual probabilities separately and then multiply.

More generally, random variables X and Y are **independent** if whenever A and B are respective events for X and Y, A and B are independent (i.e., $\Pr(A \text{ and } B) = \Pr(A) \times \Pr(B)$).

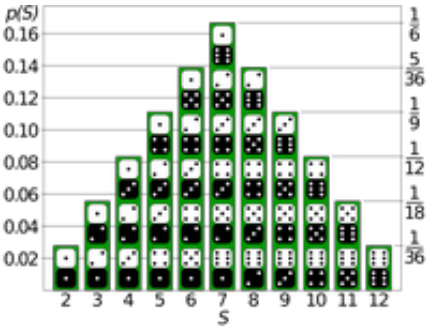


Figure 2. The probability of each outcome for the sum of the values on two rolled dice (black and white), broken down depending on the number of pips showing on each die. You can verify that 18 of the 36 equally probable possibilities result in an odd sum.

As an example of how helpful independence can be for calculating probabilities, let X and Y represent the numbers showing on two six-sided dice. Intuitively, the number of pips showing on one die should not affect the number showing on the other die. If we want to find the probability that $X + Y$ is odd, then we don't need to draw a tree diagram and consider all possibilities. We simply first note that for $X + Y$ to be odd, either X is even and Y is odd or X is odd and Y is even. In terms of probability, $\Pr(X + Y \text{ is odd}) = \Pr(X \text{ is even and } Y \text{ is odd}) + \Pr(X \text{ is odd and } Y \text{ is even})$. Using independence, this becomes

$[\Pr(X \text{ is even}) \times \Pr(Y \text{ is odd})] + [\Pr(X \text{ is odd}) \times \Pr(Y \text{ is even})]$, or $\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 = \frac{1}{2}$.

You can verify this result in **Figure 2**, which shows all 36 outcomes for rolling two dice.

Given: Two positive integers k ($k \leq 7$) and N ($N \leq 2^k$). In this problem, we begin with Tom, who in the 0th generation has genotype Aa Bb. Tom has two children in the 1st generation, each of whom has two children, and so on. Each organism always mates with an organism having genotype Aa Bb.

Return: The probability that at least N Aa Bb organisms will belong to the k-th generation of Tom's family tree (don't count the Aa Bb mates at each level). Assume that Mendel's second law holds for the factors.

Sample Dataset

```
2 1
```

Sample Output

```
0.684
```

An Example of Dependent Random Variables

Two random variables are **dependent** if they are not independent. For an example of dependent random variables, recall our example in "Mendel's First Law" of drawing two balls from a bag containing 3 red balls and 2 blue balls. If X represents the color of the first ball drawn and Y is the color of the second ball drawn (without replacement), then let A be the event "X is red" and B be the event Y is blue. In this case, the probability tree diagram illustrated in **Figure 3** demonstrates that $\Pr(A \text{ and } B) = \frac{3}{10}$. Yet we can also see that

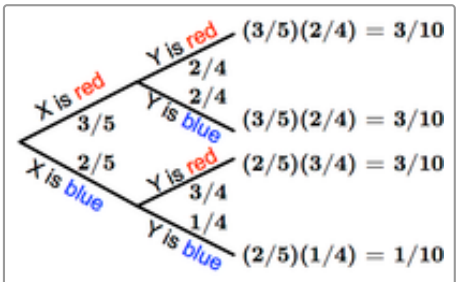


Figure 3. The probability of any outcome (leaf) in a probability tree diagram is given by the product of probabilities from the start of the tree

$\Pr(A) = \frac{3}{5}$ and $\Pr(B) = \frac{3}{10} + \frac{1}{10} = \frac{2}{5}$. We can now see that $\Pr(A \text{ and } B) \neq \Pr(A) \times \Pr(B)$.

to the outcome. For example, the probability that X is blue and Y is red is equal to (2/5)(1/4), or 1/10.

Problem 22

Finding a Protein Motif

Motif Implies Function

As mentioned in "Translating RNA into Protein", proteins perform every practical function in the cell. A structural and functional unit of the protein is a **domain**: in terms of the protein's primary structure, the domain is an interval of amino acids that can evolve and function independently.

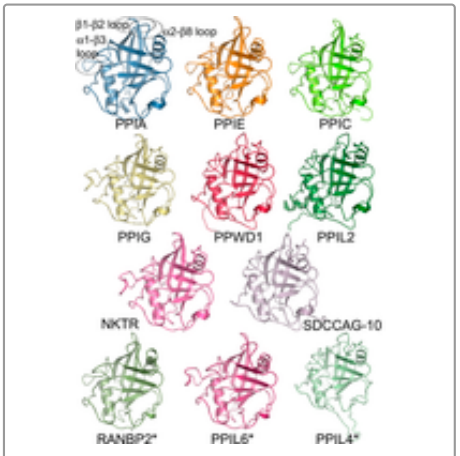


Figure 1. The human cyclophilin family, as represented by the structures of the isomerase domains of some of its members.

Each domain usually corresponds to a single function of the protein (e.g., binding the protein to DNA, creating or breaking specific chemical bonds, etc.). Some proteins, such as myoglobin and the Cytochrome complex, have only one domain, but many proteins are multifunctional and therefore possess several domains. It is even possible to artificially fuse different domains into a protein molecule with definite properties, creating a **chimeric protein**.

Just like species, proteins can evolve, forming homologous groups called **protein families**. Proteins from one family usually have the same set of domains, performing similar functions; see **Figure 1**.

A component of a domain essential for its function is called a motif, a term that in general has the same meaning as it does in nucleic acids, although many other terms are also used (blocks, signatures, fingerprints, etc.) Usually protein motifs are evolutionarily conservative, meaning that they appear without much change in different species.

Proteins are identified in different labs around the world and gathered into freely accessible databases. A central repository for protein data is UniProt, which provides detailed protein annotation, including function description, domain structure, and post-translational modifications. UniProt also supports protein similarity search, taxonomy analysis, and literature citations.

Problem

To allow for the presence of its varying forms, a protein motif is represented by a shorthand as follows: [XY] means "either X or Y" and {X} means "any amino acid except X." For example, the N-glycosylation motif is written as N{P}[ST]{P}.

You can see the complete description and features of a particular protein by its access ID "uniprot_id" in the UniProt database, by inserting the ID number into

```
http://www.uniprot.org/uniprot/uniprot_id
```

Alternatively, you can obtain a protein sequence in FASTA format by following

```
http://www.uniprot.org/uniprot/uniprot_id.fasta
```

For example, the data for protein B5ZC00 can be found at http://www.uniprot.org/uniprot/B5ZC00.

Given: At most 15 UniProt Protein Database access IDs.

Return: For each protein possessing the N-glycosylation motif, output its given access ID followed by a list of locations in the protein string where the motif can be found.

Sample Dataset

```
A2Z669
B5ZC00
P07204_TRBM_HUMAN
P20840_SAG1_YEAST
```

Sample Output

```
B5ZC00
85 118 142 306 395
P07204_TRBM_HUMAN
47 115 116 382 409
P20840_SAG1_YEAST
79 109 135 248 306 348 364 402 485 501 614
```

> ### Note
>
> Some entries in UniProt have one primary (citable) accession number and some secondary numbers, appearing due to merging or demerging entries. In this problem, you may be given any type of ID. If you type the secondary ID into the UniProt query, then you will be automatically redirected to the page containing the primary ID. You can find more information about UniProt IDs here.

Problem 23

Inferring mRNA from Protein

> ### Pitfalls of Reversing Translation
>
> When researchers discover a new protein, they would like to infer the strand of mRNA from which this protein could have been translated, thus allowing them to locate genes associated with this

protein on the genome.

Unfortunately, although any RNA string can be translated into a unique protein string, reversing the process yields a huge number of possible RNA strings from a single protein string because most amino acids correspond to multiple RNA codons (see the RNA Codon Table).

> Because of memory considerations, most data formats that are built into languages have upper bounds on how large an integer can be: in some versions of Python, an "int" variable may be required to be no larger than $2^{31} - 1$, or 2,147,483,647. As a result, to deal with very large numbers in Rosalind, we need to devise a system that allows us to manipulate large numbers without actually having to store large numbers.

Problem

For positive integers a and n, a **modulo** n (written $a \mod n$ in shorthand) is the remainder when a is divided by n. For example, $29 \mod 11 = 7$ because $29 = 11 \times 2 + 7$.

Modular arithmetic is the study of addition, subtraction, multiplication, and division with respect to the modulo operation. We say that a and b are **congruent** modulo n if $a \mod n = b \mod n$; in this case, we use the notation $a \equiv b \mod n$.

Two useful facts in modular arithmetic are that if $a \equiv b \mod n$ and $c \equiv d \mod n$, then $a + c \equiv b + d \mod n$ and $a \times c \equiv b \times d \mod n$. To check your understanding of these rules, you may wish to verify these relationships for $a = 29$, $b = 73$, $c = 10$, $d = 32$, and $n = 11$.

As you will see in this exercise, some Rosalind problems will ask for a (very large) integer solution modulo a smaller number to avoid the computational pitfalls that arise with storing such large numbers.

Given: A protein string of length at most 1000 aa.

Return: The total number of different RNA strings from which the protein could have been translated, modulo 1,000,000. (Don't neglect the importance of the stop codon in protein translation.)

Sample Dataset

```
MA
```

Sample Output

```
12
```

> #### Hint
>
> What does it mean intuitively to take a number modulo 1,000,000?

Problem 24

Open Reading Frames

Transcription May Begin Anywhere

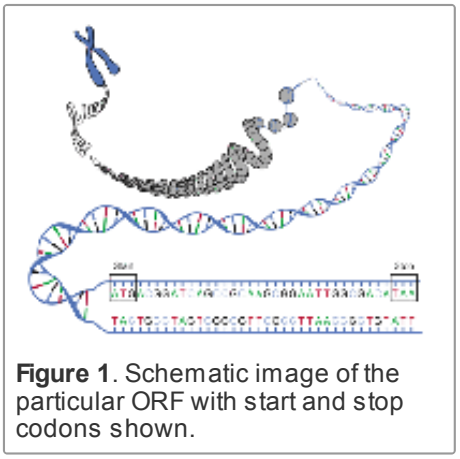


Figure 1. Schematic image of the particular ORF with start and stop codons shown.

In "Transcribing DNA into RNA", we discussed the transcription of DNA into RNA, and in "Translating RNA into Protein", we examined the translation of RNA into a chain of amino acids for the construction of proteins. We can view these two processes as a single step in which we directly translate a DNA string into a protein string, thus calling for a **DNA codon table**.

However, three immediate wrinkles of complexity arise when we try to pass directly from DNA to proteins. First, not all DNA will be transcribed into RNA: so-called **junk DNA** appears to have no practical purpose for cellular function. Second, we can begin translation at any position along a strand of RNA, meaning that any substring of a DNA string can serve as a template for translation, as long as it begins with a start codon, ends with a stop codon, and has no other stop codons in the middle. See **Figure 1**. As a result, the same RNA string can actually be translated in three different ways, depending on how we group triplets of symbols into codons. For example, ...AUGCUGAC... can be translated as ...AUGCUG..., ...UGCUGA..., and ...GCUGAC..., which will typically produce wildly different protein strings.

Problem

Either strand of a DNA double helix can serve as the coding strand for RNA transcription. Hence, a given DNA string implies six total **reading frames**, or ways in which the same region of DNA can be translated into amino acids: three reading frames result from reading the string itself, whereas three more result from reading its reverse complement.

An **open reading frame** (ORF) is one which starts from the start codon and ends by stop codon, without any other stop codons in between. Thus, a candidate protein string is derived by translating an open reading frame into amino acids until a stop codon is reached.

Given: A DNA string s of length at most 1 kbp in FASTA format.

Return: Every distinct candidate protein string that can be translated from ORFs of s. Strings can be returned in any order.

Sample Dataset

```
>Rosalind_99
AGCCATGTAGCTAACTCAGGTTACATGGGGATGACCCCGCGACTTGGATTAGAGTCTCTTTTGGAATAAGC
CTGAATGATCCGAGTAGCATCTCAG
```

Sample Output

```
MLLGSFRLIPKETLIQVAGSSPCNLS
M
MGMTPRLGLESLLE
```

```
MTPRLGLESLLE
```

Problem 25

Enumerating Gene Orders

Rearrangements Power Large-Scale Genomic Changes

Point mutations can create changes in populations of organisms from the same species, but they lack the power to create and differentiate entire species. This more arduous work is left to larger mutations called **genome rearrangements**, which move around huge blocks of DNA. Rearrangements cause major genomic change, and most rearrangements are fatal or seriously damaging to the mutated cell and its descendants (many cancers derive from rearrangements). For this reason, rearrangements that come to influence the genome of an entire species are very rare.

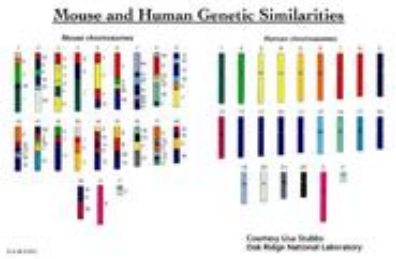


Figure 1. Similar regions in mouse and human chromosomes. Image credit: U.S. Department of Energy Human Genome Program

Because rearrangements that affect species evolution occur infrequently, two closely related species will have very similar genomes. Thus, to simplify comparison of two such genomes, researchers first identify similar intervals of DNA from the species, called **synteny blocks**; over time, rearrangements have created these synteny blocks and heaved them around across the two genomes (often separating blocks onto different chromosomes, see **Figure 1**.).

A pair of synteny blocks from two different species are not strictly identical (they are separated by the action of point mutations or very small rearrangements), but for the sake of studying large-scale rearrangements, we consider them to be equivalent. As a result, we can label each synteny block with a positive integer; when comparing two species' genomes/chromosomes, we then only need to specify the order of its numbered synteny blocks.

Problem

A **permutation** of length n is an ordering of the positive integers $\{1, 2, \ldots, n\}$. For example, $\pi = (5, 3, 2, 1, 4)$ is a permutation of length 5.

Given: A positive integer $n \leq 7$.

Return: The total number of permutations of length n, followed by a list of all such permutations (in any order).

Sample Dataset

```
3
```

Sample Output

```
6
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Problem 26

Calculating Protein Mass

Chaining the Amino Acids

In "Translating RNA into Protein", we examined the translation of RNA into an amino acid chain for the construction of a protein. When two amino acids link together, they form a **peptide bond**, which releases a molecule of water; see **Figure 1**. Thus, after a series of amino acids have been linked together into a polypeptide, every pair of adjacent amino acids has lost one molecule of water, meaning that a polypeptide containing n amino acids has had $n - 1$ water molecules removed.

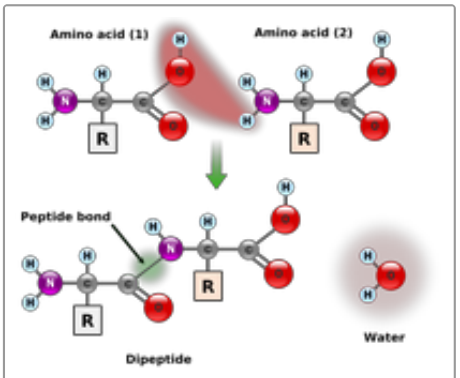


Figure 1. Formation of a peptide bond

More generally, a **residue** is a molecule from which a water molecule has been removed; every amino acid in a protein are residues except the leftmost and the rightmost ones. These outermost amino acids are special in that one has an "unstarted" peptide bond, and the other has an "unfinished" peptide bond. Between them, the two molecules have a single "extra" molecule of water (see the atoms marked in blue in **Figure 2**). Thus, the mass of a protein is the sum of masses of all its residues plus the mass of a single water molecule.

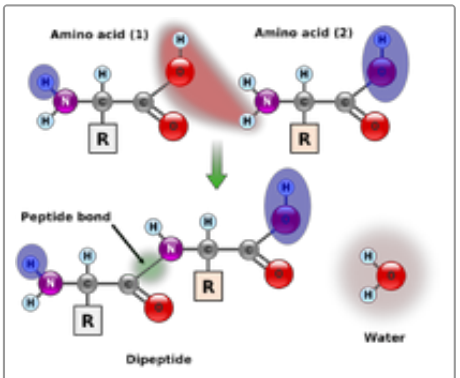


Figure 2. Outermost acids

There are two standard ways of computing the mass of a residue by summing the masses of its individual atoms. Its **monoisotopic mass** is computed by using the principal (most abundant) isotope of each atom in the amino acid, whereas its **average mass** is taken by taking the average mass of each atom in the molecule (over all naturally appearing isotopes).

Many applications in proteomics rely on **mass spectrometry**, an analytical chemical technique used to determine the mass, elemental composition, and structure of molecules. In mass spectrometry, monoisotopic mass is used more often than average mass, and so all amino acid masses are assumed to be monoisotopic unless otherwise stated.

The standard unit used in mass spectrometry for measuring mass is the **atomic mass unit**, which is also called the **dalton** (Da) and is defined as one twelfth of the mass of a neutral atom of carbon-12. The mass of a protein is the sum of the monoisotopic masses of its amino acid residues plus the mass of a single water molecule (whose monoisotopic mass is 18.01056 **Da**).

In the following several problems on applications of mass spectrometry, we avoid the complication of having to distinguish between residues and non-residues by only considering peptides excised from the middle of the protein. This is a relatively safe assumption because in practice, peptide analysis is often performed in **tandem mass spectrometry**. In this special class of mass spectrometry, a protein is first divided into peptides, which are then broken into ions for mass analysis.

Problem

In a **weighted alphabet**, every symbol is assigned a positive real number called a **weight**. A string formed from a weighted alphabet is called a **weighted string**, and its **weight** is equal to the sum of the weights of its symbols.

The standard weight assigned to each member of the 20-symbol amino acid alphabet is the monoisotopic mass of the corresponding amino acid.

Given: A protein string P of length at most 1000 aa.

Return: The total weight of P. Consult the **monoisotopic mass table**.

Sample Dataset

```
SKADYEK
```

Sample Output

```
821.392
```

Problem 27

Locating Restriction Sites

The Billion-Year War

The war between viruses and bacteria has been waged for over a billion years. Viruses called **bacteriophages** (or simply phages) require a bacterial host to propagate, and so they must somehow infiltrate the bacterium; such deception can only be achieved if the phage understands the genetic framework underlying the bacterium's cellular functions. The phage's goal is to insert DNA that will be replicated within the bacterium and lead to the reproduction of as many copies of the phage as possible, which sometimes also involves the bacterium's demise.

To defend itself, the bacterium must either obfuscate its cellular functions so that the phage cannot infiltrate it, or better yet, go on the counterattack by calling in the air force. Specifically, the bacterium employs aerial scouts called **restriction enzymes**, which operate by cutting through viral DNA to cripple the phage. But what kind of DNA are restriction enzymes looking for?

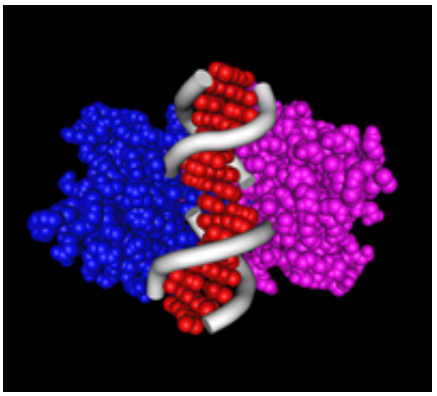


Figure 1. DNA cleaved by EcoRV restriction enzyme

The restriction enzyme is a **homodimer**, which means that it is composed of two identical substructures. Each of these structures separates from the restriction enzyme in order to bind to and cut one strand of the phage DNA molecule; both substructures are pre-programmed with the same target string containing 4 to 12 nucleotides to search for within the phage DNA (see **Figure 1**.). The chance that both strands of phage DNA will be cut (thus crippling the phage) is greater if the target is located on both strands of phage DNA, as close to each other as possible. By extension, the best chance of disarming the phage occurs when the two target copies appear directly across from each other along the phage DNA, a phenomenon that occurs precisely when the target is equal to its own reverse complement. Eons of evolution have made sure that most restriction enzyme targets now have this form.

Problem

A DNA string is a **reverse palindrome** if it is equal to its reverse complement. For instance, GCATGC is a reverse palindrome because its reverse complement is GCATGC. See **Figure 2**.

```
5'...GAT|ATC...3'
3'...CTA|TAG...5'
```

Figure 2. Palindromic recognition site

Given: A DNA string of length at most 1 kbp in FASTA format.

Return: The position and length of every reverse palindrome in the string having length between 4 and 12. You may return these pairs in any order.

Sample Dataset

```
>Rosalind_24
TCAATGCATGCGGGTCTATATGCAT
```

Sample Output

```
4 6
5 4
6 6
7 4
17 4
18 4
20 6
21 4
```

Extra Information

You may be curious how the bacterium prevents its own DNA from being cut by restriction enzymes. The short answer is that it locks itself from being cut through a chemical process called **DNA methylation**.

Problem 28

RNA Splicing

Genes are Discontiguous

In "Transcribing DNA into RNA", we mentioned that a strand of DNA is copied into a strand of RNA during transcription, but we neglected to mention how transcription is achieved.

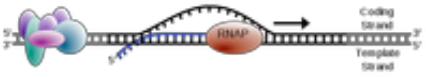


Figure 1. The elongation of a pre-mRNA by RNAP as it moves down the template strand of DNA.

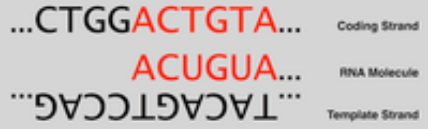


Figure 2. RNA is identical to the coding strand except for the replacement of thymine with uracil.

In the nucleus, an enzyme (i.e., a molecule that accelerates a chemical reaction) called **RNA polymerase** (RNAP) initiates transcription by breaking the bonds joining complementary bases of DNA. It then creates a molecule called **precursor mRNA**, or pre-mRNA, by using one of the two strands of DNA as a **template strand**: moving down the template strand, when RNAP encounters the next nucleotide, it adds the complementary base to the growing RNA strand, with the provision that uracil must be used in place of thymine; see **Figure 1**.

Because RNA is constructed based on complementarity, the second strand of DNA, called the **coding strand**, is identical to the new strand of RNA except for the replacement of thymine with uracil. See **Figure 2** and recall "Transcribing DNA into RNA".

After RNAP has created several nucleotides of RNA, the first separated complementary DNA bases then bond back together. The overall effect is very similar to a pair of zippers traversing the DNA double helix, unzipping the two strands and then quickly zipping them back together while the strand of pre-mRNA is produced.

For that matter, it is not the case that an entire substring of DNA is transcribed into RNA and then translated into a peptide one codon at a time. In reality, a pre-mRNA is first chopped into smaller segments called **introns** and **exons**; for the purposes of protein translation, the introns are thrown out, and the exons are glued together sequentially to produce a final strand of mRNA. This cutting and pasting process is called **splicing**, and it is facilitated by a collection of RNA and proteins called a **spliceosome**. The fact that the spliceosome is made of RNA and proteins despite regulating the splicing of RNA to create proteins is just one manifestation of a molecular chicken-and-egg scenario that has yet to be fully resolved.

In terms of DNA, the exons deriving from a gene are collectively known as the gene's **coding region**.

Problem

After identifying the exons and introns of an RNA string, we only need to delete the introns and concatenate the exons to form a new string ready for translation.

Given: A DNA string s (of length at most 1 kbp) and a collection of substrings of s acting as introns. All strings are given in FASTA format.

Return: A protein string resulting from transcribing and translating the exons of s. (Note: Only one solution will exist for the dataset provided.)

Sample Dataset

```
>Rosalind_10
ATGGTCTACATAGCTGACAAACAGCACGTAGCAATCGGTCGAATCTCGAGAGGCATATGGTCACATGATCG
GTCGAGCGTGTTTCAAAGTTTGCGCCTAG
>Rosalind_12
ATCGGTCGAA
>Rosalind_15
ATCGGTCGAGCGTGT
```

Sample Output

```
MVYIADKQHVASREAYGHMFKVCA
```

Problem 29

Enumerating k-mers Lexicographically

Organizing Strings

When cataloguing a collection of genetic strings, we should have an established system by which to organize them. The standard method is to organize strings as they would appear in a dictionary, so that "APPLE" precedes "APRON", which in turn comes before "ARMOR".

Problem

Assume that an alphabet $\mathscr{A}$ has a predetermined order; that is, we write the alphabet as a permutation $\mathscr{A} = (a_1, a_2, \ldots, a_k)$, where $a_1 < a_2 < \cdots < a_k$. For instance, the English alphabet is organized as $(\mathrm{A}, \mathrm{B}, \ldots, \mathrm{Z})$.

Given two strings s and t having the same length n, we say that s precedes t in the **lexicographic order** (and write $s <_{\mathrm{Lex}} t$) if the first symbol $s[j]$ that doesn't match $t[j]$ satisfies $s_j < t_j$ in $\mathscr{A}$.

Given: A collection of at most 10 symbols defining an ordered alphabet, and a positive integer n (

$n \leq 10$).

Return: All strings of length n that can be formed from the alphabet, ordered lexicographically.

Sample Dataset

```
T A G C
2
```

Sample Output

```
TT
TA
TG
TC
AT
AA
AG
AC
GT
GA
GG
GC
CT
CA
CG
CC
```

> **Note**
>
> As illustrated in the sample, the alphabet order in this problem is defined by the order in which symbols are provided in the dataset, which is not necessarily the traditional order of the English alphabet.

Problem 30

Longest Increasing Subsequence

A Simple Measure of Gene Order Similarity

In "Enumerating Gene Orders", we started talking about comparing the order of genes on a chromosome taken from two different species and moved around by rearrangements throughout the course of evolution.

> One very simple way of comparing genes from two chromosomes is to search for the largest

collection of genes that are found in the same order in both chromosomes. To do so, we will need to apply our idea of permutations. Say that two chromosomes share n genes; if we label the genes of one chromosome by the numbers 1 through n in the order that they appear, then the second chromosome will be given by a permutation of these numbered genes. To find the largest number of genes appearing in the same order, we need only to find the largest collection of increasing elements in the permutation.

Problem

A **subsequence** of a permutation is a collection of elements of the permutation in the order that they appear. For example, (5, 3, 4) is a subsequence of (5, 1, 3, 4, 2).

A subsequence is **increasing** if the elements of the subsequence increase, and **decreasing** if the elements decrease. For example, given the permutation (8, 2, 1, 6, 5, 7, 4, 3, 9), an increasing subsequence is (2, 6, 7, 9), and a decreasing subsequence is (8, 6, 5, 4, 3). You may verify that these two subsequences are as long as possible.

Given: A positive integer $n \leq 10000$ followed by a permutation π of length n.

Return: A longest increasing subsequence of π, followed by a longest decreasing subsequence of π.

Sample Dataset

```
5
5 1 4 2 3
```

Sample Output

```
1 2 3
5 4 2
```

Citation

Adapted from Jones & Pevzner, *An Introduction to Bioinformatics Algorithms, Problem 6.48.

Problem 31

Genome Assembly as Shortest Superstring

Introduction to Genome Sequencing

Recall from "Computing GC Content" that almost all humans share approximately 99.9% of the same nucleotides on the genome. Thus, if we know only a few complete genomes from the species, then we already possess an important key

to unlocking the species genome.

Determining an organism's complete genome (called **genome sequencing**) forms a central task of bioinformatics. Unfortunately, we still don't possess the microscope technology to zoom into the nucleotide level and determine the sequence of a genome's nucleotides, one at a time. However, researchers can apply chemical methods to generate and identify much smaller snippets of DNA, called **reads**.

After obtaining a large collection of reads from multiple copies of the same genome, the aim is to reconstruct the desired genome from these small pieces of DNA; this process is called **fragment assembly** (see **Figure 1**).

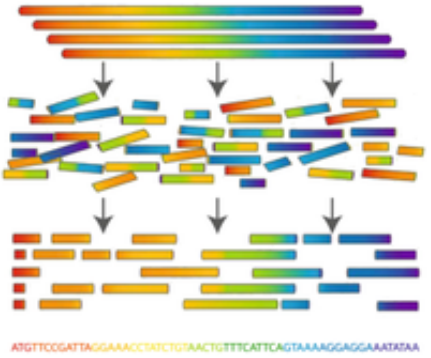


Figure 1. Fragment Assembly works by blasting many copies of the same genome into smaller, identifiable reads, which are then used to computationally assemble one copy of the genome.

Problem

For a collection of strings, a larger string containing every one of the smaller strings as a substring is called a **superstring**.

By the assumption of parsimony, a shortest possible superstring over a collection of reads serves as a candidate chromosome.

Given: At most 50 DNA strings whose length does not exceed 1 kbp in FASTA format (which represent reads deriving from the same strand of a single linear chromosome).

The dataset is guaranteed to satisfy the following condition: there exists a unique way to reconstruct the entire chromosome from these reads by gluing together pairs of reads that overlap by more than half their length.

Return: A shortest superstring containing all the given strings (thus corresponding to a reconstructed chromosome).

Sample Dataset

```
>Rosalind_56
ATTAGACCTG
>Rosalind_57
CCTGCCGGAA
>Rosalind_58
AGACCTGCCG
>Rosalind_59
GCCGGAATAC
```

Sample Output

```
ATTAGACCTGCCGGAATAC
```

Extra Information

Although the goal of fragment assembly is to produce an entire genome, in practice it is only

possible to construct several contiguous portions of each chromosome, called **contigs**. Furthermore, the assumption made above that reads all derive from the same strand is also practically unrealistic; in reality, researchers will not know the strand of DNA from which a given read has been sequenced.

Problem 32

Perfect Matchings and RNA Secondary Structures

Introduction to RNA Folding

Because RNA is single-stranded, you may have wondered if the cytosine and guanine bases bond to each other like in DNA. The answer is yes, as do adenine and uracil, and the resulting base pairs define the secondary structure of the RNA molecule; recall that its primary structure is just the order of its bases.

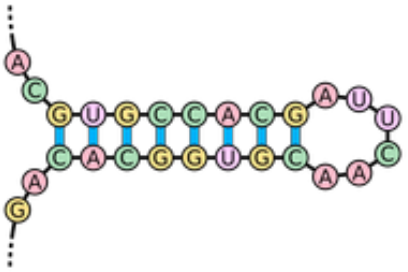


Figure 1. A hairpin loop is formed when consecutive elements from two different regions of an RNA molecule base pair.

In the greater three-dimensional world, the base pairing interactions of an RNA molecule cause it to twist around on itself in a process called **RNA folding**. When two complementary intervals of bases located close to each other on the strand bond to each other, they form a structure called a **hairpin loop** (or stem loop), shown in **Figure 1**.

The same RNA molecule may base pair differently at different points in time, thus adopting many different secondary structures. Our eventual goal is to classify which of these structures are practically feasible, and which are not. To this end, we will ask natural combinatorial questions about the number of possible different RNA secondary structures. In this problem, we will first consider the (impractical) simplified case in which every nucleotide forms part of a base pair in the RNA molecule.

Problem

A **matching** in a graph G is a collection of edges of G for which no node belongs to more than one edge in the collection. See **Figure 2** for examples of matchings. If G contains an even number of nodes (say $2n$), then a matching on G is **perfect** if it contains n edges, which is clearly the maximum possible. An example of a graph containing a perfect matching is shown in **Figure 3**.

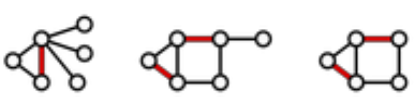


Figure 2. Three matchings (highlighted in red) shown in three different graphs.

First, let K_n denote the **complete graph** on $2n$ labeled nodes, in which every node is connected to every other node with an edge, and let p_n denote the total number of perfect matchings in K_n. For a given node x, there are $2n-1$ ways to join x to the other nodes in the graph, after which point we must form a perfect matching on the remaining $2n-2$ nodes. This reasoning provides us with the recurrence relation $p_n = (2n-1) \cdot p_{n-1}$; using the fact that p_1 is 1, this recurrence relation implies the closed equation $p_n = (2n-1)(2n-3)(2n-5)\cdots(3)(1)$.

Given an RNA string $s = s_1 \dots s_n$, a **bonding graph** for s is formed as follows. First, assign each symbol of s to a node, and arrange these nodes in order around a circle, connecting them with edges called **adjacency edges**. Second, form all possible edges {A, U} and {C, G}, called **basepair edges**; we will represent basepair edges with dashed edges, as illustrated by the bonding graph in **Figure 4**.

Note that a matching contained in the basepair edges will represent one possibility for base pairing interactions in s, as shown in **Figure 5**. For such a matching to exist, s must have the same number of 'A's as 'U's and the same number of 'C's as 'G's.

Given: An RNA string s of length at most 80 bp having the same number of occurrences of 'A' as 'U' and the same number of occurrences of 'C' as 'G'.

Return: The total possible number of *perfect* matchings of basepair edges in the bonding graph of s.

Sample Dataset

```
>Rosalind_23
AGCUAGUCAU
```

Sample Output

```
12
```

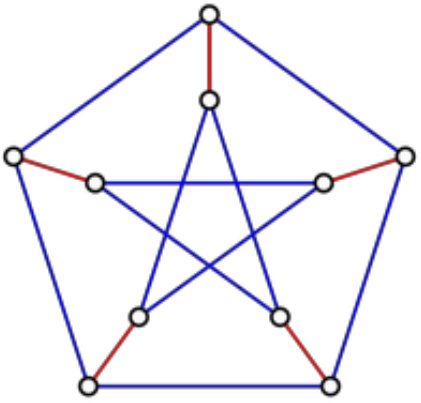


Figure 3. This graph contains 10 nodes; the five edges forming a perfect matching on these nodes are highlighted in red.

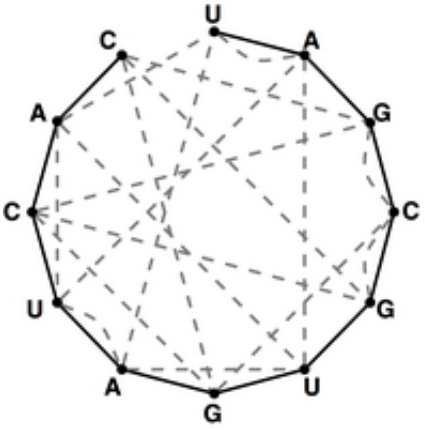


Figure 4. The bonding graph for the RNA string s = UAGCGUGAUCAC.

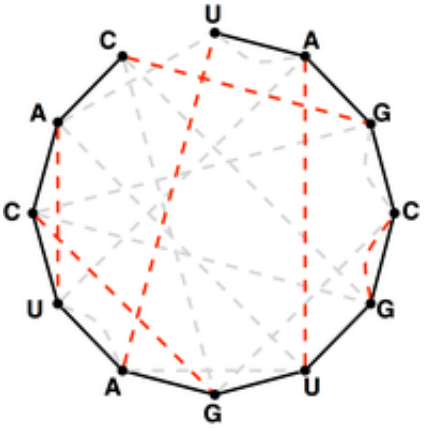


Figure 5. A perfect matching on the basepair edges is highlighted in red and represents a candidate secondary structure for the RNA strand.

Problem 33

Partial Permutations

Partial Gene Orderings

Similar species will share many of the same genes, possibly with modifications. Thus, we can compare two genomes by analyzing the orderings of their genes, then inferring which rearrangements have separated the genes.

> In "Enumerating Gene Orders", we used permutations to model gene orderings. Yet two genomes will have evolved along their own separate paths, and so they won't share all of the same genes. As a result, we should modify the notion of permutation in order to quantify the notion of partial gene orderings.

Problem

A **partial permutation** is an ordering of only k objects taken from a collection containing n objects (i.e., $k \leq n$). For example, one partial permutation of three of the first eight positive integers is given by $(5, 7, 2)$.

The statistic $P(n, k)$ counts the total number of partial permutations of k objects that can be formed from a collection of n objects. Note that $P(n, n)$ is just the number of permutations of n objects, which we found to be equal to $n! = n(n-1)(n-2)\cdots(3)(2)$ in "Enumerating Gene Orders".

Given: Positive integers n and k such that $100 \geq n > 0$ and $10 \geq k > 0$.

Return: The total number of partial permutations $P(n, k)$, modulo 1,000,000.

Sample Dataset

```
21 7
```

Sample Output

```
51200
```

Problem 34

Introduction to Random Strings

Modeling Random Genomes

We already know that the genome is not just a random strand of nucleotides; recall from "Finding a Motif in DNA" that motifs recur commonly across individuals and species. If a DNA motif occurs in many different organisms, then chances are good that it serves an important function.

> At the same time, if you form a long enough DNA string, then you should theoretically be able to

locate every possible short substring in the string. And genomes are very long; the human genome contains about 3.2 billion base pairs. As a result, when analyzing an unknown piece of DNA, we should try to ensure that a motif does not occur out of random chance.

To conclude whether motifs are random or not, we need to quantify the likelihood of finding a given motif randomly. If a motif occurs randomly with high probability, then how can we really compare two organisms to begin with? In other words, all very short DNA strings will appear randomly in a genome, and very few long strings will appear; what is the critical motif length at which we can throw out random chance and conclude that a motif appears in a genome for a reason?

In this problem, our first step toward understanding random occurrences of strings is to form a simple model for constructing genomes randomly. We will then apply this model to a somewhat simplified exercise: calculating the probability of a given motif occurring randomly at a *fixed* location in the genome.

Problem

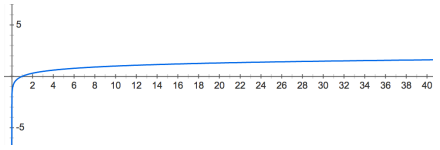


Figure 1. The graph of the common logarithm function of x. For a given x-value, the corresponding y-value is the exponent to which we must raise 10 to obtain x. Note that x-values between 0 and 1 get mapped to y-values between -infinity and 0.

An **array** is a structure containing an ordered collection of objects (numbers, strings, other arrays, etc.). We let $A[k]$ denote the k-th value in array A. You may like to think of an array as simply a matrix having only one row.

A **random string** is constructed so that the probability of choosing each subsequent symbol is based on a fixed underlying symbol frequency.

GC-content offers us natural symbol frequencies for constructing random DNA strings. If the GC-content is x, then we set the symbol frequencies of C and G equal to $\frac{x}{2}$ and the symbol frequencies of A and T equal to $\frac{1-x}{2}$. For example, if the GC-content is 40%, then as we construct the string, the next symbol is 'G'/'C' with probability 0.2, and the next symbol is 'A'/'T' with probability 0.3.

In practice, many probabilities wind up being very small. In order to work with small probabilities, we may plug them into a function that "blows them up" for the sake of comparison. Specifically, the **common logarithm** of x (defined for $x > 0$ and denoted $\log_{10}(x)$) is the exponent to which we must raise 10 to obtain x.

See **Figure 1** for a graph of the common logarithm function $y = \log_{10}(x)$. In this graph, we can see that the logarithm of x-values between 0 and 1 always winds up mapping to y-values between $-\infty$ and 0: x-values near 0 have logarithms close to $-\infty$, and x-values close to 1 have logarithms close to 0. Thus, we will select the common logarithm as our function to "blow up" small probability values for comparison.

Given: A DNA string s of length at most 100 bp and an array A containing at most 20 numbers between 0 and 1.

Return: An array B having the same length as A in which $B[k]$ represents the common logarithm of the probability that a random string constructed with the GC-content found in $A[k]$ will match s exactly.

Sample Dataset

```
ACGATACAA
0.129 0.287 0.423 0.476 0.641 0.742 0.783
```

Sample Output

```
-5.737 -5.217 -5.263 -5.360 -5.958 -6.628 -7.009
```

> #### Hint
>
> One property of the logarithm function is that for any positive numbers x and y, $\log_{10}(x \cdot y) = \log_{10}(x) + \log_{10}(y)$.

Problem 35

Enumerating Oriented Gene Orderings

Synteny Blocks Have Orientations

In "Enumerating Gene Orders", we introduced synteny blocks for two different species, which are very similar areas of two species genomes that have been flipped and moved around by rearrangements. In that problem, we used the permutation to model the order of synteny blocks on a single chromosome.

> However, each strand of a DNA molecule has an orientation (as RNA transcription only occurs in one direction), and so to more prudently model chromosomes using synteny blocks, we should provide each block with an orientation to indicate the strand on which it is located. Adding orientations to synteny blocks requires us to expand our notion of permutation so that each index in the permutation has its own orientation.

Problem

A **signed permutation** of length n is some ordering of the positive integers $\{1, 2, \ldots, n\}$ in which each integer is then provided with either a positive or negative sign (for the sake of simplicity, we omit the positive sign). For example, $\pi = (5, -3, -2, 1, 4)$ is a signed permutation of length 5.

Given: A positive integer $n \leq 6$.

Return: The total number of signed permutations of length n, followed by a list of all such permutations (you may list the signed permutations in any order).

Sample Dataset

```
2
```

Sample Output

```
8
```

```
-1 -2
-1 2
1 -2
1 2
-2 -1
-2 1
2 -1
2 1
```

Problem 36

Finding a Spliced Motif

Motifs Are Rarely Contiguous

In "Finding a Motif in DNA", we searched for occurrences of a motif as a substring of a larger database genetic string. However, because a DNA strand coding for a protein is often interspersed with introns (see "RNA Splicing"), we need a way to recognize a motif that has been chopped up into pieces along a chromosome.

Problem

A **subsequence** of a string is a collection of symbols contained in order (though not necessarily contiguously) in the string (e.g., ACG is a subsequence of TATGCTAAGATC). The **indices** of a subsequence are the positions in the string at which the symbols of the subsequence appear; thus, the indices of ACG in TATGCTAAGATC can be represented by (2, 5, 9).

As a substring can have multiple locations, a subsequence can have multiple collections of indices, and the same index can be reused in more than one appearance of the subsequence; for example, ACG is a subsequence of AACCGGTT in 8 different ways.

Given: Two DNA strings s and t (each of length at most 1 kbp) in FASTA format.

Return: One collection of indices of s in which the symbols of t appear as a subsequence of s. If multiple solutions exist, you may return any one.

Sample Dataset

```
>Rosalind_14
ACGTACGTGACG
>Rosalind_18
GTA
```

Sample Output

```
3 8 10
```

Extra Information

For the mathematically inclined, we may equivalently say that $t = t_1 t_2 \cdots t_m$ is a subsequence of $s = s_1 s_2 \cdots s_n$ if the characters of t appear in the same order within s. Even more formally, a subsequence of s is a string $s_{i_1} s_{i_2} \ldots s_{i_k}$, where $1 \leq i_1 < i_2 \cdots < i_k \leq n$.

Problem 37

Transitions and Transversions

Certain Point Mutations are More Common

Point mutations occurring in DNA can be divided into two types: **transitions** and **transversions**. A transition substitutes one purine for another ($\mathrm{A} \leftrightarrow \mathrm{G}$) or one pyrimidine for another ($\mathrm{C} \leftrightarrow \mathrm{T}$); that is, a transition does not change the structure of the nucleobase. Conversely, a transversion is the interchange of a purine for a pyrimidine base, or vice-versa. See **Figure 1**. Transitions and transversions can be defined analogously for RNA mutations.

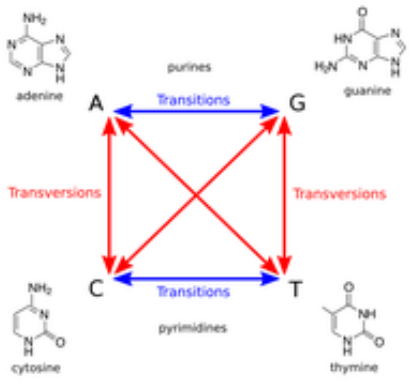


Figure 1. Illustration of transitions and transversions.

Because transversions require a more drastic change to the base's chemical structure, they are less common than transitions. Across the entire genome, the ratio of transitions to transversions is on average about 2. However, in coding regions, this ratio is typically higher (often exceeding 3) because a transition appearing in coding regions happens to be less likely to change the encoded amino acid, particularly when the substituted base is the third member of a codon (feel free to verify this fact using the DNA codon table). Such a substitution, in which the organism's protein makeup is unaffected, is known as a **silent substitution**.

Because of its potential for identifying coding DNA, the ratio of transitions to transversions between two strands of DNA offers a quick and useful statistic for analyzing genomes.

Problem

For DNA strings s_1 and s_2 having the same length, their **transition/transversion ratio** $R(s_1, s_2)$ is the ratio of the total number of transitions to the total number of transversions, where symbol substitutions are inferred from mismatched corresponding symbols as when calculating Hamming distance (see "Counting Point Mutations").

Given: Two DNA strings s_1 and s_2 of equal length (at most 1 kbp).

Return: The transition/transversion ratio $R(s_1, s_2)$.

Sample Dataset

```
>Rosalind_0209
GCAACGCACAACGAAAACCCTTAGGGACTGGATTATTTCGTGATCGTTGTAGTTATTGGA
AGTACGGGCATCAACCCAGTT
>Rosalind_2200
TTATCTGACAAAGAAAGCCGTCAACGGCTGGATAATTTCGCGATCGTGCTGGTTACTGGC
GGTACGAGTGTTCCTTTGGGT
```

Sample Output

```
1.21428571429
```

Problem 38

Completing a Tree

The Tree of Life

"As buds give rise by growth to fresh buds, and these, if vigorous, branch out and overtop on all sides many a feebler branch, so by generation I believe it has been with the great Tree of Life, which fills with its dead and broken branches the crust of the earth, and covers the surface with its ever-branching and beautiful ramifications."
Charles Darwin, ***The Origin of Species***

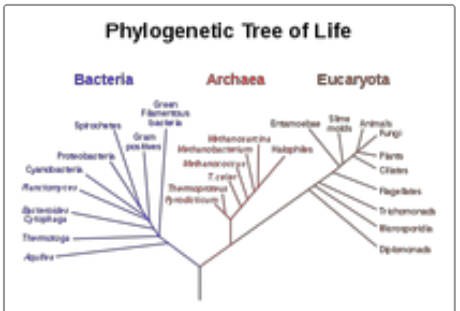


Figure 1. A phylogeny illustrating proposed evolutionary relationships among the three domains of life: Bacteria, Archaea, and Eukaryota.

A century and a half has passed since the publication of Darwin's magnum opus, and yet the construction of a single **Tree of Life** uniting life on Earth still has not been completed, with perhaps as many as 90% of all living species not yet catalogued (although a beautiful interactive animation has been produced by OneZoom).

To get an insight about state-of-art attempts to build this tree, you may take a look at the Tree of Life Web Project - collaborative effort of biologists from around the world to combine information about diversity of life on Earth. It is a peer-reviewed ongoing project started in 1995, now it holds more than 10,000 pages with characteristics of different groups of organisms and their evolutionary history, and tree still grows.

Instead of trying to construct the entire Tree of Life all at once, we often wish to form a simpler tree in which a collection of species have been clumped together for the sake of simplicity; such a group is called a **taxon** (pl. taxa). For a given collection of taxa, a **phylogeny** is a treelike diagram that best represents the evolutionary connections between the taxa: the construction of a particular phylogeny depends on our specific assumptions regarding how these evolutionary relationships

should be interpreted. See **Figure 1**.

Problem

An undirected graph is **connected** if there is a path connecting any two nodes. A **tree** is a connected (undirected) graph containing no cycles; this definition forces the tree to have a branching structure organized around a central core of nodes, just like its living counterpart. See **Figure 2**.

We have already grown familiar with trees in "Mendel's First Law", where we introduced the probability tree diagram to visualize the outcomes of a random variable.

In the creation of a phylogeny, taxa are encoded by the tree's **leaves**, or nodes having degree 1. A node of a tree having degree larger than 1 is called an **internal node**.

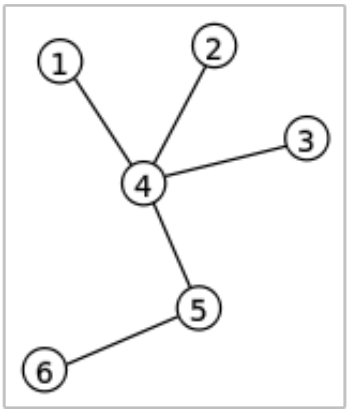


Figure 2. A labeled tree with 6 vertices and 5 edges.

Given: A positive integer n ($n \leq 1000$) and an adjacency list corresponding to a graph on n nodes that contains no cycles.

Return: The minimum number of edges that can be added to the graph to produce a tree.

Sample Dataset

```
10
1 2
2 8
4 10
5 9
6 10
7 9
```

Sample Output

```
3
```

> #### Extra Information
>
> After solving this problem, a standard mathematical exercise for the technically minded is to verify that every tree having 2 or more nodes must contain at least two leaves.

Problem 39

Catalan Numbers and RNA Secondary Structures

The Human Knot

You may have had the misfortune to participate in a team-building event that featured the "human knot," in which everyone joins hands with two other people, and the group must undo the giant knot of arms without letting go (see **Figure 1**).

Figure 1. Knot fun. Courtesy El Photo Studio.

Let's consider a simplified version of the human knot. Say that we have an even number of people at a party who are standing in a circle, and they pair off and shake hands at the same time. One combinatorial question at hand asks us to count the total number of ways that the guests can shake hands without any two pairs interfering with each other by crossing arms.

This silly little counting problem is actually an excellent analogy for RNA folding. In practice, base pairing can occur anywhere along the RNA molecule, but the secondary structure of RNA often forbids base pairs crossing over each other, which forms a structure called a **pseudoknot** (see **Figure 2**)). Pseudoknots are not technically knots, but they nevertheless cause RNA to fold over itself.

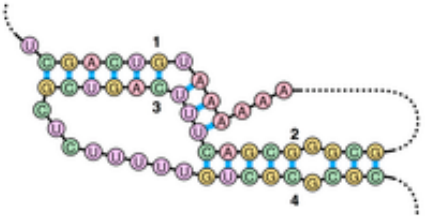


Figure 2. This pseudoknot was formed when bonding occurred at the endpoints of overlapping intervals [1,3] and [2, 4].

Forbidding pseudoknots offers an interesting wrinkle to the problem of counting potential RNA secondary structures that we started working with in "Perfect Matchings and RNA Secondary Structures", in which every possible nucleotide of a strand of RNA must base pair with another nucleotide.

Problem

A matching in a graph is **noncrossing** if none of its edges cross each other. If we assume that the n nodes of this graph are arranged around a circle, and if we label these nodes with positive integers between 1 and n, then a matching is noncrossing as long as there are not edges $\{i, j\}$ and $\{k, l\}$ such that $i < k < j < l$.

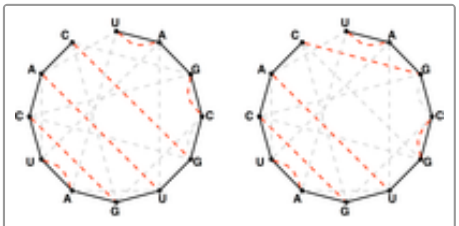


Figure 3. The only two noncrossing perfect matchings of basepair edges (shown in red) for the RNA string UAGCGUGAUCAC.

A noncrossing matching of basepair edges in the bonding graph corresponding to an RNA string will correspond to a possible secondary structure of the underlying RNA strand that lacks pseudoknots, as shown in **Figure 3**.

In this problem, we will consider counting noncrossing perfect matchings of basepair edges. As a motivating example of how to count noncrossing perfect matchings, let c_n denote the number of noncrossing perfect matchings in the complete graph K_{2n}. After setting $c_0 = 1$, we can see that c_1 should equal 1 as well. As for the case of a general n, say that the nodes of K_{2n} are labeled with the positive integers from 1 to $2n$. We can join node 1 to any of the remaining $2n - 1$ nodes; yet once we have chosen this node (say m), we cannot add another edge to the matching that crosses the

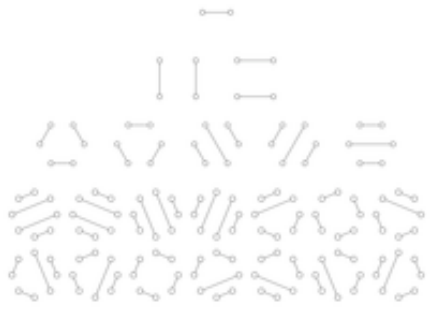


Figure 4. This figure shows all possible noncrossing perfect matchings in complete graphs on 2,

edge $\{1, m\}$. As a result, we must match all the edges on one side of $\{1, m\}$ to each other. This requirement forces m to be even, so that we can write $m = 2k$ for some positive integer k.

4, 6, and 8 nodes; the total number of such matchings are 1, 2, 5, and 14, respectively. Courtesy Tom Davis.

There are $2k - 2$ nodes on one side of $\{1, m\}$ and $2n - 2k$ nodes on the other side of $\{1, m\}$, so that in turn there will be $c_{k-1} \cdot c_{n-k}$ different ways of forming a perfect matching on the remaining nodes of K_{2n}. If we let m vary over all possible $n - 1$ choices of even numbers between 1 and $2n$, then we obtain the recurrence relation $c_n = \sum_{k=1}^{n} c_{k-1} \cdot c_{n-k}$. The resulting numbers c_n counting noncrossing perfect matchings in K_{2n} are called the **Catalan numbers**, and they appear in a huge number of other settings. See **Figure 4** for an illustration counting the first four Catalan numbers.

Given: An RNA string s having the same number of occurrences of 'A' as 'U' and the same number of occurrences of 'C' as 'G'. The length of the string is at most 300 bp.

Return: The total number of noncrossing perfect matchings of basepair edges in the bonding graph of s, modulo 1,000,000.

Sample Dataset

```
>Rosalind_57
AUAU
```

Sample Output

```
2
```

> ### Hint
>
> Write a function that counts Catalan numbers via dynamic programming. How can we modify this function to apply to our given problem?

Problem 40

Error Correction in Reads

Genome Sequencing Isn't Perfect

In "Genome Assembly as Shortest Superstring", we introduce the problem of assembling a genome from a collection of reads. Even though genome sequencing is a multi-billion dollar enterprise, sequencing machines that identify reads still produce errors a substantial percentage of the time. To make matters worse, these errors are unpredictable; it is difficult to determine if the machine has made an error, let alone where in the read the error has occurred. For this reason, error correction in reads is typically a vital first step in genome assembly.

Problem

As is the case with point mutations, the most common type of sequencing error occurs when a single nucleotide from a read is interpreted incorrectly.

Given: A collection of up to 1000 reads of equal length (at most 50 bp) in FASTA format. Some of these reads were generated with a single-nucleotide error. For each read s in the dataset, one of the following applies:

- s was correctly sequenced and appears in the dataset at least twice (possibly as a reverse complement);
- s is incorrect, it appears in the dataset exactly once, and its Hamming distance is 1 with respect to exactly one correct read in the dataset (or its reverse complement).

Return: A list of all corrections in the form "[old read]->[new read]". (Each correction must be a single symbol substitution, and you may return the corrections in any order.)

Sample Dataset

```
>Rosalind_52
TCATC
>Rosalind_44
TTCAT
>Rosalind_68
TCATC
>Rosalind_28
TGAAA
>Rosalind_95
GAGGA
>Rosalind_66
TTTCA
>Rosalind_33
ATCAA
>Rosalind_21
TTGAT
>Rosalind_18
TTTCC
```

Sample Output

```
TTCAT->TTGAT
GAGGA->GATGA
TTTCC->TTTCA
```

Problem 41

Counting Phylogenetic Ancestors

Culling the Forest

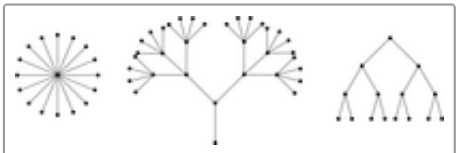


Figure 1. Trees come in lots of different shapes.

In "Completing a Tree", we introduced the tree for the purposes of constructing phylogenies. Yet the definition of tree as a connected graph with no cycles produces a huge class of different graphs, from simple paths and star-like graphs to more familiar arboreal structures (see **Figure 1**). Which of these graphs are appropriate for phylogenetic study?

Modern evolutionary theory (beginning with Darwin) indicates that the only way a new species can be created is if it splits off from an existing species after a population is isolated for an extended period of time. This model of species evolution implies a very specific type of phylogeny, in which internal nodes represent branching points of evolution where an ancestor species either evolved into a new species or split into two new species: therefore, one edge of this internal node therefore connects the node to its most recent ancestor, whereas one or two new edges connect it to its immediate descendants. This framework offers a much clearer notion of how to characterize phylogenies.

Problem

A **binary tree** is a tree in which each node has degree equal to at most 3. The binary tree will be our main tool in the construction of phylogenies.

A **rooted tree** is a tree in which one node (the **root**) is set aside to serve as the pinnacle of the tree. A standard graph theory exercise is to verify that for any two nodes of a tree, exactly one path connects the nodes. In a rooted tree, every node v will therefore have a single **parent**, or the unique node w such that the path from v to the root contains $\{v, w\}$. Any other node x adjacent to v is called a **child** of v because v must be the parent of x; note that a node may have multiple children. In other words, a rooted tree possesses an ordered hierarchy from the root down to its leaves, and as a result, we may often view a rooted tree with undirected edges as a directed graph in which each edge is oriented from parent to child. We should already be familiar with this idea; it's how the Rosalind problem tree works!

Even though a binary tree can include nodes having degree 2, an **unrooted binary tree** is defined more specifically: all internal nodes have degree 3. In turn, a **rooted binary tree** is such that only the root has degree 2 (all other internal nodes have degree 3).

Given: A positive integer n ($3 \leq n \leq 10000$).

Return: The number of internal nodes of any unrooted binary tree having n leaves.

Sample Dataset

```
4
```

Sample Output

```
2
```

Hint

In solving "Completing a Tree", you may have formed the conjecture that a graph with no cycles and n nodes is a tree precisely when it has $n - 1$ edges. This is indeed a theorem of graph theory.

Problem 42

k-Mer Composition

Generalizing GC-Content

A length k substring of a genetic string is commonly called a **k-mer**. A genetic string of length n can be seen as composed of $n - k + 1$ overlapping k-mers. The **k-mer composition** of a genetic string encodes the number of times that each possible k-mer occurs in the string. See **Figure 1**. The 1-mer composition is a generalization of the GC-content of a strand of DNA, and the 2-mer, 3-mer, and 4-mer compositions of a DNA string are also commonly known as its di-nucleotide, tri-nucleotide, and tetra-nucleotide compositions.

AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
3	5	0	8	6	2	1	3	2	2	0	4	5	3	7	8

Figure 1. The 2-mer composition of TTGATTACCTTATTTGATCATTACAC

The biological significance of k-mer composition is manyfold. GC-content is helpful not only in helping to identify a piece of unknown DNA (see "Computing GC Content"), but also because a genomic region having high GC-content compared to the rest of the genome signals that it may belong to an exon. Analyzing k-mer composition is vital to fragment assembly as well.

In "Computing GC Content", we also drew an analogy between analyzing the frequency of characters and identifying the underlying language. For larger values of k, the k-mer composition offers a more robust fingerprint of a string's identity because it offers an analysis on the scale of substrings (i.e., words) instead of that of single symbols. As a basis of comparison, in language analysis, the k-mer composition of a text can be used not only to pin down the language, but also often the *author*.

Problem

For a fixed positive integer k, order all possible k-mers taken from an underlying alphabet lexicographically.

Then the k-mer composition of a string s can be represented by an array A for which $A[m]$ denotes the number of times that the mth k-mer (with respect to the lexicographic order) appears in s.

Given: A DNA string s in FASTA format (having length at most 100 kbp).

Return: The 4-mer composition of s.

Sample Dataset

```
>Rosalind_6431
CTTCGAAAGTTTGGGCCGAGTCTTACAGTCGGTCTTGAAGCAAAGTAACGAACTCCACGG
```

```
CCCTGACTACCGAACCAGTTGTGAGTACTCAACTGGGTGAGAGTGCAGTCCCTATTGAGT
TTCCGAGACTCACCGGGATTTTCGATCCAGCCTCAGTCCAGTCTTGTGGCCAACTCACCA
AATGACGTTGGAATATCCCTGTCTAGCTCACGCAGTACTTAGTAAGAGGTCGCTGCAGCG
GGGCAAGGAGATCGGAAAATGTGCTCTATATGCGACTAAAGCTCCTAACTTACACGTAGA
CTTGCCCGTGTTAAAAACTCGGCTCACATGCTGTCTGCGGCTGGCTGTATACAGTATCTA
CCTAATACCCTTCAGTTCGCCGCACAAAAGCTGGGAGTTACCGCGGAAATCACAG
```

Sample Output

```
4 1 4 3 0 1 1 5 1 3 1 2 2 1 2 0 1 1 3 1 2 1 3 1 1 1 1 2 2 5 1 3 0 2 2 1
 1 1 1 3 1 0 0 1 5 5 1 5 0 2 0 2 1 2 1 1 1 2 0 1 0 0 1 1 3 2 1 0 3 2 3
0 0 2 0 8 0 0 1 0 2 1 3 0 0 0 1 4 3 2 1 1 3 1 2 1 3 1 2 1 2 1 1 1 2 3 2
 1 1 0 1 1 3 2 1 2 6 2 1 1 1 2 3 3 3 2 3 0 3 2 1 1 0 0 1 4 3 0 1 5 0 2
0 1 2 1 3 0 1 2 2 1 1 0 3 0 0 4 5 0 3 0 2 1 1 3 0 3 2 2 1 1 0 2 1 0 2 2
 1 2 0 2 2 5 2 2 1 1 2 1 2 2 2 2 1 1 3 4 0 2 1 1 0 1 2 2 1 1 1 5 2 0 3
2 1 1 2 2 3 0 3 0 1 3 1 2 3 0 2 1 2 2 1 2 3 0 1 2 3 1 1 3 1 0 1 1 3 0 2
 1 2 2 0 2 1 1
```

Problem 43

Speeding Up Motif Finding

Shortening the Motif Search

In "Finding a Motif in DNA", we discussed the problem of searching a genome for a known motif. Because of the large scale of eukaryotic genomes, we need to accomplish this computational task as efficiently as possible.

The standard method for locating one string t as a substring of another string s (and perhaps one you implemented in "Finding a Motif in DNA") is to move a sliding window across the larger string, at each step starting at $s[k]$ and matching subsequent symbols of t to symbols of s. After we have located a match or mismatch, we then shift the window backwards to begin searching at $s[k+1]$.

The potential weakness of this method is as follows: say we have matched 100 symbols of t to s before reaching a mismatch. The window-sliding method would then move back 99 symbols of s and start comparing t to s; can we avoid some of this sliding?

For example, say that we are looking for $t = \text{ACGTACGT}$ in $s = \text{TAGGTACGTACGGCATCACG}$. From $s[6]$ to $s[12]$, we have matched seven symbols of t, and yet $s[13]$ = G produces a mismatch with $t[8]$ = T. We don't need to go all the way back to $s[7]$ and start matching with t because $s[7] = \text{C}$, $s[8] = \text{G}$, and $s[9] = \text{T}$ are all different from $t[1] = \text{A}$. What about $s[10]$? Because $t[1:4] = t[5:8] = \text{ACGT}$, the previous mismatch of $s[13] = \text{G}$ and $t[8] = \text{T}$ guarantees the *same* mismatch with $s[13]$ and $t[4]$. Following this analysis, we may advance directly to $s[14]$ and continue sliding our window, without ever having to move it backward.

This method can be generalized to form the framework behind the **Knuth-Morris-Pratt algorithm** (KMP), which was published in 1977 and offers an efficiency boost for determining whether a given

motif can be located within a larger string.

Problem

A **prefix** of a length n string s is a substring $s[1:j]$; a **suffix** of s is a substring $s[k:n]$.

The **failure array** of s is an array P of length n for which $P[k]$ is the length of the longest substring $s[j:k]$ that is equal to some prefix $s[1:k-j+1]$, where j cannot equal 1 (otherwise, $P[k]$ would always equal k). By convention, $P[1]=0$.

Given: A DNA string s (of length at most 100 kbp) in FASTA format.

Return: The failure array of s.

Sample Dataset

```
>Rosalind_87
CAGCATGGTATCACAGCAGAG
```

Sample Output

```
0 0 0 1 2 0 0 0 0 0 0 1 2 1 2 3 4 5 3 0 0
```

Extra Information

If you would like a more precise technical explanation of the Knuth-Morris-Pratt algorithm, please take a look at this site

Problem 44

Finding a Shared Spliced Motif

Locating Motifs Despite Introns

In "Finding a Shared Motif", we discussed searching through a database containing multiple genetic strings to find a longest common substring of these strings, which served as a motif shared by the two strings. However, as we saw in "RNA Splicing", coding regions of DNA are often interspersed by introns that do not code for proteins.

We therefore need to locate shared motifs that are separated across exons, which means that the motifs are not required to be contiguous. To model this situation, we need to enlist subsequences.

Problem

A string u is a **common subsequence** of strings s and t if the symbols of u appear in order as a subsequence of both s and t. For example, "ACTG" is a common subsequence of "A**A**C**C**T**T**G**G**" and "**A**CA**CTG**TGA".

Analogously to the definition of longest common substring, u is a **longest common subsequence** of s and t if there does not exist a longer common subsequence of the two strings. Continuing our above example, "ACCTTG" is a longest common subsequence of "A**ACCTTG**G" and "**AC**A**CT**G**TG**A", as is "AACTGG".

Given: Two DNA strings s and t (each having length at most 1 kbp) in FASTA format.

Return: A longest common subsequence of s and t. (If more than one solution exists, you may return any one.)

Sample Dataset

```
>Rosalind_23
AACCTTGG
>Rosalind_64
ACACTGTGA
```

Sample Output

```
AACTGG
```

Problem 45

Ordering Strings of Varying Length Lexicographically

Organizing Strings of Different Lengths

> In "Enumerating k-mers Lexicographically", we introduced the lexicographic order for strings of the same length constructed from some ordered underlying alphabet. However, our experience with dictionaries suggests that we should be able to order strings of different lengths just as easily. That is, we already have an intuitive sense that "APPLE" comes before "APPLET", which comes before "ARTS," and so we should be able to apply this intuition toward cataloguing genetic strings of varying lengths.

Problem

Say that we have strings $s = s_1 s_2 \cdots s_m$ and $t = t_1 t_2 \cdots t_n$ with $m < n$. Consider the substring $t' = t[1:m]$. We have two cases:

1. If $s = t'$, then we set $s <_{\text{Lex}} t$ because s is shorter than t (e.g., $\text{APPLE} < \text{APPLET}$).
2. Otherwise, $s \neq t'$. We define $s <_{\text{Lex}} t$ if $s <_{\text{Lex}} t'$ and define $s >_{\text{Lex}} t$ if $s >_{\text{Lex}} t'$ (e.g., $\text{APPLET} <_{\text{Lex}} \text{ARTS}$ because $\text{APPL} <_{\text{Lex}} \text{ARTS}$).

Given: A permutation of at most 12 symbols defining an ordered alphabet $\mathscr{A}$ and a positive integer n ($n \leq 4$).

Return: All strings of length at most n formed from $\mathscr{A}$, ordered lexicographically. (Note: As in "Enumerating k-mers Lexicographically", alphabet order is based on the order in which the symbols are given.)

Sample Dataset

```
D N A
3
```

Sample Output

```
D
DD
DDD
DDN
DDA
DN
DND
DNN
DNA
DA
DAD
DAN
DAA
N
ND
NDD
NDN
NDA
NN
NND
NNN
NNA
NA
NAD
NAN
NAA
A
AD
ADD
ADN
ADA
AN
AND
ANN
ANA
AA
AAD
AAN
```

AAA

Extra Information

We can combine conditions (1) and (2) above into a single condition by adding a blank character $\emptyset$ to the beginning of our ordered alphabet. Then, if s is shorter than t, we simply add as many instances of $\emptyset$ as necessary to make s and t the same length.

Problem 46

Maximum Matchings and RNA Secondary Structures

Breaking the Bonds

In "Perfect Matchings and RNA Secondary Structures", we considered a problem that required us to assume that every possible nucleotide is involved in base pairing to induce an RNA secondary structure. Yet the only way this could occur is if the frequency of adenine in our RNA strand is equal to the frequency of uracil and if the same holds for guanine and cytosine.

We will therefore begin to explore ways of counting secondary structures in which this condition is not required. A more general combinatorial problem will ask instead for the total number of secondary structures of a strand having a maximum possible number of base pairs.

Problem

The graph theoretical analogue of the quandary stated in the introduction above is that if we have an RNA string s that does not have the same number of 'C's as 'G's and the same number of 'A's as 'U's, then the bonding graph of s cannot possibly possess a perfect matching among its basepair edges. For example, see **Figure 1**; in fact, most bonding graphs will not contain a perfect matching.

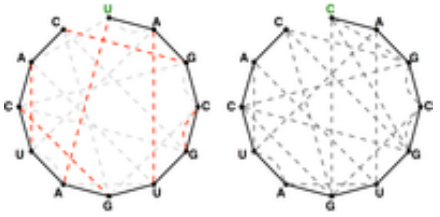


Figure 1. The bonding graph of s = UAGCGUGAUCAC (left) has a perfect matching of basepair edges, but this is not the case for t = CAGCGUGAUCAC, in which one symbol has been replaced.

In light of this fact, we define a **maximum matching** in a graph as a matching containing as many edges as possible. See **Figure 2** for three maximum matchings in graphs.

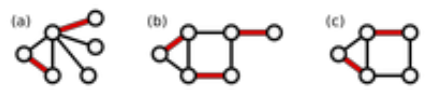


Figure 2. A maximum matching (highlighted in red) is shown in each of the three graphs above. Verify that no other matching can contain more edges.

A maximum matching of basepair edges will correspond to a way of forming as many base pairs as possible in an RNA string, as shown in **Figure 3**.

Given: An RNA string s of length at most 100.

Return: The total possible number of maximum matchings of basepair edges in the bonding graph of s.

Sample Dataset

```
>Rosalind_92
AUGCUUC
```

Sample Output

```
6
```

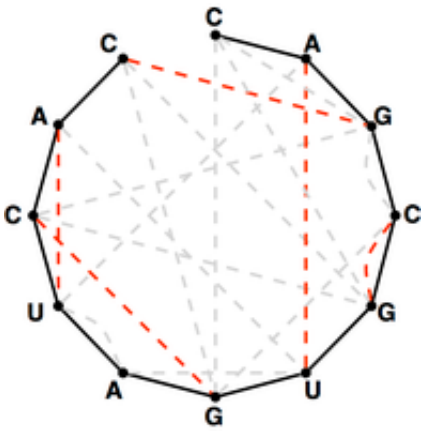


Figure 3. A red maximum matching of basepair edges in the bonding graph for t = CAGCGUGAUCAC.

Problem 47

Creating a Distance Matrix

Introduction to Distance-Based Phylogeny

A number of different approaches are used to build phylogenies, each one featuring its own computational strengths and weaknesses. One of these measures is **distance-based phylogeny**, which constructs a tree from evolutionary distances calculated between pairs of taxa.

> A wide assortment of different measures exist for quantifying this evolutionary distance. Once we have selected a distance function and used it to calculate the distance between every pair of taxa, we place these pairwise distance functions into a table.
>
> In this problem, we will consider an evolutionary function based on Hamming distance. Recall from "Counting Point Mutations" that this function compares two homologous strands of DNA by counting the minimum possible number of point mutations that could have occurred on the evolutionary path between the two strands.

Problem

For two strings s_1 and s_2 of equal length, the **p-distance** between them, denoted $d_p(s_1, s_2)$, is the proportion of corresponding symbols that differ between s_1 and s_2.

For a general distance function d on n taxa $s_1, s_2, \ldots, s_n$ (taxa are often represented by genetic strings), we may encode the distances between pairs of taxa via a **distance matrix** D in which $D_{i,j} = d(s_i, s_j)$.

Given: A collection of n ($n \leq 10$) DNA strings $s_1, \ldots, s_n$ of equal length (at most 1 kbp). Strings are given in FASTA format.

Return: The matrix D corresponding to the p-distance d_p on the given strings. As always, note that your answer is allowed an absolute error of 0.001.

Sample Dataset

```
>Rosalind_9499
TTTCCATTTA
>Rosalind_0942
GATTCATTTC
>Rosalind_6568
TTTCCATTTT
>Rosalind_1833
GTTCCATTTA
```

Sample Output

```
0.00000 0.40000 0.10000 0.10000
0.40000 0.00000 0.40000 0.30000
0.10000 0.40000 0.00000 0.20000
0.10000 0.30000 0.20000 0.00000
```

Problem 48

Reversal Distance

Rearrangements Power Large-Scale Genomic Changes

> Perhaps the most common type of genome rearrangement is an **inversion**, which flips an entire interval of DNA found on the same chromosome. As in the case of calculating Hamming distance (see "Counting Point Mutations"), we would like to determine the minimum number of inversions that have occurred on the evolutionary path between two chromosomes. To do so, we will use the model introduced in "Enumerating Gene Orders" in which a chromosome is represented by a permutation of its synteny blocks.

Problem

A **reversal** of a permutation creates a new permutation by inverting some interval of the permutation; $(5, 2, 3, 1, 4)$, $(5, 3, 4, 1, 2)$, and $(4, 1, 2, 3, 5)$ are all reversals of $(5, 3, 2, 1, 4)$. The **reversal distance** between two permutations π and σ, written $d_{\text{rev}}(\pi, \sigma)$, is the minimum number of reversals required to transform π into σ (this assumes that π and σ have the same length).

Given: A collection of at most 5 pairs of permutations, all of which have length 10.

Return: The reversal distance between each permutation pair.

Sample Dataset

```
1 2 3 4 5 6 7 8 9 10
3 1 5 2 7 4 9 6 10 8

3 10 8 2 5 4 7 1 6 9
5 2 3 1 7 4 10 8 6 9

8 6 7 9 4 1 3 10 2 5
8 2 7 6 9 1 5 3 10 4

3 9 10 4 1 8 6 7 5 2
2 9 8 5 1 7 3 4 6 10

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Sample Output

```
9 4 5 7 0
```

> **Hint**
>
> Don't be afraid to try an ugly solution.

Problem 49

Matching Random Motifs

More Random Strings

In "Introduction to Random Strings", we discussed searching for motifs in large genomes, in which random occurrences of the motif are possible. Our aim is to quantify just how frequently random motifs occur.

> One class of motifs of interest are **promoters**, or regions of DNA that initiate the transcription of a gene. A promoter is usually located shortly before the start of its gene, and it contains specific intervals of DNA that provide an initial binding site for RNA polymerase to initiate transcription. Finding a promoter is usually the second step in gene prediction after establishing the presence of an ORF (see "Open Reading Frames").
>
> Unfortunately, there is no quick rule for identifying promoters. In *Escherichia coli*, the promoter contains two short intervals (TATAAT and TTGACA), which are respectively located 10 and 35 base pairs upstream from the beginning of the gene's ORF. Yet even these two short intervals are consensus strings (see "Consensus and Profile"): they represent average-case strings that are not found intact in most promoters. Bacterial promoters further vary in that some contain additional intervals used to bind to specific proteins or to change the intensity of transcription.

> Eukaryotic promoters are even more difficult to characterize. Most have a **TATA box** (consensus sequence: TATAAA), preceded by an interval called a **B recognition element**, or BRE. These elements are typically located within 40 bp of the start of transcription. For that matter, eukaryotic promoters can hold a larger number of additional "regulatory" intervals, which can be found as far as several thousand base pairs upstream of the gene.

Problem

Our aim in this problem is to determine the probability with which a given motif (a known promoter, say) occurs in a randomly constructed genome. Unfortunately, finding this probability is tricky; instead of forming a long genome, we will form a large collection of smaller random strings having the same length as the motif; these smaller strings represent the genome's substrings, which we can then test against our motif.

Given a probabilistic event A, the **complement** of A is the collection A^c of outcomes not belonging to A. Because A^c takes place precisely when A does not, we may also call A^c "not A."

For a simple example, if A is the event that a rolled die is 2 or 4, then $\Pr(A) = \frac{1}{3}$. A^c is the event that the die is 1, 3, 5, or 6, and $\Pr(A^c) = \frac{2}{3}$. In general, for any event we will have the identity that $\Pr(A) + \Pr(A^c) = 1$.

Given: A positive integer $N \leq 100000$, a number x between 0 and 1, and a DNA string s of length at most 10 bp.

Return: The probability that if N random DNA strings having the same length as s are constructed with GC-content x (see "Introduction to Random Strings"), then at least one of the strings equals s. We allow for the same random string to be created more than once.

Sample Dataset

```
90000 0.6
ATAGCCGA
```

Sample Output

```
0.689
```

Problem 50

Counting Subsets

Characters and SNPs

A **character** is any feature (genetic, physical, etc.) that divides a collection of organisms into two separate groups. One commonly used genetic character is the possession of a **single-nucleotide polymorphism**, or SNP.

> In a process called **genotyping**, the SNP markers taken from a large number of human donors have been used very successfully to catalogue the migration and differentiation of human populations over the last 200,000 years. For $199, you can participate in *National Geographic*'s Genographic Project, and discover your own genetic heritage.
>
> Whether we use genetic or physical characters, we may think of a collection of n characters as a collection of "ON"/"OFF" switches. An organism is said to possess a character in the "ON" position (although often the assignment of "ON"/"OFF" is arbitrary). Given a collection of taxa, we may represent a character by the collection of taxa possessing the character.

Problem

A **set** is the mathematical term for a loose collection of objects, called **elements**. Examples of sets include $\{\text{the moon}, \text{the sun}, \text{Wilford Brimley}\}$ and $\mathbb{R}$, the set containing all real numbers. We even have the **empty set**, represented by $\emptyset$ or $\{\}$, which contains no elements at all. Two sets are equal when they contain the same elements. In other words, in contrast to permutations, the ordering of the elements of a set is unimportant (e.g., $\{\text{the moon}, \text{the sun}, \text{Wilford Brimley}\}$ is equivalent to $\{\text{Wilford Brimley}, \text{the moon}, \text{the sun}\}$). Sets are not allowed to contain duplicate elements, so that $\{\text{Wilford Brimley}, \text{the sun}, \text{the sun}\}$ is not a set. We have already used sets of 2 elements to represent edges from a graph.

A set A is a **subset** of B if every element of A is also an element of B, and we write $A \subseteq B$. For example, $\{\text{the sun}, \text{the moon}\} \subseteq \{\text{the sun}, \text{the moon}, \text{Wilford Brimley}\}$, and $\emptyset$ is a subset of *every* set (including itself!).

As illustrated in the biological introduction, we can use subsets to represent the collection of taxa possessing a character. However, the number of applications is endless; for example, an event in probability can now be defined as a subset of the set containing all possible outcomes.

Our first question is to count the total number of possible subsets of a given set.

Given: A positive integer n ($n \leq 1000$).

Return: The total number of subsets of $\{1, 2, \ldots, n\}$ modulo 1,000,000.

Sample Dataset

```
3
```

Sample Output

```
8
```

> ### Hint
>
> What does counting subsets have to do with characters and "ON"/"OFF" switches?

Problem 51

Introduction to Alternative Splicing

The Baby and the Bathwater

In "RNA Splicing", we described the process by which the exons are spliced out from a molecule of pre-mRNA and reassembled to yield a final mRNA for the purposes of protein translation.

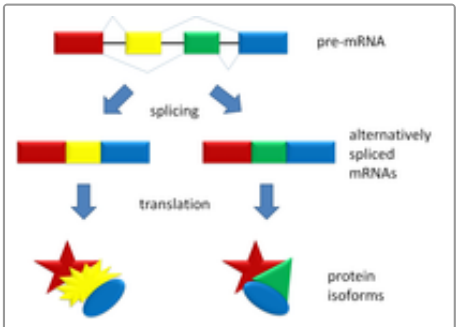


Figure 1. Alternative splicing induces different protein isoforms.

However, the chaining of exons does not always proceed in the same manner; **alternative splicing** describes the fact that all the exons from a gene are not necessarily joined together in order to produce an mRNA. The most common form of alternative splicing is **exon skipping**, in which certain exons are omitted along with introns.

Alternative splicing serves a vital evolutionary purpose, as it greatly increases the number of different proteins that can be translated from a given gene; different proteins produced from the same gene as a result of alternative splicing are called **protein isoforms**; see **Figure 1** In fact, about 95% of human genes are commonly spliced in more than one way. At the same time, when alternative splicing goes wrong, it can create the same negative effects caused by mutations, and it has been blamed for a number of genetic disorders.

In this problem, we will consider a simplified model of alternative splicing in which any of a collection of exons can be chained together to create a final molecule of mRNA, under the condition that we use a minimum number of exons (m) whose order is fixed. Because the exons are not allowed to move around, we need only select a subset of at least m of our exons to chain into an mRNA.

The implied computational question is to count the total number of such subsets, which will provide us with the total possible number of alternatively spliced isoforms for this model.

Problem

In "Counting Subsets", we saw that the total number of subsets of a set S containing n elements is equal to 2^n.

However, if we intend to count the total number of subsets of S having a fixed size k, then we use the **combination** statistic $C(n, k)$, also written $\binom{n}{k}$.

Given: Positive integers n and m with $0 \leq m \leq n \leq 2000$.

Return: The sum of combinations $C(n, k)$ for all k satisfying $m \leq k \leq n$, modulo 1,000,000. In shorthand, $\sum_{k=m}^{n} \binom{n}{k}$.

Sample Dataset

```
6 3
```

Sample Output

```
42
```

Problem 52

Edit Distance

Point Mutations Include Insertions and Deletions

In "Counting Point Mutations", we saw that Hamming distance gave us a preliminary notion of the evolutionary distance between two DNA strings by counting the minimum number of single nucleotide substitutions that could have occurred on the evolutionary path between the two strands.

> However, in practice, homologous strands of DNA or protein are rarely the same length because point mutations also include the insertion or deletion of a single nucleotide (and single amino acids can be inserted or deleted from peptides). Thus, we need to incorporate these insertions and deletions into the calculation of the minimum number of point mutations between two strings. One of the simplest models charges a unit "cost" to any single-symbol insertion/deletion, then (in keeping with parsimony) requests the minimum cost over all transformations of one genetic string into another by point substitutions, insertions, and deletions.

Problem

Given two strings s and t (of possibly different lengths), the **edit distance** $d_{\mathrm{E}}(s,t)$ is the minimum number of **edit operations** needed to transform s into t, where an edit operation is defined as the substitution, insertion, or deletion of a single symbol.

The latter two operations incorporate the case in which a contiguous interval is inserted into or deleted from a string; such an interval is called a **gap**. For the purposes of this problem, the insertion or deletion of a gap of length k still counts as k distinct edit operations.

Given: Two protein strings s and t in FASTA format (each of length at most 1000 aa).

Return: The edit distance $d_{\mathrm{E}}(s,t)$.

Sample Dataset

```
>Rosalind_39
PLEASANTLY
>Rosalind_11
MEANLY
```

Sample Output

```
5
```

Problem 53

Expected Number of Restriction Sites

A Shot in the Dark

In "Locating Restriction Sites", we first familiarized ourselves with restriction enzymes. Recall that these enzymes are used by bacteria to cut through both strands of viral DNA, thus disarming the virus: the viral DNA locations where these cuts are made are known as **restriction sites**. Recall also that every restriction enzyme is preprogrammed with a reverse palindromic interval of DNA to which it will bind and cut, called a **recognition sequence**. These even length intervals are usually either 4 or 6 base pairs long, although longer ones do exist; **rare-cutter enzymes** have recognition sequences of 8 or more base pairs.

In this problem, we will ask a simple question: how does the bacterium "know" that it will probably succeed in finding a restriction site within the virus's DNA? The answer is that the short length of recognition sequences guarantees a large number of matches occurring *randomly*.

Intuitively, we would expect for a recognition sequence of length 6 to occur on average once every $4^6 = 4,096$ base pairs. Note that this fact does not imply that the associated restriction enzyme will cut the viral DNA every 4,096 bp; it may find two restriction sites close together, then not find a restriction site for many thousand nucleotides.

In this problem, we will generalize the problem of finding an average number of restriction sites to take into account the GC-content of the underlying string being analyzed.

Problem

Say that you place a number of bets on your favorite sports teams. If their chances of winning are 0.3, 0.8, and 0.6, then you should expect on average to win 0.3 + 0.8 + 0.6 = 1.7 of your bets (of course, you can never win exactly 1.7!)

More generally, if we have a collection of events $A_1, A_2, \ldots, A_n$, then the expected number of events occurring is $\Pr(A_1) + \Pr(A_2) + \cdots + \Pr(A_n)$ (consult the note following the problem for a precise explanation of this fact). In this problem, we extend the idea of finding an expected number of events to finding the expected number of times that a given string occurs as a substring of a random string.

Given: A positive integer n ($n \leq 1,000,000$), a DNA string s of even length at most 10, and an array A of length at most 20, containing numbers between 0 and 1.

Return: An array B having the same length as A in which $B[i]$ represents the expected number of times that s will appear as a substring of a random DNA string t of length n, where t is formed with GC-content $A[i]$ (see "Introduction to Random Strings").

Sample Dataset

```
10
AG
```

```
0.25 0.5 0.75
```

Sample Output

```
0.422 0.563 0.422
```

> ### The Mathematical Details
>
> In this problem, we are speaking of an expected number of events; how can we tie this into the definition of expected value that we already have from "Calculating Expected Offspring"?
>
> The answer relies on a slick mathematical trick. For any event A, we can form a random variable for A, called an **indicator random variable** I_A. For an outcome x, $I_A(x) = 1$ when x belongs to A and $I_A(x) = 0$ when x belongs to A^c.
>
> For an indicator random variable $I_A(x) = 1$, verify that $\mathrm{E}(I_A) = \Pr(A)$.
>
> You should also verify from our original formula for expected value that for any two random variables X and Y, $\mathrm{E}(X + Y)$ is equal to $\mathrm{E}(X) + \mathrm{E}(Y)$. As a result, the expected number of events $A_1, A_2, \ldots, A_m$ occurring, or $\mathrm{E}(I_{A_1} + I_{A_2} + \cdots + I_{A_m})$, reduces to $\Pr(A_1) + \Pr(A_2) + \cdots + \Pr(A_m)$.

Problem 54

Motzkin Numbers and RNA Secondary Structures

> ### Dull, Unhappy People
>
> In "Catalan Numbers and RNA Secondary Structures", we talked about counting the number of ways for an even number of people to shake hands at a party without crossing hands. However, in the real world, parties only contain an even number of people about 40% of the time, and mathematicians don't enjoy socializing. So we should instead count the total number of ways for *some* of the people at the party to shake hands without crossing.
>
> In the biological world, people are far more social, but not every nucleotide in a strand of RNA winds up base pairing with another nucleotide during RNA folding. As a result, we want to loosen this assumption and count the total number of different secondary structures of an RNA strand whose base pairs don't overlap (i.e., we still forbid pseudoknots in the strand).

Problem

Similarly to our definition of the Catalan numbers, the n-th **Motzkin number** m_n counts the number of ways to form a (not necessarily perfect) noncrossing matching in the complete graph K_n containing n nodes. For example, **Figure 1** demonstrates that $m_5 = 21$. Note in this figure that technically, the "trivial" matching that contains no edges at all is considered to be a matching,

because it satisfies the defining condition that no two edges are incident to the same node.

How should we compute the Motzkin numbers? As with Catalan numbers, we will take $m_0 = m_1 = 1$. To calculate m_n in general, assume that the nodes of K_n are labeled around the outside of a circle with the integers between 1 and n, and consider node 1, which may or may not be involved in a matching. If node 1 is *not* involved in a matching, then there are m_{n-1} ways of matching the remaining $n-1$ nodes. If node 1 *is* involved in a matching, then say it is matched to node k: this leaves $k-2$ nodes on one side of edge $\{1,k\}$ and $n-k$ nodes on the other side; as with the Catalan numbers, no edge can connect the two sides, which gives us $m_{k-2} \cdot m_{n-k}$ ways of matching the remaining edges. Allowing k to vary between 2 and n yields the following recurrence relation for the Motzkin numbers: $m_n = m_{n-1} + \sum_{k=2}^{n} m_{k-2} \cdot m_{n-k}$.

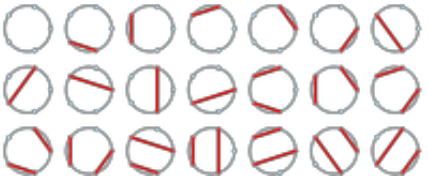


Figure 1. The 21 distinct ways to form a noncrossing matching on 5 labeled nodes.

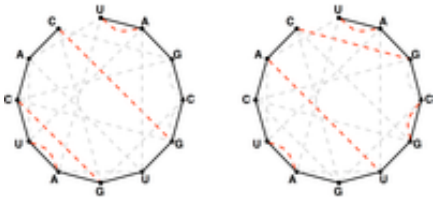


Figure 2. Two possible noncrossing matchings of basepair edges in the bonding graph corresponding to RNA string UAGCGUGAUCAC.

To count all possible secondary structures of a given RNA string that do not contain pseudoknots, we need to modify the Motzkin recurrence so that it counts only matchings of basepair edges in the bonding graph corresponding to the RNA string; see **Figure 2**

Given: An RNA string s of length at most 300 bp.

Return: The total number of noncrossing matchings of basepair edges in the bonding graph of s, modulo 1,000,000.

Sample Dataset

```
>Rosalind_57
AUAU
```

Sample Output

```
7
```

Problem 55

Distances in Trees

Paths in Trees

For any two nodes of a tree, a unique path connects the nodes; more specifically, there is a unique path connecting any pair of leaves. Why must this be the case? If more than one path connected two nodes, then they would necessarily form a cycle, which would violate the definition of tree.

> The uniqueness of paths connecting nodes in a tree is helpful in phylogenetic analysis because a rudimentary measure of the separation between two taxa is the distance between them in the tree, which is equal to the number of edges on the unique path connecting the two leaves corresponding to the taxa.

Problem

Newick format is a way of representing trees even more concisely than using an adjacency list, especially when dealing with trees whose internal nodes have not been labeled.

First, consider the case of a rooted tree T. A collection of leaves $v_1, v_2, \ldots, v_n$ of T are **neighbors** if they are all adjacent to some internal node u. Newick format for T is obtained by iterating the following key step: delete all the edges $\{v_i, u\}$ from T and label u with $(v_1, v_2, \ldots, v_n)u$. This process is repeated all the way to the root, at which point a semicolon signals the end of the tree.

A number of variations of Newick format exist. First, if a node is not labeled in T, then we simply leave blank the space occupied by the node. In the key step, we can write $(v_1, v_2, \ldots, v_n)$ in place of $(v_1, v_2, \ldots, v_n)u$ if the v_i are labeled; if none of the nodes are labeled, we can write $(, , \ldots,)$.

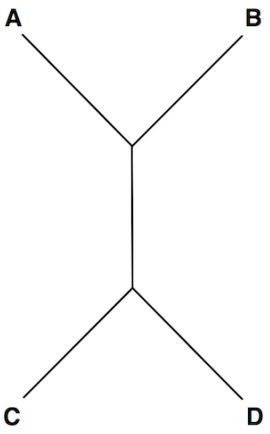


Figure 1. This tree can be represented in Newick format in a number of ways, including (C, D, (A, B)); (A, (D, C), B); and (((A, B), C))D;.

A second variation of Newick format occurs when T is unrooted, in which case we simply select any internal node to serve as the root of T. A particularly peculiar case of Newick format arises when we choose a leaf to serve as the root.

Note that there will be a large number of different ways to represent T in Newick format; see **Figure 1**.

Given: A collection of n trees ($n \leq 40$) in Newick format, with each tree containing at most 200 nodes; each tree T_k is followed by a pair of nodes x_k and y_k in T_k.

Return: A collection of n positive integers, for which the kth integer represents the distance between x_k and y_k in T_k.

Sample Dataset

```
(cat)dog;
dog cat

(dog,cat);
dog cat
```

Sample Output

```
1 2
```

Problem 56

Interleaving Two Motifs

Two Motifs, One Gene

Recall that in "Finding a Shared Spliced Motif", we found the longest motif that could have been shared by two genetic strings, allowing for the motif to be split onto multiple exons in the process. As a result, we needed to find a longest common subsequence of the two strings (which extended the problem of finding a longest common substring from "Finding a Shared Motif").

> In this problem, we consider an inverse problem of sorts in which we are given two different motifs and we wish to interleave them together in such a way as to produce a *shortest* possible genetic string in which both motifs occur as subsequences.

Problem

A string s is a **supersequence** of another string t if s contains t as a subsequence.

A **common supersequence** of strings s and t is a string that serves as a supersequence of both s and t. For example, "GACCTAGGAACTC" serves as a common supersequence of "ACGTC" and "ATAT". A **shortest common supersequence** of s and t is a supersequence for which there does not exist a shorter common supersequence. Continuing our example, "ACGTACT" is a shortest common supersequence of "ACGTC" and "ATAT".

Given: Two DNA strings s and t.

Return: A shortest common supersequence of s and t. If multiple solutions exist, you may output any one.

Sample Dataset

```
ATCTGAT
TGCATA
```

Sample Output

```
ATGCATGAT
```

Problem 57

Sorting by Reversals

Reconstructing Evolutionary Histories

When we calculate the Hamming distance between two genetic strings, we can easily infer a collection of point mutations that occurred on the evolutionary path between the two strings by simply examining the mismatched symbols. However, when calculating the reversal distance (see "Reversal Distance"), we only have the minimum *number* of reversals separating two permutations.

> The computational problem of **sorting by reversals** demands instead that we provide a minimum list of reversals transforming one permutation into another. As a result, sorting by reversals subsumes the calculation of reversal distance: once we have a minimum collection of reversals, the reversal distance follows immediately.

Problem

A reversal of a permutation can be encoded by the two indices at the endpoints of the interval that it inverts; for example, the reversal that transforms $(4, 1, 2, 6, 3, 5)$ into $(4, 1, 3, 6, 2, 5)$ is encoded by $[3, 5]$.

A collection of reversals **sorts** π into γ if the collection contains $d_{\text{rev}}(\pi, \gamma)$ reversals, which when successively applied to π yield γ.

Given: Two permutations π and γ, each of length 10.

Return: The reversal distance $d_{\text{rev}}(\pi, \gamma)$, followed by a collection of reversals sorting π into γ. If multiple collections of such reversals exist, you may return any one.

Sample Dataset

```
1 2 3 4 5 6 7 8 9 10
1 8 9 3 2 7 6 5 4 10
```

Sample Output

```
2
4 9
2 5
```

Problem 58

Inferring Protein from Spectrum

> ### Introduction to Mass Spectrometry

In "Calculating Protein Mass", we briefly mentioned an analytic chemical method called mass spectrometry, which aims to measure the mass-to-charge ratio of a particle or a molecule. In a mass spectrometer, a sample is vaporized (turned into gas), and then particles from the sample are ionized. The resulting ions are placed into an electromagnetic field, which separates them based on their charge and mass. The output of the mass spectrometer is a **mass spectrum**, or a plot of ions' possible mass-to-charge ratio values with the **intensity** (actual observed frequency) of ions having these mass-to-charge values.

> For the moment, we will ignore charge and consider a list of the ions' monoisotopic masses as a **simplified spectrum**. Researchers do not possess cheap technology to go in and examine a protein one amino acid at a time (molecules are too submicroscopic). Instead, to determine a protein's structure, we will split several copies of the protein into smaller pieces, then weigh the resulting fragments. To do this, we assume that each **cut** (breakage point) occurs between two amino acids and that we can measure the mass of the resulting pieces for all possible cuts.
>
> For example, the (unknown) protein "PRTEIN" can be cut in five possible ways: "P" and "RTEIN"; "PR" and "TEIN"; "PRT" and "EIN"; "PRTE" and "IN"; "PRTEI" and "N". We then can measure the masses of all fragments, including the entire string. The "left" end of a protein is called its **N-terminus**, and the ions corresponding to the protein string's prefixes (P, PR, PRT, PRTE, PRTEI) are called **b-ions**. The "right" end of the protein is called its **C-terminus**, and the ions corresponding to the string's suffixes (N, IN, EIN, TEIN, RTEIN) are called **y-ions**. The difference in the masses of two adjacent b-ions (or y-ions) gives the mass of one amino acid residue; for example, the difference between the masses of "PRT" and "PR" must be the mass of "T." By extension, knowing the masses of every b-ion of a protein allows us to deduce the protein's identity.

Problem

The **prefix spectrum** of a weighted string is the collection of all its prefix weights.

Given: A list L of n ($n \leq 100$) positive real numbers.

Return: A protein string of length $n - 1$ whose prefix spectrum is equal to L (if multiple solutions exist, you may output any one of them). Consult the monoisotopic mass table.

Sample Dataset

```
3524.8542
3710.9335
3841.974
3970.0326
4057.0646
```

Sample Output

```
WMQS
```

Problem 59

Introduction to Pattern Matching

If At First You Don't Succeed...

We introduced the problem of finding a motif in a genetic string in "Finding a Motif in DNA". More commonly, we will have a collection of motifs that we may wish to find in a larger string, for example when searching a genome for a collection of known genes.

> This application sets up the algorithmic problem of **pattern matching**, in which we are searching a large string (called a **text**) for instances of a collection of smaller strings, called **patterns**. For the moment, we will focus on requiring that all matches should be exact.
>
> The most obvious method for finding exact patterns in a text is to simply apply a simple "sliding window" algorithm for each pattern. However, this method is time-consuming if we have a large number of patterns to consider (which will often be the case when dealing with a database of genes). It would be better if instead of traversing the genome for every pattern, we could somehow only traverse it once. To this end, we will need a data structure that can efficiently organize a collection of patterns.

Problem

Given a collection of strings, their **trie** (often pronounced "try" to avoid ambiguity with the general term tree) is a rooted tree formed as follows. For every unique first symbol in the strings, an edge is formed connecting the root to a new vertex. This symbol is then used to label the edge.

We may then iterate the process by moving down one level as follows. Say that an edge connecting the root to a node v is labeled with 'A'; then we delete the first symbol from every string in the collection beginning with 'A' and then treat v as our root. We apply this process to all nodes that are adjacent to the root, and then we move down another level and continue. See **Figure 1** for an example of a trie.

Figure 1. The trie corresponding to the strings 'apple', 'apropos', 'banana', 'bandana', and 'orange'. Each path from root to leaf encodes one of these strings.

As a result of this method of construction, the symbols along the edges of any path in the trie from the root to a leaf will spell out a unique string from the collection, as long as no string is a prefix of another in the collection (this would cause the first string to be encoded as a path terminating at an internal node).

Given: A list of at most 100 DNA strings of length at most 100 bp, none of which is a prefix of another.

Return: The adjacency list corresponding to the trie T for these patterns, in the following format. If T has n nodes, first label the root with 1 and then label the remaining nodes with the integers 2 through n in any order you like. Each edge of the adjacency list of T will be encoded by a triple containing the integer representing the edge's parent node, followed by the integer representing the edge's child node, and finally the symbol labeling the edge.

Sample Dataset

```
ATAGA
ATC
GAT
```

Sample Output

```
1 2 A
2 3 T
3 4 A
4 5 G
5 6 A
3 7 C
1 8 G
8 9 A
9 10 T
```

Problem 60

Comparing Spectra with the Spectral Convolution

Comparing Spectra

Suppose you have two mass spectra, and you want to check if they both were obtained from the same protein; you will need some notion of spectra similarity. The simplest possible metric would be to count the number of peaks in the mass spectrum that the spectra share, called the **shared peaks count**; its analogue for simplified spectra is the number of masses that the two spectra have in common.

> The shared peaks count can be useful in the simplest cases, but it does not help us if, for example, one spectrum corresponds to a peptide contained inside of another peptide from which the second spectrum was obtained. In this case, the two spectra are very similar, but the shared peaks count will be very small. However, if we shift one spectrum to the right or left, then shared peaks will align. In the case of simplified spectra, this means that there is some shift value x such that adding x to the weight of every element in one spectrum should create a large number of matches in the other spectrum.

Problem

A **multiset** is a generalization of the notion of set to include a collection of objects in which each object may occur more than once (the order in which objects are given is still unimportant). For a multiset S, the **multiplicity** of an element x is the number of times that x occurs in the set; this multiplicity is denoted $S(x)$. Note that every set is included in the definition of multiset.

The **Minkowski sum** of multisets S_1 and S_2 containing real numbers is the new multiset $S_1 \oplus S_2$ formed by taking all possible sums $s_1 + s_2$ of an element s_1 from S_1 and an element s_2 from S_2. The Minkowski sum could be defined more concisely as

$S_1 \oplus S_2 = s_1 + s_2 : s_1 \in S_1, s_2 \in S_2$, The **Minkowski difference** $S_1 \ominus S_2$ is defined analogously by taking all possible differences $s_1 - s_2$.

If S_1 and S_2 represent simplified spectra taken from two peptides, then $S_1 \ominus S_2$ is called the **spectral convolution** of S_1 and S_2. In this notation, the shared peaks count is represented by $(S_2 \ominus S_1)(0)$, and the value of x for which $(S_2 \ominus S_1)(x)$ has the maximal value is the shift value maximizing the number of shared masses of S_1 and S_2.

Given: Two multisets of positive real numbers S_1 and S_2. The size of each multiset is at most 200.

Return: The largest multiplicity of $S_1 \ominus S_2$, as well as the absolute value of the number x maximizing $(S_1 \ominus S_2)(x)$ (you may return any such value if multiple solutions exist).

Sample Dataset

```
186.07931 287.12699 548.20532 580.18077 681.22845 706.27446 782.27613 9
68.35544 968.35544
101.04768 158.06914 202.09536 318.09979 419.14747 463.17369
```

Sample Output

```
3
85.03163
```

Note

Observe that $S_1 \oplus S_2$ is equivalent to $S_2 \oplus S_1$, but it is not usually the case that $S_1 \ominus S_2$ is the same as $S_2 \ominus S_1$; in this case, one multiset can be obtained from the other by negating every element.

Problem 61

Creating a Character Table

Introduction to Character-Based Phylogeny

Before the modern genetics revolution, phylogenies were constructed from physical characters resulting from direct structural comparison of taxa. A great deal of analysis relied on the fossil record, as fossils provided the only concrete framework for studying the appearance of extinct species and for inferring how they could have evolved into present-day organisms.

A classic case illustrating the utility of the fossil record is the case of dinosaur pelvic bones. In 1887, Harry Seeley proposed a new classification of dinosaurs into two orders, Saurischia and Ornithischia: the former possessed hip bones shaped like those found in reptiles, whereas the latter had a much different hip shape that resembled birds. Seeley's pelvic classification has

> survived to the present day as the principal division of dinosaurs.
>
> The key point is that hip bone shape is a physical character that separates all dinosaurs into two different groups. Our hope is to construct a phylogeny solely from a collection of characters. Throughout character-based phylogeny, our two-part assumption is that all taxa possessing a character must have evolved from a single ancestor that introduced this character, and conversely, any taxon not possessing the character cannot be descended from this ancestor.

Problem

Given a collection of n taxa, any subset S of these taxa can be seen as encoding a character that divides the taxa into the sets S and S^c; we can represent the character by $S \mid S^c$, which is called a **split**. Alternately, the character can be represented by a **character array** A of length n for which $A[j] = 1$ if the jth taxon belongs to S and $A[j] = 0$ if the jth taxon belongs to S^c (recall the "ON"/"OFF" analogy from "Counting Subsets").

At the same time, observe that the removal of an edge from an unrooted binary tree produces two separate trees, each one containing a subset of the original taxa. So each edge may also be encoded by a split $S \mid S^c$.

A **trivial character** isolates a single taxon into a group of its own. The corresponding split $S \mid S^c$ must be such that S or S^c contains only one element; the edge encoded by this split must be incident to a leaf of the unrooted binary tree, and the array for the character contains exactly one 0 or exactly one 1. Trivial characters are of no phylogenetic interest because they fail to provide us with information regarding the relationships of taxa to each other. All other characters are called **nontrivial characters** (and the associated splits are called **nontrivial splits**).

A **character table** is a matrix C in which each row represents the array notation for a nontrivial character. That is, entry $C_{i,j}$ denotes the "ON"/"OFF" position of the ith character with respect to the jth taxon.

Given: An unrooted binary tree T in Newick format for at most 200 species taxa.

Return: A character table having the same splits as the edge splits of T. The columns of the character table should encode the taxa ordered lexicographically; the rows of the character table may be given in any order. Also, for any given character, the particular subset of taxa to which 1s are assigned is arbitrary.

Sample Dataset

```
(dog,((elephant,mouse),robot),cat);
```

Sample Output

```
00110
00111
```

Problem 62

Constructing a De Bruijn Graph

Wading Through the Reads

Because we use multiple copies of the genome to generate and identify reads for the purposes of fragment assembly, the total length of all reads will be much longer than the genome itself. This begs the definition of **read coverage** as the average number of times that each nucleotide from the genome appears in the reads. In other words, if the total length of our reads is 30 billion bp for a 3 billion bp genome, then we have 10x read coverage.

> To handle such a large number of k-mers for the purposes of sequencing the genome, we need an efficient and simple structure.

Problem

Consider a set S of $(k+1)$-mers of some unknown DNA string. Let S^{rc} denote the set containing all reverse complements of the elements of S. (recall from "Counting Subsets" that sets are not allowed to contain duplicate elements).

The **de Bruijn graph** B_k of order k corresponding to $S \cup S^{\text{rc}}$ is a digraph defined in the following way:

- Nodes of B_k correspond to all k-mers that are present as a substring of a $(k+1)$-mer from $S \cup S^{\text{rc}}$.
- Edges of B_k are encoded by the $(k+1)$-mers of $S \cup S^{\text{rc}}$ in the following way: for each $(k+1)$-mer r in $S \cup S^{\text{rc}}$, form a directed edge $(r[1:k], r[2:k+1])$.

Given: A collection of up to 1000 DNA strings of equal length (not exceeding 50 bp) corresponding to a set S of $(k+1)$-mers.

Return: The adjacency list corresponding to the de Bruijn graph corresponding to $S \cup S^{\text{rc}}$.

Sample Dataset

```
TGAT
CATG
TCAT
ATGC
CATC
CATC
```

Sample Output

```
(ATC, TCA)
(ATG, TGA)
(ATG, TGC)
(CAT, ATC)
(CAT, ATG)
(GAT, ATG)
```

```
(GCA, CAT)
(TCA, CAT)
(TGA, GAT)
```

Problem 63

Edit Distance Alignment

Reconstructing Edit Distance

In “Counting Point Mutations”, the calculation of Hamming distance gave us a clear way to model the sequence of point mutations transforming one genetic string into another. By simply writing one string directly over the other, we could count each mismatched symbol as a substitution.

> However, in the calculation of edit distance (see “Edit Distance”), the two strings can have different lengths; thus, simply superimposing one string over the other does us no good when it comes to visualizing a sequence of edit operations transforming one string into the other. To remedy this, we will introduce a new symbol to serve as a placeholder representing an inserted or deleted symbol; furthermore, this placeholder will allow us to align two strings of differing lengths.

Problem

An **alignment** of two strings s and t is defined by two strings s' and t' satisfying the following three conditions: 1. s' and t' must be formed from adding **gap symbols** "-" to each of s and t, respectively; as a result, s and t will form subsequences of s' and t'. 2. s' and t' must have the same length. 3. Two gap symbols may not be aligned; that is, if $s'[j]$ is a gap symbol, then $t'[j]$ cannot be a gap symbol, and vice-versa.

We say that s' and t' **augment** s and t. Writing s' directly over t' so that symbols are *aligned* provides us with a scenario for transforming s into t. Mismatched symbols from s and t correspond to symbol substitutions; a gap symbol $s'[j]$ aligned with a non-gap symbol $t'[j]$ implies the insertion of this symbol into t; a gap symbol $t'[j]$ aligned with a non-gap symbol $s'[j]$ implies the deletion of this symbol from s.

Thus, an alignment represents a transformation of s into t via edit operations. We define the corresponding **edit alignment score** of s' and t' as $d_{\mathrm{H}}(s', t')$ (Hamming distance is used because the gap symbol has been introduced for insertions and deletions). It follows that $d_{\mathrm{E}}(s, t) = \min_{s', t'} d_{\mathrm{H}}(s', t')$, where the minimum is taken over all alignments of s and t. We call such a minimum score alignment an **optimal alignment** (with respect to edit distance).

Given: Two protein strings s and t in FASTA format (with each string having length at most 1000 aa).

Return: The edit distance $d_{\mathrm{E}}(s, t)$ followed by two augmented strings s' and t' representing an optimal alignment of s and t.

Sample Dataset

```
>Rosalind_43
PRETTY
>Rosalind_97
PRTTEIN
```

Sample Output

```
4
PRETTY--
PR-TTEIN
```

Problem 64

Inferring Peptide from Full Spectrum

Ions Galore

In "Inferring Protein from Spectrum", we inferred a protein string from a list of b-ions. In practice, biologists have no way of distinguishing between b-ions and y-ions in the simplified spectrum of a peptide. However, we will often possess a pair of masses in the spectrum corresponding to a single cut. The two corresponding ions complement each other: for example, mass("PR") + mass("TEIN") = mass("PRTEIN"). As a result, we can easily infer the mass of a b-ion from its complementary y-ion and vice versa, as long as we already know the **parent mass**, i.e., the mass of the entire peptide.

The theoretical simplified spectrum for a protein P of length n is constructed as follows: form all possible cuts, then compute the mass of the b-ion and the y-ion at each cut. Duplicate masses are allowed. You might guess how we could modify "Inferring Protein from Spectrum" to infer a peptide from its theoretical simplified spectrum; here we consider a slightly modified form of this problem in which we attempt to identify the interior region of a peptide given only b-ions and y-ions that are cut within this region. As a result, we will have constant masses at the beginning and end of the peptide that will be present in the mass of every b-ion and y-ion, respectively.

Problem

Say that we have a string s containing t as an internal substring, so that there exist nonempty substrings s_1 and s_2 of s such that s can be written as $s_1 t s_2$. A **t-prefix** contains all of s_1 and none of s_2; likewise, a **t-suffix** contains all of s_2 and none of s_1.

Given: A list L containing $2n + 3$ positive real numbers ($n \leq 100$). The first number in L is the parent mass of a peptide P, and all other numbers represent the masses of some b-ions and y-ions of P (in no particular order). You may assume that if the mass of a b-ion is present, then so is that of its complementary y-ion, and vice-versa.

Return: A protein string t of length n for which there exist two positive real numbers w_1 and w_2 such that for every prefix p and suffix s of t, each of $w(p) + w_1$ and $w(s) + w_2$ is equal to an

element of L. (In other words, there exists a protein string whose t-prefix and t-suffix weights correspond to the non-parent mass values of L.) If multiple solutions exist, you may output any one.

Sample Dataset

```
1988.21104821
610.391039105
738.485999105
766.492149105
863.544909105
867.528589105
992.587499105
995.623549105
1120.6824591
1124.6661391
1221.7188991
1249.7250491
1377.8200091
```

Sample Output

```
KEKEP
```

Problem 65

Independent Segregation of Chromosomes

Mendel's Work Examined

Mendel's laws of heredity were initially ignored, as only 11 papers have been found that cite his paper between its publication in 1865 and 1900. One reason for Mendel's lack of popularity is that information did not move quite so readily as in the modern age; perhaps another reason is that as a friar in an Austrian abbey, Mendel was already isolated from Europe's university community.

It is fair to say that no one who did initially read Mendel's work fully believed that traits for more complex organisms, like humans, could be broken down into discrete units of heredity (i.e., Mendel's factors). This skepticism was well-founded in empirical studies of inheritance, which indicated a far more complex picture of heredity than Mendel's theory dictated. The friar himself admitted that representing every trait with a single factor was overly simplistic, and so he proposed that some traits are **polymorphic**, or encoded by multiple different factors.

Yet any hereditary model would ultimately be lacking without an understanding of how traits are physically passed from organisms to their offspring. This physical mechanism was facilitated by Walther Flemming's 1879 discovery of chromosomes in salamander eggs during cell division, followed by Theodor Boveri's observation that sea urchin embryos with chromatin removed failed to

develop correctly (implying that traits must somehow be encoded on chromosomes). By the turn of the 20th century, Mendel's work had been rediscovered by Hugo de Vries and Carl Correns, but it was still unclear how Mendel's hereditary model could be tied to chromosomes.

Fortunately, Walter Sutton demonstrated that grasshopper chromosomes occur in matched pairs called **homologous chromosomes**, or homologs. We now know that the DNA found on homologous chromosomes is identical except for minor variations attributable to SNPs and small rearrangements, which are typically insertions and deletions. Sutton himself, working five decades before Watson & Crick and possessing no real understanding of DNA, actually surmised that variations to homologous chromosomes should somehow correspond to Mendel's alleles.

Yet it still remained to show how chromosomes themselves are inherited. Most multicellular organisms are **diploid**, meaning that their cells possess two sets of chromosomes; humans are included among diploid organisms, having 23 homologous chromosome pairs.

Gametes (i.e., sex cells) in diploid organisms form an exception and are **haploid**, meaning that they only possess one chromosome from each pair of homologs. During the fusion of two gametes of opposite sex, a diploid embryo is formed by simply uniting the two gametes' halved chromosome sets.

Mendel's first law can now be explained by the fact that during the meiosis each gamete randomly selects one of the two available alleles of the particular gene.

Mendel's second law follows from the fact that gametes select *nonhomologous* chromosomes independently of each other; however, this law will hold only for factors encoded on nonhomologous chromosomes, which leaves open the inheritance of factors encoded on homologous chromosomes.

Problem

Consider a collection of coin flips. One of the most natural questions we can ask is if we flip a coin 92 times, what is the probability of obtaining 51 "heads", vs. 27 "heads", vs. 92 "heads"?

Each coin flip can be modeled by a uniform random variable in which each of the two outcomes ("heads" and "tails") has probability equal to 1/2. We may assume that these random variables are independent (see "Independent Alleles"); in layman's terms, the outcomes of the two coin flips do not influence each other.

A **binomial random variable** X takes a value of k if n consecutive "coin flips" result in k total "heads" and $n - k$ total "tails." We write that $X \in \text{Bin}(n, 1/2)$.

Given: A positive integer $n \leq 50$.

Return: An array A of length $2n$ in which $A[k]$ represents the common logarithm of the probability that two diploid siblings share at least k of their $2n$ chromosomes (we do not consider recombination for now).

Sample Dataset

```
5
```

Sample Output

```
0.000 -0.004 -0.024 -0.082 -0.206 -0.424 -0.765 -1.262 -1.969 -3.010
```

Problem 66

Finding Disjoint Motifs in a Gene

Disjoint Motifs

In this problem, we will consider an algorithmic (but not particularly practical) variant of motif finding for multiple motifs. Say we have two motifs corresponding to the coding regions of genes, and we want to know whether these motifs can be found in genes occupying the same region of the genome. To prevent exons from coinciding, we further stipulate that the two motifs are nonoverlapping.

In this problem, we will ask whether two disjoint motifs can be located in a given string. We considered a similar problem in "Interleaving Two Motifs", which asked us to find a shortest possible string containing two motifs; however, in that problem, the motifs were allowed to coincide.

Problem

Given three strings s, t, and u, we say that t and u can be **interwoven** into s if there is some substring of s made up of t and u as disjoint subsequences.

For example, the strings "ACAG" and "CCG" can be interwoven into "GACCACGGTT". However, they cannot be interwoven into "GACCACAAAAGGTT" because of the appearance of the four 'A's in the middle of the subsequences. Similarly, even though both "ACACG" is a shortest common supersequence of ACAG and CCG, it is not possible to interweave these two strings into "ACACG" because the two desired subsequences must be disjoint; see "Interleaving Two Motifs" for details on finding a shortest common supersequence of two strings.

Given: A text DNA string s of length at most 10 kbp, followed by a collection of n ($n \leq 10$) DNA strings of length at most 10 bp acting as patterns.

Return: An $n \times n$ matrix M for which $M_{j,k} = 1$ if the jth and kth pattern strings can be interwoven into s and $M_{j,k} = 0$ otherwise.

Sample Dataset

```
GACCACGGTT
ACAG
GT
CCG
```

Sample Output

```
0 0 1
0 1 0
1 0 0
```

Citation

This problem follows Jones & Pevzner, *An Introduction to Bioinformatics Algorithms*, Problem 6.31

Problem 67

Finding the Longest Multiple Repeat

Long Repeats

We saw in "Introduction to Pattern Matching" that a data structure commonly used to encode the relationships among a collection of strings was the trie, which is particularly useful when the strings represent a collection of patterns that we wish to match to a larger text.

> The trie is helpful when processing multiple strings at once, but when we want to analyze a single string, we need something different.
>
> In this problem, we will use a new data structure to handle the problem of finding long repeats in the genome. Recall from "Finding a Motif in DNA" that cataloguing these repeats is a problem of the utmost interest to molecular biologists, as a natural correlation exists between the frequency of a repeat and its influence on RNA transcription. Our aim is therefore to identify long repeats that occur more than some predetermined number of times.

Problem

A **repeated substring** of a string s of length n is simply a substring that appears in more than one location of s; more specifically, a **k-fold substring** appears in at least k distinct locations.

Figure 1. The suffix tree for s = GTCCGAAGCTCCGG. Note that the dollar sign has been appended to a substring of the tree to mark the end of s. Every path from the root to a leaf corresponds to a unique suffix of GTCCGAAGCTCCGG, and each leaf is labeled with the location in s of the suffix ending at that leaf.

The **suffix tree** of s, denoted $T(s)$, is defined as follows:

- $T(s)$ is a rooted tree having exactly n leaves.
- Every edge of $T(s)$ is labeled with a substring of s^*, where s^* is the string formed by adding a placeholder symbol `$` to the end of s.
- Every internal node of $T(s)$ other than the root has at least two children; i.e., it has degree at least 3.
- The substring labels for the edges leading from a node to its children must begin with different symbols.
- By concatenating the substrings along edges, each path from the root to a leaf corresponds to a unique suffix of s^*.

See **Figure 1** for an example of a suffix tree.

Given: A DNA string s (of length at most 20 kbp) with `$` appended, a positive integer k, and a list of edges defining the suffix tree of s. Each edge is represented by four components:

1. the label of its parent node in $T(s)$;

2. the label of its child node in $T(s)$;
3. the location of the substring t of s^* assigned to the edge; and
4. the length of t.

Return: The longest substring of s that occurs at least k times in s. (If multiple solutions exist, you may return any single solution.)

Sample Dataset

```
CATACATAC$
2
node1 node2 1 1
node1 node7 2 1
node1 node14 3 3
node1 node17 10 1
node2 node3 2 4
node2 node6 10 1
node3 node4 6 5
node3 node5 10 1
node7 node8 3 3
node7 node11 5 1
node8 node9 6 5
node8 node10 10 1
node11 node12 6 5
node11 node13 10 1
node14 node15 6 5
node14 node16 10 1
```

Sample Output

```
CATAC
```

> ### Hint
>
> How can repeated substrings of s be located in $T(s)$?

Problem 68

Newick Format with Edge Weights

> ### Weighting the Tree
>
> A vital goal of creating phylogenies is to quantify a molecular clock that indicates the amount of evolutionary time separating two members of the phylogeny. To this end, we will assign numbers to the edges of a tree so that the number assigned to an edge represents the amount of time

separating the two species at each end of the edge. More generally, the evolutionary time between any two species will be given by simply adding the individual times connecting the nodes.

Problem

In a **weighted tree**, each edge is assigned a (usually positive) number, called its **weight**. The distance between two nodes in a weighted tree becomes the sum of the weights along the unique path connecting the nodes.

To generalize Newick format to the case of a weighted tree T, during our repeated "key step," if leaves $v_1, v_2, \ldots, v_n$ are neighbors in T, and all these leaves are incident to u, then we replace u with $(v_1 : d_1, v_2 : d_2, \ldots, v_n : d_n)u$, where d_i is now the weight on the edge $\{v_i, u\}$.

Given: A collection of n weighted trees ($n \leq 40$) in Newick format, with each tree containing at most 200 nodes; each tree T_k is followed by a pair of nodes x_k and y_k in T_k.

Return: A collection of n numbers, for which the kth number represents the distance between x_k and y_k in T_k.

Sample Dataset

```
(dog:42,cat:33);
cat dog

((dog:4,cat:3):74,robot:98,elephant:58);
dog elephant
```

Sample Output

```
75 136
```

Problem 69

Wobble Bonding and RNA Secondary Structures

Don't Look Down

We have discussed the problem of counting RNA secondary structures in previous problems. In this problem, we will add some assumptions to those used in "Motzkin Numbers and RNA Secondary Structures" to provide ourselves with an ultimately robust way of counting feasible RNA secondary structures.

> First, in addition to the classic Watson and Crick base pairing of adenine with uracil and cytosine with guanine, uracil sometimes bonds with guanine, forming what is called a **wobble base pair**. As a result, we would like to allow wobble base pairing.

> Second, although RNA folds over itself during base pairing, the structure of the molecule is rigid enough to prevent bases located very close to each other on the strand from bonding to each other.

Problem

Given an RNA string s, we will augment the bonding graph of s by adding basepair edges connecting all occurrences of 'U' to all occurrences of 'G' in order to represent possible wobble base pairs.

We say that a matching in the bonding graph for s is **valid** if it is noncrossing (to prevent pseudoknots) and has the property that a basepair edge in the matching cannot connect symbols s_j and s_k unless $k \geq j + 4$ (to prevent nearby nucleotides from base pairing).

See **Figure 1** for an example of a valid matching if we allow wobble base pairs. In this problem, we will wish to count all possible valid matchings in a given bonding graph; see **Figure 2** for all possible valid matchings in a small bonding graph, assuming that we allow wobble base pairing.

Given: An RNA string s (of length at most 200 bp).

Return: The total number of distinct valid matchings of basepair edges in the bonding graph of s. Assume that wobble base pairing is allowed.

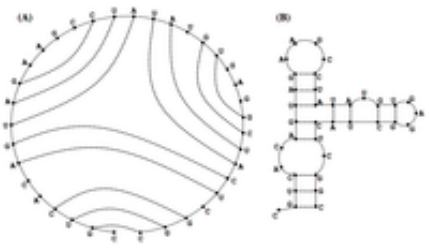


Figure 1. A valid matching of basepair edges in the bonding graph of an RNA string, followed by a diagram of how this bonding will induce the resulting folded RNA.

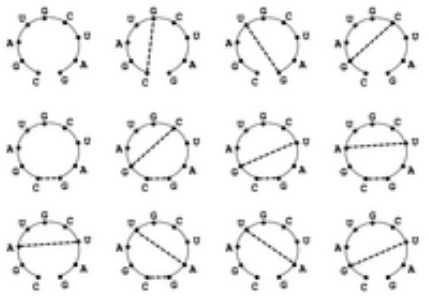


Figure 2. All 12 possible valid basepair matchings in the bonding graph corresponding to the string s = CGAUGCUAG (including the trivial matching in which no edges are matched.) Courtesy Brian Tjaden.

Sample Dataset

```
AUGCUAGUACGGAGCGAGUCUAGCGAGCGAUGUCGUGAGUAC
UAUAUAUGCGCAUAAGCCACGU
```

Sample Output

```
284850219977421
```

Problem 70

Counting Disease Carriers

Genetic Drift and the Hardy-Weinberg Principle

Mendel's laws of segration and independent assortment are excellent for the study of individual organisms and their progeny, but they say nothing about how alleles move through a population over time. Our first question is: when can we assume that the ratio of an allele in a population, called the **allele frequency**, is

stable?

G. H. Hardy and Wilhelm Weinberg independently considered this question at the turn of the 20th Century, shortly after Mendel's ideas had been rediscovered. They concluded that the percentage of an allele in a population of individuals is in **genetic equilibrium** when five conditions are satisfied:

1. The population is so large that random changes in the allele frequency are negligible.
2. No new mutations are affecting the gene of interest;
3. The gene does not influence survival or reproduction, so that natural selection is not occurring;
4. **Gene flow**, or the change in allele frequency due to migration into and out of the population, is negligible.
5. Mating occurs randomly with respect to the gene of interest.

The **Hardy-Weinberg principle** states that if a population is in genetic equilibrium for a given allele, then its frequency will remain constant and evenly distributed through the population. Unless the gene in question is important to survival or reproduction, Hardy-Weinberg usually offers a reasonable enough model of population genetics.

One of the many benefits of the Mendelian theory of inheritance and simplifying models like Hardy-Weinberg is that they help us predict the probability with which genetic diseases will be inherited, so as to take appropriate preventative measures. Genetic diseases are usually caused by mutations to chromosomes, which are passed on to subsequent generations.

The simplest and most widespread case of a genetic disease is a **single gene disorder**, which is caused by a single mutated gene. Over 4,000 such human diseases have been identified, including **cystic fibrosis** and **sickle-cell anemia**. In both of these cases, the individual must possess two recessive alleles for a gene in order to contract the disease. Thus, carriers can live their entire lives without knowing that they can pass the disease on to their children.

The above introduction to genetic equilibrium leaves us with a basic and yet very practical question regarding gene disorders: if we know the number of people who have a disease encoded by a recessive allele, can we predict the number of carriers in the population?

Problem

To model the Hardy-Weinberg principle, assume that we have a population of N diploid individuals. If an allele is in genetic equilibrium, then because mating is random, we may view the $2N$ chromosomes as receiving their alleles uniformly. In other words, if there are m dominant alleles, then the probability of a selected chromosome exhibiting the dominant allele is simply $p = \frac{m}{2N}$.

Because the first assumption of genetic equilibrium states that the population is so large as to be ignored, we will assume that N is infinite, so that we only need to concern ourselves with the value of p.

Given: An array A for which $A[k]$ represents the proportion of homozygous recessive individuals for the k-th Mendelian factor in a diploid population. Assume that the population is in genetic equilibrium for all factors.

Return: An array B having the same length as A in which $B[k]$ represents the probability that a randomly selected individual carries at least one copy of the recessive allele for the k-th factor.

Sample Dataset

```
0.1 0.25 0.5
```

Sample Output

```
0.532 0.75 0.914
```

Problem 71

Creating a Character Table from Genetic Strings

Phylogeny from Genetic Characters

In "Creating a Character Table", we introduced the character table as a way of representing a number of characters simultaneously. In that problem, we found a character table representing an unrooted binary tree on a collection of taxa.

> Of course, in practice, the problem operates in reverse. We first need to generate a character table before we can model a phylogeny on this table. In modern genetics, a reliable way to obtain a large number of characters is by using SNPs. As mentioned in "Counting Subsets", for a given SNP, we can divide taxa into two sets depending on which of two bases is present at the nucleotide, thus defining the split of a character.

Problem

A collection of strings is **characterizable** if there are at most two possible choices for the symbol at each position of the strings.

Given: A collection of at most 100 characterizable DNA strings, each of length at most 300 bp.

Return: A character table for which each nontrivial character encodes the symbol choice at a single position of the strings. (Note: the choice of assigning '1' and '0' to the two states of each SNP in the strings is arbitrary.)

Sample Dataset

```
ATGCTACC
CGTTTACC
ATTCGACC
AGTCTCCC
CGTCTATC
```

Sample Output

```
10110
10100
```

> **Note**
>
> Recall that the character table does not encode trivial characters.

Problem 72

Counting Optimal Alignments

Beware of Alignment Inference

In "Edit Distance Alignment", we introduced the concept of an alignment of two genetic strings having differing lengths with respect to edit distance. This provided us with a way of visualizing a specific collection of symbol substitutions, insertions, and deletions that could have taken place on the evolutionary path between the two strings.

However, simply finding one optimal alignment and declaring that it represents a true evolutionary history is a dangerous idea because the actual evolutionary picture may be suboptimal. For that matter, the collection of all optimal alignments may be huge, and the characteristics of these alignments could differ widely.

In order to begin analyzing the collection of optimal alignments for a pair of strings, the first question we will ask is simple: just how many optimal alignments exist?

Problem

Recall from "Edit Distance Alignment" that if s' and t' are the augmented strings corresponding to an alignment of strings s and t, then the edit alignment score of s' and t' was given by the Hamming distance $d_{\mathrm{H}}(s', t')$ (because s' and t' have the same length and already include gap symbols to denote insertions/deletions).

As a result, we obtain $d_{\mathrm{E}}(s, t) = \min_{s', t'} d_{\mathrm{H}}(s', t')$, where the minimum is taken over all alignments of s and t. Strings s' and t' achieving this minimum correspond to an optimal alignment with respect to edit alignment score.

Given: Two protein strings s and t in FASTA format, each of length at most 1000 aa.

Return: The total number of optimal alignments of s and t with respect to edit alignment score, modulo 134,217,727 (2^{27}-1).

Sample Dataset

```
>Rosalind_78
PLEASANTLY
>Rosalind_33
MEANLY
```

Sample Output

```
4
```

> #### Why Are We Counting Modulo 134,217,727?
>
> As a simple example, say that we attempt to count some statistic modulo 10. If computing the statistic requires us to multiply a collection of integers, and at any point we multiply by a multiple of 10, then the statistic will automatically become a multiple of 10 and thus congruent to 0 modulo 10. A similar issue can arise when we count a huge number modulo any composite number; however, if we count modulo a large prime number p (i.e., one without any divisors other than itself), then problems can only ever arise if when counting our statistic, we multiply by a multiple of p.

Problem 73

Counting Unrooted Binary Trees

Counting Trees

A natural question is to be able to count the total number of distinct unrooted binary trees having n leaves, where each leaf is labeled by some taxon. Before we can count all these trees, however, we need to have a notion of when two such trees are the same.

> Our tool will be the split. Recall from "Creating a Character Table" that removing any edge from a tree T separates its leaves into sets S and S^c, so that each edge of T can be labeled by this split $S \mid S^c$. As a result, an unrooted binary tree can be represented uniquely by its collection of splits.

Problem

Two unrooted binary trees T_1 and T_2 having the same n labeled leaves are considered to be equivalent if there is some assignment of labels to the internal nodes of T_1 and T_2 so that the adjacency lists of the two trees coincide. As a result, note that T_1 and T_2 must have the same splits; conversely, if the two trees do not have the same splits, then they are considered distinct.

Let $b(n)$ denote the total number of distinct unrooted binary trees having n labeled leaves.

Given: A positive integer n ($n \leq 1000$).

Return: The value of $b(n)$ modulo 1,000,000.

Sample Dataset

```
5
```

Sample Output

```
15
```

Problem 74

Global Alignment with Scoring Matrix

Generalizing the Alignment Score

The edit alignment score in "Edit Distance Alignment" counted the total number of edit operations implied by an alignment; we could equivalently think of this scoring function as assigning a cost of 1 to each such operation. Another common scoring function awards matched symbols with 1 and penalizes substituted/inserted/deleted symbols equally by assigning each one a score of 0, so that the maximum score of an alignment becomes the length of a longest common subsequence of s and t (see "Finding a Shared Spliced Motif"). In general, the **alignment score** is simply a scoring function that assigns costs to edit operations encoded by the alignment.

One natural way of adding complexity to alignment scoring functions is by changing the alignment score based on which symbols are substituted; many methods have been proposed for doing this. Another way to do so is to vary the penalty assigned to the insertion or deletion of symbols.

In general, alignment scores can be either maximized or minimized depending on how scores are established. The general problem of optimizing a particular alignment score is called **global alignment**.

Problem

To penalize symbol substitutions differently depending on which two symbols are involved in the substitution, we obtain a **scoring matrix** S in which $S_{i,j}$ represents the (negative) score assigned to a substitution of the ith symbol of our alphabet $\mathscr{A}$ with the jth symbol of $\mathscr{A}$.

A **gap penalty** is the component deducted from alignment score due to the presence of a gap. A gap penalty may be a function of the length of the gap; for example, a **linear gap penalty** is a constant g such that each inserted or deleted symbol is charged g; as a result, the cost of a gap of length L is equal to gL.

Given: Two protein strings s and t in FASTA format (each of length at most 1000 aa).

Return: The maximum alignment score between s and t. Use:

- The BLOSUM62 scoring matrix.
- Linear gap penalty equal to 5 (i.e., a cost of -5 is assessed for each gap symbol).

Sample Dataset

```
>Rosalind_67
PLEASANTLY
>Rosalind_17
MEANLY
```

Sample Output

```
8
```

Problem 75

Genome Assembly with Perfect Coverage

Cyclic Chromosomes

Recall that although chromosomes taken from eukaryotes have a linear structure, many bacterial chromosomes are actually circular. We represented a linear chromosome with a DNA string, so we only need to modify the definition of string to model circular chromosomes.

> **Perfect coverage** is the phenomenon in fragment assembly of having a read (or k-mer) begin at every possible location in the genome. Unfortunately, perfect coverage is still difficult to achieve, but fragment assembly technology continues to improve by leaps and bounds, and perfect coverage is perhaps not the fantasy it once was.

Problem

A **circular string** is a string that does not have an initial or terminal element; instead, the string is viewed as a necklace of symbols. We can represent a circular string as a string enclosed in parentheses. For example, consider the circular DNA string (ACGTAC), and note that because the string "wraps around" at the end, this circular string can equally be represented by (CGTACA), (GTACAC), (TACACG), (ACACGT), and (CACGTA). The definitions of substrings and superstrings are easy to generalize to the case of circular strings (keeping in mind that substrings are allowed to wrap around).

Given: A collection of (error-free) DNA k-mers ($k \leq 50$) taken from the same strand of a circular chromosome. In this dataset, all k-mers from this strand of the chromosome are present, and their de Bruijn graph consists of exactly one **simple cycle**.

Return: A cyclic superstring of minimal length containing the reads (thus corresponding to a candidate cyclic chromosome).

Sample Dataset

```
ATTAC
TACAG
```

```
GATTA
ACAGA
CAGAT
TTACA
AGATT
```

Sample Output

```
GATTACA
```

> **Note**
>
> The assumption made above that all reads derive from the same strand is practically unrealistic; in reality, researchers will not know the strand of DNA from which a given read has been sequenced.

Problem 76

Matching a Spectrum to a Protein

> **Searching the Protein Database**
>
> Many proteins have already been identified for a wide variety of organisms. Accordingly, there are a large number of protein databases available, and so the first step after creating a mass spectrum for an unidentified protein is to search through these databases for a known protein with a highly similar spectrum. In this manner, many similar proteins found in different species have been identified, which aids researchers in determining protein function.
>
> In "Comparing Spectra with the Spectral Convolution", we introduced the spectral convolution and used it to measure the similarity of simplified spectra. In this problem, we would like to extend this idea to find the most similar protein in a database to a spectrum taken from an unknown protein. Our plan is to use the spectral convolution to find the largest possible number of masses that each database protein shares with our candidate protein after shifting, and then select the database protein having the largest such number of shared masses.

Problem

The **complete spectrum** of a weighted string s is the multiset $S[s]$ containing the weights of every prefix and suffix of s.

Given: A positive integer n followed by a collection of n protein strings s_1, s_2, ..., s_n and a multiset R of positive numbers (corresponding to the complete spectrum of some unknown protein string).

Return: The maximum multiplicity of $R \ominus S[s_k]$ taken over all strings s_k, followed by the string s_k for which this maximum multiplicity occurs (you may output any such value if multiple solutions

exist).

Sample Dataset

```
4
GSDMQS
VWICN
IASWMQS
PVSMGAD
445.17838
115.02694
186.07931
314.13789
317.1198
215.09061
```

Sample Output

```
3
IASWMQS
```

Problem 77

Quartets

Incomplete Characters

The modern revolution in genome sequencing has produced a huge amount of genetic data for a wide variety of species. One ultimate goal of possessing all this information is to be able to construct complete phylogenies via direct genome analysis.

> For example, say that we have a gene shared by a number of taxa. We could create a character based on whether species are known to possess the gene or not, and then use a huge character table to construct our desired phylogeny. However, the present bottleneck with such a method is that it assumes that we already possess complete genome information for all possible species. The race is on to sequence as many species genomes as possible; for instance, the **Genome 10K Project** aims to sequence 10,000 species genomes over the next decade. Yet for the time being, possessing a complete genomic picture of all Earth's species remains a dream.
>
> As a result of these practical limitations, we need to be able to work with **partial characters**, which divide taxa into three separate groups: those possessing the character, those not possessing the character, and those for which we do not yet have conclusive information.

Problem

A **partial split** of a set S of n taxa models a partial character and is denoted by $A \mid B$, where A and B are still the two disjoint subsets of taxa divided by the character. Unlike in the case of splits, we do not necessarily require that $A \cup B = S$; $(A \cup B)^c$ corresponds to those taxa for which we lack conclusive evidence regarding the character.

We can assemble a collection of partial characters into a generalized **partial character table** C in which the symbol x is placed in $C_{i,j}$ if we do not have conclusive evidence regarding the jth taxon with respect to the ith partial character.

A **quartet** is a partial split $A \mid B$ in which both A and B contain precisely two elements. For the sake of simplicity, we often will consider quartets instead of partial characters. We say that a quartet $A \mid B$ is inferred from a partial split $C \mid D$ if $A \subseteq C$ and $B \subseteq D$ (or equivalently $A \subseteq D$ and $B \subseteq C$). For example, $\{1,3\} \mid \{2,4\}$ and $\{3,5\} \mid \{2,4\}$ can be inferred from $\{1,3,5\} \mid \{2,4\}$.

Given: A partial character table C.

Return: The collection of all quartets that can be inferred from the splits corresponding to the underlying characters of C.

Sample Dataset

```
cat dog elephant ostrich mouse rabbit robot
01xxx00
x11xx00
111x00x
```

Sample Output

```
{elephant, dog} {rabbit, robot}
{cat, dog} {mouse, rabbit}
{mouse, rabbit} {cat, elephant}
{dog, elephant} {mouse, rabbit}
```

Problem 78

Using the Spectrum Graph to Infer Peptides

Getting Real with Spectra

In "Inferring Peptide from Full Spectrum", we considered an idealized version of the simplified spectrum in which every cut through a given peptide was produced, so that the spectrum possessed all possible b-ions and y-ions cutting the peptide. In reality, not every cut will be produced in a spectrum, which may also contain errors. As a result, it is difficult or impossible to recover an entire peptide from a single spectrum.

In the more practical case of a mass spectrum, where intensity is plotted against ions' mass-charge ratios, inferring the protein is also greatly complicated by the presence of erratic peaks in the spectrum.

Problem

For a weighted alphabet $\mathscr{A}$ and a collection L of positive real numbers, the **spectrum graph** of L is a digraph constructed in the following way. First, create a node for every real number in L. Then, connect a pair of nodes with a directed edge (u, v) if $v > u$ and $v - u$ is equal to the weight of a single symbol in $\mathscr{A}$. We may then label the edge with this symbol.

In this problem, we say that a weighted string $s = s_1 s_2 \cdots s_n$ matches L if there is some increasing sequence of positive real numbers $(w_1, w_2, \ldots, w_{n+1})$ in L such that $w(s_1) = w_2 - w_1$, $w(s_2) = w_3 - w_2$, ..., and $w(s_n) = w_{n+1} - w_n$.

Given: A list L (of length at most 100) containing positive real numbers.

Return: The longest protein string that matches the spectrum graph of L (if multiple solutions exist, you may output any one of them). Consult the monoisotopic mass table.

Sample Dataset

```
3524.8542
3623.5245
3710.9335
3841.974
3929.00603
3970.0326
4026.05879
4057.0646
4083.08025
```

Sample Output

```
WMSPG
```

> #### Hint
>
> How can our question be rephrased in terms of the spectrum graph?

Problem 79

Encoding Suffix Trees

Creating a Suffix Tree

In "Finding the Longest Multiple Repeat", we introduced the suffix tree. This data structure has a wide array of applications, one of which was to help us identify long repeats in a genome. In that problem, we provided the tree as part of the

dataset, but a vital computational exercise is to create the suffix tree solely from a string.

Problem

Given a string s having length n, recall that its suffix tree $T(s)$ is defined by the following properties:

- $T(s)$ is a rooted tree having exactly n leaves.
- Every edge of $T(s)$ is labeled with a substring of s^*, where s^* is the string formed by adding a placeholder symbol `$` to the end of s.
- Every internal node of $T(s)$ other than the root has at least two children; i.e., it has degree at least 3.
- The substring labels for the edges leading down from a node to its children must begin with different symbols.
- By concatenating the substrings along edges, each path from the root to a leaf corresponds to a unique suffix of s^*.

Figure 1. The suffix tree for s = GTCCGAAGCTCCGG. Note that the dollar sign has been appended to a substring of the tree to mark the end of s. Every path from the root to a leaf corresponds to a unique suffix of GTCCGAAGCTCCGG, and each leaf is labeled with the location in s of the suffix ending at that leaf.

Figure 1 contains an example of a suffix tree.

Given: A DNA string s of length at most 1kbp.

Return: The substrings of s^* encoding the edges of the suffix tree for s. You may list these substrings in any order.

Sample Dataset

```
ATAAATG$
```

Sample Output

```
AAATG$
G$
T
ATG$
TG$
A
A
AAATG$
G$
T
G$
$
```

Problem 80

Character-Based Phylogeny

Introduction to Character-Based Phylogeny

In "Creating a Character Table", we discussed the construction of a character table from a collection of characters represented by subsets of our taxa. However, the ultimate goal is to be able to construct a phylogeny from this character table.

> The issues at hand are that we want to ensure that we have enough characters to actually construct a phylogeny, and that our characters do not conflict with each other.

Problem

Because a tree having n nodes has $n - 1$ edges (see "Completing a Tree"), removing a single edge from a tree will produce two smaller, disjoint trees. Recall from "Creating a Character Table" that for this reason, each edge of an unrooted binary tree corresponds to a split $S \mid S^c$, where S is a subset of the taxa.

A **consistent character table** is one whose characters' splits do not conflict with the edge splits of some unrooted binary tree T on the n taxa. More precisely, $S_1 \mid S_1^c$ conflicts with $S_2 \mid S_2^c$ if all four intersections $S_1 \cap S_2$, $S_1 \cap S_2^c$, $S_1^c \cap S_2$, and $S_1^c \cap S_2^c$ are nonempty. As a simple example, consider the conflicting splits $\{a, b\} \mid \{c, d\}$ and $\{a, c\} \mid \{b, d\}$.

More generally, given a consistent character table C, an unrooted binary tree T "models" C if the edge splits of T agree with the splits induced from the characters of C.

Given: A list of n species ($n \leq 80$) and an n-column character table C in which the jth column denotes the jth species.

Return: An unrooted binary tree in Newick format that models C.

Sample Dataset

```
cat dog elephant mouse rabbit rat
011101
001101
001100
```

Sample Output

```
(dog,(cat,rabbit),(rat,(elephant,mouse)));
```

Problem 81

Counting Quartets

> ### Introduction to Quartet-Based Phylogeny

In "Quartets", we introduced partial splits modeling partial characters on a collection of taxa. Our aim is to use the quartets inferred from partial splits to construct a phylogeny on the taxa. This procedure is called **quartet-based phylogeny**.

We could construct a phylogeny directly from a collection of partial splits, but it is not immediately clear how many different splits we would need. Hence, our first question is to ask how many quartets are required to be able to infer a tree; in this problem we will ask the reverse question of how many quartets can be inferred from a known tree.

Problem

A quartet $AB \mid CD$ is **consistent** with a binary tree T if the quartet can be inferred from one of the splits of T (see "Quartets" for a description of inferring quartets from splits).

Let $q(T)$ denote the total number of quartets that are consistent with T.

Given: A positive integer n ($4 \leq n \leq 5000$), followed by an unrooted binary tree T in Newick format on n taxa.

Return: The value of $q(T)$ modulo 1,000,000.

Sample Dataset

```
6
(lobster,(cat,dog),(caterpillar,(elephant,mouse)));
```

Sample Output

```
15
```

Problem 82

Enumerating Unrooted Binary Trees

Seeing the Forest

In "Counting Unrooted Binary Trees", we found a way to count the number of unrooted binary trees representing phylogenies on n taxa. Our observation was that two such trees are considered distinct when they do not share the same collection of splits.

Counting all these trees is one task, but actually understanding how to write them out in a list (i.e., enumerating them) is another, which will be the focus of this problem.

Problem

Recall the definition of Newick format from "Distances in Trees" as a way of encoding trees. See **Figure 1** for an example of Newick format applied to an unrooted binary tree whose five leaves are labeled (note that the same tree can have multiple Newick representations).

Given: A collection of species names representing n taxa.

Return: A list containing all unrooted binary trees whose leaves are these n taxa. Trees should be given in Newick format, with one tree on each line; the order of the trees is unimportant.

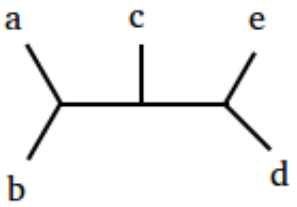


Figure 1. This unrooted binary tree may be represented in Newick format by (((a,b),c),(d,e)); another way of encoding it is ((a,b),(c,(d,e))).

Sample Dataset

```
dog cat mouse elephant
```

Sample Output

```
(((mouse,cat),elephant))dog;
(((elephant,mouse),cat))dog;
(((elephant,cat),mouse))dog;
```

Problem 83

Genome Assembly Using Reads

Putting the Puzzle Together

In practical genome sequencing, even if we assume that reads have been sequenced without errors, we have no idea of knowing immediately the particular strand of DNA a read has come from.

> Also, our reads may not have the same length. In 1995, Idury and Waterman proposed a way to boost read coverage and achieve uniform read length by breaking long reads into overlapping k-mers for some fixed value of k. For example, a 100 bp read could be split into 51 overlapping 50-mers.

Problem

A **directed cycle** is simply a cycle in a directed graph in which the head of one edge is equal to the tail of the next (so that every edge in the cycle is traversed in the same direction).

For a set of DNA strings S and a positive integer k, let S_k denote the collection of all possible k-mers of the strings in S.

Given: A collection S of (error-free) reads of equal length (not exceeding 50 bp). In this dataset, for

some positive integer k, the de Bruijn graph B_k on $S_{k+1} \cup S_{k+1}^{\text{rc}}$ consists of exactly two directed cycles.

Return: A cyclic superstring of minimal length containing every read or its reverse complement.

Sample Dataset

```
AATCT
TGTAA
GATTA
ACAGA
```

Sample Output

```
GATTACA
```

> **Note**
>
> The reads "AATCT" and "TGTAA" are not present in the answer, but their reverse complements "AGATT" and "TTACA" are present in the circular string (GATTACA).

Problem 84

Global Alignment with Constant Gap Penalty

Penalizing Large Insertions and Deletions

In dealing with global alignment in "Global Alignment with Scoring Matrix", we encountered a linear gap penalty, in which the insertion or deletion of a gap is penalized by some constant times the length of the gap. However, this model is not necessarily the most practical model, as one large rearrangement could have inserted or deleted a long gap in a single step to transform one genetic string into another.

Problem

In a **constant gap penalty**, every gap receives some predetermined constant penalty, regardless of its length. Thus, the insertion or deletion of 1000 contiguous symbols is penalized equally to that of a single symbol.

Given: Two protein strings s and t in FASTA format (each of length at most 1000 aa).

Return: The maximum alignment score between s and t. Use:

- The BLOSUM62 scoring matrix.
- Constant gap penalty equal to 5.

Sample Dataset

```
>Rosalind_79
PLEASANTLY
>Rosalind_41
MEANLY
```

Sample Output

```
13
```

Problem 85

Linguistic Complexity of a Genome

Getting Repetitive

We have seen that every genome contains a large number of repeats and noted that the Alu repeat recurs around a million times on the human genome. Yet exactly how repetitive is the human genome?

> To frame such a vague question mathematically, we first need to make the observation that if the genome were formed by adding nucleobases randomly, with each base having a 1/4 probability of being added at each nucleotide position, then we should expect to see a huge number of different substrings in the genome. Yet (to take a simple case) the genome containing only adenosine and forming the DNA string "AAAAAA...AAA" has relatively very few distinct substrings.
>
> Now, real genomes are formed by a process that chooses nucleotides somewhere in between these two extreme cases, and so to quantify just how random this process is, we need to take the percentage of distinct substrings appearing in a genome with respect to the maximum possible number of distinct substrings that could appear in a genome of the same length.

Problem

Given a length n string s formed over an alphabet $\mathscr{A}$ of size a, let the "substring count" $\mathrm{sub}(s)$ denote the total number of distinct substrings of s. Furthermore, let the "maximum substring count" $m(a,n)$ denote the maximum number of distinct substrings that could appear in a string of length n formed over $\mathscr{A}$.

The **linguistic complexity** of s (written $\mathrm{lc}(s)$) is equal to $\frac{\mathrm{sub}(s)}{m(a,n)}$; in other words, $\mathrm{lc}(s)$ represents the percentage of observed substrings of s to the total number that are theoretically possible. Note that $0 < \mathrm{lc}(s) < 1$, with smaller values of $\mathrm{lc}(s)$ indicating that s is more repetitive.

As an example, consider the DNA string ($a = 4$) $s = \mathrm{ATTTGGATT}$. In the following table, we demonstrate that $\mathrm{lc}(s) = \frac{35}{40} = 0.875$ by considering the number of observed and possible length k

substrings of s, which are denoted by $\mathrm{sub}_k(s)$ and $m(a,k,n)$, respectively. (Observe that $m(a,n) = \sum_{k=1}^{n} m(a,k,n) = 40$ and $\mathrm{sub}(s) = \sum_{k=1}^{n} \mathrm{sub}_k(s) = 35$.)

k	$\mathrm{sub}_k(s)$	$m(a,k,n)$
1	3	4
2	5	8
3	6	7
4	6	6
5	5	5
6	4	4
7	3	3
8	2	2
9	1	1
Total	35	40

Given: A DNA string s of length at most 100 kbp.

Return: The linguistic complexity $\mathrm{lc}(s)$.

Sample Dataset

```
ATTTGGATT
```

Sample Output

```
0.875
```

> **Hint**
>
> Why does this problem follow "Encoding Suffix Trees"?

Problem 86

Local Alignment with Scoring Matrix

Aligning Similar Substrings

Whereas global alignment (see "Global Alignment with Scoring Matrix") can be helpful for comparing genetic strings of similar length that resemble each other, often we will be presented with strings that are mostly dissimilar except for some unknown region of the strings, which may represent a shared gene. To find such genes, we need to modify global alignment to instead search for shared motifs in the form of locally similar regions (recall "Finding a Shared Motif" and "Finding a Shared Spliced Motif").

> Using global alignment often fails to find shared motifs hidden in larger strings because (especially if the similar region is found on different ends of the string) aligning the strings causes gap penalties to rack up.
>
> If we are only interested in comparing the regions of similarity, then we would like to have some way of disregarding the parts of the strings that don't resemble each other. The way to do this is to produce alignment scores for all possible pairs of substrings.

Problem

A **local alignment** of two strings s and t is an alignment of substrings r and u of s and t, respectively. Let $\mathrm{opt}(r, u)$ denote the score of an optimal alignment of r and u with respect to some predetermined alignment score.

Given: Two protein strings s and t in FASTA format (each having length at most 1000 aa).

Return: A maximum alignment score along with substrings r and u of s and t, respectively, which produce this maximum alignment score (multiple solutions may exist, in which case you may output any one). Use:

- The PAM250 scoring matrix.
- Linear gap penalty equal to 5.

Sample Dataset

```
>Rosalind_80
MEANLYPRTEINSTRING
>Rosalind_21
PLEASANTLYEINSTEIN
```

Sample Output

```
23
LYPRTEINSTRIN
LYEINSTEIN
```

Problem 87

Inferring Genotype from a Pedigree

Lying in Wait

Single gene disorders can be encoded by either dominant or recessive alleles. In the latter case, the affected person usually has two healthy carrier parents, who were usually unaware that their child could inherit a deadly or debilitating genetic condition from them.

We know from Mendel's first law that any offspring of two heterozygous carriers has a 25% chance of inheriting a recessive disorder. Knowing your own genotype is therefore important when deciding to have children, and genetic screening will prove vital for preventive medicine in the coming years.

In this problem, we will consider an exercise in which we determine the probability of an organism exhibiting each possible genotype for a factor knowing only the genotypes of the organism's ancestors.

Problem

A rooted binary tree can be used to model the pedigree of an individual. In this case, rather than time progressing from the root to the leaves, the tree is viewed upside down with time progressing from an individual's ancestors (at the leaves) to the individual (at the root).

An example of a pedigree for a single factor in which only the genotypes of ancestors are given is shown in **Figure 1**.

Given: A rooted binary tree T in Newick format encoding an individual's pedigree for a Mendelian factor whose alleles are A (dominant) and a (recessive).

Return: Three numbers between 0 and 1, corresponding to the respective probabilities that the individual at the root of T will exhibit the "AA", "Aa" and "aa" genotypes.

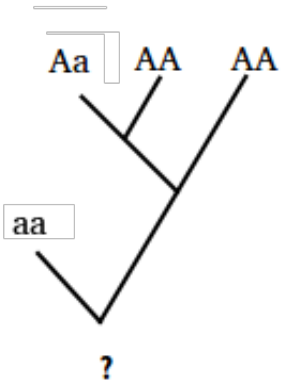


Figure 1. The rooted binary tree whose Newick format is (aa,((Aa,AA),AA)). Each leaf encodes the genotype of an ancestor for the given individual, which is represented by '?'.

Sample Dataset

```
((((Aa,aa),(Aa,Aa)),((aa,aa),(aa,AA))),Aa);
```

Sample Output

```
0.156 0.5 0.344
```

Problem 88

Maximizing the Gap Symbols of an Optimal Alignment

Adjusting Alignment Parameters

As we change the parameters contributing to alignment score, the nature of alignments achieving the maximum score may change. One feature of maximum-score alignments worthy of consideration is the number and size of their gaps. In this problem, we would like to determine the maximum number of possible gaps in any optimal alignment based solely on the parameter values chosen.

Problem

For the computation of an alignment score generalizing the edit alignment score, let m denote the score assigned to matched symbols, d denote the score assigned to mismatched non-gap symbols, and g denote the score assigned a symbol matched to a gap symbol '-' (i.e., g is a linear gap penalty).

Given: Two DNA strings s and t in FASTA format (each of length at most 5000 bp).

Return: The maximum number of gap symbols that can appear in any maximum score alignment of s and t with score parameters satisfying $m > 0$, $d < 0$, and $g < 0$.

Sample Dataset

```
>Rosalind_92
AACGTA
>Rosalind_47
ACACCTA
```

Sample Output

```
3
```

Problem 89

Identifying Maximal Repeats

Spies in the War Against Phages

Figure 1. A genomic region containing a CRISPR. Red substrings correspond to CRISPR repeats, and blue substrings correspond to unique spacers. Repeats are highly palindromic and fold into a hairpin loop when transcribed.

In "Locating Restriction Sites", we saw how one weapon used by bacteria in their age-old fight with phages is the use of restriction enzymes. Another defense mechanism found in the genomes of most bacteria and archaea centers on intervals of DNA called **CRISPRs** (Clustered Regularly Interspaced Short Palindromic Repeats), which allow the cell to distinguish its own DNA from that of phages or plasmids.

Specifically, a CRISPR is an interval of DNA consisting of identical repeats (approximately 23 to 47 bp long), alternating with unique intervals (approximately 21 to 72 bp long) called **spacers**; see **Figure 1**. Spacers correspond to fragments of foreign DNA that were integrated into the genome between repeats and serve as a memory bank for genetic material captured from invading phages. As a result, spacers can be used to recognize and silence invasive elements.

Specifically, CRISPRs are transcribed into RNA molecules, each consisting of a spacer flanked by

partial repeats. The small CRISPR RNAs, together with associated proteins translated from this RNA, target foreign DNA that matches the CRISPR spacer. In eukaryotes, a similar process is achieved by a process called **RNA interference** (RNAi).

To locate a CRISPR in a genome, we need to search for its repeats. We have already located long repeats in "Finding the Longest Multiple Repeat", but the case here is different because of the repeats appearing in CRISPRS are relatively short. Instead, we are looking for repeated intervals that cannot be lengthened in either direction (otherwise, we would intersect with a spacer).

Problem

A **maximal repeat** of a string s is a repeated substring t of s having two occurrences t_1 and t_2 such that t_1 and t_2 cannot be extended by one symbol in either direction in s and still agree.

For example, "**AG**" is a maximal repeat in "T**AG**TT**AG**CG**AG**A" because even though the first two occurrences of "AG" can be extended left into "TAG", the first and third occurrences differ on both sides of the repeat; thus, we conclude that "AG" is a maximal repeat. Note that "**TAG**" is also a maximal repeat of "**TAG**T**TAG**CGAGA", since its only two occurrences do not still match if we extend them in either direction.

Given: A DNA string s of length at most 1 kbp.

Return: A list containing all maximal repeats of s having length at least 20.

Sample Dataset

```
TAGAGATAGAATGGGTCCAGAGTTTTGTAATTTCCATGGGTCCAGAGTTTTGTAATTTATTATATAGAGAT
AGAATGGGTCCAGAGTTTTGTAATTTCCATGGGTCCAGAGTTTTGTAATTTAT
```

Sample Output

```
TAGAGATAGAATGGGTCCAGAGTTTTGTAATTTCCATGGGTCCAGAGTTTTGTAATTTAT
ATGGGTCCAGAGTTTTGTAATTT
```

Hint

How can we use the suffix tree of s to find maximal repeats?

Problem 90

Multiple Alignment

Comparing Multiple Strings Simultaneously

In "Consensus and Profile", we generalized the notion of Hamming distance to find an average case

for a collection of nucleic acids or peptides. However, this method only worked if the polymers had the same length. As we have already noted in "Edit Distance", homologous strands of DNA have varying lengths because of the effect of mutations inserting and deleting intervals of genetic material; as a result, we need to generalize the notion of alignment to cover multiple strings.

Problem

A **multiple alignment** of a collection of three or more strings is formed by adding gap symbols to the strings to produce a collection of augmented strings all having the same length.

A **multiple alignment score** is obtained by taking the sum of an alignment score over all possible pairs of augmented strings. The only difference in scoring the alignment of two strings is that two gap symbols may be aligned for a given pair (requiring us to specify a score for matched gap symbols).

Given: A collection of four DNA strings of length at most 10 bp in FASTA format.

Return: A multiple alignment of the strings having maximum score, where we score matched symbols 0 (including matched gap symbols) and all mismatched symbols -1 (thus incorporating a linear gap penalty of 1).

Sample Dataset

```
>Rosalind_7
ATATCCG
>Rosalind_35
TCCG
>Rosalind_23
ATGTACTG
>Rosalind_44
ATGTCTG
```

Sample Output

```
-18
ATAT-CCG
-T---CCG
ATGTACTG
ATGT-CTG
```

Problem 91

Creating a Restriction Map

> ### Genetic Fingerprinting

Recall that a restriction enzyme cuts the endpoints of a specific interval of DNA, which must form a reverse palindrome that typically has length 4 or 6. The interval of DNA cleaved by a given restriction enzyme is called its recognition sequence.

A single human chromosome is so long that a given recognition sequence will occur frequently throughout the chromosome (recall from "Expected Number of Restriction Sites" that a recognition sequence would be expected to occur several times even in a short chromosome). Nevertheless, the small-scale mutations that create diversity in the human genome (chiefly SNPs) will cause each human to have a different collection of recognition sequences for a given restriction enzyme.

Genetic fingerprinting is the term applied to the general process of forming a limited picture of a person's genetic makeup (which was traditionally cheaper than sequencing). The earliest application of genetic fingerprinting inexpensive enough to be widely used in common applications, like forensics and paternity tests, relied on a process called **restriction digest**. In this technique, a sample of DNA is replicated artificially, then treated with a given restriction enzyme; when the enzyme cuts the DNA at restriction sites, it forms a number of fragments. A second process called **gel electrophoresis** then separates these fragments along a membrane based on their size, with larger pieces tending toward one end and smaller pieces tending toward the other. When the membrane is stained or viewed with an X-ray machine, the fragments create a distinct banding pattern, which typically differs for any two individuals.

These intervals can be thought of simply as the collection of distances between restriction sites in the genome. Before the rapid advances of genome sequencing, biologists wanted to know if they could use only these distances to reconstruct the actual locations of restriction sites in the genome, forming a **restriction map**. Restriction maps were desired in the years before the advent of sequencing, when any information at all about genomic makeup was highly coveted. The application of forming a restriction map from cleaved restriction fragments motivates the following problem.

Problem

For a set X containing numbers, the **difference multiset** of X is the multiset ΔX defined as the collection of all *positive* differences between elements of X. As a quick example, if $X = \{2, 4, 7\}$, then we will have that $\Delta X = \{2, 3, 5\}$.

If X contains n elements, then ΔX will contain one element for each pair of elements from X, so that ΔX contains $\binom{n}{2}$ elements (see combination statistic). You may note the similarity between the difference multiset and the Minkowski difference $X \ominus X$, which contains the elements of ΔX and their negatives. For the above set X, $X \ominus X$ is $\{-5, -3, -2, 2, 3, 5\}$.

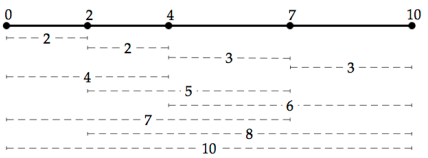


Figure 1. In the simplified figure above, we know that the dashed segments came from a chromosome; we desire a collection of numbers whose differences match the lengths of the dotted lines, which will correspond to the locations of restriction sites on the unknown chromosome. Taken from Jones & Pevzner, An Introduction to Bioinformatics Algorithms.

In practical terms, we can easily obtain a multiset L corresponding to the distances between restriction sites on a chromosome. If we can find a set X whose difference multiset ΔX is equal to L, then X will represent possible locations of these restriction sites. For an example, consult **Figure 1**.

Given: A multiset L containing $\binom{n}{2}$ positive integers for some positive integer n.

Return: A set X containing n nonnegative integers such that $\Delta X = L$.

Sample Dataset

```
2 2 3 3 4 5 6 7 8 10
```

Sample Output

```
0 2 4 7 10
```

Problem 92

Counting Rooted Binary Trees

From Unrooted to Rooted Trees

Recall that a **rooted binary tree** is a binary tree for which the root is the only node of degree 2. Such a tree differs from an unrooted binary tree only in the existence of the root.

> Different phylogenetic methods may be better suited to rooted or unrooted trees. If a method produces an unrooted tree, then the root (i.e., the common ancestor of all taxa) could theoretically be placed anywhere. Thus, there will be more rooted binary trees than unrooted binary trees on the same number of taxa. The question is: how many more rooted trees are there?

Problem

As in the case of unrooted trees, say that we have a fixed collection of n taxa labeling the leaves of a rooted binary tree T. You may like to verify that (by extension of "Counting Phylogenetic Ancestors") such a tree will contain $n - 1$ internal nodes and $2n - 2$ total edges. Any edge will still encode a split of taxa; however, the two splits corresponding to the edges incident to the root of T will be equal. We still consider two trees to be equivalent if they have the same splits (which requires that they must also share the same duplicated split to be equal).

Let $B(n)$ represent the total number of distinct rooted binary trees on n labeled taxa.

Given: A positive integer n ($n \leq 1000$).

Return: The value of $B(n)$ modulo 1,000,000.

Sample Dataset

```
4
```

Sample Output

```
15
```

Problem 93

Sex-Linked Inheritance

Chromosomes Determine Sex

In "Independent Segregation of Chromosomes", we discussed how chromosomes in diploid organisms form pairs of homologs. It turns out that this is not the case for one pair of chromosomes in animals. In 1905, Nettie Stevens and Edmund Wilson independently discovered that male animals possess one chromosome that is shorter than its partner, whereas female animals instead have two long chromosomes. The shorter chromosome earned the title of **Y chromosome** for its stunted shape, whereas the longer chromosome became known as the **X chromosome**. These two chromosomes are aptly termed **sex chromosomes**, or allosomes, and we write the female sex chromosome genotype as XX and the male genotype as XY. The remaining homologous chromosome pairs are called **autosomes**.

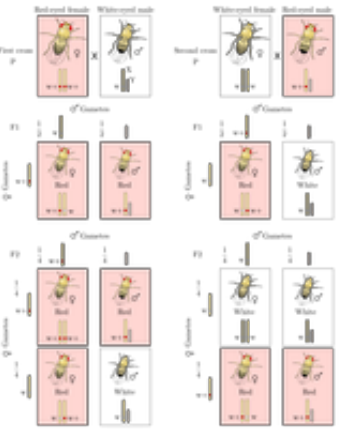


Figure 1. Morgan's two experiments on fruit fly eye color. In the first experiment, a white-eyed male is crossed with a purebred red-eyed female; in the second experiment, a red-eyed male is crossed with a white-eyed female. The results of Morgan's expermients demonstrate that eye color must be encoded by a recessive allele on the X chromosome.

Sex chromosomes are still passed on to gametes based on the outcome of a coin flip, but egg cells (deriving from females) must always possess an X chromosome, so that the sex of an individual is determined by whether it receives an X or a Y chromosome from its father's sperm cell.

Fast-forward five years to 1910 and the lab of Thomas Hunt Morgan, who is often considered the first modern geneticist because of his tireless work to place Mendel's work on sound footing. One of Morgan's many experiments with fruit flies (genus *Drosophila*) began as he noticed a number of white-eyed males. When these white-eyed flies were crossed with purebred red-eyed females, their progeny were all red-eyed, and yet crossing the second generation's red-eyed individuals with each other produced some white-eyed males but exclusively red-eyed females. Strange results indeed.

Morgan's experiments are summarized in **Figure 1**, after which he concluded that the trait for eye color in fruit flies must be **sex linked**, or encoded on a sex chromosome. More specifically, the factor for white eye color is encoded by a recessive allele on the X chromosome. Because a male only has one copy of the X chromosome, having only one recessive allele will cause the individual to exhibit white eyes, whereas a female fly requires both copies of the recessive allele to possess white eyes.

X-linked recessive traits are manifested in males much more often than in females, because a male only needs to receive a recessive allele from his mother to exhibit the trait: in the case of genetic conditions, half of all male children born to carrier mothers will inherit the condition.

Problem

The **conditional probability** of an event A given another event B, written $\Pr(A \mid B)$, is equal to $\Pr(A \text{ and } B)$ divided by $\Pr(B)$.

Note that if A and B are independent, then $\Pr(A \text{ and } B)$ must be equal to $\Pr(A) \times \Pr(B)$, which results in $\Pr(A \mid B) = \Pr(A)$. This equation offers an intuitive view of independence: the probability of A, given the occurrence of event B, is simply the probability of A (which does not depend on B).

In the context of sex-linked traits, genetic equilibrium requires that the alleles for a gene k are uniformly distributed over the males and females of a population. In other words, the distribution of alleles is *independent* of sex.

Given: An array A of length n for which $A[k]$ represents the proportion of males in a population exhibiting the k-th of n total recessive X-linked genes. Assume that the population is in genetic equilibrium for all n genes.

Return: An array B of length n in which $B[k]$ equals the probability that a randomly selected female will be a carrier for the k-th gene.

Sample Dataset

```
0.1 0.5 0.8
```

Sample Output

```
0.18 0.5 0.32
```

Problem 94

Phylogeny Comparison with Split Distance

Quantifying Binary Tree Comparison

We may often obtain two different phylogenies on the same collection of taxa from different sets of data. As a result, we would like to have a way of quantifying how much the two phylogenies differ. In the simplest case, we would like to compare the characters of two phylogenies.

> Recall from "Counting Unrooted Binary Trees" that two unrooted binary trees are equivalent when they have the same set of splits; recall also (by extension of "Counting Phylogenetic Ancestors") that any unrooted binary tree on n taxa must have $n - 3$ nontrivial splits.

Problem

Define the **split distance** between two unrooted binary trees as the number of nontrivial splits contained in one tree but not the other.

Formally, if $s(T_1, T_2)$ denotes the number of nontrivial splits shared by unrooted binary trees T_1 and T_2, Then their split distance is $d_{\text{split}}(T_1, T_2) = 2(n-3) - 2s(T_1, T_2)$.

Given: A collection of at most 3,000 species taxa and two unrooted binary trees T_1 and T_2 on these taxa in Newick format.

Return: The split distance $d_{\text{split}}(T_1, T_2)$.

Sample Dataset

```
dog rat elephant mouse cat rabbit
(rat,(dog,cat),(rabbit,(elephant,mouse)));
(rat,(cat,dog),(elephant,(mouse,rabbit)));
```

Sample Output

```
2
```

Problem 95

The Wright-Fisher Model of Genetic Drift

Hardy-Weinberg Revisited

The principle of genetic equilibrium is an idealistic model for population genetics that simply cannot hold for all genes in practice. For one, evolution has proven too powerful for equilibrium to possibly hold. At the same time, evolution works on the scale of eons, and at any given moment in time, most populations are essentially stable.

Yet we could overlook the inevitable effects of simple random chance in disrupting the allelic frequency for a given gene, a phenomenon called **genetic drift**.

In this problem, we would like to obtain a simple mathematical model of genetic drift, and so we will need to make a number of simplifying assumptions. First, assume that individuals from different generations do not mate with each other, so that generations exist as discrete, non-overlapping quantities. Second, rather than selecting pairs of mating organisms, we simply randomly select the *alleles* for the individuals of the next generation based on the allelic frequency in the present generation. Third, the population size is stable, so that we do not need to take into account the population growing or shrinking between generations. Taken together, these three assumptions make up the **Wright-Fisher model** of genetic drift.

Problem

Consider flipping a weighted coin that gives "heads" with some fixed probability p (i.e., p is not necessarily equal to 1/2).

We generalize the notion of binomial random variable from "Independent Segregation of Chromosomes" to quantify the sum of the weighted coin flips. Such a random variable X takes a value of k if a sequence of n independent "weighted coin flips" yields k "heads" and $n - k$ "tails." We write that

$X \in \text{Bin}(n, p)$.

To quantify the Wright-Fisher Model of genetic drift, consider a population of N diploid individuals, whose $2N$ chromosomes possess m copies of the dominant allele. As in "Counting Disease Carriers", set $p = \frac{m}{2N}$. Next, recall that the next generation must contain exactly N individuals. These individuals' $2N$ alleles are selected independently: a dominant allele is chosen with probability p, and a recessive allele is chosen with probability $1 - p$.

Given: Positive integers N ($N \leq 7$), m ($m \leq 2N$), g ($g \leq 6$) and k ($k \leq 2N$).

Return: The probability that in a population of N diploid individuals initially possessing m copies of a dominant allele, we will observe after g generations at least k copies of a recessive allele. Assume the Wright-Fisher model.

Sample Dataset

```
4 6 2 1
```

Sample Output

```
0.772
```

Problem 96

Alignment-Based Phylogeny

From Characters Toward Alignments

In "Creating a Character Table from Genetic Strings", we used strings to create a collection of characters from which we could create a phylogeny. However, the strings all had to share the same length, which was a problem. In practice, we would like to create a phylogeny from genetic strings having differing lengths; specifically, our aim is to construct a phylogeny from a multiple alignment.

Unfortunately, constructing a phylogeny from the ground up based only on an alignment can be difficult. In order to produce an efficient solution, we will need to assume that the structure of the phylogeny has already been provided (perhaps from character-based methods), and our aim instead is to reconstruct the genetic strings corresponding to the internal nodes (i.e., ancestors) in the tree.

The ancestor strings should have the property that the total number of point mutations separating adjacent nodes in the tree is minimized (in keeping with parsimony).

Problem

Say that we have n taxa represented by strings $s_1, s_2, \ldots, s_n$ with a multiple alignment inducing corresponding augmented strings $\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_n$.

Recall that the number of single-symbol substitutions required to transform one string into another is

the Hamming distance between the strings (see "Counting Point Mutations"). Say that we have a rooted binary tree T containing $\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_n$ at its leaves and additional strings $\bar{s}_{n+1}, \bar{s}_{n+2}, \ldots, \bar{s}_{2n-1}$ at its internal nodes, including the root (the number of internal nodes is $n-1$ by extension of "Counting Phylogenetic Ancestors"). Define $d_{\mathrm{H}}(T)$ as the sum of $d_{\mathrm{H}}(\bar{s}_i, \bar{s}_j)$ over all edges $\{\bar{s}_i, \bar{s}_j\}$ in T:

$$d_{\mathrm{H}}(T) = \sum_{\{\bar{s}_i, \bar{s}_j\} \in E(T)} d_{\mathrm{H}}(\bar{s}_i, \bar{s}_j)$$

Thus, our aim is to minimize $d_{\mathrm{H}}(T)$.

Given: A rooted binary tree T on n ($n \leq 500$) species, given in Newick format, followed by a multiple alignment of m ($m \leq n$) augmented DNA strings having the same length (at most 300 bp) corresponding to the species and given in FASTA format.

Return: The minimum possible value of $d_{\mathrm{H}}(T)$, followed by a collection of DNA strings to be assigned to the internal nodes of T that will minimize $d_{\mathrm{H}}(T)$ (multiple solutions will exist, but you need only output one).

Sample Dataset

```
(((ostrich,cat)rat,(duck,fly)mouse)dog,(elephant,pikachu)hamster)robot;
>ostrich
AC
>cat
CA
>duck
T-
>fly
GC
>elephant
-T
>pikachu
AA
```

Sample Output

```
8
>rat
AC
>mouse
TC
>dog
AC
>hamster
AT
>robot
AC
```

Note

Given internal strings minimizing $d_{\mathrm{H}}(T)$, the alignment between any two adjacent strings is not

necessarily an optimal global paired alignment. In other words, it may not be the case that $d_{\mathrm{H}}(\bar{s}_i, \bar{s}_j)$ is equal to the edit distance $d_{\mathrm{E}}(s_i, s_j)$.

Problem 97

Assessing Assembly Quality with N50 and N75

How Well Assembled Are Our Contigs?

As we have stated, the goal of genome sequencing is to create contigs that are as long as possible. Thus, after fragment assembly, it is important to possess statistics quantifying how well-assembled our contigs are.

> First and foremost, we demand a measure of what percentage of the assembled genome is made up of long contigs. Our first question is then: if we select contigs from our collection, how long do the contigs need to be to cover 50% of the genome?

Problem

Given a collection of DNA strings representing contigs, we use the **N statistic** NXX (where XX ranges from 01 to 99) to represent the maximum positive integer L such that the total number of nucleotides of all contigs having length $\geq L$ is at least XX% of the sum of contig lengths. The most commonly used such statistic is **N50**, although N75 is also worth mentioning.

Given: A collection of at most 1000 DNA strings (whose combined length does not exceed 50 kbp).

Return: N50 and N75 for this collection of strings.

Sample Dataset

```
GATTACA
TACTACTAC
ATTGAT
GAAGA
```

Sample Output

```
7 6
```

> ### Extra Information
>
> For an explanation of the results obtained in the sample above, contigs of length at least 7 total 7 + 9 = 16 bp, which is more than 50% of the total 27). Contigs of length at least 8 total only 9 bp (less than 50%).

Contigs of length at least 6 total 6 + 7 + 9 = 22 bp, which is more than 75% of all base pairs. Contigs of length at least 7 total only 16 bp (less than 75%).

Problem 98

Fixing an Inconsistent Character Set

Pitfalls of Character-Based Phylogeny

In "Character-Based Phylogeny", we asked for the construction of an unrooted binary tree from a consistent character table. However, the assumption of consistency is often inaccurate, as many character collections derived from real data are inconsistent, owing to the fact that the reinforcement of mutations by evolution can cause species to lose features over time, evolve to produce the same character on different evolutionary paths, or **revert** to a past character. This issue arises even when using genetic characters taken from SNPs, as point mutations can be undone.

As an example of why using characters can lead us astray, let's return to our first example of a character introduced in "Character-Based Phylogeny". There, we learned that dinosaurs may be divided into the two Orders Saurischia and Ornithischia depending on hip-bone shape: the former have "lizard hips," whereas the latter have "bird hips." Adding to this information the fact that birds are widely believed to descend from dinosaurs, we would guess that birds derive from ornithischians. Yet this is not the case: birds derive from theropods, a suborder of the saurischians! The shared hip bone shape with ornithischians is either simply coincidence or caused by the "convergence" of bird hip shape with that of ornithischians along their different evolutionary paths.

Another example of a character that would be ill-suited for phylogenetic analysis is the presence or absence of wings in insects. Many wingless modern species wingless modern species have evolved from wildly differing ancestors that lost their wings independently of each other.

If we divided a collection of taxa based on either of these characters, many different taxa would be lumped together when they do not in fact share a recent common ancestor, which could have disastrous consequences when trying to assign characters to the splits of a phylogeny.

The moral is that we must select our characters carefully, although the Catch-22 is that we don't know in advance which characters are the most appropriate to use until we actually start constructing phylogenies. At the same time, if we err on the side of caution, then using too few characters might not provide us with enough splits to generate an unrooted binary tree, thus inducing an enormous number of possible phylogenies (recall "Counting Unrooted Binary Trees" and how quickly the total number of trees grows with the number of taxa).

Problem

A **submatrix** of a matrix M is a matrix formed by selecting rows and columns from M and taking only those entries found at the intersections of the selected rows and columns. We may also think of a submatrix as formed by deleting the remaining rows and columns from M.

Given: An inconsistent character table C on at most 100 taxa.

Return: A submatrix of C' representing a consistent character table on the same taxa and formed

by deleting a single row of C. (If multiple solutions exist, you may return any one.)

Sample Dataset

```
100001
000110
111000
100111
```

Sample Output

```
000110
100001
100111
```

Problem 99

Wright-Fisher's Expected Behavior

Reaching Population Equilibrium

In "The Wright-Fisher Model of Genetic Drift", we introduced the Wright-Fisher model of genetic drift. Although the effects of genetic drift are inevitable, we should be able to quantify how many alleles for a given trait will remain in the next generation.

> Intuitively, because Wright-Fisher demands that we randomly and independently select alleles for the next generation based off the allele frequency of the present generation, we would hope that on average this frequency would illustrate a stabilizing effect: that is, the expected frequency in the next generation should equal the allele frequency in the current generation. In this problem, we will see if the mathematics matches our intuition.

Problem

In "The Wright-Fisher Model of Genetic Drift", we generalized the concept of a binomial random variable $\text{Bin}(n, p)$ as a "weighted coin flip." It is only natural to calculate the expected value of such a random variable.

For example, in the case of unweighted coin flips (i.e., $p = 1/2$), our intuition would indicate that $\text{E}(\text{Bin}(n, 1/2)$ is $n/2$; what should be the expected value of a binomial random variable?

Given: A positive integer n ($n \leq 1000000$) followed by an array P of length m ($m \leq 20$) containing numbers between 0 and 1. Each element of P can be seen as representing a probability corresponding to an allele frequency.

Return: An array B of length m for which $B[k]$ is the expected value of $\text{Bin}(n, P[k])$; in terms of Wright-Fisher, it represents the expected allele frequency of the next generation.

Sample Dataset

```
17
0.1 0.2 0.3
```

Sample Output

```
1.7 3.4 5.1
```

Problem 100

The Founder Effect and Genetic Drift

Strength in Numbers

Charles Darwin is known first and foremost for his notion of **natural selection**, the elegant statistical fact that changes in populations are attributable to the observation that organisms better equipped to handle their environment are more likely to survive and reproduce, thus passing on their beneficial traits to the next generation. As a result of natural selection, populations can change greatly over a long time.

A lesser known aspect of Darwin's evolutionary theory dictates how new species are actually created. Darwin noted that the only way for a population to grow so distinct that it would actually split off and form a new species would be if the population were isolated for a very long period. This notion that isolation forms new species was validated by Darwin's observation that the tiny Galapagos islands in the South Pacific enjoy a diversity of species rivaling that of a much larger ecosystem.

Isolated populations also tend to be small, strengthening the effects of genetic drift. To take an extreme example, consider a population of only 2 organisms that are both heterozygous for a given factor. Note that there is a 1/8 chance that 2 offspring of these organisms will possess only recessive alleles or only dominant alleles for the factor, thus wiping out the other allele completely.

In general, the principle stating that mutations (both positive and negative) can randomly attain higher proportions in small, isolated communities than they would in large populations, is known as the **founder effect**. An infamous example of the founder effect on human populations occurs in Pennsylvania, where the Amish community is at risk for a much greater incidence of **Ellis-van Creveld syndrome**, a single gene disorder causing a slew of defects, including additional fingers and toes (polydactyly). The condition has been traced to a single couple in the original Amish settlers, and it is still preserved in elevated percentages because of the community's isolationism.

In this problem, we would like to apply the Wright-Fisher model of genetic drift to understand the power of the founder effect. Specifically, we will quantify the likelihood that an allele will be completely annihilated in a small population after a number of generations.

Problem

Given: Two positive integers N and m, followed by an array A containing k integers between 0 and $2N$. $A[j]$ represents the number of recessive alleles for the j-th factor in a population of N diploid individuals.

Return: An $m \times k$ matrix B for which $B_{i,j}$ represents the common logarithm of the probability that after i generations, no copies of the recessive allele for the j-th factor will remain in the population. Apply the Wright-Fisher model.

Sample Dataset

```
4 3
0 1 2
```

Sample Output

```
0.0 -0.463935575821 -0.999509892866
0.0 -0.301424998891 -0.641668367342
0.0 -0.229066698008 -0.485798552456
```

Problem 101

Global Alignment with Scoring Matrix and Affine Gap Penalty

Mind the Gap

In "Global Alignment with Scoring Matrix", we considered a linear gap penalty, in which each inserted/deleted symbol contributes the exact same amount to the calculation of alignment score. However, as we mentioned in "Global Alignment with Constant Gap Penalty", a single large insertion/deletion (due to a rearrangement is then punished very strictly, and so we proposed a constant gap penalty.

> Yet large insertions occur far more rarely than small insertions and deletions. As a result, a more practical method of penalizing gaps is to use a hybrid of these two types of penalties in which we charge one constant penalty for *beginning* a gap and another constant penalty for every *additional* symbol added or deleted.

Problem

An **affine gap penalty** is written as $a + b \cdot (L - 1)$, where L is the length of the gap, a is a positive constant called the **gap opening penalty**, and b is a positive constant called the **gap extension penalty**.

We can view the gap opening penalty as charging for the first gap symbol, and the gap extension penalty as charging for each subsequent symbol added to the gap.

For example, if $a = 11$ and $b = 1$, then a gap of length 1 would be penalized by 11 (for an average cost of 11 per gap symbol), whereas a gap of length 100 would have a score of 110 (for an average cost of 1.10 per gap symbol).

Consider the strings "PRTEINS" and "PRTWPSEIN". If we use the BLOSUM62 scoring matrix and an affine gap penalty with $a = 11$ and $b = 1$, then we obtain the following optimal alignment.

```
PRT---EINS
|||   |||
PRTWPSEIN-
```

Matched symbols contribute a total of 32 to the calculation of the alignment's score, and the gaps cost 13 and 11 respectively, yielding a total score of 8.

Given: Two protein strings s and t in FASTA format (each of length at most 100 aa).

Return: The maximum alignment score between s and t, followed by two augmented strings s' and t' representing an optimal alignment of s and t. Use:

- The BLOSUM62 scoring matrix.
- Gap opening penalty equal to 11.
- Gap extension penalty equal to 1.

Sample Dataset

```
>Rosalind_49
PRTEINS
>Rosalind_47
PRTWPSEIN
```

Sample Output

```
8
PRT---EINS
PRTWPSEIN-
```

Problem 102

Genome Assembly with Perfect Coverage and Repeats

Repeats: A Practical Assembly Difficulty

Genome assembly is straightforward if we know in advance that the de Bruijn graph has exactly one directed cycle (see "Genome Assembly with Perfect Coverage").

> In practice, a genome contains repeats longer than the length of the k-mers that we wish to use to assemble the genome. Such repeats increase the number of cycles present in the de Bruijn graph

for these k-mers, thus preventing us from assembling the genome uniquely.

For example, consider the circular string (ACCTCCGCC), along with a collection S of error-free reads of length 3, exhibiting perfect coverage and taken from the same strand of an interval of DNA. The corresponding de Bruijn graph B_2 (where edges correspond to 3-mers and nodes correspond to 2-mers) has at least two directed cycles: one giving the original circular string (ACCTCCGCC), and another corresponding to the misfit (ACCGCCTCC).

Also, note that these cycles are not simple cycles, as the node corresponding to "CC" is visited three times in each cycle.

To generalize the problem of genome assembly from a de Bruijn graph to the case of genomes containing repeats, we therefore must add a constraint: in a cycle corresponding to a valid assembly, every 3-mer must appear as many times in the cycle as it does in our collection of reads (which correspond to all 3-mers in the original string).

Problem

Recall that a directed cycle is a cycle in a directed graph in which the head of one edge is equal to the tail of the following edge.

In a de Bruijn graph of k-mers, a circular string s is constructed from a directed cycle $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_i \rightarrow s_1$ is given by $s_1 + s_2[k] + \ldots + s_{i-k}[k] + s_{i-k+1}[k]$. That is, because the final $k-1$ symbols of s_1 overlap with the first $k-1$ symbols of s_2, we simply tack on the k-th symbol of s_2 to s, then iterate the process.

For example, the circular string assembled from the cycle "AC" $\rightarrow$ "CT" $\rightarrow$ "TA" $\rightarrow$ "AC" is simply (ACT). Note that this string only has length three because the 2-mers "wrap around" in the string.

If every k-mer in a collection of reads occurs as an edge in a de Bruijn graph cycle the same number of times as it appears in the reads, then we say that the cycle is "complete."

Given: A list S_{k+1} of error-free DNA $(k+1)$-mers ($k \leq 5$) taken from the same strand of a circular chromosome (of length ≤ 50).

Return: All circular strings assembled by complete cycles in the de Bruijn graph B_k of S_{k+1}. The strings may be given in any order, but each one should begin with the first $(k+1)$-mer provided in the input.

Sample Dataset

```
CAG
AGT
GTT
TTT
TTG
TGG
GGC
GCG
CGT
GTT
TTC
TCA
CAA
AAT
ATT
```

```
TTC
TCA
```

Sample Output

```
CAGTTCAATTTGGCGTT
CAGTTCAATTGGCGTTT
CAGTTTCAATTGGCGTT
CAGTTTGGCGTTCAATT
CAGTTGGCGTTCAATTT
CAGTTGGCGTTTCAATT
```

Problem 103

Overlap Alignment

Overlapping Reads with Errors

As also mentioned in "Error Correction in Reads", the sequencing machines that identify reads can make errors. However, the problem that we considered in "Genome Assembly as Shortest Superstring" assumed that all reads are error-free.

> Thus, rather than trying to overlap reads exactly, we will instead do so approximately. The key to do this is to move toward methods that incorporate alignments. Yet neither global nor local alignment is appropriate for this task. Global alignment will attempt to align the entire reads, when we know that only the overlapping parts of the reads are relevant. For that matter, we may identify an optimal local alignment that does not correspond to an overlap.
>
> As a result, we need a specific type of local alignment that aligns only the overlapping parts of two strings.

Problem

An **overlap alignment** between two strings s and t is a local alignment of a suffix of s with a prefix of t. An optimal overlap alignment will therefore maximize an alignment score over all such substrings of s and t.

The term "overlap alignment" has also been used to describe what Rosalind defines as a semiglobal alignment. See "Semiglobal Alignment" for details.

Given: Two DNA strings s and t in FASTA format, each having length at most 10 kbp.

Return: The score of an optimal overlap alignment of s and t, followed by an alignment of a suffix s' of s and a prefix t' of t achieving this optimal score. Use an alignment score in which matching symbols count +1, substitutions count -2, and there is a linear gap penalty of 2. If multiple optimal alignments exist, then you may return any one.

Sample Dataset

```
>Rosalind_54
CTAAGGGATTCCGGTAATTAGACAG
>Rosalind_45
ATAGACCATATGTCAGTGACTGTGTAA
```

Sample Output

```
1
ATTAGAC-AG
AT-AGACCAT
```

> ### Citation
>
> This problem follows Jones & Pevzner, *An Introduction to Bioinformatics Algorithms*, problem 6.22.

Problem 104

Quartet Distance

Another Tree Distance

In "Phylogeny Comparison with Split Distance", we examined the split distance for comparison of different phylogenies on the same collection of taxa.

> Yet quartet-based phylogeny offers another way in which two phylogenies can be compared (see "Quartets" and "Counting Quartets"). Specifically, we wonder how many quartets can be inferred from one tree but not inferred from the other.

Problem

In "Counting Quartets", we found an expression for $q(T)$, the number of quartets that can be inferred from an unrooted binary tree containing n taxa.

If T_1 and T_2 are both unrooted binary trees on the same n taxa, then we now let $q(T_1, T_2)$ denote the number of inferred quartets that are common to both trees. The **quartet distance** between T_1 and T_2, $d_{\mathrm{q}}(T_1, T_2)$ is the number of quartets that are only inferred from one of the trees. More precisely, $d_{\mathrm{q}}(T_1, T_2) = q(T_1) + q(T_2) - 2q(T_1, T_2)$.

Given: A list containing n taxa ($n \leq 2000$) and two unrooted binary trees T_1 and T_2 on the given taxa. Both T_1 and T_2 are given in Newick format.

Return: The quartet distance $d_{\mathrm{q}}(T_1, T_2)$.

Sample Dataset

```
A B C D E
(A,C,((B,D),E));
(C,(B,D),(A,E));
```

Sample Output

```
4
```

Problem 105

Finding a Motif with Modifications

Finding Mutated Motifs

We have discussed at length the importance of motif finding in biology for genetic strings. However, searching for exact substring matches is of little use in applications because a motif can vary under the effect of mutation. Fortunately, we already possess functions like edit distance for quantifying the similarity of two strings.

> Furthermore, recall that each chromosome is made up of a large number of genes (on average, each human chromosome contains over 1,000 genes). Therefore, to determine whether a newly sequenced chromosome contains a given gene, neither local nor global alignment applies.
>
> One possible alignment variant for finding genes is semiglobal alignment, which we discuss in "Semiglobal Alignment"; yet semiglobal alignment only allows us to disregard gaps at the end of the alignment. To find a known gene in a new chromosome, we need to instead align the gene against *intervals* of the chromosome, a problem that calls for an entirely new algorithmic variation of alignment.

Problem

Given a string s and a motif t, an alignment of a substring of s against all of t is called a **fitting alignment**. Our aim is to find a substring s' of s that maximizes an alignment score with respect to t.

Note that more than one such substring of s may exist, depending on the particular strings and alignment score used. One candidate for scoring function is the one derived from edit distance; In this problem, we will consider a slightly different alignment score, in which all matched symbols count as +1 and all mismatched symbols (including insertions and deletions) receive a cost of -1. Let's call this scoring function

	global	local	fitting
v	GTAGGCTTAAGGTTA	TAG	TAGGCTTA
w	-TAG----A---T-A	TAG	TAGA--TA
score	−3	3	2

Figure 1. Global, local, and fitting alignments of strings v = GTAGGCTTAAGGTTA and w = TAGATA with respect to mismatch score. Note that in the fitting alignment, a substring of v must be aligned against all of w. Taken from Jones & Pevzner, An Introduction to Bioinformatics Algorithms

the **mismatch score**. See **Figure 1** for a comparison of global, local, and fitting alignments with respect to mismatch score.

Given: Two DNA strings s and t, where s has length at most 10 kbp and t represents a motif of length at most 1 kbp.

Return: An optimal fitting alignment score with respect to the mismatch score defined above, followed by an optimal fitting alignment of a substring of s against t. If multiple such alignments exist, then you may output any one.

Sample Dataset

```
>Rosalind_54
GCAAACCATAAGCCCTACGTGCCGCCTGTTTAAACTCGCGAACTGAATCTTCTGCTTCACGGTGAAAGTAC
CACAATGGTATCACACCCCAAGGAAAC
>Rosalind_46
GCCGTCAGGCTGGTGTCCG
```

Sample Output

```
5
ACCATAAGCCCTACGTG-CCG
GCCGTCAGGC-TG-GTGTCCG
```

> ### Citation
>
> This problem follows Jones & Pevzner, *An Introduction to Bioinformatics Algorithms*, Problem 6.23.

Problem 106

Semiglobal Alignment

Gaps on the Ends are Free

We have covered both global and local alignments. However, sometimes we need a hybrid approach that avoids the weaknesses of these two methods. One such alternate approach is that of fitting alignments outlined in "Finding a Motif with Modifications".

> Another tactic is to allow ourselves to trim off gaps appearing on the ends of a global alignment for free; this is relevant if one of our strings to be aligned happens to contain additional symbols on the ends that are not relevant for the particular alignment at hand.

Problem

A **semiglobal alignment** of strings s and t is an alignment in which any gaps appearing as prefixes or suffixes of s and t do not contribute to the alignment score.

Semiglobal alignment has sometimes also been called "overlap alignment". Rosalind defines overlap alignment differently (see "Overlap Alignment").

Given: Two DNA strings s and t in FASTA format, each having length at most 10 kbp.

Return: The maximum semiglobal alignment score of s and t, followed by an alignment of s and t achieving this maximum score. Use an alignment score in which matching symbols count +1, substitutions count -1, and there is a linear gap penalty of 1. If multiple optimal alignments exist, then you may return any one.

Sample Dataset

```
>Rosalind_79
CAGCACTTGGATTCTCGG
>Rosalind_98
CAGCGTGG
```

Sample Output

```
4
CAGCA-CTTGGATTCTCGG
---CAGCGTGG--------
```

> ### Citation
>
> This problem follows Jones & Pevzner, *An Introduction to Bioinformatics Algorithms*, Problem 6.24.

Problem 107

Finding All Similar Motifs

> ### The Case of Mutated Repeats
>
> In "Finding a Motif with Modifications", we considered a problem in which we were given a motif and a long string (perhaps representing a genome), and we aimed to find the "closest" substring of the long string to the motif. In that problem, "closest" was defined as a minimum with respect to edit distance.

Yet there may be multiple substring candidates from the genome that achieve the minimum distance to the motif; this situation might occur in practice when the motif forms a repeat that occurs multiple times with variations deriving from mutations.

In this problem, we would like to find *all* substrings of a genome that are within a certain fixed distance of the desired motif.

Problem

Given: A positive integer k ($k \leq 50$), a DNA string s of length at most 5 kbp representing a motif, and a DNA string t of length at most 50 kbp representing a genome.

Return: All substrings t' of t such that the edit distance $d_{\mathrm{E}}(s, t')$ is less than or equal to k. Each substring should be encoded by a pair containing its location in t followed by its length.

Sample Dataset

```
2
ACGTAG
ACGGATCGGCATCGT
```

Sample Output

```
1 4
1 5
1 6
```

Problem 108

Local Alignment with Affine Gap Penalty

Building Upon Local Alignments

We have thus far worked with local alignments with a linear gap penalty and global alignments with affine gap penalties (see "Local Alignment with Scoring Matrix" and "Global Alignment with Scoring Matrix and Affine Gap Penalty").

> It is only natural to take the intersection of these two problems and find an optimal local alignment given an affine gap penalty.

Problem

Given: Two protein strings s and t in FASTA format (each having length at most 10,000 aa).

Return: The maximum local alignment score of s and t, followed by substrings r and u of s and t, respectively, that correspond to the optimal local alignment of s and t. Use:

- The BLOSUM62 scoring matrix.
- Gap opening penalty equal to 11.
- Gap extension penalty equal to 1.

If multiple solutions exist, then you may output any one.

Sample Dataset

```
>Rosalind_8
PLEASANTLY
>Rosalind_18
MEANLY
```

Sample Output

```
12
LEAS
MEAN
```

Problem 109

Isolating Symbols in Alignments

How Much Does it Cost to Align Two Symbols?

As we saw in "Counting Optimal Alignments", there will usually be a huge number of different optimal alignments of two given strings. In this problem, which represents a first attempt to understand how much optimal alignments can differ, we will select two symbols at a time from the two strings and ask how much the maximum alignment score can differ from the optimal score if we demand that these two symbols must be aligned (i.e., implying that one symbol must be substituted for the other).

Problem

Say that we have two strings s and t of respective lengths m and n and an alignment score. Let's define a matrix M corresponding to s and t by setting $\mathrm{M}_{j,k}$ equal to the maximum score of any alignment that aligns $s[j]$ with $t[k]$. So each entry in M can be equal to at most the maximum score of any alignment of s and t.

Given: Two DNA strings s and t in FASTA format, each having length at most 1000 bp.

Return: The maximum alignment score of a global alignment of s and t, followed by the sum of all elements of the matrix M corresponding to s and t that was defined above. Apply the mismatch score introduced in "Finding a Motif with Modifications".

Sample Dataset

```
>Rosalind_35
ATAGATA
>Rosalind_5
```

```
ACAGGTA
```

Sample Output

```
3
-139
```

Citation

This problem follows Jones & Pevzner, *An Introduction to Bioinformatics Algorithms*, Problem 6.21

Hint

For the sample dataset $\mathrm{M} = \begin{bmatrix} 3 & 0 & -1 & -4 & -5 & -10 & -11 \\ 0 & 3 & 0 & -1 & -4 & -7 & -10 \\ -1 & 0 & 3 & -2 & -1 & -6 & -7 \\ -4 & -1 & 0 & 3 & 0 & -3 & -6 \\ -7 & -4 & -3 & 2 & 3 & 0 & -3 \\ -10 & -5 & -6 & -3 & 0 & 3 & -2 \\ -11 & -10 & -5 & -6 & -1 & -2 & 3 \end{bmatrix}$

Problem 110

Identifying Reversing Substitutions

Reversions Complicate Phylogenies

In "Fixing an Inconsistent Character Set", we mentioned how the construction of a phylogeny can be complicated by a the reversion of a character to a past state. For that matter, in calculating Hamming distance and edit distance, the assumption of parsimony required us to assume that if some nucleotide base or amino acid is aligned with an identical symbol in two genetic strings, then it has not changed on the evolutionary path between the two taxa.

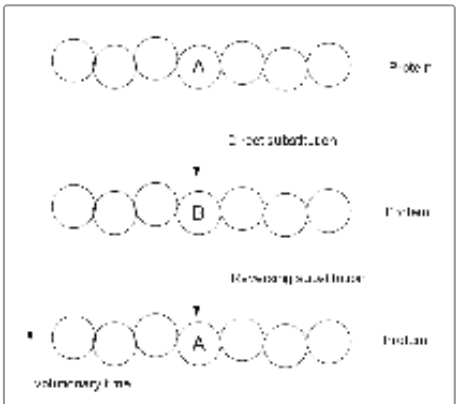


Figure 1. Illustration of an amino acid's reversing substitution after two point mutations.

However, this model is too strict in practice, where a base or amino acid can change to another state and then change back as the result of two point mutations, which is called a **reversing substitution**; see **Figure 1**. In the case of DNA, the presence of only four bases makes randomly occurring reversing substitutions common; these substitutions will carry over into amino acid language via transcription and

translation.

Unfortunately, with the possible exception of experimental evolution in bacteria, we lack the luxury of knowing the ancestral state of a nucleic acid strand. Instead, we must infer its most probable ancestor from homologous strands. To do so, we may use characters to construct a phylogeny (see "Character-Based Phylogeny"), then apply alignment-based phylogeny to infer strings for the tree's internal nodes (see "Alignment-Based Phylogeny"). Only once we have an adequate picture of the entire phylogeny, including its internal nodes, can we hope to identify reversing substitutions.

Problem

For a rooted tree T whose internal nodes are labeled with genetic strings, our goal is to identify reversing substitutions in T. Assuming that all the strings of T have the same length, a reversing substitution is defined formally as two parent-child string pairs (s, t) and (v, w) along with a position index i, where:

- there is a path in T from s down to w;
- $s[i] = w[i] \neq v[i] = t[i]$; and
- if u is on the path connecting t to v, then $t[i] = u[i]$.

In other words, the third condition demands that a reversing substitution must be contiguous: no other substitutions can appear between the initial and reversing substitution.

Given: A rooted binary tree T with labeled nodes in Newick format, followed by a collection of at most 100 DNA strings in FASTA format whose labels correspond to the labels of T. We will assume that the DNA strings have the same length, which does not exceed 400 bp).

Return: A list of all reversing substitutions in T (in any order), with each substitution encoded by the following three items:

- the name of the species in which the symbol is first changed, followed by the name of the species in which it changes back to its original state
- the position in the string at which the reversing substitution occurs; and
- the reversing substitution in the form original_symbol->substituted_symbol->reverted_symbol.

Sample Dataset

```
(((ostrich,cat)rat,mouse)dog,elephant)robot;
>robot
AATTG
>dog
GGGCA
>mouse
AAGAC
>rat
GTTGT
>cat
GAGGC
>ostrich
GTGTC
>elephant
AATTC
```

Sample Output

```
dog mouse 1 A->G->A
dog mouse 2 A->G->A
rat ostrich 3 G->T->G
rat cat 3 G->T->G
dog rat 3 T->G->T
```