

两个例子：调度 问题与投资问题

例1：调度问题

问题 有 n 项任务，每项任务加工时间已知。
从 0 时刻开始陆续安排到一台机器上加工。
每个任务的完成时间是从 0 时刻到任务加工截止的时间。
求：总完成时间（所有任务完成时间之和）
最短的安排方案。

实例

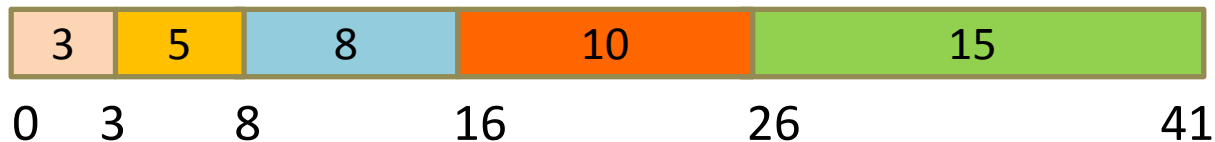
任务集 $S = \{1, 2, 3, 4, 5\}$,

加工时间： $t_1=3$, $t_2=8$, $t_3=5$, $t_4=10$, $t_5=15$

贪心法的解

算法：按加工时间 (3,8,5,10,15) 从小到大安排

解： 1, 3, 2, 4, 5



总完成时间：

$$\begin{aligned} t &= 3 + (3+5) + (3+5+8) + (3+5+8+10) + (3+5+8+10+15) \\ &= 3 \times 5 + 5 \times 4 + 8 \times 3 + 10 \times 2 + 15 \\ &= 94 \end{aligned}$$

问题建模

输入：任务集： $S=\{1, 2, \dots, n\}$,

第 j 项任务加工时间： $t_j \in \mathbb{Z}^+$, $j=1, 2, \dots, n$.

输出：调度 I , S 的排列 i_1, i_2, \dots, i_n ,

目标函数： I 的完成时间, $t(I) = \sum_{k=1}^n (n-k+1)t_{i_k}$

解 I^* : 使得 $t(I^*)$ 达到最小, 即

$$t(I^*) = \min \{ t(I) \mid I \text{ 为 } S \text{ 的排列} \}$$

贪心算法

设计策略：加工时间短的先做

算法：根据加工时间从小到大排序,依次加工

算法正确性：对所有输入实例都得到最优解

证：假如调度 f 第 i, j 项任务相邻且有逆序，
即 $t_i > t_j$ 。交换任务 i 和 j 得到调度 g ，



总完成时间 $t(g) - t(f) = t_j - t_i < 0$

直觉不一定是正确的

反例

有4 件物品要装入背包, 物品重量和价值如下:

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

背包重量限制是 6, 问如何选择物品, 使得不超重的情况下装入背包的物品价值达到最大?

实例的解

贪心法： 单位重量价值大的优先，总重不超 6

按照 $\frac{v_i}{w_i}$ 从大到小排序：1, 2, 3, 4

$$\frac{7}{3} > \boxed{\frac{9}{4}} > \frac{9}{5} > \boxed{\frac{2}{2}}$$

贪心法的解：{ 1, 4 }, 重量 5, 价值为 9.

更好的解：{ 2, 4 }, 重量 6, 价值 11.

算法设计

1. 问题建模
2. 选择什么算法？如何描述这个方法？
3. 这个方法是否对所有实例都得到最优解？
如何证明？
4. 如果不是，能否找到反例？

例2：投资问题

问题： m 元钱，投资 n 个项目. 效益函数 $f_i(x)$, 表示第 i 个项目投 x 元的效益, $i=1, 2, \dots, n$. 求如何分配每个项目的钱数使得总效益最大?

实例： 5 万元，投资给 4 个项目，效益函数：

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

建模

输入: $n, m, f_i(x), i = 1, 2, \dots, n, x = 1, 2, \dots, m$

解: n 维向量 $\langle x_1, x_2, \dots, x_n \rangle$, x_i 是第 i 个项目的钱数, 使得下述条件满足:

目标函数 $\max \sum_{i=1}^n f_i(x_i),$

约束条件 $\sum_{i=1}^n x_i = m, \quad x_i \in \mathbb{N}$

蛮力算法

对所有满足下述条件的向量 $\langle x_1, x_2, \dots, x_n \rangle$

$$x_1 + x_2 + \dots + x_n = m$$

x_i 为非负整数, $i = 1, 2, \dots, n$

计算相应的效益

$$f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)$$

从中确认效益最大的向量.

实例计算

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

$$x_1 + x_2 + x_3 + x_4 = 5$$

$$s_1 = \langle 0, 0, 0, 5 \rangle, v(s_1) = 24$$

$$s_2 = \langle 0, 0, 1, 4 \rangle, v(s_2) = 25$$

$$s_3 = \langle 0, 0, 2, 3 \rangle, v(s_3) = 32$$

...

$$s_{56} = \langle 5, 0, 0, 0 \rangle, v(s_{56}) = 15$$

解: $s = \langle 1, 0, 3, 1 \rangle$,

最大效益: $11 + 30 + 20 = 61$

蛮力算法的效率

方程 $x_1 + x_2 + \dots + x_n = m$ 的非负整数解
 $\langle x_1, x_2, \dots, x_n \rangle$ 的个数估计:

可行解表示成 0-1 序列: m 个1, $n-1$ 个 0

$\underbrace{1 \dots 1}_{x_1 \uparrow} 0 \underbrace{1 \dots 1}_{x_2 \uparrow} 0 \dots 0 \underbrace{1 \dots 1}_{x_n \uparrow}$

$n=4, m=7$

可行解 $\langle 1, 2, 3, 1 \rangle$

\Leftrightarrow 序列 $1 0 1 1 0 1 1 1 0 1$

蛮力算法的效率

序列个数是输入规模的指数函数

$$\begin{aligned} & C(m+n-1, m) \\ &= \frac{(m+n-1)!}{m!(n-1)!} \\ &= \Omega((1+\varepsilon)^{m+n-1}) \end{aligned}$$



有没有更好的算法？

小结

问题求解的关键

- 建模：对输入参数和解给出形式化或半形式化的描述
- 设计算法：
采用什么算法设计技术
正确性——是否对所有的实例都得到正确的解
- 分析算法——效率

问题计算复杂度 的界定:排序问题

例3 排序算法的效率

以元素比较作基本运算

算法	最坏情况下	平均情况下
插入排序	$O(n^2)$	$O(n^2)$
冒泡排序	$O(n^2)$	$O(n^2)$
快速排序	$O(n^2)$	$O(n\log n)$
堆排序	$O(n\log n)$	$O(n\log n)$
二分归并排序	$O(n\log n)$	$O(n\log n)$

插入排序的插入操作

前面已经排好，插入2

输入

5	7	1	3	6	2	4
---	---	---	---	---	---	---

插入2

1	3	5	6	7	2	4
---	---	---	---	---	---	---

插入后

1	2	3	5	6	7	4
---	---	---	---	---	---	---

插入排序运行实例

输入

5	7	1	3	6	2	4
---	---	---	---	---	---	---

初始

5	7	1	3	6	2	4
---	---	---	---	---	---	---

插入7

5	7	1	3	6	2	4
---	---	---	---	---	---	---

插入1

1	5	7	3	6	2	4
---	---	---	---	---	---	---

插入3

1	3	5	7	6	2	4
---	---	---	---	---	---	---

插入6

1	3	5	6	7	2	4
---	---	---	---	---	---	---

插入2

1	2	3	5	6	7	4
---	---	---	---	---	---	---

插入4

1	2	3	4	5	6	7
---	---	---	---	---	---	---

冒泡排序的一次巡回

巡回前

5	1	6	2	8	3	4	7
---	---	---	---	---	---	---	---

巡回

5	1	6	2	8	3	4	7
---	---	---	---	---	---	---	---

巡回后

1	5	2	6	3	4	7	8
---	---	---	---	---	---	---	---

冒泡排序运行实例

5	8	1	3	6	2	4	7
---	---	---	---	---	---	---	---

巡回1	5	1	3	6	2	4	7	8
-----	---	---	---	---	---	---	---	---

巡回2	1	3	5	2	4	6	7	8
-----	---	---	---	---	---	---	---	---

巡回3	1	3	2	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

巡回4	1	2	3	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

巡回5	1	2	3	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

快速排序一次递归运行

5	8	1	3	6	2	4	7
---	---	---	---	---	---	---	---

交换1

5	8	1	3	6	2	4	7
---	---	---	---	---	---	---	---

交换2

5	4	1	3	6	2	8	7
---	---	---	---	---	---	---	---

划分

5	4	1	3	2	6	8	7
---	---	---	---	---	---	---	---

子问题

2	4	1	3	5	6	8	7
---	---	---	---	---	---	---	---

二分归并排序运行实例

5	8	1	3	6	2	4	7
---	---	---	---	---	---	---	---

划分

5	8	1	3	6	2	4	7
---	---	---	---	---	---	---	---

递归
排序

1	3	5	8	2	4	6	7
---	---	---	---	---	---	---	---

1	3	5	8	2	4	6	7
---	---	---	---	---	---	---	---

合并后的输出

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

问题的计算复杂度分析

问题:

哪个排序算法效率最高?

是否可找到更好的排序算法?

排序问题计算难度如何?

其他问题的计算复杂度

问题计算复杂度估计方法

n^2

插入排序
冒泡排序
快速排序

$n\log n$

堆排序
归并排序

?

更好的算
法下界



那个排序算法效率最高?

排序问题的难度?

小结

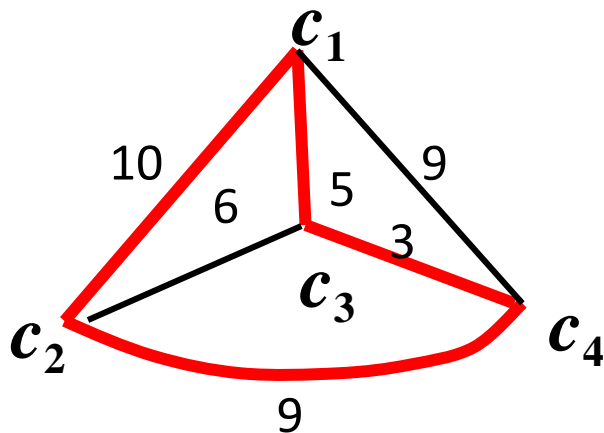
- 几种排序算法简介
 - 插入排序
 - 冒泡排序
 - 快速排序
 - 归并排序
- 排序问题的难度估计——界定什么是最好的排序算法

货郎问题与计算 复杂性理论

例4 货郎问题

问题:

有 n 个城市, 已知任两个城市之间的距离. 求一条每个城市恰好经过1次的回路, 使得总长度最小.



建模与算法

- 输入

有穷个城市的集合 $C = \{c_1, c_2, \dots, c_n\}$,

距离 $d(c_i, c_j) = d(c_j, c_i) \in \mathbb{Z}^+$, $1 \leq i < j \leq n$

- 解: $1, 2, \dots, n$ 的排列 k_1, k_2, \dots, k_n 使得:

$$\min \left\{ \sum_{i=1}^{n-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_n}, c_{k_1}) \right\}$$

- 现状: 至今没找到有效的算法

0-1背包问题

0-1背包问题：有 n 个件物品要装入背包，第 i 件物品的重量 w_i ，价值 v_i ， $i=1,2,\dots,n$. 背包最多允许装入的重量为 B ，问如何选择装入背包的物品，使得总价值达到最大？

实例： $n=4$ ， $B=6$ ，物品的重量和价值如下

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

0-1背包问题建模

问题的解: 0-1向量 $\langle x_1, x_2, \dots, x_n \rangle$

$x_i=1 \Leftrightarrow$ 物品 i 装入背包

目标函数 $\max \sum_{i=1}^n v_i x_i$

约束条件 $\sum_{i=1}^n w_i x_i \leq B$

$x_i = 0, 1, i = 1, 2, \dots, n$

双机调度

双机调度问题：有 n 项任务，任务 i 的加工时间为 $t_i, t_i \in \mathbb{Z}^+, i=1,2,\dots,n$. 用两台相同的机器加工，从0时刻开始计时，完成时间是后停止加工机器的停机时间. 问如何把这些任务分配到两台机器上，使得完成时间达到最小？

实例：任务集 $S = \{1,2,3,4,5,6\}$

$$t_1=3, t_2=10, t_3=6, \underline{t_4=2}, t_5=1, t_6=7$$

解：机器 1 的任务：1, 2, 4

机器 2 的任务：3, 5, 6

完成时间： $\max\{3+10+2, 6+1+7\}=15$

双机调度建模

解: 0-1向量 $\langle x_1, x_2, \dots, x_n \rangle$, $x_i=1$ 表示任务 i 分配到第一台机器, $i=1,2,\dots,n$.

不妨设机器1的加工时间 \leq 机器2的加工时间令 $T=t_1+t_2+\dots+t_n$, $D=\lfloor T/2 \rfloor$, 机器1的加工时间不超过 D , 且达到最大.



如何对该问题建模? 目标函数与约束条件是什么?

NP-hard问题

- 这样的问题有数千个，大量存在于各个应用领域.
- 至今没找到有效算法：现有的算法的运行时间是输入规模的指数或更高阶函数.
- 至今没有人能够证明对于这类问题不存在多项式时间的算法.
- 从是否存在多项式时间算法的角度看，这些问题彼此是等价的. 这些问题的难度处于可有效计算的边界.

Algorithm + Data Structure = Programming

好的算法

提高求解问题的效率

节省存储空间

算法的研究目标

问题→建模并寻找算法

算法→算法的评价

算法类→问题复杂度估计

问题类→能够求解的边界

算法设计技术

算法分析方法

问题复杂度分析

计算复杂性理论

课程主要内容

近似算法

随机算法

NP 完全理论简介

算法分析与问题的计算复杂性

分治
策略

动态
规划

贪心
算法

回溯与
分支限界

数学基础、数据结构

计算复杂性理论:

NP完全理论

其他算法

算法设计:

算法分析方法

算法设计技术

基础知识

算法研究的重要性

算法设计与分析技术在计算机科学与技术领域有着重要的应用背景

算法设计分析与计算复杂性理论研究是计算机科学技术的核心研究领域

- 1966-2005期间，Turing奖获奖50人，其中10人以算法设计，7人以计算理论、自动机和复杂性研究领域的杰出贡献获奖
- 计算复杂性理论的核心课题“ $P=NP?$ ”是本世纪 7个最重要的数学问题之一

提高学生素质和分析问题解决问题的能力，
培养计算思维

小结

- NP-hard问题的几个例子：货郎问题
0-1背包问题、双机调度问题等
- NP-hard问题的计算现状
- 计算复杂性理论的核心——NP完全理论
- 算法研究的主要内容及重要意义

算法及其 时间复杂度

问题及实例

- 问题

需要回答的一般性提问，通常含若干参数

- 问题描述

定义问题参数(集合,变量,函数,序列等)

说明每个参数的取值范围及参数间的关系

定义问题的解

说明解满足的条件(优化目标或约束条件)

- 问题实例

参数的一组赋值可得到问题的一个实例

算法

- 算法

有限条指令的序列

这个指令序列确定了解决某个问题的一系列运算或操作

- 算法 A 解决问题 P

把问题 P 的任何实例作为算法 A 的输入

每步计算是确定性的

A 能够在有限步停机

输出该实例的正确的解

基本运算与输入规模

- **算法时间复杂度**: 针对指定**基本运算**, 计数算法所做运算次数
- **基本运算**: 比较, 加法, 乘法, 置指针, 交换...
- **输入规模**: 输入串编码长度
通常用下述参数度量: 数组元素多少, 调度问题的任务个数, 图的顶点数与边数等.
- **算法基本运算次数可表为输入规模的函数**
- **给定问题和基本运算就决定了一个算法类**

输入规模

- 排序：数组中元素个数 n
- 检索：被检索数组的元素个数 n
- 整数乘法：两个整数的位数 m, n
- 矩阵相乘：矩阵的行列数 i, j, k
- 图的遍历：图的顶点数 n , 边数 m
- ...

基本运算

- 排序: 元素之间的比较
- 检索: 被检索元素 x 与数组元素的比较
- 整数乘法: 每位数字相乘(位乘) 1 次
 m 位和 n 位整数相乘要做 mn 次位乘
- 矩阵相乘: 每对元素乘 1 次
 $i \times j$ 矩阵与 $j \times k$ 矩阵相乘要做 ijk 次乘法
- 图的遍历: 置指针
- ...

算法的两种时间复杂度

对于相同输入规模的不同实例，算法的基本运算次数也不一样，可定义两种时间复杂度

最坏情况下的时间复杂度 $W(n)$

算法求解输入规模为 n 的实例所需要的最长时间

平均情况下的时间复杂度 $A(n)$

在给定同样规模为 n 的输入实例的概率分布下，算法求解这些实例所需要的平均时间

$A(n)$ 计算公式

平均情况下的时间复杂度 $A(n)$

设 S 是规模为 n 的实例集

实例 $I \in S$ 的概率是 P_I

算法对实例 I 执行的基本运算次数是 t_I

$$A(n) = \sum_{I \in S} P_I t_I$$

在某些情况下可以假定每个输入实例概率相等

例子：检索

检索问题

输入：非降顺序排列的数组 L , 元素数 n ,
数 x

输出： j

若 x 在 L 中, j 是 x 首次出现的下标;

否则 $j = 0$

基本运算： x 与 L 中元素的比较

顺序检索算法

$j=1$, 将 x 与 $L[j]$ 比较. 如果 $x=L[j]$, 则算法停止, 输出 j ; 如果不等, 则把 j 加1, 继续 x 与 $L[j]$ 的比较, 如果 $j>n$, 则停机并输出0.

实例

1	2	3	4	5
---	---	---	---	---

$x = 4$, 需要比较 4 次

$x = 2.5$, 需要比较 5 次

最坏情况的时间估计

不同的输入有 $2n + 1$ 个，分别对应：

$$x = L[1], x = L[2], \dots, x = L[n]$$

$$x < L[1], L[1] < x < L[2], L[2] < x < L[3], \dots, L[n] < x$$

最坏情况下时间： $W(n) = n$

最坏的输入： x 不在 L 中或 $x = L[n]$

要做 n 次比较

平均情况的时间估计

输入实例的概率分布：

假设 x 在 L 中概率是 p ,且每个位置概率相等

$$A(n) = \sum_{i=1}^n i \frac{p}{n} + (1-p)n$$
$$= \frac{p(n+1)}{2} + (1-p)n$$

等差数列求和

当 $p=1/2$ 时,

$$A(n) = \frac{n+1}{4} + \frac{n}{2} \approx \underline{\frac{3n}{4}}$$

改进顺序检索算法

$j=1$, 将 x 与 $L[j]$ 比较. 如果 $x=L[j]$, 则算法停止, 输出 j ; 如果 $x > L[j]$, 则把 j 加1, 继续 x 与 $L[j]$ 的比较; 如果 $x < L[j]$, 则停机并输出0. 如果 $j > n$, 则停机并输出 0.

实例

1	2	3	4	5
---	---	---	---	---

$x = 4$, 需要比较 4 次

$x = 2.5$, 需要比较 3 次

时间估计

最坏情况下: $W(n) = n$

平均情况下

输入实例的概率分布: 假设 x 在 L 中每个位置与空隙的概率都相等



改进检索算法平均时间复杂度是多少?

小结:

- 算法最坏和平均情况下的时间复杂度定义
- 如何计算上述时间复杂度



算法的伪码表示

算法的伪码描述

赋值语句: \leftarrow

分支语句: if ...then ... [else...]

循环语句: while, for, repeat until

转向语句: goto

输出语句: return

调用: 直接写过程的名字

注释: //...

例：求最大公约数

算法 **Euclid** (m, n)

输入：非负整数 m, n , 其中 m 与 n 不全为 0

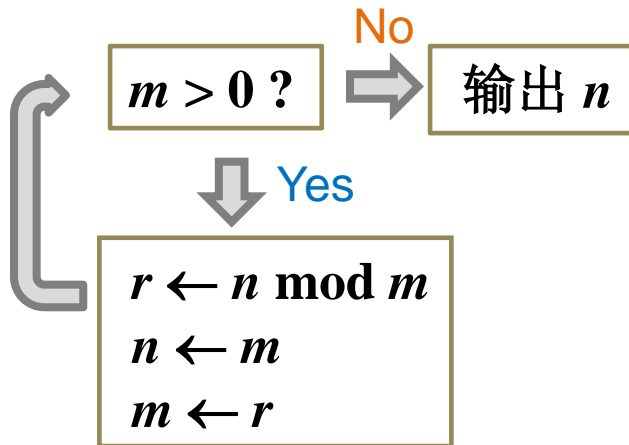
输出： m 与 n 的最大公约数

1. while $m > 0$ do
2. $r \leftarrow n \bmod m$
3. $n \leftarrow m$
4. $m \leftarrow r$
5. return n

运行实例: $n=36, m=15$

while	n	m	r
第1次	36	15	6
第2次	15	6	3
第3次	6	3	0
	3	0	0

↓
输出3



例：改进的顺序检索

算法 **Search (L, x)**

输入：数组 $L[1..n]$, 元素从小到大排列, 数 x .

输出：若 x 在 L 中, 输出 x 的位置下标 j ;
否则输出0.

1. $j \leftarrow 1$
2. while $j \leq n$ and $x > L[j]$ do $j \leftarrow j+1$
3. if $x < L[j]$ or $j > n$ then $j \leftarrow 0$
4. return j

例：插入排序

算法 **Insert Sort** (A, n)

输入： n 个数的数组 A

输出： 按照递增顺序排好序的数组 A

1. for $j \leftarrow 2$ to n do
2. $x \leftarrow A[j]$
3. $i \leftarrow j-1$ //3-7 行把 $A[j]$ 插入 $A[1..j-1]$
4. while $i > 0$ and $x < A[i]$ do
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow x$

运行实例

2	4	1	5	3
---	---	---	---	---

$j = 3, x = A[3] = 1$

$i = 2, A[2] = 4$

$i > 0, x < A[2] \quad \checkmark$

2	4	4	5	3
---	---	---	---	---

$A[3] = 4, i = 1, x = 1$

$i > 0, x < A[1] \quad \checkmark$

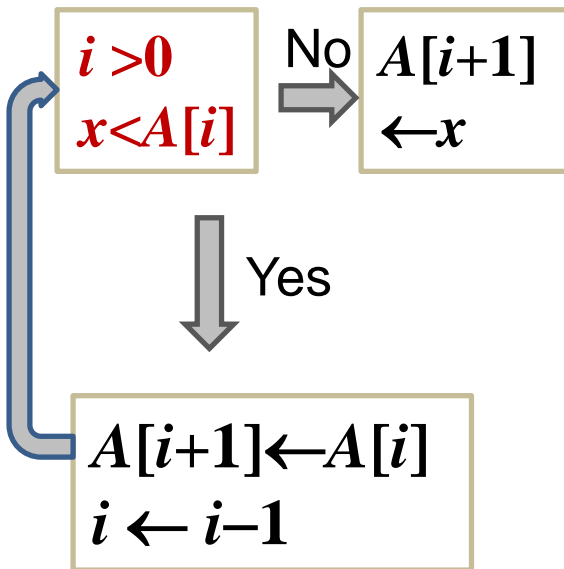
2	2	4	5	3
---	---	---	---	---

$A[2] = 2, i = 0, x = 1$

$i > 0 \quad \times$

1	2	4	5	3
---	---	---	---	---

4. while $i > 0$ and $x < A[i]$ do
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow x$



例：二分归并排序

MergeSort (A, p, r)

输入：数组 $A[p..r]$

输出：按递增顺序排序的数组 A

1. if $p < r$
2. then $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort (A, p, q)
4. MergeSort ($A, q+1, r$)
5. Merge (A, p, q, r)

MergeSort有递归调用,也调用Merge过程

例：算法A的伪码

算法 A

输入：实数的数组 $P[0..n]$ ，实数 x

输出： y

1. $y \leftarrow P[0]$; $power \leftarrow 1$
2. for $i \leftarrow 1$ to n do
3. $power \leftarrow power * x$
4. $y \leftarrow y + P[i] * power$
5. return y



算法
A计
算什么值
？

对 $i = 1, 2, \dots, n$ \Rightarrow

$$\begin{aligned} power &\leftarrow power * x \\ y &\leftarrow y + P[i] * power \end{aligned}$$

	i	$power$	y
初值		1	$P[0]$
循环	1	x	$P[0] + P[1]*x$
	2	x^2	$P[0] + P[1]*x + P[2]*x^2$
	3	x^3	$P[0] + P[1]*x + P[2]*x^2 + P[3]*x^3$
			...

输入 $P[0..n]$ 是 n 次多项式 $P(x)$ 的系数
 算法 A 计算该多项式在 x 的值

小结

用伪码表示算法

- 伪码不是程序代码，只是给出算法的主要步骤
- 伪码中有哪些关键字？
- 伪码中允许过程调用

函数的渐近的界

大O符号

定义： 设 f 和 g 是定义域为自然数集 \mathbf{N} 上的函数. 若存在正数 c 和 n_0 , 使得对一切 $n \geq n_0$ 有

$$0 \leq f(n) \leq c g(n)$$

成立, 则称 $f(n)$ 的渐近的上界是 $g(n)$, 记作

$$f(n) = O(g(n))$$

例子

设 $f(n) = n^2 + n$, 则

$f(n) = O(n^2)$, 取 $c = 2$, $n_0 = 1$ 即可

$f(n) = O(n^3)$, 取 $c = 1$, $n_0 = 2$ 即可

1. $f(n) = O(g(n))$, $f(n)$ 的阶不高于 $g(n)$ 的阶.
2. 可能存在多个正数 c , 只要指出一个即可.
3. 对前面有限个值可以不满足不等式.
4. 常函数可以写作 $O(1)$.

大 Ω 符号

定义： 设 f 和 g 是定义域为自然数集 \mathbb{N} 上的函数. 若存在正数 c 和 n_0 , 使得对一切 $n \geq n_0$ 有

$$\underline{0 \leq cg(n) \leq f(n)}$$

成立, 则称 $f(n)$ 的渐近的下界是 $g(n)$, 记作

$$f(n) = \Omega(g(n))$$

例子

设 $f(n) = n^2 + n$, 则

$f(n) = \Omega(n^2)$, 取 $c = 1, n_0 = 1$ 即可

$f(n) = \Omega(100n)$, 取 $c = 1/100, n_0 = 1$ 即可

1. $f(n) = \Omega(g(n))$, $f(n)$ 的阶不低于 $g(n)$ 的阶.
2. 可能存在多个正数 c , 指出一个即可.
3. 对前面有限个 n 值可以不满足上述不等式.

小o符号

定义 设 f 和 g 是定义域为自然数集 \mathbf{N} 上的函数. 若对于任意正数 c 都存在 n_0 , 使得对一切 $n \geq n_0$ 有

$$\underline{0 \leq f(n) < c g(n)}$$

成立, 则记作

$$f(n) = o(g(n))$$

例子

例子: $f(n)=n^2+n$, 则

$$f(n)=o(n^3)$$

$c \geq 1$ 显然成立, 因为 $n^2+n < cn^3$ ($n_0=2$)

任给 $1 > c > 0$, 取 $n_0 > \lceil 2/c \rceil$ 即可. 因为

$$cn \geq \underline{cn_0} > 2 \quad (\text{当 } n \geq n_0)$$

$$n^2+n < 2n^2 < cn^3$$

1. $f(n) = o(g(n))$, $f(n)$ 的阶低于 $g(n)$ 的阶
2. 对不同正数 c , n_0 不一样. c 越小 n_0 越大.
3. 对前面有限个 n 值可以不满足不等式.

小 ω 符号

定义： 设 f 和 g 是定义域为自然数集 \mathbf{N} 上的函数. 若对于 任意正数 c 都存在 n_0 , 使得对一切 $n \geq n_0$ 有

$$\underline{0 \leq cg(n) < f(n)}$$

成立, 则记作

$$f(n) = \omega(g(n))$$

例子

设 $f(n) = n^2 + n$, 则

$$f(n) = \omega(n),$$

不能写 $f(n) = \omega(n^2)$, 因为取 $c = 2$, 不存在 n_0 使得对一切 $n \geq n_0$ 有下式成立

$$c n^2 = 2n^2 < n^2 + n \quad \times$$

1. $f(n) = \omega(g(n))$, $f(n)$ 的阶高于 $g(n)$ 的阶.
2. 对不同的正数 c , n_0 不等, c 越大 n_0 越大.
3. 对前面有限个 n 值可以不满足不等式.

Θ 符号

若 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$,
则记作

$$f(n) = \Theta(g(n))$$

例子: $f(n) = n^2 + n$, $g(n) = 100n^2$, 那么有

$$f(n) = \Theta(g(n))$$

1. $f(n)$ 的阶与 $g(n)$ 的阶相等.
2. 对前面有限个 n 值可以不满足条件.

例子：素数测试

算法 **PrimalityTest**(n)

输入： n , 大于2的奇整数

输出： true 或者 false

1. $s \leftarrow \lfloor n^{1/2} \rfloor$

2. for $j \leftarrow 2$ to s

3. if j 整除 n

4. then return false

5. return true

问题：

若 $n^{1/2}$ 可在 $O(1)$

计算, 基本运算是整除, 以下表示是否正确？

$W(n) = O(n^{1/2})$



$W(n) = \Theta(n^{1/2})$



为什么？

小结

- 五种表示函数的阶的符号
 $O, \Omega, o, \omega, \Theta$
- 定义
- 如何用定义证明函数的阶？

有关函数渐 近的界的定理

定理1

定理 设 f 和 g 是定义域为自然数集合的函数.

(1) 如果 $\lim_{n \rightarrow \infty} f(n)/g(n)$ 存在, 并且等于某个

常数 $c > 0$, 那么 $f(n) = \Theta(g(n))$.

(2) 如果 $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$, 那么

$f(n) = o(g(n))$.

(3) 如果 $\lim_{n \rightarrow \infty} f(n)/g(n) = +\infty$, 那么

$f(n) = \omega(g(n))$.

证明用到 Θ, o, ω 定义

证明定理1(1)

根据极限定义, 对于给定正数 ε 存在某个 n_0 , 只要 $n \geq n_0$, 就有

$$|f(n)/g(n) - c| < \varepsilon$$

$$c - \varepsilon < f(n)/g(n) < c + \varepsilon$$

取 $\varepsilon = c/2$,

$$c/2 < f(n)/g(n) < 3c/2 < 2c$$

对所有 $n \geq n_0$, $f(n) \leq 2cg(n)$, 于是 $f(n) = O(g(n))$;

对所有 $n \geq n_0$, $f(n) \geq (c/2)g(n)$, 于是 $f(n) = \Omega(g(n))$.

从而 $f(n) = \Theta(g(n))$.

例：估计函数的阶

例1 设 $f(n) = \frac{1}{2}n^2 - 3n$, 证明 $f(n) = \Theta(n^2)$.

证 因为

$$\lim_{n \rightarrow \infty} \frac{f(n)}{n^2} = \lim_{n \rightarrow \infty} \frac{\frac{1}{2}n^2 - 3n}{n^2} = \frac{1}{2}$$

根据定理1, 有 $f(n) = \Theta(n^2)$.

一些重要结果

可证明：多项式函数的阶低于指数函数的阶

$$n^d = o(r^n), \quad r > 1, \quad d > 0$$

证 不妨设 d 为正整数,

$$\lim_{n \rightarrow \infty} \frac{n^d}{r^n} = \lim_{n \rightarrow \infty} \frac{dn^{d-1}}{r^n \ln r} = \lim_{n \rightarrow \infty} \frac{d(d-1)n^{d-2}}{r^n (\ln r)^2}$$

$$= \dots = \lim_{n \rightarrow \infty} \frac{d!}{r^n (\ln r)^d} = 0$$

分子分母
求导数

一些重要结果(续)

可证明：对数函数的阶低于幂函数的阶

$$\ln n = o(n^d), \quad d > 0$$

证

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\ln n}{n^d} &= \lim_{n \rightarrow \infty} \frac{\frac{1}{n}}{dn^{d-1}} \\ &= \lim_{n \rightarrow \infty} \frac{1}{dn^d} = 0 \end{aligned}$$

定理 2

定理 设函数 f, g, h 的定义域为自然数集合,

- (1) 如果 $f=O(g)$ 且 $g=O(h)$, 那么 $f=O(h)$.
- (2) 如果 $f=\Omega(g)$ 且 $g=\Omega(h)$, 那么 $f=\Omega(h)$.
- (3) 如果 $f=\Theta(g)$ 和 $g=\Theta(h)$, 那么 $f=\Theta(h)$

函数的阶之间的关系具有传递性

例子

按照阶从高到低排序以下函数：

$$f(n)=(n^2+n)/2, \quad g(n)=10n$$

$$h(n)=1.5^n, \quad t(n)=n^{1/2}$$

$$h(n) = \omega(f(n)),$$

$$f(n) = \omega(g(n)),$$

$$g(n) = \omega(t(n)),$$

排序 $h(n), f(n), g(n), t(n)$

定理3

定理 假设函数 f 和 g 的定义域为自然数集,
若对某个其它函数 h , 有 $f=O(h)$ 和 $g=O(h)$,
那么 $f + g = O(h)$.

该性质可以推广到有限个函数.

算法由有限步骤构成. 若每一步的时间复杂度函数的上界都是 $h(n)$, 那么该算法的时间复杂度函数可以写作 $O(h(n))$.

小结

- 估计函数的阶的方法：
计算极限
阶具有传递性
- 对数函数的阶低于幂函数的阶，多项式函数的阶低于指数函数的阶.
- 算法的时间复杂度是各步操作时间之和，在常数步的情况下取最高阶的函数即可.

几类重要的函数

基本函数类

阶的高低

至少指数级: $2^n, 3^n, n!, \dots$

多项式级: $n, n^2, n \log n, n^{1/2}, \dots$

对数多项式级: $\log n, \log^2 n, \log \log n, \dots$

对数函数

符号:

$$\log n = \log_2 n$$

$$\log^k n = (\log n)^k$$

$$\log \log n = \log(\log n)$$

性质:

$$(1) \log_2 n = \Theta(\log_l n)$$

$$(2) \log_b n = o(n^\alpha) \quad \alpha > 0$$

$$(3) a^{\log_b n} = n^{\log_b a}$$

有关性质 (1) 的证明

$$\log_k n = \frac{\log_l n}{\log_l k} \quad \log_l k \text{ 为常数}$$

$$\lim_{n \rightarrow \infty} \frac{\log_k n}{\log_l n} = \lim_{n \rightarrow \infty} \frac{\log_l n}{\log_l k \cdot \log_l n} = \frac{1}{\log_l k}$$

根据定理, $\log_k n = \Theta(\log_l n)$

有关性质 (2) (3) 的说明

$$\log_b n = \Theta(\ln n)$$

$$\ln n = o(n^\alpha) \Rightarrow \log_b n = o(n^\alpha) \quad \alpha > 0$$

$$a^{\log_b n} = n^{\log_b a}$$



$$\log_b n \log_b a = \log_b a \log_b n$$

指数函数与阶乘

Stirling公式 $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n \log n)$$

应用：估计搜索空间大小

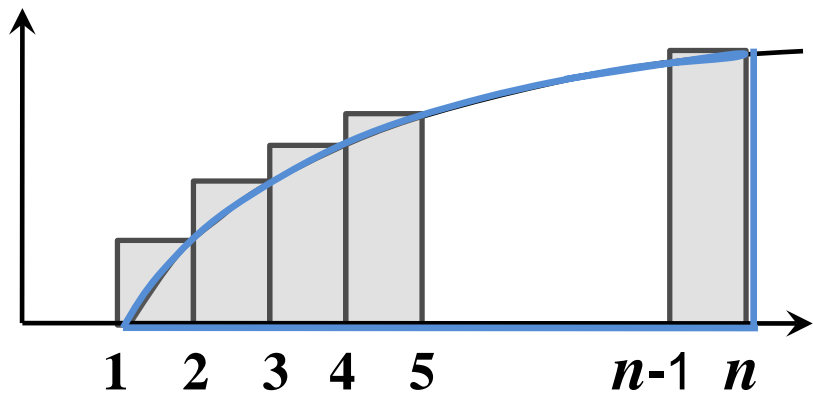
$$C_{m+n-1}^m = \frac{(m+n-1)!}{m!(n-1)!}$$

m 元钱、投资
 n 个项目的分配方
案数（视频001）

$$= \frac{\sqrt{2\pi(m+n-1)}(m+n-1)^{m+n-1}(1+\Theta(\frac{1}{m+n-1}))}{\sqrt{2\pi m}m^m(1+\Theta(\frac{1}{m}))\sqrt{2\pi(n-1)}(n-1)^{n-1}(1+\Theta(\frac{1}{n-1}))}$$

$$= \Theta((1+\varepsilon)^{m+n-1})$$

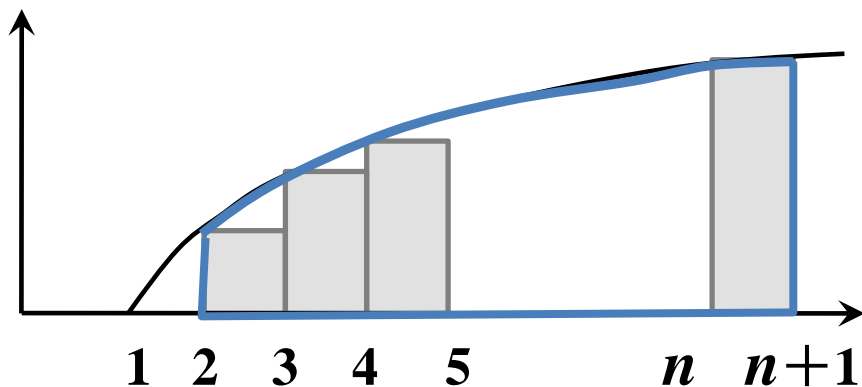
$\log(n!) = \Omega(n \log n)$ 的证明



$$\log(n!) = \sum_{k=1}^n \log k \geq \int_1^n \log x dx$$

$$= \log e(n \ln n - n + 1) = \Omega(n \log n)$$

$\log(n!) = O(n \log n)$ 的证明



$$\log(n!) = \sum_{k=1}^n \log k \leq \int_2^{n+1} \log x dx = O(n \log n)$$

取整函数

取整函数的定义

$\lfloor x \rfloor$: 表示小于等于 x 的最大的整数

$\lceil x \rceil$: 表示大于等于 x 的最小的整数

实例

$$\lfloor 2.6 \rfloor = 2$$

$$\lceil 2.6 \rceil = 3$$

$$\lfloor 2 \rfloor = \lceil 2 \rceil = 2$$

应用: 二分检索

输入数组长度: n

中位数的位置: $\lfloor n/2 \rfloor$

与中位数比较后子问

题大小: $\lfloor n/2 \rfloor$

取整函数的性质

$$(1) \quad \underline{x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1}$$

$$(2) \quad \underline{\lfloor x+n \rfloor = \lfloor x \rfloor + n, \lceil x+n \rceil = \lceil x \rceil + n, n \text{ 为整数}}$$

$$(3) \quad \underline{\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n}$$

$$(4) \quad \underline{\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil, \left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor}$$

证明(1)

(1) 如果 x 是整数 n , 根据定义 $\lfloor x \rfloor = \lceil x \rceil = n$,

$$x-1 < \lfloor x \rfloor = x = \lceil x \rceil < x+1$$

如果 $n < x < n+1$, n 为整数, 那么

$$\lfloor x \rfloor = n, \lceil x \rceil = n+1,$$

从而有

$$x-1 < n = \lfloor x \rfloor, \quad n < x < n+1 = \lceil x \rceil$$

$$\Rightarrow x-1 < n = \lfloor x \rfloor < x < \lceil x \rceil = n+1 < x+1$$

例：按照阶排序

$\log(n!), \log^2 n, 1, n!, n2^n, n^{1/\log n},$

$(3/2)^n, \sqrt{\log n}, (\log n)^{\log n}, 2^{2^n},$

$n^{\log \log n}, n^3, \log \log n, n \log n, n,$

$2^{\log n}, \log n$

例：按照阶排序

$$2^{2^n}, n!, n2^n, (3/2)^n, (\log n)^{\log n} = n^{\log \log n},$$

$$n^3, \log(n!) = \Theta(n \log n), n = 2^{\log n},$$

$$\log^2 n, \log n, \sqrt{\log n}, \log \log n,$$

$$n^{1/\log n} = 1$$

小结

- 几类常用函数的阶的性质
 - 对数函数
 - 指数函数
 - 阶乘函数
 - 取整函数
- 如何利用上述性质估计函数的阶？

序列求和的方法

数列求和公式

等差、等比数列与调和级数

$$\sum_{k=1}^n a_k = \frac{n(a_1 + a_n)}{2}$$

$$\sum_{k=0}^n aq^k = \frac{a(1-q^{n+1})}{1-q}, \quad \sum_{k=0}^{\infty} aq^k = \frac{a}{1-q} (q < 1)$$

$$\sum_{k=1}^n \frac{1}{k} = \ln n + O(1)$$

求和的例子

$$\sum_{t=1}^k t 2^{t-1} = \sum_{t=1}^k t(2^t - 2^{t-1})$$

拆项

$$= \sum_{t=1}^k t 2^t - \sum_{t=1}^k t 2^{t-1} = \sum_{t=1}^k t 2^t - \sum_{t=0}^{k-1} (t+1) 2^t$$

$$= \sum_{t=1}^k t 2^t - \sum_{t=0}^{k-1} t 2^t - \sum_{t=0}^{k-1} 2^t$$

变限

拆项

$$= k 2^k - (2^k - 1) = (k-1) 2^k + 1$$

二分检索算法

算法 BinarySearch (T, l, r, x)

输入：数组 T ，下标从 l 到 r ；数 x

输出： j

1. $l \leftarrow 1; r \leftarrow n$

2. while $l \leq r$ do

3. $m \leftarrow \lfloor (l+r)/2 \rfloor$

4. if $T[m]=x$ then return m // x 中位元素

5. else if $T[m] > x$ then $r \leftarrow m-1$

6. else $l \leftarrow m+1$

7. return 0

二分检索运行实例

1	2	3	4	5	6	7
			3.5			

第1次
 $3.5 < 4$

1	2	3	4	5	6	7
	3.5					

第2次
 $3.5 > 2$

1	2	3	4	5	6	7
		3.5				

第3次
 $3.5 > 3$

$2n+1$ 个输入

假设 $n = 2^k - 1$, 输入有 $2n + 1$ 种:

$$x = T[1]$$

$$x = T[2]$$

...

$$x = T[n-1]$$

$$x = T[n]$$

x 在 T 中

$$x < T[1]$$

$$T[1] < x < T[2]$$

...

$$T[n-1] < x < T[n]$$

$$T[n] < x$$

x 不在 T 中

比较 t 次的输入个数



比较1次:1个



比较2次:2个



比较3次:4个

对 $t = 1, 2, \dots, k-1$, 比较 t 次: 2^{t-1} 个

比较 k 次的输入有 $2^{k-1} + n + 1$ 个

总次数: 对每个输入乘以次数并求和


二分检索平均时间复杂度

假设 $n = 2^k - 1$, 各种输入概率相等

$$\begin{aligned} A(n) &= \frac{1}{2n+1} \left[\sum_{t=1}^{k-1} t 2^{t-1} + k(2^{k-1} + n + 1) \right] \\ &= \frac{1}{2n+1} \left[\sum_{t=1}^k t 2^{t-1} + k(n+1) \right] \\ &= \frac{1}{2n+1} \left[(k-1)2^k + 1 + k(n+1) \right] \\ &= \frac{k2^k - 2^k + 1 + k2^k}{2^{k+1} - 1} \approx k - \frac{1}{2} = \lfloor \log n \rfloor + \frac{1}{2} \end{aligned}$$

估计和式上界的放大法

放大法:

1. $\sum_{k=1}^n a_k \leq n a_{\max}$ 

2. 假设存在常数 $r < 1$, 使得 对一切 $k \geq 0$ 有 $a_{k+1}/a_k \leq r$ 成立

$$\sum_{k=0}^n a_k \leq \sum_{k=0}^{\infty} a_0 r^k = a_0 \sum_{k=0}^{\infty} r^k = \frac{a_0}{1-r}$$



$a_1 \leq a_0 r, a_2 \leq a_1 r \leq a_0 r^2, \dots$

放大法的例子

估计 $\sum_{k=1}^n \frac{k}{3^k}$ 的上界.

解 由 $a_k = \frac{k}{3^k}$, $a_{k+1} = \frac{k+1}{3^{k+1}}$

$$\frac{a_{k+1}}{a_k} = \frac{1}{3} \frac{k+1}{k} \leq \frac{2}{3}$$

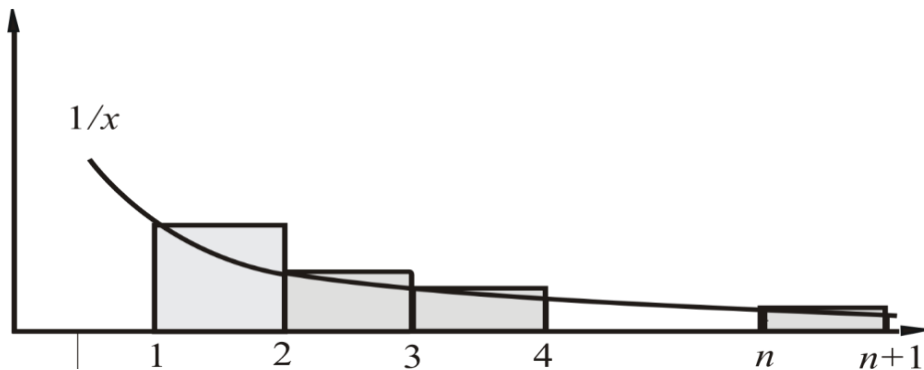
得

$$\sum_{k=1}^n \frac{k}{3^k} \leq \sum_{k=1}^{\infty} \frac{1}{3} \left(\frac{2}{3}\right)^{k-1} = \frac{1}{3} \frac{1}{1 - \frac{2}{3}} = 1$$

估计和式渐近的界

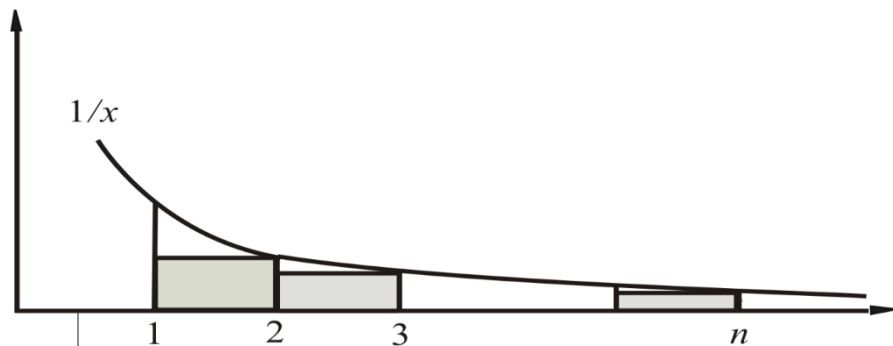
估计 $\sum_{k=1}^n \frac{1}{k}$ 的渐近的界.

$$\sum_{k=1}^n \frac{1}{k} \geq \int_1^{n+1} \frac{dx}{x} = \ln(n+1)$$



估计和式渐近的界

$$\sum_{k=1}^n \frac{1}{k} = \frac{1}{1} + \sum_{k=2}^n \frac{1}{k} \leq 1 + \int_1^n \frac{dx}{x} \\ = \ln n + 1$$



$$\sum_{k=1}^n \frac{1}{k} = \Theta(\log n)$$

小结

- 序列求和基本公式：
等差数列
等比数列
调和级数
- 估计序列和：
放大法求上界
用积分做和式的渐近的界
- 应用：计数循环过程的基本运算次数

递推方程与 算法分析

递推方程

设序列 $a_0, a_1, \dots, a_n, \dots$, 简记为 $\{a_n\}$,
一个把 a_n 与某些个 $a_i (i < n)$ 联系起来的
等式叫做关于序列 $\{a_n\}$ 的递推方程

递推方程的求解:

给定关于序列 $\{a_n\}$ 的递推方程和若干初值, 计算 a_n

递推方程的例子

Fibonacci数

1, 1, 2, 3, 5, 8, 13, 21,
34, 55, ...



数学家Fibonacci
意大利1170-1240

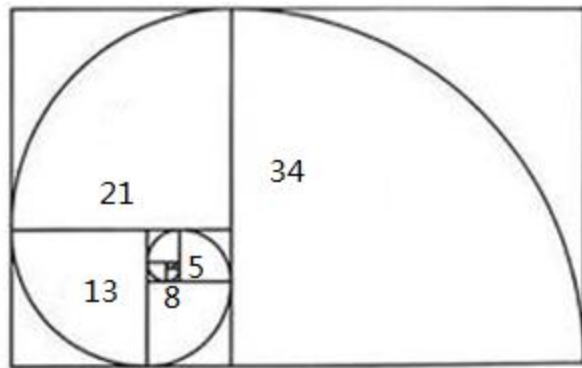
递推方程: $f_n = f_{n-1} + f_{n-2}$

初值: $f_0 = 1, f_1 = 1$

解:

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^{n+1}$$

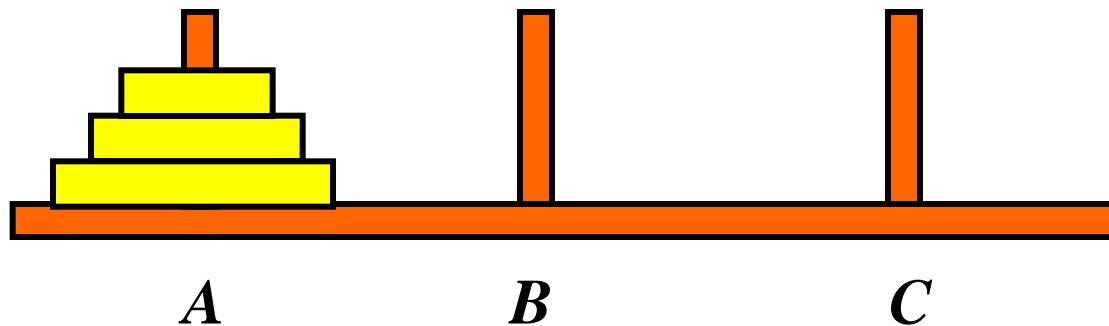
Fibonacci数的存在



55



Hanoi塔问题



n 个盘子从大到小顺序放在A 柱上，
要把它们从 A 移到 C，每次移动 1个
盘子，移动时不允许大盘压在小盘上。
设计一种移动方法。

递归算法

算法 Hanoi (A, C, n) // n 个盘子 A 到 C

1. if $n=1$ then move (A, C) // 1个盘子 A 到 C
2. else Hanoi ($A, B, n-1$)
3. move (A, C)
4. Hanoi ($B, C, n-1$)

设 n 个盘子的移动次数为 $T(n)$

$$T(n) = 2 T(n-1) + 1,$$

$$T(1) = 1,$$

分析算法

$$T(n) = 2 T(n-1) + 1, \quad T(1) = 1,$$

解

$$T(n) = 2^n - 1$$

1 秒移1个，64个盘子要多少时间？

5000亿年！千万亿次/秒，4个多小时



有没有更好的算法？

没有！这是一个难解的问题，不存在多项式时间的算法！

插入排序

算法 Insert Sort (A, n)

1. for $j \leftarrow 2$ to n
2. $x \leftarrow A[j]$
3. $i \leftarrow j-1$ // 把 $A[j]$ 插入
4. while $i > 0$ and $x < A[i]$ do
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow x$

最坏情况下时间复杂度

插入排序：

设基本运算是元素比较，对规模为 n 的输入最坏情况下的时间复杂度 $W(n)$

$$W(n) = W(n-1) + n - 1$$

$$W(1) = 0$$

解为

$$W(n) = n(n-1)/2$$

小结

- 递推方程的定义及初值
- 递推方程与算法时间复杂度的关系

Hanoi塔的递归算法

插入排序的迭代算法

迭代法求解 递推方程

迭代法

- 不断用递推方程的右部替换左部
- 每次替换，随着 n 的降低在和式中多出一项
- 直到出现初值停止迭代
- 将初值代入并对和式求和
- 可用数学归纳法验证解的正确性

Hanoi 塔算法

$$\begin{aligned}T(n) &= 2 T(n-1) + 1 \\T(1) &= 1\end{aligned}$$

$$T(n) = 2 \underline{T(n-1)} + 1$$

$$= 2 [\underline{2T(n-2)} + 1] + 1$$

$$= 2^2 T(n-2) + 2 + 1$$

$$= \dots$$

$$= 2^{n-1}T(1) + 2^{n-2} + 2^{n-3} + \dots + 2 + 1$$

$$= 2^{n-1} + 2^{n-1} - 1$$

$$= 2^n - 1$$

代入初值
求和

插入排序算法

$$\begin{cases} W(n) = W(n-1) + n - 1 \\ W(1) = 0 \end{cases}$$

$$W(n) = W(n-1) + n - 1$$

$$= [W(n-2) + n - 2] + n - 1$$

$$= W(n-2) + n - 2 + n - 1$$

$$= \dots$$

$$= W(1) + 1 + 2 + \dots + (n-2) + (n-1)$$

$$= 1 + 2 + \dots + (n-2) + (n-1)$$

$$= n(n-1)/2$$

换元迭代

- 将对 n 的递推式换成对其他变元 k 的递推式
- 对 k 直接迭代
- 将解 (关于 k 的函数) 转换成关于 n 的函数

二分归并排序

MergeSort (A, p, r)

输入：数组 $A[p..r]$

输出：按递增顺序排序的数组 A

1. if $p < r$
2. then $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort (A, p, q)
4. MergeSort ($A, q+1, r$)
5. Merge (A, p, q, r)

换元

假设 $n=2^k$, 递推方程如下:

$$W(n)=2W(n/2)+n-1$$

$$W(1)=0$$

换元:

$$W(2^k) = 2W(2^{k-1}) + 2^k - 1$$

$$W(0) = 0$$

迭代求解

$$W(2^k) = 2W(2^{k-1}) + 2^k - 1$$

$$\begin{aligned} W(n) &= 2W(2^{k-1}) + 2^k - 1 \\ &= 2[2W(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1 \end{aligned}$$

换元，
对 k 迭代

$$\begin{aligned} &= 2^2 W(2^{k-2}) + 2^k - 2 + 2^k - 1 \\ &= 2^2 [2W(2^{k-3}) + 2^{k-2} - 1] + 2^k - 2 + 2^k - 1 \end{aligned}$$

$= \dots$

$$= 2^k W(1) + k 2^k - (2^{k-1} + 2^{k-2} + \dots + 2 + 1)$$

$$= k 2^k - 2^k + 1$$

$$= n \log n - n + 1$$

解的正确性-归纳验证

证明:下述递推方程的解是 $W(n)=n(n-1)/2$

$$W(n)=W(n-1)+n-1$$

$$W(1)=0$$

方法: 数学归纳法

证 $n=1$, $W(1)=1\times(1-1)/2 = 0$

假设对于 n , 解满足方程, 则

$$\begin{aligned} & W(n+1) \\ &= \underline{W(n)+n} = \underline{n(n-1)/2} + n \\ &= n[(n-1)/2+1] = n(n+1)/2 \end{aligned}$$

小结

迭代法求解递推方程

- 直接迭代，代入初值，然后求和
- 对递推方程和初值进行换元，然后求和，求和后进行相反换元，得到原始递推方程的解
- 验证方法——数学归纳法

差消法化简 高阶递推方程

快速排序

- 假设 $A[p..r]$ 的元素彼此不等

以首元素 $A[p]$ 对数组 $A[p..r]$ 划分,使得:

小于 x 的元素放在 $A[p..q-1]$

大于 x 的元素放在 $A[q+1..r]$

- 递归对 $A[p..q-1]$ 和 $A[q+1..r]$ 排序

工作量: 子问题工作量+划分工作量

输入情况

- 有 n 种可能的输入

x 排好序位置	子问题 1 规模	子问题 2 规模
1	0	$n-1$
2	1	$n-2$
3	2	$n-3$
...
$n-1$	$n-2$	1
n	$n-1$	0

对每个输入，划分的比较次数都是 $n-1$

工作量总和

$$T(0) + T(n-1) + n-1$$

$$T(1) + T(n-2) + n-1$$

$$T(2) + T(n-3) + n-1$$

...

$$+ T(n-1) + T(0) + n-1$$

$$2[T(1)+...+T(n-1)]+n(n-1)$$

快速排序平均工作量

假设首元素排好序在每个位置是等概率的

$$T(n) = \frac{2}{n} \sum_{i=1}^{n-1} T(i) + O(n), n \geq 2$$

$$T(1) = 0$$

全部历史递推方程

对于高阶方程应该先化简，然后迭代

差消化简

利用两个方程相减，将右边的项尽可能消去，以达到降阶的目的

$$T(n) = \frac{2}{n} \sum_{i=1}^{n-1} T(i) + cn$$

$$nT(n) = 2 \sum_{i=1}^{n-1} T(i) + cn^2$$

$$(n-1)T(n-1) = 2 \sum_{i=1}^{n-2} T(i) + c(n-1)^2$$

差消化简

$$\begin{aligned} & nT(n) - \underline{(n-1)T(n-1)} \\ = & \underline{2T(n-1)} + cn^2 - c(n-1)^2 \end{aligned}$$



$$nT(n) = (n+1)T(n-1) + c_1n$$



$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c_1}{n+1}$$

迭代求解

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \boxed{\frac{c_1}{n+1}} = \dots$$

$$= c_1 \left[\frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right] + \frac{T(1)}{2}$$

$$= c_1 \left[\frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right]$$

代入
初值

$$= \Theta(\log n)$$

$$T(n) = \Theta(n \log n)$$

小结

- 对于高阶递推方程先用差消法化简为一阶方程
- 迭代求解

递归树

递归树的概念

- 递归树是迭代计算的模型.
- 递归树的生成过程与迭代过程一致.
- 递归树上所有项恰好是迭代之后产生和式中的项.
- 对递归树上的项求和就是迭代后方程的解.

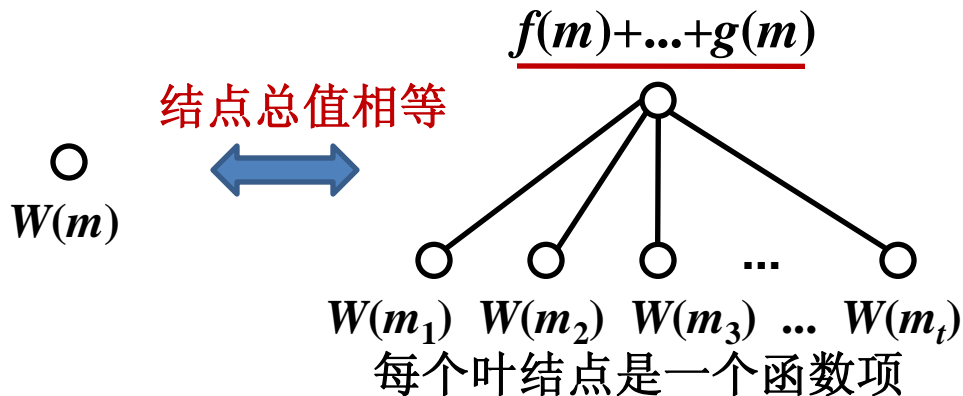
迭代在递归树中的表示

如果递归树上某结点标记为 $W(m)$

$$W(m) = W(m_1) + \dots + W(m_t)$$

$$+ \underline{f(m) + \dots + g(m)}, \quad m_1, \dots, m_t < m$$

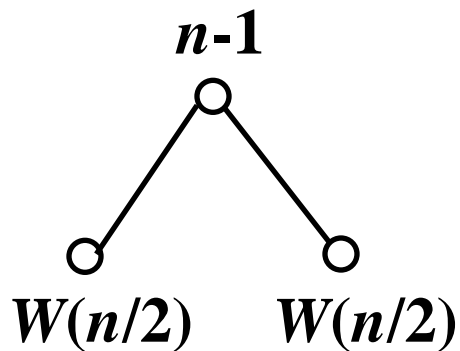
其中 $W(m_1), \dots, W(m_t)$ 称为函数项。



二层子树的例子

二分归并排序

$$W(n) = 2W(n/2) + \underline{n-1}$$

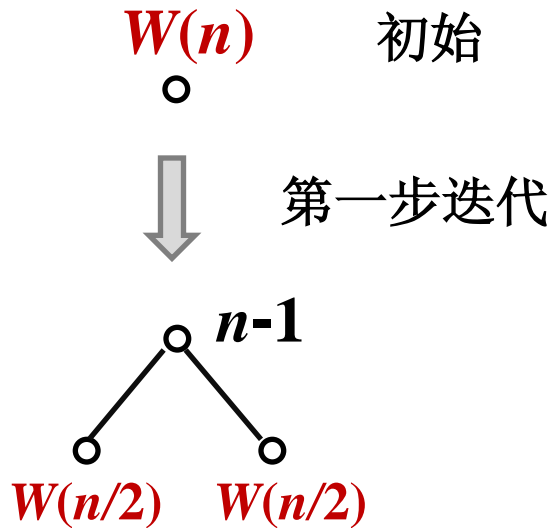


递归树的生成规则

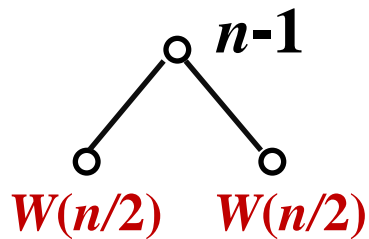
- 初始，递归树只有根结点，其值为 $W(n)$
- 不断继续下述过程：
将函数项叶结点的迭代式 $W(m)$ 表示成二层子树
用该子树替换该叶结点
- 继续递归树的生成，直到树中无函数项(只有初值)为止.

递归树生成实例

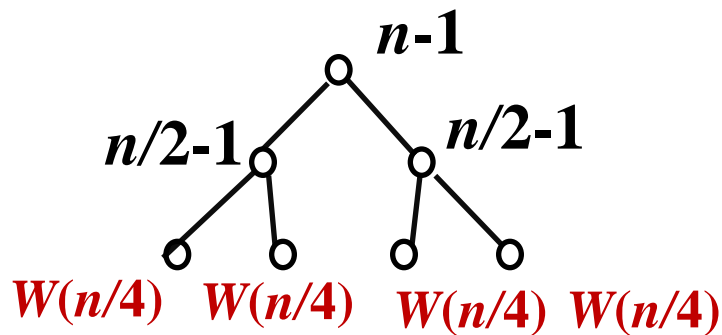
$$W(n) = 2W(n/2) + n-1$$



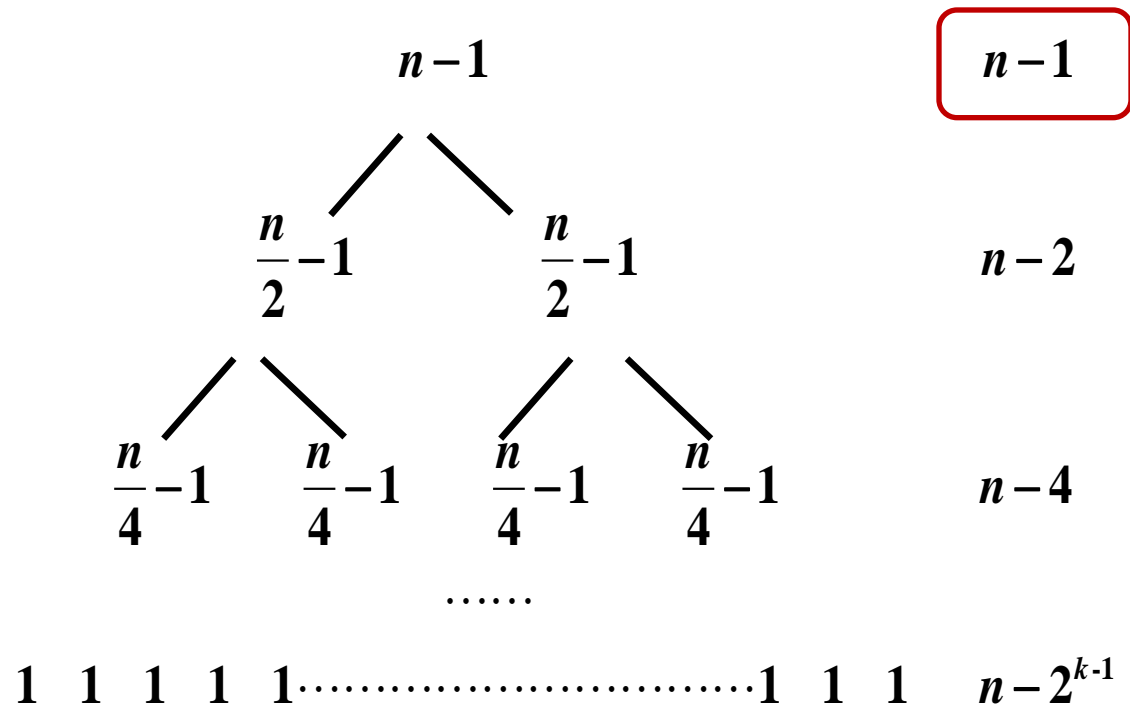
递归树生成实例



第二步迭代



递归树



对递归树上的量求和

$$W(n) = 2W(n/2) + n - 1, \quad n = 2^k,$$

$$W(1) = 0$$

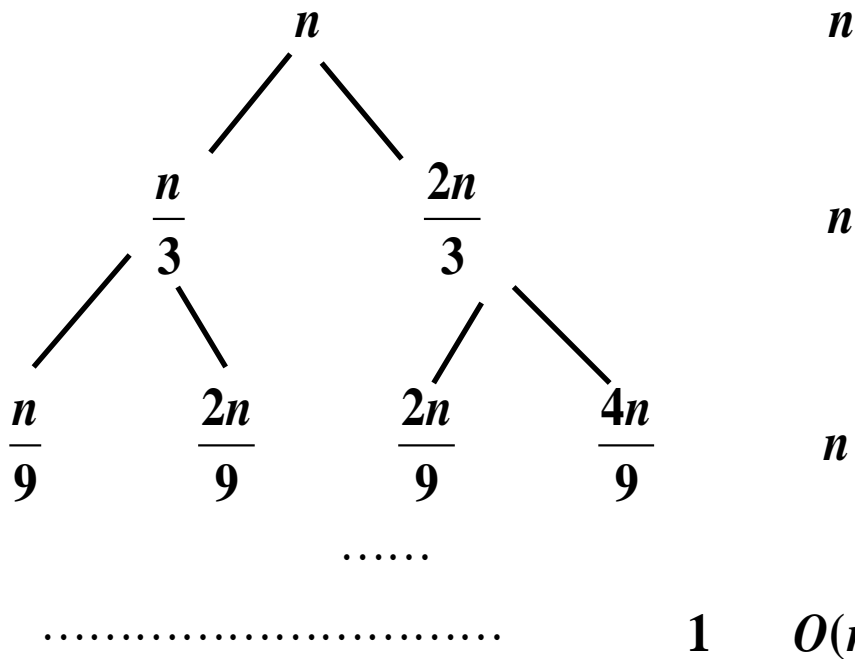
$$W(n) = n - 1 + n - 2 + \dots + n - 2^{k-1}$$

$$= kn - (2^k - 1)$$

$$= n \log n - n + 1$$

递归树应用实例

求解方程: $T(n) = T(n/3) + T(2n/3) + n$



求和

方程: $T(n)=T(n/3)+T(2n/3)+n$

递归树层数 k , 每层 $O(n)$

$$n(2/3)^k = 1$$

$$\Rightarrow (3/2)^k = n$$

$$\Rightarrow k = O(\log_{3/2} n)$$

$$T(n)=O(n\log n)$$

小结

- 递归树是迭代的图形表述
- 递归树的生成规则
- 如何利用递归树求解递推方程?

主定理及其证明

主定理的应用背景

求解递推方程

$$T(n) = a T(n/b) + f(n)$$

a : 归约后的子问题个数

n/b : 归约后子问题的规模

$f(n)$: 归约过程及组合子问题的解的工作量

二分检索: $T(n) = T(n/2) + 1$

二分归并排序: $T(n) = 2T(n/2) + n - 1$

主定理

定理： 设 $a \geq 1, b > 1$ 为常数, $f(n)$ 为函数, $T(n)$ 为非负整数, 且 $T(n) = aT(n/b) + f(n)$, 则

1. 若 $f(n) = O(n^{\log_b a - \varepsilon})$, $\varepsilon > 0$, 那么

$$T(n) = \Theta(n^{\log_b a})$$

存在 ε

2. 若 $f(n) = \Theta(n^{\log_b a})$, 那么

$$T(n) = \Theta(n^{\log_b a} \log n)$$

存在 ε

3. 若 $f(n) = \Omega(n^{\log_b a + \varepsilon})$, $\varepsilon > 0$, 且对于某个常数 $c < 1$ 和充分大的 n 有 $af(n/b) \leq cf(n)$, 那么

$$T(n) = \Theta(f(n))$$

存在 c
和 n_0

迭代

$$T(n)=aT(n/b)+f(n)$$

设 $n=b^k$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$$= a[aT(\frac{n}{b^2}) + f(\frac{n}{b})] + f(n)$$

$$= a^2T(\frac{n}{b^2}) + af(\frac{n}{b}) + f(n)$$

$$= \dots$$

迭代结果

$$a^{\log_b n} = n^{\log_b a}$$

$$= a^k T\left(\frac{n}{b^k}\right) + a^{k-1} f\left(\frac{n}{b^{k-1}}\right) + \dots + a f\left(\frac{n}{b}\right) + f(n)$$

$$= \underline{a^k T(1)} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right)$$

$$= \underline{c_1 n^{\log_b a}} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right) \quad T(1) = c_1$$

- 第一项为所有最小子问题的计算工作量
- 第二项为迭代过程归约到子问题及综合解的工作量

Case1

$$f(n) = O(n^{\log_b a - \varepsilon})$$

$$T(n) = c_1 n^{\log_b a} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right)$$

$$= c_1 n^{\log_b a} + O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right)$$

$$= c_1 n^{\log_b a} + O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} \frac{a^j}{(b^{\log_b a - \varepsilon})^j}\right)$$

Case1(续)

$$\frac{1}{(b^{\log_b a - \varepsilon})^j} = \frac{b^{\varepsilon j}}{(b^{\log_b a})^j} = \frac{b^{\varepsilon j}}{a^j}$$

$$= c_1 n^{\log_b a} + O(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} \frac{a^j}{(b^{\log_b a - \varepsilon})^j})$$

$$= c_1 n^{\log_b a} + O(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} (b^{\varepsilon})^j)$$

$$= c_1 n^{\log_b a} + O(n^{\log_b a - \varepsilon} \frac{b^{\varepsilon \log_b n} - 1}{b^{\varepsilon} - 1})$$

$$= c_1 n^{\log_b a} + O(n^{\log_b a - \varepsilon} \underline{n^{\varepsilon}}) = \Theta(n^{\log_b a})$$

Case2

$$f(n) = \Theta(n^{\log_b a})$$

$$T(n) = c_1 n^{\log_b a} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right)$$

$$= c_1 n^{\log_b a} + \underbrace{\Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right)}$$

$$= c_1 n^{\log_b a} + \underbrace{\Theta\left(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \frac{a^j}{a^j}\right)}$$

$$= c_1 n^{\log_b a} + \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_b a} \log n)$$

Case3

$$f(n) = \Omega(n^{\log_b a + \varepsilon}) \quad (1)$$

$$af(n/b) \leq cf(n) \quad (2)$$

$$\begin{aligned} T(n) &= c_1 n^{\log_b a} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right) \\ &\leq c_1 n^{\log_b a} + \sum_{j=0}^{\log_b n - 1} c^j f(n) \end{aligned}$$

$$a^j f\left(\frac{n}{b^j}\right) \leq a^{j-1} cf\left(\frac{n}{b^{j-1}}\right)$$

$$\leq ca^{j-1} f\left(\frac{n}{b^{j-1}}\right) \leq \dots \leq c^j f(n)$$

Case3 (续)

$$T(n) \leq c_1 n^{\log_b a} + \sum_{j=0}^{\log_b n - 1} c^j f(n)$$

$$= c_1 n^{\log_b a} + f(n) \frac{c^{\log_b n} - 1}{c - 1}$$

$$= c_1 n^{\log_b a} + \Theta(f(n))$$

$$= \Theta(f(n))$$

小结

- 主定理的应用背景
- 主定理的内容
- 主定理的证明

主定理的应用

求解递推方程：例1

例1 求解递推方程

$$T(n) = 9T(n/3) + n$$

解 上述递推方程中的

$$a = 9, \quad b = 3, \quad f(n) = n$$

$$\underline{n^{\log_3 9} = n^2, \quad f(n) = O(n^{\log_3 9 - 1})}$$

相当于主定理的**case1**，其中 $\varepsilon=1$ 。

根据定理得到 $T(n) = \Theta(n^2)$

求解递推方程：例2

例2 求解递推方程

$$T(n) = T(2n/3) + 1$$

解 上述递推方程中的

$$\underline{a = 1, b = 3/2, f(n) = 1,}$$

$$n^{\log_{3/2} 1} = n^0 = 1$$

相当于主定理的**Case2** .

根据定理得到 $T(n) = \Theta(\log n)$

求解递推方程：例3

例3 求解递推方程

$$T(n) = 3T(n/4) + n\log n$$

解 上述递推方程中的

$$\underline{a = 3, \quad b = 4, \quad f(n) = n\log n}$$

$$n\log n = \Omega(n^{\log_4 3 + \varepsilon}) \approx \Omega(n^{0.793 + \varepsilon})$$

取 $\varepsilon = 0.2$ 即可.

条件验证

要使 $a f(n/b) \leq c f(n)$ 成立,

代入 $f(n) = n \log n$, 得到

$$\underline{3 (n/4) \log (n/4) \leq c n \log n}$$

只要 $\underline{c \geq 3/4}$, 上述不等式可以对所有充分大的 n 成立. 相当于主定理的 **Case3**.

因此有 $T(n) = \Theta(f(n)) = \Theta(n \log n)$

递归算法分析

二分检索: $W(n)=W(n/2)+1, W(1)=1$

$$a=1, b=2, \underline{n^{\log_2 1}=1}, f(n)=1,$$

属于Case2,

$$W(n)=\Theta(\log n)$$

二分归并排序:

$$W(n)=2W(n/2)+n-1, W(1)=0$$

$$a=2, b=2, \underline{n^{\log_2 2}=n}, f(n)=n-1$$

属于Case2,

$$W(n)=\Theta(n \log n)$$

不能使用主定理的例子

例4 求解 $T(n)=2T(n/2)+n\log n$

解 $a=b=2$, $n^{\log_b a}=n$, $f(n)=n\log n$

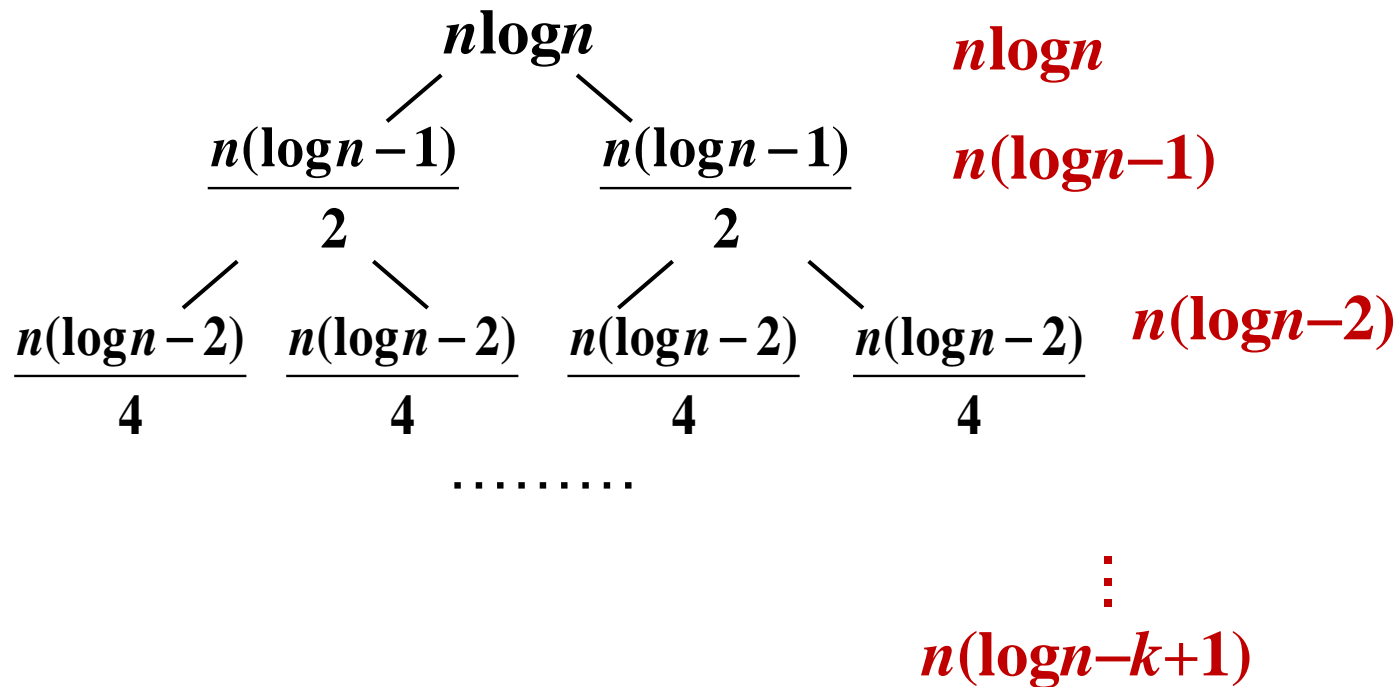
不存在 $\varepsilon > 0$ 使下式成立

$$n \log n = \Omega(n^{1+\varepsilon})$$

不存在 $c < 1$ 使 $af(n/b) \leq cf(n)$ 对所有充分大的 n 成立

$$2(n/2)\log(n/2) = \underline{n(\log n - 1)} \leq cn \log n$$

递归树求解



求和

$$T(n)$$

$$= n \log n + n(\log n - 1) + n(\log n - 2)$$

$$+ \dots + n(\log n - k + 1)$$

$$= (n \log n) \log n - n \underline{(1 + 2 + \dots + k - 1)}$$

$$= n \log^2 n - n \underline{k(k - 1) / 2} = O(n \log^2 n)$$

小结

- 使用主定理求解递推方程需要满足什么条件？
- 主定理怎样用于算法复杂度分析？

分治策略 的设计思想

分治策略的基本思想

分治策略（ Divide and Conquer ）

1. 将原始问题划分或者归结为规模较小的子问题
1. 递归或迭代求解每个子问题
2. 将子问题的解综合得到原问题的解

注意：

1. 子问题与原始问题性质完全一样
2. 子问题之间可彼此独立地求解
3. 递归停止时子问题可直接求解

二分检索

算法 Binary Search (T, l, r, x)

输入：数组 T ，下标从 l 到 r ；数 x

输出： j // 若 x 在 T 中, j 为下标; 否则为 0

1. $l \leftarrow 1; r \leftarrow n$
2. while $l \leq r$ do
3. $m \leftarrow \lfloor (l + r) / 2 \rfloor$ // m 为中间位置
4. if $T[m] = x$ then return m // x 是中位数
5. else if $T[m] > x$ then $r \leftarrow m - 1$
6. else $l \leftarrow m + 1$
7. return 0

二分检索算法设计思想

- 通过 x 与中位数的比较，将原问题归结为规模减半的子问题，如果 x 小于中位数，则子问题由小于 x 的数构成，否则子问题由大于 x 的数构成。
- 对子问题进行二分检索。
- 当子问题规模为 1 时，直接比较 x 与 $T[m]$ ，若相等则返回 m ，否则返回 0。



是否能够递归实现？

二分检索时间复杂度分析

二分检索问题最坏情况下时间复杂度

$$W(n) = W(\lfloor n/2 \rfloor) + 1$$

$$W(1) = 1$$

可以解出

$$W(n) = \lfloor \log n \rfloor + 1$$

二分归并排序

算法 Merge Sort (A, p, r)

输入：数组 $A[p .. r]$

输出：元素按从小到大排序的数组 A

1. if $p < r$
2. then $q \leftarrow \lfloor (p + r)/2 \rfloor$ 对半划分
3. Merge Sort (A, p, q) 子问题1
4. Merge Sort ($A, q+1, r$) 子问题 2
5. Merge (A, p, q, r) 综合解

二分归并排序设计思想

- 划分将原问题归结为规模为 $n/2$ 的 2 个子问题
- 继续划分，将原问题归结为规模为 $n/4$ 的 4 个子问题。继续...，当子问题规模为 1 时，划分结束。
- 从规模 1 到 $n/2$ ，陆续归并被排好序的两个子数组。每归并一次，数组规模扩大一倍，直到原始数组。

二分归并排序时 间复杂度分析

假设 n 为2的幂，二分归并排序最坏情况下时间复杂度

$$W(n) = 2W(n/2) + n - 1$$

$$W(1) = 0$$

可以解出

$$W(n) = n \log n - n + 1$$

Hanoi塔的递归算法

算法 $\text{Hanoi}(A, C, n)$ // n 个盘子A到C

1. if $n=1$ then move (A, C) //1个盘子A到C

2. else $\text{Hanoi}(A, B, n-1)$

3. move (A, C)

4. $\text{Hanoi}(B, C, n-1)$

设 n 个盘子的移动次数为 $T(n)$

$$T(n) = 2 T(n-1) + 1,$$

$$T(1) = 1,$$

$$T(n)=2^n-1$$

算法设计思想

- 将原问题归结为规模为 $n-1$ 的2个子问题.
- 继续归约, 将原问题归结为规模为 $n-2$ 的 4 个子问题. 继续..., 当子问题规模为1 时, 归约过程截止.
- 从规模 1到 $n-1$, 陆续组合两个子问题的解. 直到规模为 n .

小结

通过几个例子展示分治算法的特点：

- 将原问题归约为规模小的子问题，
子问题与原问题具有相同的性质。
- 子问题规模足够小时可直接求解。
- 算法可以递归也可以迭代实现。
- 算法的分析方法：递推方程。

分治算法的一般 描述和分析方法

分治算法的一般性描述

分治算法 Divide-and-Conquer(P)

1. if $|P| \leq c$ then $S(P)$
2. divide P into P_1, P_2, \dots, P_k
3. for $i \leftarrow 1$ to k
4. $y_i \leftarrow$ Divide-and-Conquer(P_i)
5. Return Merge (y_1, y_2, \dots, y_k)

划分

求解子
问题

综合
解

设计要点

- 原问题可以划分或者归约为规模较小的子问题

子问题与原问题具有相同的性质

子问题的求解彼此独立

划分时子问题的规模尽可能均衡

- 子问题规模足够小时可直接求解
- 子问题的解综合得到原问题的解
- 算法实现：递归或迭代

分治算法时间分析

时间复杂度函数的递推方程

$$W(n) = W(|P_1|) + W(|P_2|) + \dots + W(|P_k|) + f(n)$$

$$W(c) = C$$

- P_1, P_2, \dots, P_k 为划分后产生的子问题
- $f(n)$ 为划分子问题以及将子问题的解综合得到原问题解的总工作量
- 规模为 c 的最小子问题的工作量为 C

两类常见的递推方程

$$f(n) = \sum_{i=1}^k a_i f(n-i) + g(n) \quad (1)$$

$$f(n) = af\left(\frac{n}{b}\right) + d(n) \quad (2)$$

例子:

Hanoi塔, $W(n) = 2W(n-1) + 1$

二分检索, $W(n) = W(n/2) + 1$

归并排序, $W(n) = 2W(n/2) + n - 1$

递推方程的求解

方程1
$$f(n) = \sum_{i=1}^k a_i f(n-i) + g(n)$$

方程2
$$f(n) = af\left(\frac{n}{b}\right) + d(n)$$

求解方法

方程1：迭代法、递归树

方程2：迭代法、换元法、递归树、
主定理

方程2的解

方程 $T(n) = aT(n/b) + d(n)$

$d(n)$ 为常数

$$T(n) = \begin{cases} O(n^{\log_b a}) & a \neq 1 \\ O(\log n) & a = 1 \end{cases}$$

$d(n) = cn$

$$T(n) = \begin{cases} O(n) & a < b \\ O(n \log n) & a = b \\ O(n^{\log_b a}) & a > b \end{cases}$$

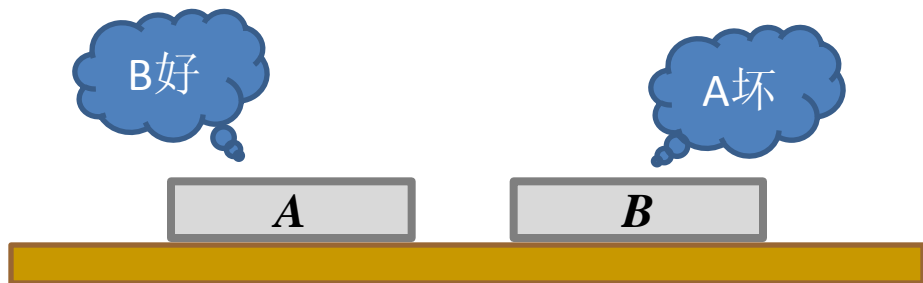
小结

- 分治算法的一般描述
 - 划分或归约为彼此独立的子问题
 - 分别求解每个子问题
 - 给出递归或迭代计算的终止条件
 - 如何由子问题的解得到原问题解
- 分治算法的分析方法
 - 求解时间复杂度的递推方程
 - 常用的递推方程的解

芯片测试

一次测试过程

测试方法：将2片芯片（*A*和*B*）置于测试台上，互相进行测试，测试报告是“好”或“坏”，只取其一。



假设：好芯片的报告一定是正确的，坏芯片的报告是不确定的（可能会出错）

测试结果分析

<i>A</i> 报告	<i>B</i> 报告	结论
<i>B</i> 是好的	<i>A</i> 是好的	<i>A, B</i> 都好或 <i>A, B</i> 都坏
<i>B</i> 是好的	<i>A</i> 是坏的	至少一片是坏的
<i>B</i> 是坏的	<i>A</i> 是好的	至少一片是坏的
<i>B</i> 是坏的	<i>A</i> 是坏的	至少一片是坏的

问题

输入：

n 片芯片，其中好芯片至少比坏芯片多 1 片。

问题：

设计一种测试方法，通过测试从 n 片芯片中挑出 1 片好芯片。

要求：使用最少的测试次数。

判定芯片A的好坏

问题：给定芯片A，判定A的好坏

方法：用其他 $n-1$ 片芯片对 A 测试。

$n=7$ ：好芯片数 ≥ 4 。

A 好，6个报告中至少 3 个报“好”

A 坏，6个报告中至少 4 个报“坏”

n 是**奇数**：好芯片数 $\geq (n+1)/2$ 。

A 好，至少有 $(n-1)/2$ 个报“好”

A 坏，至少有 $(n+1)/2$ 个报告“坏”

结论：至少一半报“好”，A是好芯片，
超过一半报“坏”，A是坏芯片。

判定芯片A的好坏

$n=8$: 好芯片数 ≥ 5 .

A 好, 7个报告中至少 4 个报“好”

A 坏, 7个报告中至少 5 个报“坏”

n 是偶数: 好芯片数 $\geq n/2+1$.

A 好, 至少有 $n/2$ 个报告“好”

A 坏, 至少有 $n/2+1$ 个报告“坏”

结论: $n-1$ 份报告中,
至少一半报“好”, 则 A 为好芯片
超过一半报“坏”, 则 A 为坏芯片

蛮力算法

测试方法：任取 1 片测试，如果是好芯片，测试结束；如果是坏芯片，抛弃，再从剩下芯片中任取 1 片测试，直到得到 1 片好芯片。

时间估计：

第 1 片坏芯片，最多测试 $n-2$ 次，

第 2 片坏芯片，最多测试 $n-3$ 次，

...

总计 $\Theta(n^2)$

分治算法设计思想

假设 n 为偶数，将 n 片芯片两两一组做测试淘汰，剩下芯片构成子问题，进入下一轮分组淘汰。

淘汰规则：

"好, 好" \Rightarrow 任留 1 片，进入下轮
其他情况 \Rightarrow 全部抛弃

递归截止条件： $n \leq 3$

3 片芯片，1 次测试可得到好芯片。

1 或 2 片芯片，不再需要测试。

分治算法的正确性

命题1 当 n 是偶数时, 在上述淘汰规则下, 经过一轮淘汰, 剩下的好芯片比坏芯片至少多1片.

证 设 A, B 都好的芯片 i 组, A 与 B 一好一坏 j 组, A 与 B 都坏的 k 组. 淘汰后好芯片至少 i 片, 坏芯片至多 k 片.

$$2i + 2j + 2k = n \quad \text{初始芯片总数}$$

$$2i + j > 2k + j \quad \text{初始好芯片多于坏芯片}$$

→ $i > k$

n 为奇数时的特殊处理

当 n 是奇数时，可能出问题

输入：

好	好	好	好	坏	坏	坏
---	---	---	---	---	---	---

分组：

好	好	好	好	坏	坏	坏
---	---	---	---	---	---	---

淘汰后：

好	好	坏	坏
---	---	---	---

处理办法：当 n 为奇数时，增加一轮对轮空芯片的单独测试。

如果该芯片为好芯片，算法结束；
如果是坏芯片，则淘汰该芯片。

伪码描述

算法 Test(n)

1. $k \leftarrow n$
2. while $k > 3$ do
3. 将芯片分成 $\lfloor k/2 \rfloor$ 组 // 轮空处理
4. for $i = 1$ to $\lfloor k/2 \rfloor$ do
5. if 2片好 then 则任取1片留下
6. else 2片同时丢掉
7. $k \leftarrow$ 剩下的芯片数
8. if $k = 3$ then
9. 任取2片芯片测试
10. if 1好1坏 then 取没测的芯片
11. else 任取1片被测芯片
- 12 if $k = 2$ or 1 then 任取1片

分组
淘汰

递归
结束

时间复杂度分析

设输入规模为 n

每轮淘汰后，芯片数至少减半

测试次数(含轮空处理): $O(n)$

时间复杂度:

$$W(n) = W(n/2) + O(n)$$

$$W(3) = 1, W(2) = W(1) = 0$$

解得 $W(n) = O(n)$

小结

- 芯片测试的分治算法

如何保证子问题与原问题性质相同:
增加额外处理

额外处理的工作量不改变函数的阶
时间复杂度为 $O(n)$

快速排序

基本思想

- 用首元素 x 作划分标准，将输入数组 A 划分成不超过 x 的元素构成的数组 A_L ，大于 x 的元素构成的数组 A_R 。其中 A_L, A_R 从左到右存放在数组 A 的位置。
- 递归地对子问题 A_L 和 A_R 进行排序，直到子问题规模为 1 时停止。

伪码

算法 Quicksort (A, p, r)

输入：数组 $A[p..r]$

输出：排好序的数组 A

1. if $p < r$
2. then $q \leftarrow \text{Partition}(A, p, r)$
3. $A[p] \leftrightarrow A[q]$
4. Quicksort ($A, p, q-1$)
5. Quicksort ($A, q+1, r$)

初始置 $p=1, r=n$ ，然后调用上述算法

划分过程

Partition (A, p, r)

- 1. $x \leftarrow A[p]$**
- 2. $i \leftarrow p$**
- 3. $j \leftarrow r + 1$**
- 4. while true do**
 - 5. repeat $j \leftarrow j - 1$**
 - 6. until $A[j] \leq x$ // 不超过首元素的**
 - 7. repeat $i \leftarrow i + 1$**
 - 8. until $A[i] > x$ // 比首元素大的**
 - 9. if $i < j$**
 - 10. then $A[i] \leftrightarrow A[j]$**
 - 11. else return j**

划分实例

27 **99** 0 8 13 64 86 16 7 10 88 **25** 90
i *j*

27 25 0 8 13 **64** 86 16 7 **10** 88 99 90
i *j*

27 25 0 8 13 10 **86** 16 **7** 64 88 99 90
i *j*

27 25 0 8 13 10 7 **16** **86** 64 88 99 90
j *i*

16 25 0 8 13 10 7 **27** 86 64 88 99 90

时间复杂度

最坏情况: $W(n) = W(n-1) + n - 1$

$$W(1) = 0$$

$$W(n) = n(n-1)/2$$

最好划分: $T(n) = 2 T(n/2) + n - 1$

$$T(1) = 0$$

$$T(n) = \Theta(n \log n)$$

均衡划分的时间复杂度

均衡划分：子问题的规模比不变
例如为 1:9

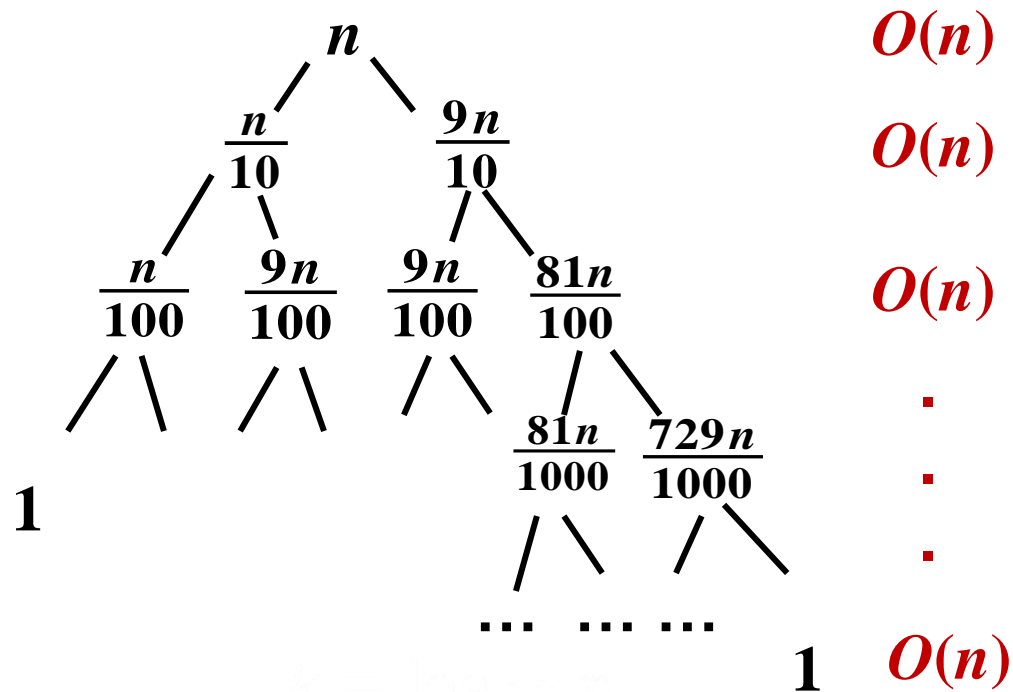
$$T(n) = T(n/10) + T(9n/10) + n$$

$$T(1) = 0$$

根据递归树，时间复杂度

$$T(n) = \Theta(n \log n)$$

递归树



$$T(n) = O(n \log n)$$

平均时间复杂度

首元素排好序后处在 $1, 2, \dots, n$

各种情况概率都为 $1/n$

首元素在位置 1: $T(0), T(n-1)$

首元素在位置 2: $T(1), T(n-2)$

....

首元素在位置 $n-1$: $T(n-2), T(1)$

首元素在位置 n : $T(n-1), T(0)$

子问题工作量 $2[T(1)+T(2)+\dots+T(n-1)]$

划分工作量 $n-1$

平均时间复杂度

$$T(n) = \frac{1}{n} \sum_{k=1}^{n-1} (T(k) + T(n-k)) + n - 1$$

$$T(n) = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + n - 1$$

$$T(1) = 0$$

$$T(n) = \Theta(n \log n)$$

首元素划分后每个位置概率相等

小结

快速排序算法

- 分治策略
- 子问题划分是由首元素决定
- 最坏情况下时间 $O(n^2)$
- 平均情况下时间为 $O(n\log n)$

幂乘算法及应用

幂乘问题

输入： a 为给定实数， n 为自然数

输出： a^n

传统算法： 顺序相乘

$$a^n = (\dots(((a \ a)a)a)\dots)a$$

乘法次数： $\Theta(n)$

分治算法——划分

n 为偶数 $\underbrace{a \dots a}_{n/2 \text{ 个}} \mid \underbrace{a \dots a}_{n/2 \text{ 个}}$

n 为奇数 $\underbrace{a \dots a}_{(n-1)/2 \text{ 个}} \mid \underbrace{a \dots a}_{(n-1)/2 \text{ 个}} / a$

$$a^n = \begin{cases} a^{n/2} \times a^{n/2} & n \text{ 为偶数} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & n \text{ 为奇数} \end{cases}$$

分治算法分析

以乘法作为基本运算

- 子问题规模：不超过 $n/2$
- 两个规模近似 $n/2$ 的子问题完全一样，只要计算1次

$$W(n) = W(n/2) + \Theta(1)$$

$$W(n) = \Theta(\log n)$$

幂乘算法的应用

Fibonacci数列: 1, 1, 2, 3, 5, 8, 13, 21, ...

增加 $F_0=0$, 得到数列

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

问题: 已知 $F_0=0, F_1=1$, 给定 n , 计算 F_n .

通常算法: 从 F_0, F_1, \dots 开始, 根据递推公式

$$F_n = F_{n-1} + F_{n-2}$$

陆续相加可得 F_n , 时间复杂度为 $\Theta(n)$

Fibonacci数的性质

定理1 设 $\{F_n\}$ 为 Fibonacci 数构成的数列，那么

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

归纳证明

$$n=1, \text{ 左边} = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \text{右边}$$

Fibonacci数的性质(续)

假设对任意正整数 n , 命题成立, 即

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

那么

$$\begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1}$$

归纳假
设代入

算法

令矩阵 $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, 用乘幂算法计算 M^n

时间复杂度:

- 矩阵乘法次数 $T(n) = \Theta(\log n)$
- 每次矩阵乘法需要做 8 次元素相乘
- 总计元素相乘次数为 $\Theta(\log n)$

小结

- 分治算法的例子——幂乘算法
- 幂乘算法的应用
 - 计算Fibonacci数
 - 通常算法 $O(n)$ ，分治算法为 $O(\log n)$

改进分治算法的途径

1: 减少子问题数

减少子问题个数的依据

分治算法的时间复杂度方程

$$W(n) = aW(n/b) + d(n)$$

a : 子问题数, n/b : 子问题规模,

$d(n)$: 划分与综合工作量.

当 a 较大, b 较小, $d(n)$ 不大时, 方程的解:

$$\underline{W(n) = \Theta(n^{\log_b a})}$$

减少 a 是降低函数 $W(n)$ 的阶的途径.

利用子问题的依赖关系, 使某些子问题的解通过组合其他子问题的解而得到.

例1：整数位乘问题

输入： X, Y 是 n 位二进制数， $n = 2^k$

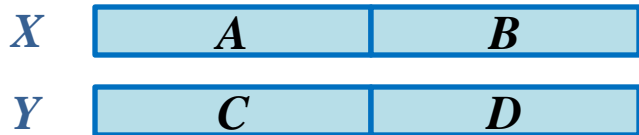
输出： XY

普通乘法： 需要 $O(n^2)$ 次位乘运算

简单划分： 令

$$X = A2^{n/2} + B, \quad Y = C2^{n/2} + D.$$

$$XY = \underline{AC} 2^n + (\underline{AD} + \underline{BC}) 2^{n/2} + \underline{BD}$$



$$W(n) = 4W(n/2) + O(n) \Rightarrow W(n) = O(n^2)$$

减少子问题个数

子问题间的依赖关系：代数变换

$$AD+BC = (\underline{A-B})(\underline{D-C}) + \underline{AC} + \underline{BD}$$

算法复杂度

$$W(n) = 3 W(n/2) + cn$$

$$W(1) = 1$$

方程的解

$$W(n) = O(n^{\log 3}) = \mathbf{O(n^{1.59})}$$

例2：矩阵相乘问题

输入： A, B 为 n 阶矩阵， $n = 2^k$

输出： $C = AB$

通常矩阵乘法：

C 中有 n^2 个元素

每个元素需要做 n 次乘法

以元素相乘为基本运算

$$W(n) = O(n^3)$$

简单分治算法

分治法 将矩阵分块，得

$$\begin{pmatrix} \boxed{A_{11}} & \boxed{A_{12}} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \boxed{B_{11}} & B_{12} \\ \boxed{B_{21}} & B_{22} \end{pmatrix} = \begin{pmatrix} \boxed{C_{11}} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

其中

$$C_{11} = \underline{A_{11}B_{11}} + \underline{A_{12}B_{21}} \quad C_{12} = \underline{A_{11}B_{12}} + \underline{A_{12}B_{22}}$$

$$C_{21} = \underline{A_{21}B_{11}} + \underline{A_{22}B_{21}} \quad C_{22} = \underline{A_{21}B_{12}} + \underline{A_{22}B_{22}}$$

递推方程 $W(n) = 8 W(n/2) + cn^2$

$$W(1) = 1$$

解

$$W(n) = \mathbf{O(n^3)}.$$

Strassen 矩阵乘法

变换方法:

设计 M_1, M_2, \dots, M_7 , 对应7个子问题

$$M_1 = A_{11} (B_{12} - B_{22})$$

$$M_2 = (A_{11} + A_{12}) B_{22}$$

$$M_3 = (A_{21} + A_{22}) B_{11}$$

$$M_4 = A_{22} (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{22}) (B_{11} + B_{22})$$

$$M_6 = (A_{12} - A_{22}) (B_{21} + B_{22})$$

$$M_7 = (A_{11} - A_{21}) (B_{11} + B_{12})$$

Strassen 矩阵乘法 (续)

利用中间矩阵，得到结果矩阵

$$C_{11} = M_5 + M_4 - M_2 + M_6$$

$$C_{12} = M_1 + M_2$$

$$C_{21} = M_3 + M_4$$

$$C_{22} = M_5 + M_1 - M_3 - M_7$$

时间复杂度函数：

$$W(n) = 7 W(n/2) + 18(n/2)^2$$

$$W(1) = 1$$

解 $W(n) = O(n^{\log 7}) = O(n^{2.8075})$

矩阵乘法的研究及应用

矩阵乘法问题的难度：

- Coppersmith–Winograd算法： $O(n^{2.376})$
目前为止最好的上界
- 目前为止最好的下界是： $\Omega(n^2)$

应用：

- 科学计算、图像处理、数据挖掘等
- 回归、聚类、主成分分析、决策树等挖掘算法常涉及大规模矩阵运算

改进途径小结

- 适用于：子问题个数多，划分和综合工作量不太大，时间复杂度函数

$$W(n) = \Theta(n^{\log_b a})$$

- 利用子问题依赖关系，用某些子问题解的代数表达式表示另一些子问题的解，减少独立计算子问题个数.
- 综合解的工作量可能会增加，但增加的工作量不影响 $W(n)$ 的阶.

改进分治算法的途径

径2：增加预处理

例子：平面点对问题

输入：平面点集 P 中有 n 个点, $n > 1$

输出： P 中的两个点，其距离最小

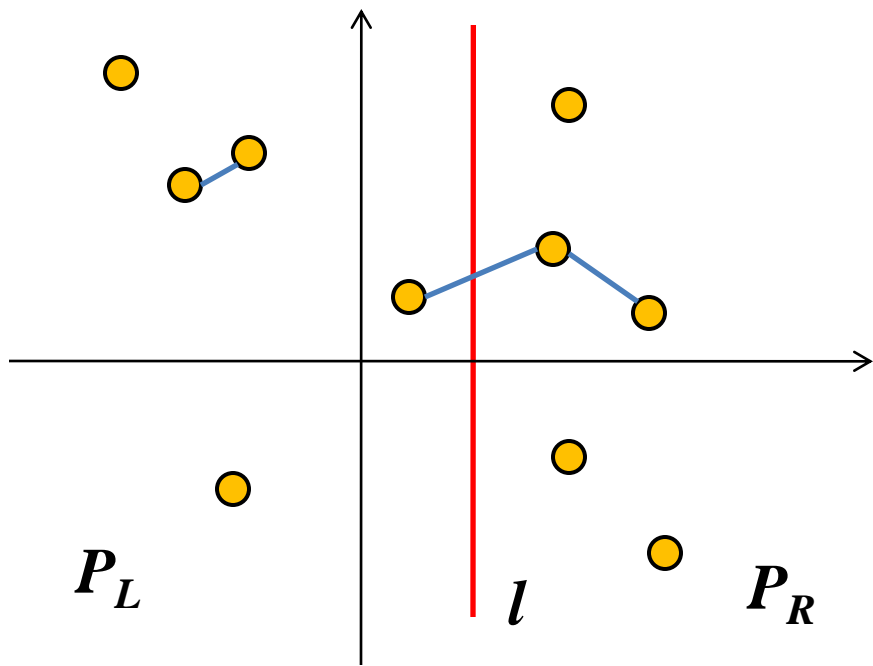
蛮力算法：

$C(n, 2)$ 个点对, 计算最小距离, $O(n^2)$

分治策略： P 划为大小相等的 P_L 和 P_R

1. 分别计算 P_L 、 P_R 中最近点对
2. 计算 P_L 与 P_R 中各一个点的最近点对
3. 上述情况下的最近点对是解

划分实例： $n=10$



算法伪码

MinDistance (P, X, Y)

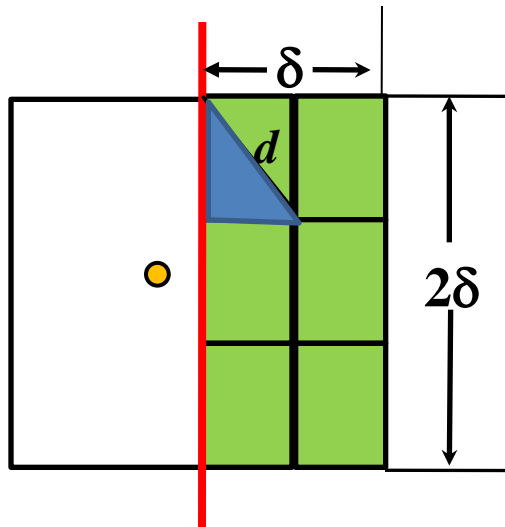
输入: 点集 P , X 和 Y 为横、纵坐标数组

输出: 最近的两个点及距离

1. 若 $|P| \leq 3$, 直接计算其最小距离
2. 排序 X, Y
3. 做中垂线 l 将 P 划分为 P_L 和 P_R
4. MinDistance (P_L, X_L, Y_L)
5. MinDistance (P_R, X_R, Y_R)
6. $\delta = \min(\delta_L, \delta_R)$ // δ_L, δ_R 为子问题的距离
7. 检查距 l 不超过 δ 两侧各1个点的距离. 若小于 δ , 修改 δ 为这个值

跨边界处理

$$\begin{aligned}d &= \sqrt{(\delta/2)^2 + (2\delta/3)^2} \\&= \sqrt{\delta^2/4 + 4\delta^2/9} \\&= \sqrt{25\delta^2/36} = 5\delta/6\end{aligned}$$



右边每个小方格至多1个点，每个点
至多比较对面的6个点，检查1个点是
常数时间， $O(n)$ 个点需要 $O(n)$ 时间

算法分析

步1 递归边界处理: $O(1)$

步2 排序: $O(n\log n)$

步3 划分: $O(1)$

步4-5子问题: $2T(n/2)$

步6确定 δ : $O(1)$

步7检查跨边界点对: $O(n)$

$$T(n) = 2T(n/2) + O(n\log n)$$

$$T(n) = O(1), n \leq 3$$

递归树求解 $T(n) = O(n\log^2 n)$

增加预处理

原算法:

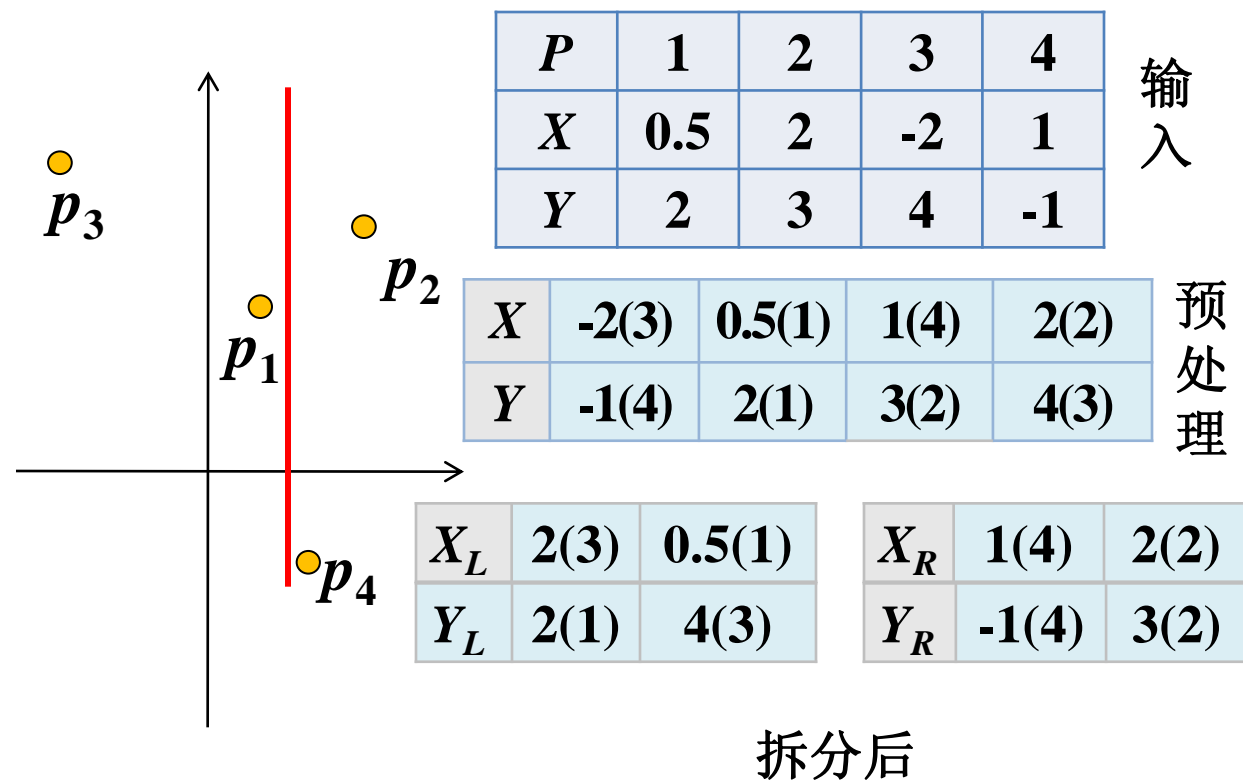
在每次划分时对子问题数组重新排序

改进算法:

1. 在递归前对 X, Y 排序, 作为预处理
2. 划分时对排序的数组 X, Y 进行拆分, 得到针对子问题 P_L 的数组 X_L, Y_L 及针对子问题 P_R 的数组 X_R, Y_R

原问题规模为 n , 拆分的时间为 $O(n)$

实例：递归中的拆分



改进算法时间复杂度

$W(n)$ 为算法时间复杂度

递归过程: $T(n)$, 预处理: $O(n\log n)$

$$W(n) = T(n) + O(n\log n)$$

$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = O(1) \quad n \leq 3$$

解得 $T(n) = O(n\log n)$

于是 $W(n) = O(n\log n)$

小结

- 依据

$$W(n) = aW(n/b) + f(n)$$

- 提高算法效率的方法：

- 减少子问题个数 a ：

$$W(n) = O(n^{\log_b a})$$

- 增加预处理，减少 $f(n)$

选最大与最小

选择问题

输入：集合 L (含 n 个不等的实数)

输出： L 中第 i 小元素

$i=1$, 称为最小元素

$i=n$, 称为最大元素

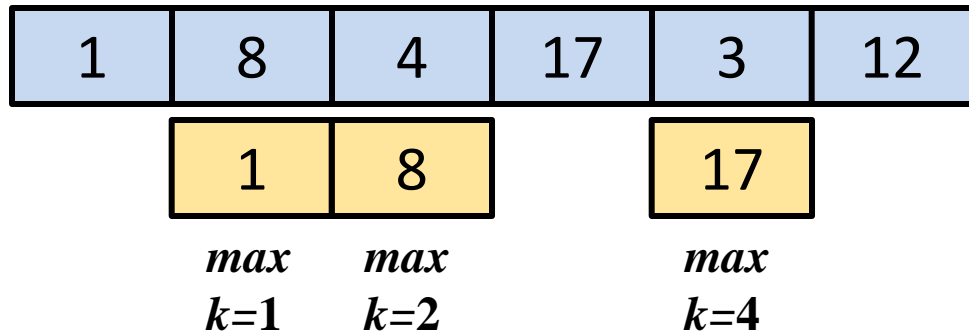
位置处在中间的元素，称为中位元素

n 为奇数，中位数唯一， $i = (n+1)/2$

n 为偶数，可指定 $i = n/2+1$

选最大

算法：顺序比较



输出： $max = 17$, $k=4$

算法最坏情况下的时间 $W(n)=n-1$

伪码

算法 Findmax

输入: n 个数的数组 L

输出: max, k

1. $max \leftarrow L[1]$

2. for $i \leftarrow 2$ to n do

3. if $max < L[i]$

4. then $max \leftarrow L[i]$

5. $k \leftarrow i$

6. return max, k

选最大最小

通常算法:

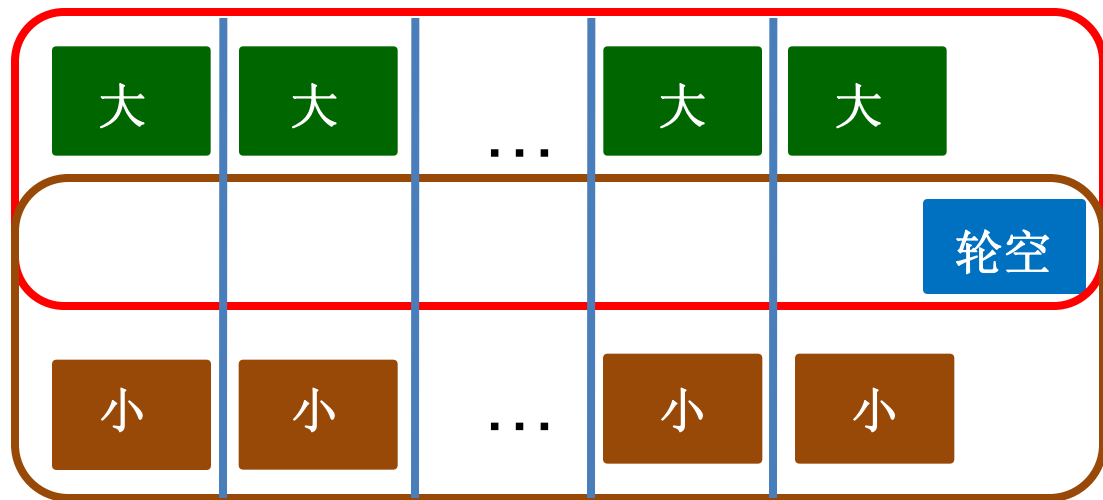
1. 顺序比较, 先选最大 max
2. 顺序比较, 在剩余数组中选最小 min , 类似于选最大算法, 但比较时保留较小的数

时间复杂性:

$$W(n) = n-1 + n-2 = 2n-3$$

分组算法

组1 组2 ... 组 $\lfloor n/2 \rfloor$



伪码

算法 FindMaxMin

输入： n 个数的数组 L

输出： max , min

1. 将 n 个元素两两一组分成 $\lfloor n/2 \rfloor$ 组
2. 每组比较，得到 $\lfloor n/2 \rfloor$ 个较小和 $\lfloor n/2 \rfloor$ 个较大
3. 在 $\lceil n/2 \rceil$ 个较大（含轮空元素）中找最大 max
4. 在 $\lceil n/2 \rceil$ 个较小（含轮空元素）中找最小 min

最坏情况时间复杂度

行2 的组内比较: $\lfloor n/2 \rfloor$ 次

行3--4 求 max 和 min 比较:

至多 $2\lceil n/2 \rceil - 2$ 次

$$W(n) = \lfloor n/2 \rfloor + 2\lceil n/2 \rceil - 2$$

$$= n + \lceil n/2 \rceil - 2$$

$$= \lceil 3n/2 \rceil - 2$$

分治算法

1. 将数组 L 从中间划分为两个子数组 L_1 和 L_2
2. 递归地在 L_1 中求最大 max_1 和 min_1
3. 递归地在 L_2 中求最大 max_2 和 min_2
4. $max \leftarrow \max\{max_1, max_2\}$
5. $min \leftarrow \min\{min_1, min_2\}$

最坏情况时间复杂度

假设 $n = 2^k$,

$$W(n) = 2W(n/2) + 2$$

$$W(2) = 1$$

解
$$\begin{aligned} W(2^k) &= 2W(2^{k-1}) + 2 \\ &= 2[2W(2^{k-2}) + 2] + 2 \\ &= 2^2W(2^{k-2}) + 2^2 + 2 = \dots \\ &= 2^{k-1} + 2^{k-1} + \dots + 2^2 + 2 \\ &= 3 \cdot 2^{k-1} - 2 = 3n/2 - 2 \end{aligned}$$

选择算法小结

选最大：顺序比较, 比较次数 $n-1$

选最大最小

- 选最大+ 选最小, 比较次数 $2n-3$
- 分组： 比较次数 $\lceil 3n/2 \rceil - 2$
- 分治： $n=2^k$, 比较次数 $3n/2-2$

选第二大

选第二大

输入： n 个数的数组 L

输出： 第二大的数 $second$

通常算法： 顺序比较

1. 顺序比较找到最大 max
2. 从剩下 $n - 1$ 个数中找最大，就是第二大 $second$

时间复杂度：

$$W(n) = n - 1 + n - 2 = 2n - 3$$

提高效率的途径

- 成为第二大数的条件：仅在与最大数的比较中被淘汰.
- 要确定第二大数，必须知道最大数.
- 在确定最大数的过程中记录下被最大数直接淘汰的元素.
- 在上述范围（被最大数直接淘汰的数）内的最大数就是第二大数.
- 设计思想： 用空间换时间.

锦标赛算法

1. 两两分组比较，大者进入下一轮，直到剩下 1 个元素 *max* 为止
2. 在每次比较中淘汰较小元素，将被淘汰元素记录在淘汰它的元素的链表上
3. 检查 *max* 的链表，从中找到最大元，即 *second*

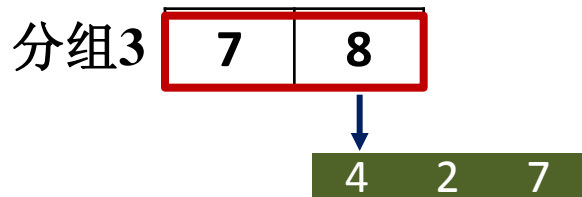
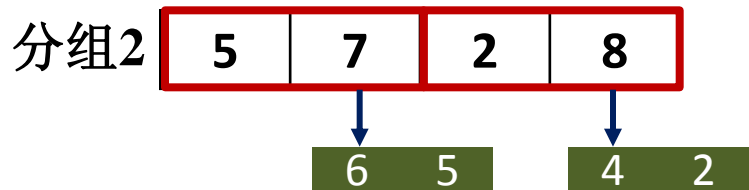
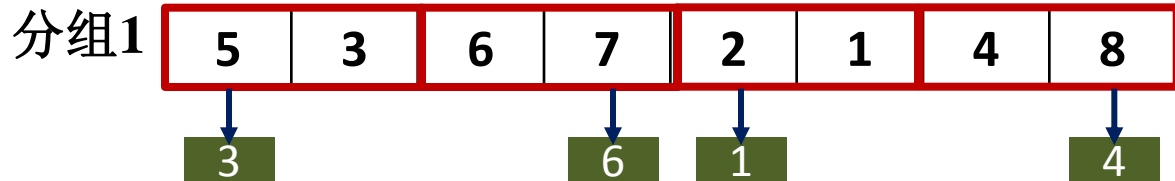
伪码

算法 FindSecond

输入: n 个数的数组 L , 输出: $second$

1. $k \leftarrow n$ // 参与淘汰的元素数
 2. 将 k 个元素两两1组, 分成 $\lfloor k/2 \rfloor$ 组
 3. 每组的2个数比较, 找到较大数
 4. 将被淘汰数记入较大数的链表
 5. if k 为奇数 then $k \leftarrow \lfloor k/2 \rfloor + 1$
 6. else $k \leftarrow \lfloor k/2 \rfloor$
 7. if $k > 1$ then goto 2
 8. $max \leftarrow$ 最大数
 9. $second \leftarrow max$ 的链表中的最大
- 一轮淘汰
- 继续分组淘汰

实例



时间复杂度分析

命题1 设参与比较的有 t 个元素，经过 i 轮淘汰后元素数至多为 $\lceil t/2^i \rceil$.

证 对 i 归纳. $i=1$, 分 $\lfloor t/2 \rfloor$ 组，淘汰 $\lfloor t/2 \rfloor$ 个元素，进入下一轮元素数是 $t - \lfloor t/2 \rfloor = \lceil t/2 \rceil$

假设 i 轮分组淘汰后元素数至多为 $\lceil t/2^i \rceil$ ，那么 $i+1$ 轮分组淘汰后元素数为

$$\lceil \lceil t/2^i \rceil / 2 \rceil = \lceil t/2^{i+1} \rceil$$

时间复杂度分析（续）

命题2 max 在第一阶段分组比较中总计进行了 $\lceil \log n \rceil$ 次比较.

证 假设到产生 max 时总计进行 k 轮淘汰, 根据命题 1 有 $\lceil n/2^k \rceil = 1$.

若 $n=2^d$, 那么有

$$k = d = \log n = \lceil \log n \rceil$$

若 $2^d < n < 2^{d+1}$, 那么

$$k = d + 1 = \lceil \log n \rceil$$

时间复杂度分析（续）

第一阶段元素数： n

比较次数： $n-1$

淘汰了 $n-1$ 个元素

第二阶段：元素数 $\lceil \log n \rceil$

比较次数： $\lceil \log n \rceil - 1$

淘汰元素数为 $\lceil \log n \rceil - 1$

时间复杂度是

$$\begin{aligned} W(n) &= n - 1 + \lceil \log n \rceil - 1 \\ &= n + \lceil \log n \rceil - 2. \end{aligned}$$

小结

求第二大算法

- 调用2次找最大: $2n-3$
 - 锦标赛算法: $n + \lceil \log n \rceil - 2$
- 主要的技术: 用空间换时间

一般选择问题 的算法设计

一般性选择问题

问题：选第 k 小.

输入：数组 S , S 的长度 n , 正整数 k ,
 $1 \leq k \leq n$.

输出：第 k 小的数

实例 1

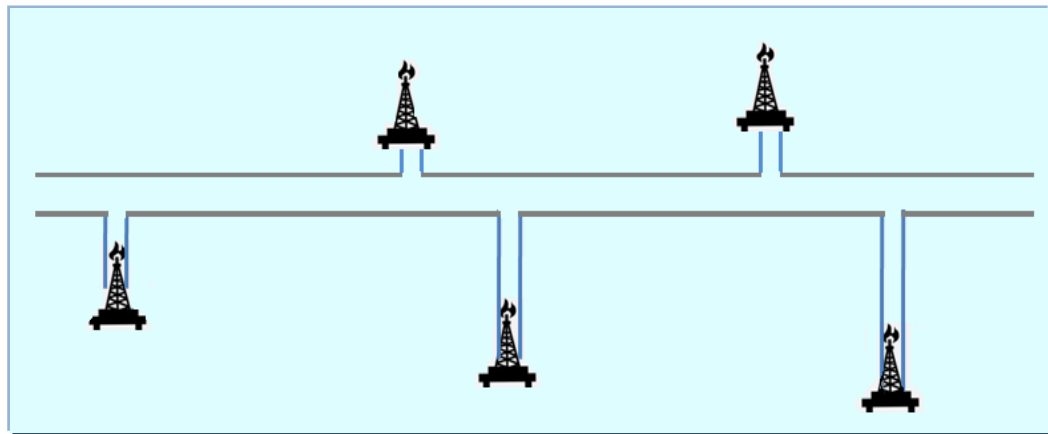
$S = \{ 3, 4, 8, 2, 5, 9, 18 \}$, $k = 4$, 解: 5

实例 2

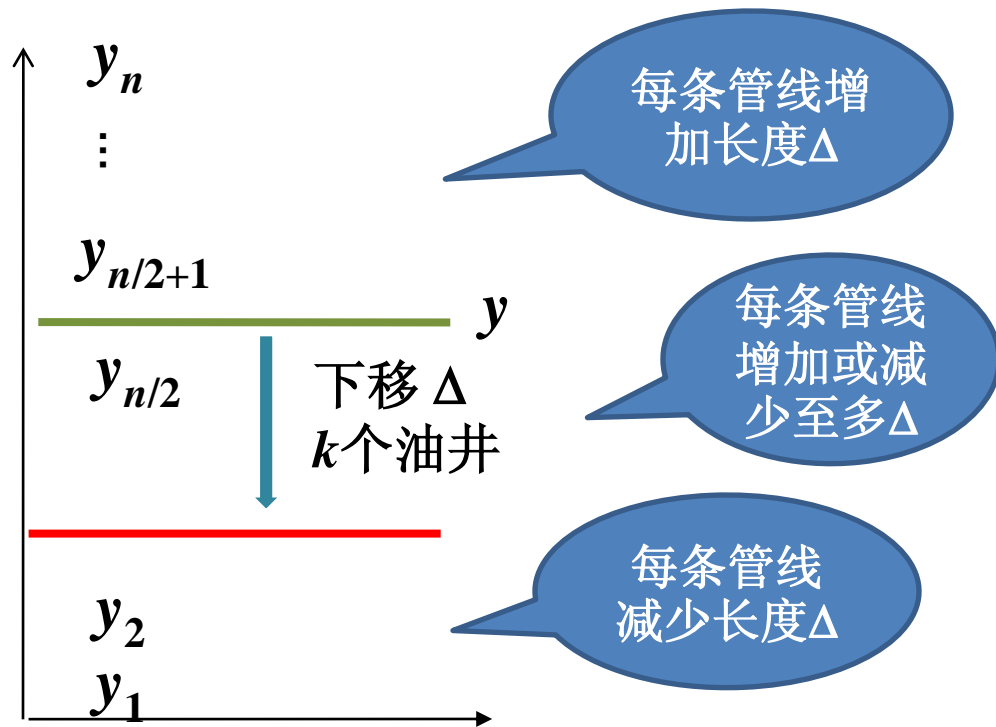
统计数据的集合 S , $|S|=n$,
选中位数, $k = \lceil n/2 \rceil$

一个应用：管道位置

问题：某区域有 n 口油井，需要修建输油管道. 根据设计要求，水平方向有一条主管道，每口油井修一条垂直方向的支管道通向主管道. 如何选择主管道的位置，以使得支管道长度的总和最小？



最优解: Y 坐标的中位数



下移后支管线总长度增加

简单的算法

算法一：

调用 k 次选最小算法

时间复杂度为 $O(kn)$

算法二：

先排序，然后输出第 k 小的数

时间复杂度为 $O(n \log n)$

分治算法

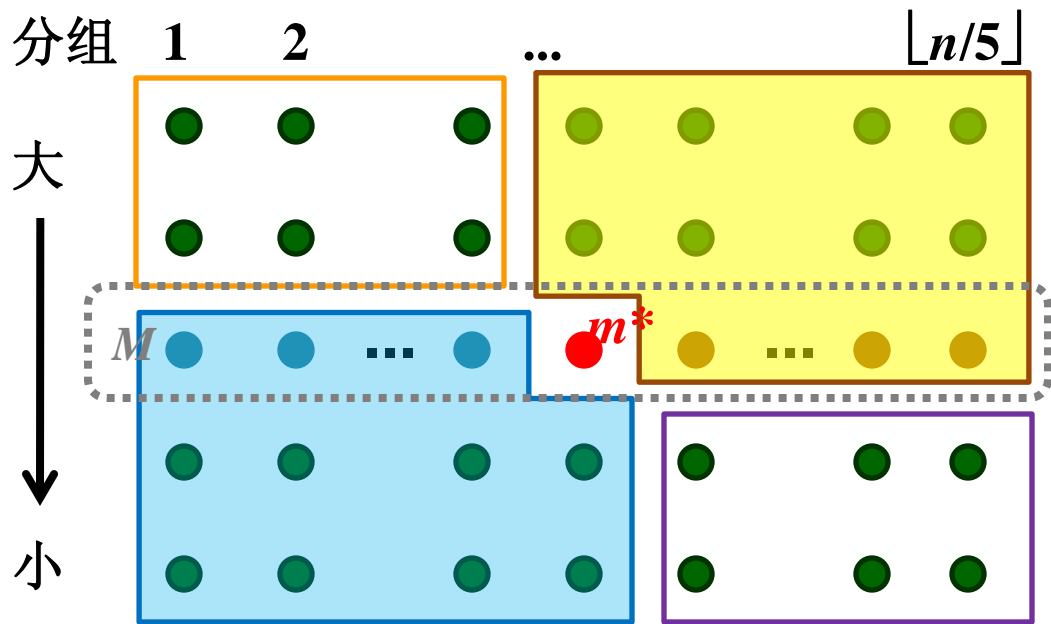
假设元素彼此不等，设计思想：

1. 用**某个元素 m^*** 作为标准将 S 划分成 S_1 与 S_2 ，其中 S_1 的元素小于 m^* ， S_2 的元素大于等于 m^* .
2. 如果 $k \leq |S_1|$ ，则在 S_1 中找第 k 小。
如果 $k = |S_1| + 1$ ，则 m^* 是第 k 小
如果 $k > |S_1| + 1$ ，则在 S_2 中找第 $k - |S_1| - 1$ 小



算法效率取决于子问题规模，
如何通过 m^* 控制子问题规模？

m^* 的选择与划分过程



A: 数需要与 m^* 比大小, **B**: 数大于 m^*

C: 数小于 m^* , **D**: 数需要与 m^* 比大小

实例: $n=15, k=6$

8	2	3	5	7	6	11	14	1	9	13	10	4	12	15
---	---	---	---	---	---	----	----	---	---	----	----	---	----	----

	8	14	15	
	7	11	13	
M	5	9	12	$m^* = 9$
	3	6	10	
	2	1	4	

	8	14	15	
A	7	11	13	B
	5	9	12	
C	3	6	10	D
	2	1	4	

8, 7, 10, 4 需要与9比较

归约为子问题

S_1	8	14	15	S_2
	7	11	13	
	5	9	12	
	3	6	10	
	2	1	4	

子问题

8 7 5 3 2 6 1 4

子问题规模 = 8, $k=6$

伪码

算法 Select (S, k)

输入：数组 S ，正整数 k ，

输出： S 中的第 k 小元素

1. 将 S 分 5 个一组，共 $n_M = \lceil n/5 \rceil$ 组
2. 每组排序，中位数放到集合 M
3. $m^* \leftarrow \text{Select}(M, \lceil |M|/2 \rceil)$ // S 分 A, B, C, D
4. A, D 元素小于 m^* 放 S_1 , 大于 m^* 放 S_2
5. $S_1 \leftarrow S_1 \cup C; S_2 \leftarrow S_2 \cup B$ 划分
6. if $k = |S_1| + 1$ then 输出 m^*
7. else if $k \leq |S_1|$ \Leftarrow 递归计算子问题
8. then Select (S_1, k)
9. else Select ($S_2, k - |S_1| - 1$)

小结

选第 k 小的算法:

- 分治策略
- 确定 m^*
- 用 m^* 划分数组归约为子问题
- 递归实现

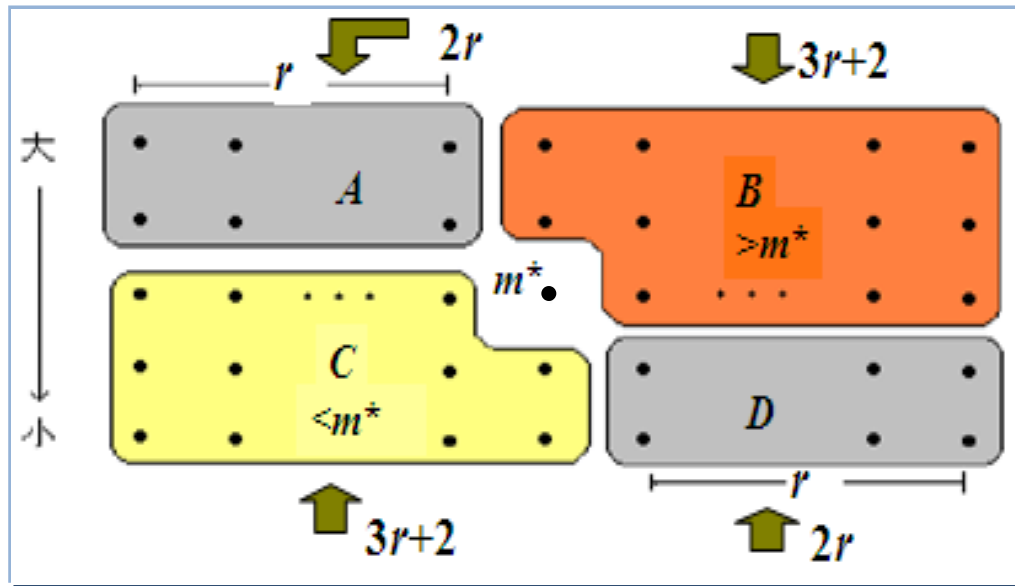
选择问题的 算法分析

伪码

算法 **Select** (S, k)

1. 将 S 分5个一组, 共 $n_M = \lceil n/5 \rceil$ 组
2. 每组排序, 中位数放到集合 M
3. $m^* \leftarrow \text{Select}(M, \lceil |M|/2 \rceil)$ // S 分 A, B, C, D
4. A, D 元素小于 m^* 放 S_1 , 大于 m^* 放 S_2
5. $S_1 \leftarrow S_1 \cup C; S_2 \leftarrow S_2 \cup B$
6. if $k = |S_1| + 1$ then 输出 m^*
7. else if $k \leq |S_1|$
8. then **Select** (S_1, k)
9. else **Select** ($S_2, k - |S_1| - 1$)

用 m^* 划分



$$n = 5(2r + 1), \quad |A| = |D| = 2r$$

子问题规模至多: $2r + 2r + 3r + 2 = 7r + 2$

子问题规模估计

不妨设 $n = 5(2r + 1)$, $|A|=|D|=2r$,

$$r = \frac{n/5 - 1}{2} = \frac{n}{10} - \frac{1}{2}$$

划分后子问题规模至多为

$$\begin{aligned} \underline{7r + 2} &= 7\left(\frac{n}{10} - \frac{1}{2}\right) + 2 \\ &= \frac{7n}{10} - \frac{3}{2} < \frac{7n}{10} \end{aligned}$$

时间复杂度递推方程

算法工作量 $W(n)$

行2: $O(n)$ //每5个数找中位数,构成 M

行3: $W(n/5)$ // M 中找中位数 m^*

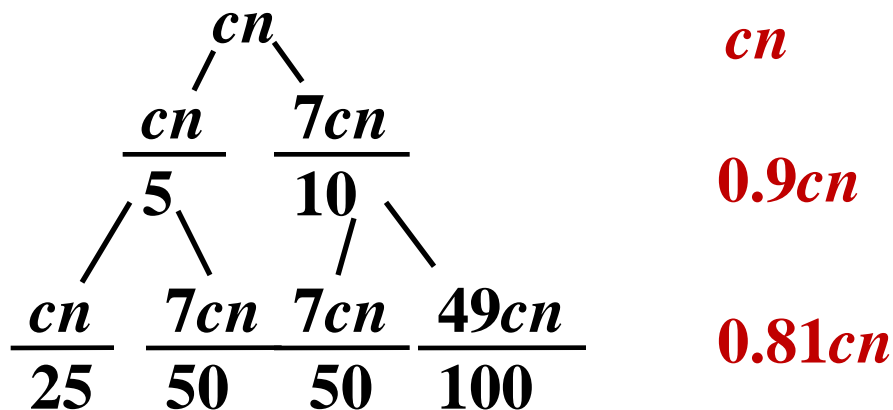
行4: $O(n)$ // 用 m^* 划分集合 S

行8-9: $W(7n/10)$ //递归

$$W(n) \leq W(n/5) + W(7n/10) + O(n)$$

递归树

$$W(n) = W(n/5) + W(7n/10) + cn$$



.....

$$W(n) \leq cn (1 + 0.9 + 0.9^2 + \dots) = O(n)$$

讨论



分组时为什么5个元素一组？

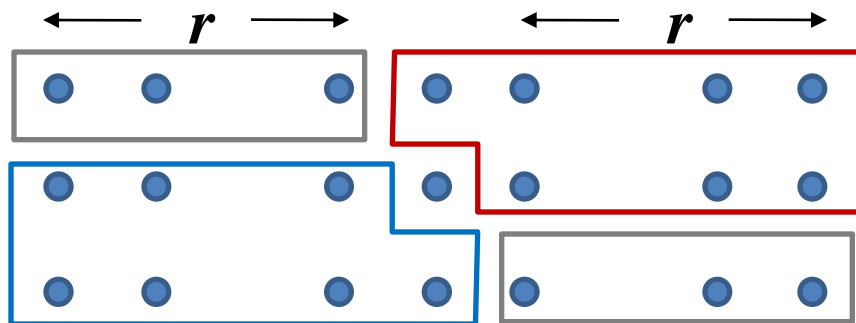
3个一组或 7个一组行不行？

分析：递归调用

1. 求 m^* 的工作量与 $|M| = n/t$ 相关, t 为每组元素数. t 大, $|M|$ 小
2. 归约后子问题大小与分组元素数 t 有关. t 大, 子问题规模大

3分组时的子问题规模

假设 $t=3$, 3个一组:



$$n = 3(2r + 1)$$

$$r = (n/3 - 1)/2 = n/6 - 1/2$$

子问题规模最多为 $4r+1 = 4n/6 - 1$

算法的时间复杂度

算法的时间复杂度满足方程

$$W(n) = W(n/3) + W(4n/6) + cn$$

由递归树得 $W(n) = \Theta(n \log n)$

关键：

$|M|$ 与归约后子问题规模之和小于 n ，
递归树每行的工作量构成公比小于 1
的等比级数， 算法复杂度才是 $O(n)$ 。

小结

选第 k 小算法的时间分析

- 递推方程
- 分组时每组元素数的多少对时间复杂度的影响

卷积及其应用

向量计算

给定向量

$$a = (a_0, a_1, \dots, a_{n-1})$$
$$b = (b_0, b_1, \dots, b_{n-1})$$

向量和

$$a+b = (a_0+b_0, a_1+b_1, \dots, a_{n-1}+b_{n-1})$$

内积

$$a \cdot b = a_0 b_0 + a_1 b_1 + \dots + a_{n-1} b_{n-1}$$

卷积

$$a * b = (c_0, c_1, \dots, c_{2n-2}), \text{ 其中}$$

$$c_k = \sum_{\substack{i+j=k \\ i,j < n}} a_i b_j, \quad k = 0, 1, \dots, 2n-2$$

卷积的含义

在下述矩阵中，每个斜线的项之和恰好是卷积中的各个分量

$$\begin{array}{ccccccc}
 & & ab_0 & ab_1 & \cdots & ab_{n-2} & ab_{n-1} \\
 a_0b & \cancel{a_0b_0} & a_0b_1 & \cdots & a_0b_{n-2} & \cancel{a_0b_{n-1}} & \\
 a_1b & a_1b_0 & a_1b_1 & \cdots & a_1b_{n-2} & a_1b_{n-1} & \\
 \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\
 a_{n-2}b & a_{n-2}b_0 & \cancel{a_{n-2}b_1} & \cdots & a_{n-2}b_{n-2} & \cancel{a_{n-2}b_{n-1}} & \\
 a_{n-1}b & \cancel{a_{n-1}b_0} & a_{n-1}b_1 & \cdots & \cancel{a_{n-1}b_{n-2}} & \cancel{a_{n-1}b_{n-1}} & \\
 & & & c_{2n-3} & c_{2n-2} & &
 \end{array}$$

Diagram illustrating the convolution of two sequences a and b . The matrix shows the products $a_i b_j$ arranged in a grid. Red diagonal lines indicate the summation paths for each output component c_k . The components c_0, c_1, \dots, c_{n-1} are shown above the first row, and c_{2n-3}, c_{2n-2} are shown below the last row. The terms $a_0b, a_1b, \dots, a_{n-1}b$ are shown to the left of the grid.

计算实例

向量 $a = (1, 2, 4, 3), \quad b = (4, 2, 8, 0)$

则 $a+b = (5, 4, 12, 3)$

$a \cdot b = (4, 4, 32, 0)$

$a * b = (4, 10, 28, 36, 38, 24, 0)$

	ab_0	ab_1	ab_2	ab_3
a_0b	1×4	1×2	1×8	1×0
a_1b	2×4	2×2	2×8	2×0
a_2b	4×4	4×2	4×8	4×0
a_3b	3×4	3×2	3×8	3×0

$$c_2 = 4 \times 4 + 2 \times 2 + 1 \times 8 = 28$$

卷积与多项式乘法

多项式乘法: $C(x) = A(x) B(x)$

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

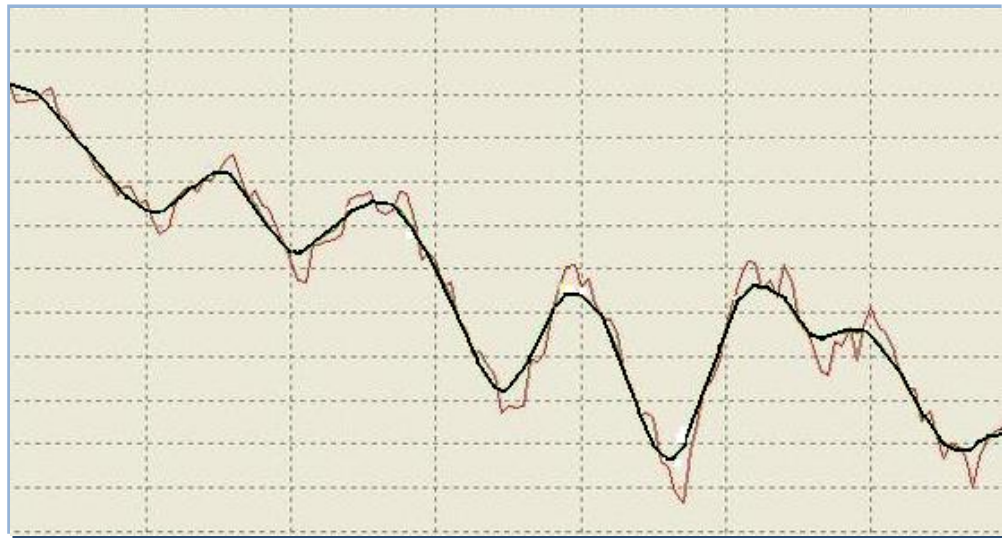
$$C(x) = \underline{a_0b_0 + (a_0b_1 + a_1b_0)x + \dots}$$
$$\underline{+ a_{m-1}b_{n-1}x^{m+n-2}}$$

其中 x^k 的系数

$$c_k = \sum_{\substack{i+j=k \\ i \in \{0,1,\dots,m-1\} \\ j \in \{0,1,\dots,n-1\}}} a_i b_j, \quad k = 0, 1, \dots, m+n-2$$

卷积应用：信号平滑处理

由于噪音干扰，对信号需要平滑处理

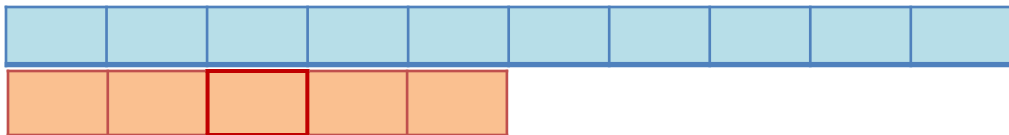


平滑处理

信号向量: $a=(a_0, a_1, \dots, a_{m-1})$

$b=(b_{2k}, b_{2k-1}, \dots, b_0) = (w_{-k}, \dots, w_k)$

$$a_i' = \sum_{s=-k}^k a_{i+s} b_{k-s} = \sum_{s=-k}^k a_{i+s} w_s$$



把向量 b 看作 $2k+1$ 长度窗口在 a 上移动计算 $a*b$, 得到 $(a_0', a_1', \dots, a_{m-1}')$. 有少数项有误差.

实例

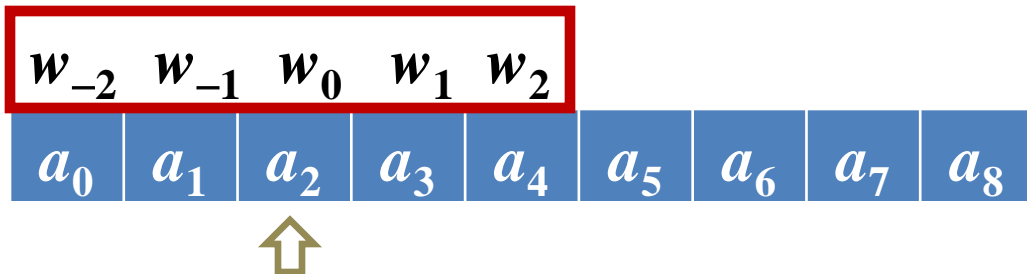
信号向量: $a = (a_0, a_1, \dots, a_8)$

步长: $k = 2$

权值: $w = (w_{-2}, w_{-1}, w_0, w_1, w_2)$
 $= (b_4, b_3, b_2, b_1, b_0)$

$$a_i' = a_{i-2}b_4 + a_{i-1}b_3 + a_ib_2 + a_{i+1}b_1 + a_{i+2}b_0$$

下标之和为 $i + k$



$$a_i' = a_{i-2}b_4 + a_{i-1}b_3 + a_ib_2 + a_{i+1}b_1 + a_{i+2}b_0$$

a_0b_0	a_0b_1	a_0b_2	a_0b_3	a_0b_4	a_2'
a_1b_0	a_1b_1	a_1b_2	a_1b_3	a_1b_4	a_3'
a_2b_0	a_2b_1	a_2b_2	a_2b_3	a_2b_4	a_4'
a_3b_0	a_3b_1	a_3b_2	a_3b_3	a_3b_4	a_5'
a_4b_0	a_4b_1	a_4b_2	a_4b_3	a_4b_4	a_6'
a_5b_0	a_5b_1	a_5b_2	a_5b_3	a_5b_4	
a_6b_0	a_6b_1	a_6b_2	a_6b_3	a_6b_4	
a_7b_0	a_7b_1	a_7b_2	a_7b_3	a_7b_4	
a_8b_0	a_8b_1	a_8b_2	a_8b_3	a_8b_4	

小结

- 卷积的定义
- 卷积与多项式乘法的关系
- 卷积的应用——信号平滑处理

卷积计算

卷积计算：蛮力算法

向量 $a=(a_0,a_1,\dots,a_{n-1})$ 和 $b=(b_0,b_1,\dots,b_{n-1})$

$$A(x)=a_0+a_1x+a_2x^2+\dots+a_{n-1}x^{n-1}$$

$$B(x)=b_0+b_1x+b_2x^2+\dots+b_{n-1}x^{n-1}$$

$$C(x) = A(x)B(x)$$

$$=a_0b_0 + (a_0b_1+a_1b_0) x + \dots + a_{n-1}b_{n-1} x^{2n-2}$$

$C(x)$ 的系数向量就是 $a*b$.

卷积 $a*b$ 计算等价于多项式相乘

蛮力算法的时间： $O(n^2)$

计算 $2n-1$ 次多项式 $C(x)$

1. 选择值 $x_0, x_1, \dots, x_{2n-1}$,

求出 $A(x_j)$ 和 $B(x_j)$,

$j = 0, 1, \dots, 2n-1$

主要步骤:
多项式求值

2. 对每个 j , 计算 $C(x_j) = A(x_j)B(x_j)$

3. 利用多项式插值方法, 由 $C(x)$ 在

$x = x_0, x_1, \dots, x_{2n-1}$

的值求出多项式 $C(x)$ 的系数



如何选择 $x_0, x_1, \dots, x_{2n-1}$ 的值?

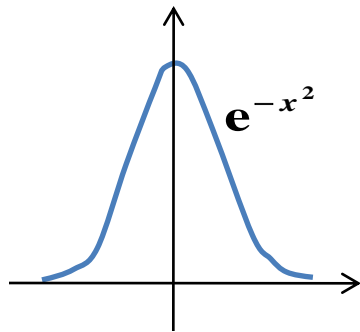
高效多项式求值算法

高斯滤波的权值函数

高斯滤波的权值函数为

$$w_s = \frac{1}{z} e^{-s^2}, \quad s = 0, \pm 1, \dots, \pm k$$

$$w = (w_{-k}, \dots, w_{-1}, w_0, w_1, \dots, w_k)$$



其中 z 用于归一化处理，使所有的权值之和为1. 处理结果

$$a_i' = \sum_{s=-k}^k a_{i+s} w_s$$

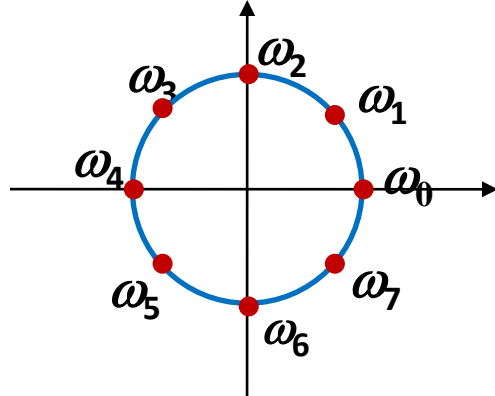
$2n$ 个数的选择:1的 $2n$ 次根

$$\omega_j = e^{\frac{2\pi j}{2n}i} = e^{\frac{\pi j}{n}i}$$

$$= \cos \frac{\pi j}{n} + i \sin \frac{\pi j}{n}$$

$$j=0,1,\dots,2n-1, \quad i=\sqrt{-1}$$

$n=4$ 的实例



$$\omega_0 = 1, \quad \omega_1 = e^{\frac{\pi}{4}i} = \sqrt{2}/2 + \sqrt{2}/2 \cdot i,$$

$$\omega_2 = e^{\frac{\pi}{2}i} = i, \quad \omega_3 = e^{\frac{3\pi}{4}i} = -\sqrt{2}/2 + \sqrt{2}/2 \cdot i,$$

$$\omega_4 = e^{\pi i} = -1, \quad \omega_5 = e^{\frac{5\pi}{4}i} = -\sqrt{2}/2 - \sqrt{2}/2 \cdot i,$$

$$\omega_6 = e^{\frac{3\pi}{2}i} = -i, \quad \omega_7 = e^{\frac{7\pi}{4}i} = \sqrt{2}/2 - \sqrt{2}/2 \cdot i$$

快速傅立叶变换FFT

1. 对 $x=1, \omega_1, \omega_2, \dots, \omega_{2n-1}$, 分别计算 $A(x), B(x)$
2. 利用步1的结果对每个 $x = 1, \omega_1, \omega_2, \dots, \omega_{2n-1}$, 计算 $C(x)$, 得到
 $C(1)=d_0, C(\omega_1)=d_1, \dots, C(\omega_{2n-1})=d_{2n-1}$
3. 构造多项式
$$D(x) = d_0 + d_1x + d_2x^2 + \dots + d_{2n-1}x^{2n-1}$$
4. 对 $x=1, \omega_1, \omega_2, \dots, \omega_{2n-1}$, 计算 $D(x)$,
 $D(1), D(\omega_1), \dots, D(\omega_{2n-1})$

快速傅立叶变换FFT (续)

可以证明:

$$D(1) = 2n c_0$$

$$D(\omega_1) = 2n c_{2n-1}$$

...

$$D(\omega_{2n-1}) = 2n c_1$$



$$c_0 = D(1)/2n$$

$$c_{2n-1} = D(\omega_1)/2n$$

...

$$c_1 = D(\omega_{2n-1})/2n$$

知道了 $D(x)$ 的值, 就能求 $C(x)$ 的系数

算法的关键

令 $x = 1, \omega_1, \omega_2, \dots, \omega_{2n-1}$,

- 步1对 $2n$ 个 x 值分别求值多项式 $A(x), B(x)$
- 步2 做 $2n$ 次乘法
- 步3 对 $2n$ 个 x 值求值多项式 $D(x)$

关键：一个对所有的 x 快速多项式求值算法

小结

卷积计算

- 蛮力算法 $O(n^2)$
- 快速傅立叶变换FFT算法
确定 x 的取值: 1 的 $2n$ 次根
关键步骤: 多项式对 x 求值



如何设计多项式求值的快速算法?

快速傅立叶 变换:FFT算法

多项式求值算法

给定多项式:

$$A(x)=a_0+a_1x+\dots+a_{n-1}x^{n-1}$$

设 x 为 1 的 $2n$ 次方根, 对所有的 x 计算 $A(x)$ 的值.

算法1: 对每个 x 做下述运算:

依次计算每个项 $a_i x^i$, $i=1, \dots, n-1$

对 $a_i x^i$ ($i=0,1,\dots,n-1$) 求和.

蛮力算法的时间复杂度

$$T_1(n) = O(n^3)$$

改进的求值算法

算法2: 依次对 每个 x 做下述计算

$$\underline{A_1(x)} = a_{n-1}$$

$$A_2(x) = a_{n-2} + x \underline{A_1(x)} = a_{n-2} + a_{n-1}x$$

$$A_3(x) = a_{n-3} + x A_2(x) = a_{n-3} + a_{n-2}x + a_{n-1}x^2$$

...

$$A_n(x) = a_0 + x A_{n-1}(x) = A(x)$$

时间复杂度

$$T_2(n) = O(n^2)$$

偶系数与奇系数多项式

$$n=4$$

$$A(x)=a_0+a_1x+a_2x^2+a_3x^3$$

$$A_{\text{even}}(x) = a_0+a_2x$$

$$A_{\text{odd}}(x) = a_1+a_3x$$

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$

$$= a_0+a_2x^2 + x(a_1+a_3x^2)$$

分治多项式求值算法

一般公式 (n 为偶数)

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{(n-2)/2}$$

$$A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{(n-2)/2}$$

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$

- x^2 也是1 的 $2n$ 次根
- 偶次数与奇次数多项式计算作为 $n/2$ 规模的子问题, 然后利用子问题的解 $A_{\text{even}}(x^2)$ 与 $A_{\text{odd}}(x^2)$ 得到 $A(x)$

分治求值算法设计

算法 3:

1. 计算 1 的所有的 $2n$ 次根

$$1, \omega_1, \omega_2, \dots, \omega_{2n-1}$$

2. 分别计算 $A_{\text{even}}(x^2)$ 与 $A_{\text{odd}}(x^2)$

3. 利用步2 的结果计算 $A(x)$

$$A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$$

注意: x^2 不需要重新计算, 所有根在单位圆间隔分布, 隔一取一即可.

分治求值算法分析

$$T(n) = T_1(n) + f(n)$$

$f(n)=O(n)$ 是步 1 计算 $2n$ 次根的时间

递归过程 $T_1(n) = 2T_1(n/2) + g(n)$

$$T_1(1) = O(1),$$

$g(n) = O(n)$ 是对所有 $2n$ 次根在步3组合解的时间

$$T_1(n)=O(n\log n)$$

$$T(n)=O(n\log n)+O(n)=O(n\log n)$$

FFT算法伪码

1. 求值 $A(\omega_j)$ 和 $B(\omega_j)$, $j=0,1,\dots,2n-1$
2. 计算 $C(\omega_j)$, $j=0, 1, \dots, 2n-1$
3. 构造多项式

$$D(x)=C(\omega_0)+C(\omega_1)x+\dots+C(\omega_{2n-1})x^{2n-1}$$

4. 计算所有的 $D(\omega_j)$, $j=0,1,\dots,2n-1$
5. 利用下式计算 $C(x)$ 的系数 c_j ,

$$D(\omega_j) = 2nc_{2n-j}$$
$$j = 0, 1, \dots, 2n-1$$

FFT算法分析

步1: 求值 $A(\omega_j)$ 和 $B(\omega_j)$ $O(n \log n)$

步2: 计算所有的 $C(\omega_j)$ $O(n)$

步3:

步4: 计算所有的 $D(\omega_j)$ $O(n \log n)$

步5: 计算所有的 c_j $O(n)$

算法时间为 $O(n \log n)$

小结

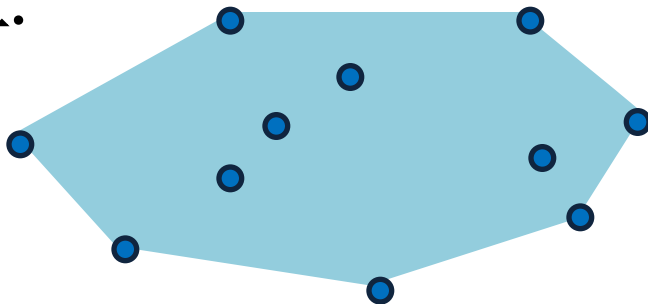
- 多项式求值算法
蛮力算法: $O(n^3)$
改进的求值算法: $O(n^2)$
FFT算法: $O(n\log n)$
- FFT算法的设计与分析

平面点集的凸包

平面点集的凸包

问题（平面点集的凸包）

给定大量离散点的集合 Q ，求一个最小的凸多边形，使得 Q 中的点在该多边形内或者边上。

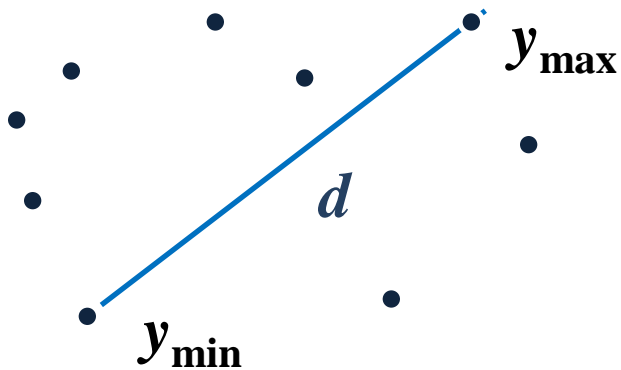


应用背景

图形处理中用于形状识别：字形识别、碰撞检测等

分治算法

1. 以 连接最大纵坐标点 y_{\max} 和最小纵坐标点 y_{\min} 的线段 $d = \{y_{\max}, y_{\min}\}$ 划分 L 为左点集 L_{left} 和右点集 L_{right}



2. Deal (L_{left}); Deal (L_{right})

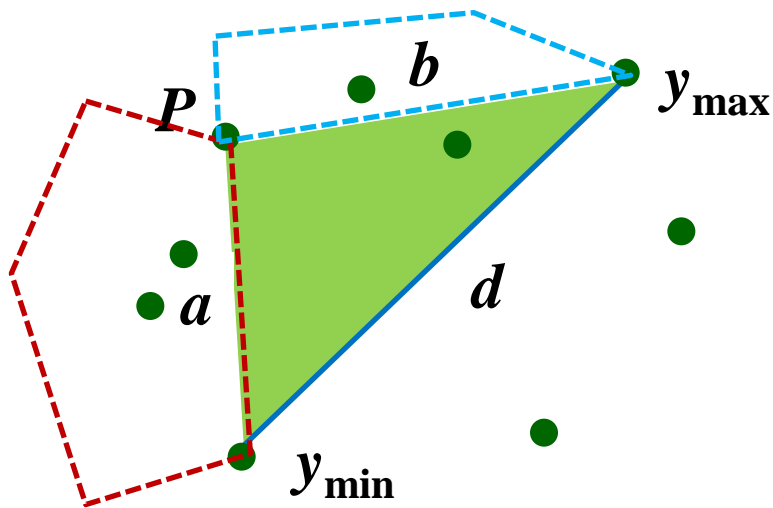
Deal (L_{left})

考虑 L_{left} : 确定距 d 最远的点 P

在三角形内的点, 删除;

a 外的点与 a 构成 L_{left} 的子问题;

b 外的点与 b 构成 L_{left} 的子问题.



伪码

Deal (L_{left})

1. 以 d 和距离 d 最远点 P 构成三角形, P 加入凸包, 另外两条边分别记作 a 和 b
2. 检查 L_{left} 中其他点是否在三角形内; 在则从 L 中删除; 否则根据在 a 或 b 边的外侧划分在两个子问题中
3. Deal (a)
4. Deal (b)

算法分析

- 初始用 d 划分 $O(n)$
- Deal 递归调用 $W(n)$
 - 找凸包顶点 P $O(n)$
 - 根据点的位置划分子问题 $O(n)$

- $$W(n) = W(n-1) + O(n)$$
$$W(3) = O(1)$$

最坏情况为 $O(n^2)$

$$T(n) = O(n) + W(n) = O(n^2)$$

- Graham扫描算法 $O(n \log n)$

小结：分治算法设计

- 将原问题归约为子问题
 - 直接划分注意尽量均衡
 - 通过计算归约为特殊的子问题
 - 子问题与原问题具有相同的性质
 - 子问题之间独立计算
- 算法实现：
 - 递归或迭代实现
 - 注意递归执行的边界

小结：分治算法的 分析及改进

- 时间复杂度分析
 - 给出关于时间复杂度函数的
递推方程和初值
 - 求解方程
- 提高效率的途径
 - 减少子问题个数
 - 预处理

重要的分治算法

- 检索算法：二分检索
- 排序算法：快速排序、二分归并排序
- 选择算法
- 快速傅立叶变换 FFT 算法
- 平面点集的凸包

动态规划 算法的例子

最短路径问题

问题:

输入: 起点集合 $\{S_1, S_2, \dots, S_n\}$,

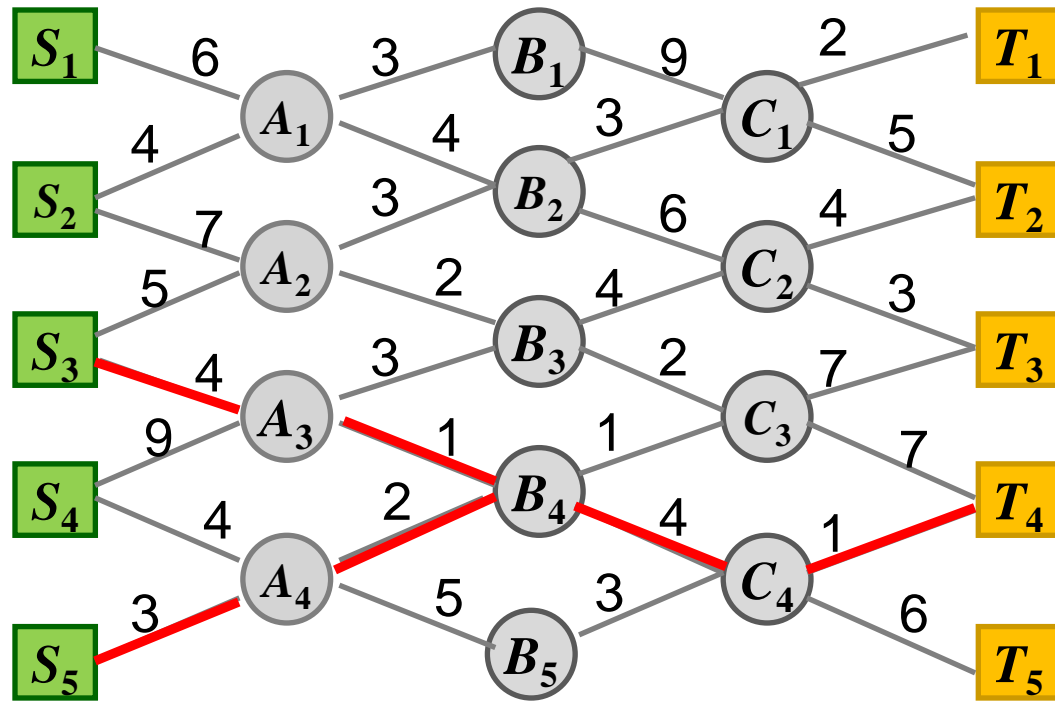
终点集合 $\{T_1, T_2, \dots, T_m\}$,

中间结点集,

边集 E , 对于任意边 e 有长度

输出: 一条从起点到终点的最短路

一个实例



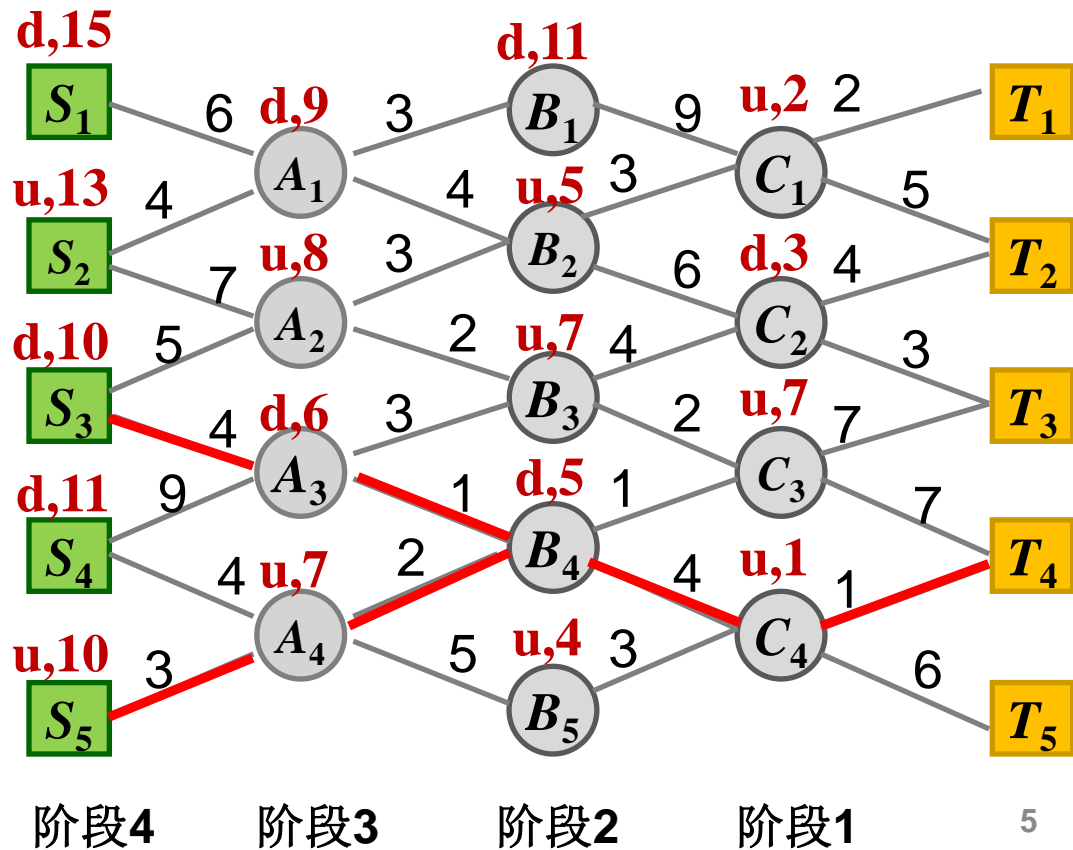
算法设计

蛮力算法：考察每一条从某个起点到某个终点的路径，计算长度，从其中找出最短路径。

在上述实例中，如果网络的层数为 k ，那么路径条数将接近于 2^k

动态规划算法：多阶段决策过程。每步求解的问题是后面阶段求解问题的子问题。每步决策将依赖于以前步骤的决策结果。

动态规划求解



子问题界定

后边界不变, 前边界前移

 决策 1

  决策 2

   决策 3

    决策 4

S_i A_j B_k C_l T_m

最短路长的依赖关系

$$\underline{F(C_l)} = \min_m \{C_l T_m\} \quad \text{决策 1}$$

$$F(B_k) = \min_l \{B_k C_l + \underline{F(C_l)}\} \quad \text{决策 2}$$

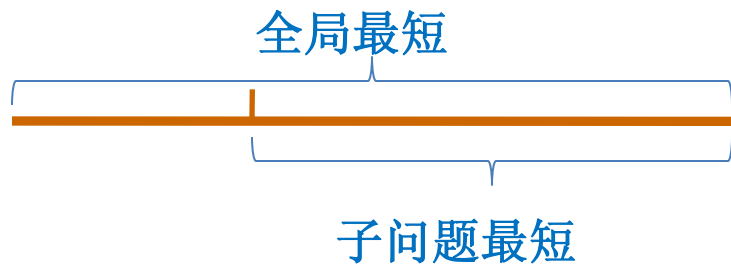
$$F(A_j) = \min_k \{A_j B_k + F(B_k)\} \quad \text{决策 3}$$

$$F(S_i) = \min_j \{S_i A_j + F(A_j)\} \quad \text{决策 4}$$

优化函数值之间存在依赖关系

优化原则:最优子结构性质

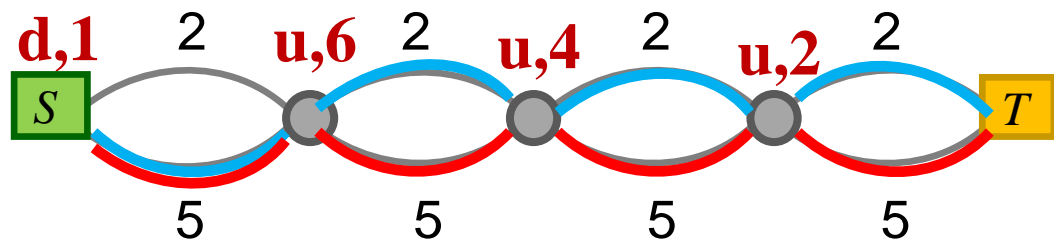
- **优化函数的特点:** 任何最短路的子路径相对于子问题始、终点最短



- **优化原则:** 一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

一个反例

求总长模10的最小路径



动态规划算法的解：下，上，上，上

最优解：下，下，下，下

不满足优化原则，不能用动态规划

小结

动态规划(Dynamic Programming)

- 求解过程是多阶段决策过程，每步处理一个子问题，可用于求解组合优化问题
- 适用条件：问题要满足优化原则或最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

动态规划 算法设计

动态规划设计要素

1. 问题建模，优化的目标函数是什么？约束条件是什么？
2. 如何划分子问题（边界）？
3. 问题的优化函数值与子问题的优化函数值存在着什么依赖关系？
(递推方程)
4. 是否满足优化原则？
5. 最小子问题怎样界定？其优化函数值，即初值等于什么？

矩阵链相乘

问题： 设 A_1, A_2, \dots, A_n 为矩阵序列， A_i 为 $P_{i-1} \times P_i$ 阶矩阵， $i = 1, 2, \dots, n$.
试确定矩阵的乘法顺序，使得元素相乘的总次数最少.

输入： 向量

$$P = \langle P_0, P_1, \dots, P_n \rangle,$$

其中 P_0, P_1, \dots, P_n 为 n 个矩阵的行数与列数

输出： 矩阵链乘法加括号的位置.

矩阵相乘基本运算次数

矩阵A: i 行 j 列, B : j 行 k 列, 以元素相乘作基本运算, 计算 AB 的工作量

$$\begin{bmatrix} \cdots \\ a_{t1} & a_{t2} & \cdots & a_{tj} \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ b_{1s} \\ b_{2s} \\ \vdots \\ b_{js} \end{bmatrix} = \begin{bmatrix} \vdots \\ c_{ts} \\ \vdots \end{bmatrix}$$

$$c_{ts} = a_{t1}b_{1s} + a_{t2}b_{2s} + \cdots + a_{tj}b_{js}$$

AB : i 行 k 列, 计算每个元素需要做 j 次乘法, 总计乘法次数 为 ijk

实例

实例: $P = \langle 10, 100, 5, 50 \rangle$

$A_1: 10 \times 100$, $A_2: 100 \times 5$, $A_3: 5 \times 50$,

乘法次序

$(A_1 A_2) A_3$: $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$

$A_1 (A_2 A_3)$: $10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000$

第一种次序计算次数最少.

蛮力算法

加 n 个括号的方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种，
是一个Catalan数，是指数级别

$$W(n) = \Omega\left(\frac{1}{n+1} \frac{(2n)!}{n!n!}\right)$$

代入
Stirling
公式

$$= \Omega\left(\frac{1}{n+1} \frac{\sqrt{2\pi 2n} \left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(\frac{n}{e}\right)^n}\right) = \Omega(2^{2n} / n^{\frac{3}{2}})$$

动态规划算法

- 子问题划分

$A_{i..j}$: 矩阵链 $A_i A_{i+1} \dots A_j$, 边界 i, j

输入向量: $\langle P_{i-1}, P_i, \dots, P_j \rangle$

其最好划分的运算次数: $m[i, j]$

- 子问题的依赖关系

最优划分最后一次相乘发生在矩阵 k 的位置, 即

$$A_{i..j} = A_{i..k} A_{k+1..j}$$

$A_{i..j}$ 最优运算次数依赖于 $A_{i..k}$ 与 $A_{k+1..j}$ 的最优运算次数

优化函数的递推方程

递推方程:

$m[i,j]$: 得到 $A_{i..j}$ 的最少的相乘次数

$m[i,j]$

$$= \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{ \underline{m[i,k]} + \underline{m[k+1,j]} + P_{i-1}P_kP_j \} & i < j \end{cases}$$

A_i	\dots	A_k	A_{k+1}	\dots	A_j
$P_{i-1} \times P_k$			$P_k \times P_j$		

该问题满足优化原则

小结

动态规划算法设计要素

- 多阶段决策过程，每步处理一个子问题，界定子问题的边界
- 列出优化函数的递推方程及初值
- 问题要满足优化原则或最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

动态规划算法 的递归实现

部分伪码

算法1 RecurMatrixChain (P, i, j)

子问题 $i-j$

1. $m[i,j] \leftarrow \infty$

2. $s[i,j] \leftarrow i$

划分位置 k

3. for $k \leftarrow i$ to $j-1$ do

4. $q \leftarrow$ RecurMatrixChain(P, i, k)
+ RecurMatrixChain($P, k+1, j$) + $p_{i-1}p_kp_j$

5. if $q < m[i,j]$

6. then $m[i,j] \leftarrow q$

7. $s[i,j] \leftarrow k$

找到更好的解

8. return $m[i,j]$

这里没有写出算法的全部描述（递归边界）

算法分析

时间复杂度的递推方程

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

$$T(n) \geq O(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k)$$

$$T(n) \geq O(n) + 2 \sum_{k=1}^{n-1} T(k)$$

时间复杂度

数学归纳法证明: $T(n) \geq 2^{n-1}$

$n=2$, 显然为真.

假设对于任何小于 n 的 k , 命题为真

$$T(n) \geq O(n) + 2 \sum_{k=1}^{n-1} T(k)$$

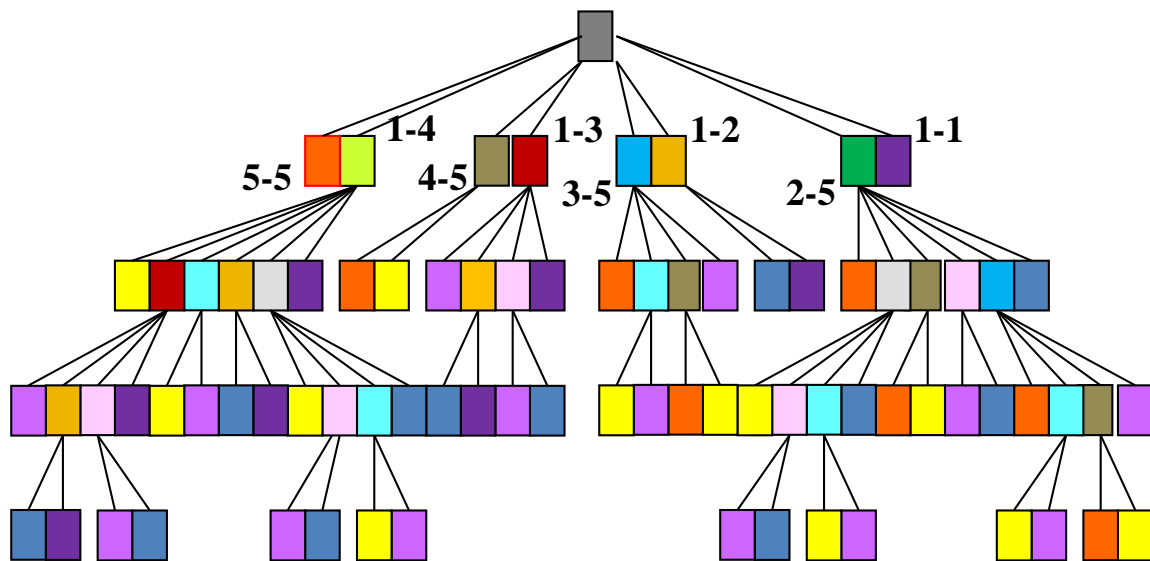
代入归纳假设

$$\geq O(n) + 2 \sum_{k=1}^{n-1} 2^{k-1}$$

等比数列求和

$$= O(n) + 2(2^{n-1} - 1) \geq 2^{n-1}$$

子问题的产生, $n=5$



划分: $A_{1..4}A_{5..5}$, $A_{1..3}A_{4..5}$, $A_{1..2}A_{3..5}$, $A_{1..1}A_{2..5}$

产生 8 个子问题, 即第一层的 8 个结点.

子问题的计数

边界	次数	边界	次数	边界	次数
1-1	8	1-2	4	2-4	2
2-2	12	2-3	5	3-5	2
3-3	14	3-4	5	1-4	1
4-4	12	4-5	4	2-5	1
5-5	8	1-3	2	1-5	1

边界不同的子问题：15 个

递归计算的子问题：81 个

结论

- 与蛮力算法相比较，动态规划算法利用了子问题优化函数间的依赖关系，时间复杂度有所降低
- 动态规划算法的递归实现效率不高，原因在于同一子问题多次重复出现，每次出现都需要重新计算一遍。
- 采用空间换时间策略，记录每个子问题首次计算结果，后面再用时就直接取值，每个子问题只算一次。

动态规划算法 的迭代实现

迭代计算的关键

- 每个子问题只计算一次
- 迭代过程
 - 从最小的子问题算起
 - 考虑计算顺序，以保证后面用到的值前面已经计算好
 - 存储结构保存计算结果——备忘录
- 解的追踪
 - 设计标记函数标记每步的决策
 - 考虑根据标记函数追踪解的算法

矩阵链乘法不同子问题

长度1: 只含1个矩阵, 有 n 个子问题
(不需要计算)

长度2: 含2个矩阵, $n-1$ 个子问题

长度3: 含3个矩阵, $n-2$ 个子问题

...

长度 $n-1$: 含 $n-1$ 个矩阵, 2个子问题

长度 n : 原始问题, 只有1个

矩阵链乘法迭代顺序

长度为1: 初值, $m[i, i] = 0$

长度为2: 1..2, 2..3, 3..4, ... , $n-1..n$

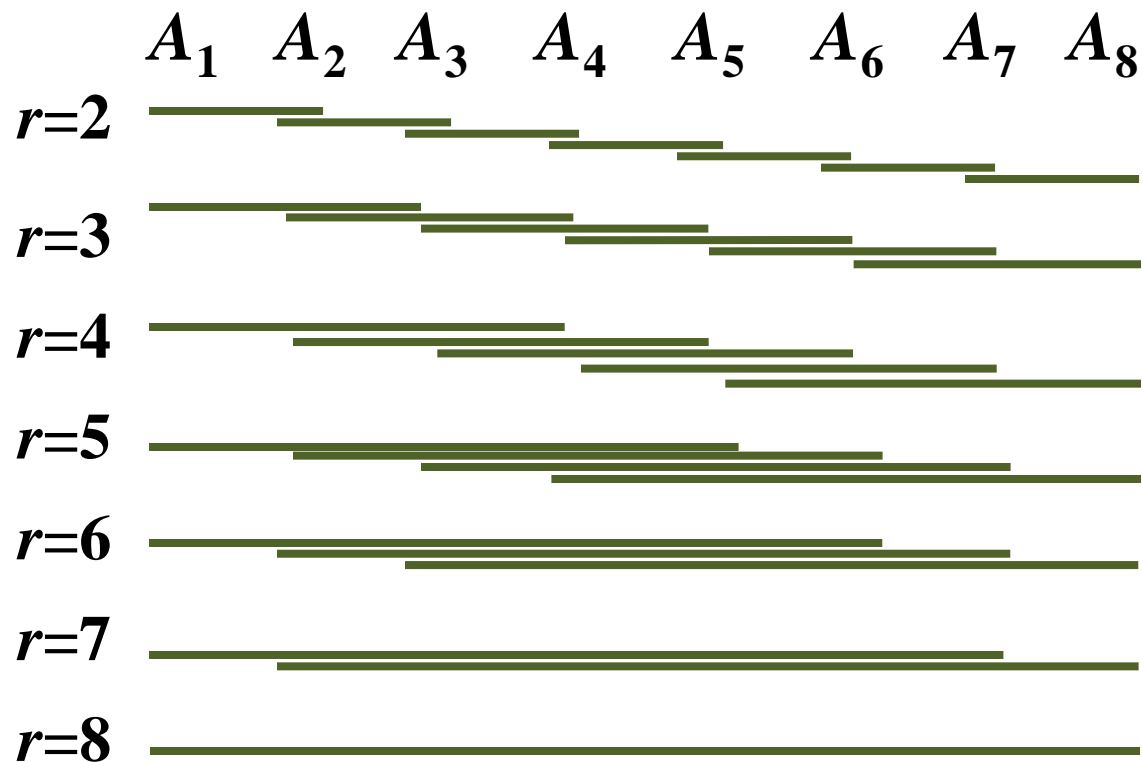
长度为3: 1..3, 2..4, 3..5, ... , $n-2..n$

...

长度为 $n-1$: 1.. $n-1$, 2.. n

长度为 n : 1.. n

$n=8$ 的子问题计算顺序



算法 **MatrixChain** (P, n)

1. 令所有的 $m[i,j]$ 初值为0
2. for $r \leftarrow 2$ to n do // r 为链长
3. for $i \leftarrow 1$ to $n-r+1$ do // 左边界 i
4. $j \leftarrow i+r-1$ // 右边界 j
5. $m[i,j] \leftarrow m[i+1,j] + p_{i-1}p_ip_j$ // $k=i$
6. $s[i,j] \leftarrow i$ // 记录 k
7. for $k \leftarrow i+1$ to $j-1$ do // 遍历 k
8. $t \leftarrow \underline{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j}$
9. if $t < m[i,j]$
10. then $m[i,j] \leftarrow t$ // 更新解
11. $s[i,j] \leftarrow k$

遍历
长 r 子
问题

遍历所
有划分

二维数组 m 与 s 为备忘录

时间复杂度

- **根据伪码**：行 2, 3, 7 都是 $O(n)$ ，循环执行 $O(n^3)$ 次，内部为 $O(1)$

$$W(n) = O(n^3)$$

- **根据备忘录**：估计每项工作量，求和。
子问题有 $O(n^2)$ 个，确定每个子问题的最少乘法次数需要对不同划分位置比较，需要 $O(n)$ 时间。

$$W(n) = O(n^3)$$

- 追踪解工作量 $O(n)$ ，总工作量 $O(n^3)$.

实例

输入: $P = \langle 30, 35, 15, 5, 10, 20 \rangle$,
 $n = 5$

矩阵链: $A_1 A_2 A_3 A_4 A_5$, 其中

$A_1: 30 \times 35$, $A_2: 35 \times 15$, $A_3: 15 \times 5$,

$A_4: 5 \times 10$, $A_5: 10 \times 20$

备忘录: 存储所有子问题的最小乘法次数及得到这个值的划分位置.

备忘录 $m[i,j]$ $P = \langle 30, \underline{35}, 15, \underline{5}, \underline{10}, 20 \rangle$

$r=1$	$m[1,1]=0$	$m[2,2]=0$	$m[3,3]=0$	$m[4,4]=0$	$m[5,5]=0$
$r=2$	$m[1,2]=15750$	$m[2,3]=2625$	$m[3,4]=750$	$m[4,5]=1000$	
$r=3$	$m[1,3]=7875$	$m[2,4]=4375$	$m[3,5]=2500$		
$r=4$	$m[1,4]=9375$	$m[2,5]=7125$			
$r=5$	$m[1,5]=11875$				

$$m[2,5] = \min\{ 0+2500+35 \times 15 \times 20, 2625+1000+35 \times 5 \times 20, \\ 4375+0+35 \times 10 \times 20 \} = 7125$$

标记函数 $s[i,j]$

$r=2$	$s[1,2]=1$	$s[2,3]=2$	$s[3,4]=3$	$s[4,5]=4$	
$r=3$	$s[1,3]=1$	$s[2,4]=3$	$s[3,5]=3$		
$r=4$	$s[1,4]=3$	$s[2,5]=3$			
$r=5$	$s[1,5]=3$				

解的追踪: $s[1,5]=3 \Rightarrow (A_1A_2A_3)(A_4A_5)$

$s[1,3]=1 \Rightarrow A_1(A_2A_3)$

输出

计算顺序: $(A_1(A_2A_3))(A_4A_5)$

最少的乘法次数: $m[1,5]=11875$

两种实现的比较

递归实现：时间复杂性高，空间较小

迭代实现：时间复杂性低，空间消耗多

原因：递归实现子问题多次重复计算，子问题计算次数呈指数增长. 迭代实现每个子问题只计算一次.

动态规划时间复杂度：

备忘录各项计算量之和 + 追踪解工作量

通常追踪工作量不超过计算工作量，是问题规模的多项式函数

动态规划算法的要素

- 划分子问题，确定子问题边界，将问题求解转 变成多步判断的过程。
- 定义优化函数,以该函数极大(或极小)值作为依据,确定是否满足优化原则。
- 列优化函数的递推方程和边界条件
- 自底向上计算，设计备忘录 (表格)
- 考虑是否需要设立标记函数
- 用递推方程或备忘录估计时间复杂度

投资问题

投资问题的建模

问题： m 元钱， n 项投资， $f_i(x)$: 将 x 元投入第 i 个项目的效益. 求使得总效益最大的投资方案.

建模：

问题的解是向量 $\langle x_1, x_2, \dots, x_n \rangle$,

x_i 是投给项目 i 的钱数, $i = 1, 2, \dots, n$.

目标函数 $\max\{f_1(x_1)+f_2(x_2)+\dots+f_n(x_n)\}$

约束条件 $x_1+x_2+\dots+x_n=m, x_i \in \mathbb{N}$

实例

- 实例：5万元钱，4个项目
效益函数如下表所示

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

子问题界定和计算顺序

子问题界定：由参数 k 和 x 界定

k ：考虑对项目 $1, 2, \dots, k$ 的投资

x ：投资总钱数不超过 x



这两个参数与矩阵链相乘问题的参数有什么区别？

原始输入： $k = n, x = m$

子问题计算顺序：

$k = 1, 2, \dots, n$

对于给定的 $k, x = 1, 2, \dots, m$

优化函数的递推方程

$F_k(x)$: x 元钱投给前 k 个项目最大效益

多步判断: 若知道 p 元钱 ($p \leq x$) 投给前 $k-1$ 个项目的最大效益 $F_{k-1}(p)$, 确定 x 元钱投给前 k 个项目的方案

递推方程和边界条件

$$F_k(x) = \max_{0 \leq x_k \leq x} \{f_k(x_k) + F_{k-1}(x - x_k)\} \quad k > 1$$

$$F_1(x) = f_1(x)$$

$k=1$ 时实例的计算

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

$k=1$ 为初值

$F_1(1)=11, F_1(2)=12, F_1(3)=13,$

$F_1(4)=14, F_1(5)=15,$

$k=2$ 时实例计算

方案(项目2,其他): (1,0), (0,1)

$$F_2(1) = \max\{f_1(1), f_2(1)\} = 11$$

x	$f_1(x)$	$f_2(x)$
0	0	0
1	11	0
2	12	5
3	13	10
4	14	15
5	15	20

方案: (2,0), (1,1), (0,2)

$$F_2(2) = \max\{f_2(2), F_1(1) + f_2(1), F_1(2)\} = 12$$

方案: (3,0), (2,1), (1,2), (0,3)

$$F_2(3) = \max\{f_2(3), F_1(1) + f_2(2), F_1(2) + f_2(1), F_1(3)\} = 16$$

类似地计算

$$F_2(4) = 21, F_2(5) = 26$$

备忘录和解

x	$F_1(x) \ x_1(x)$	$F_2(x) \ x_2(x)$	$F_3(x) \ x_3(x)$	$F_4(x) \ x_4(x)$
1	11 1	11 0	11 0	20 1
2	12 2	12 0	13 1	31 1
3	13 3	16 2	30 3	33 1
4	14 4	21 3	41 3	50 1
5	15 5	26 4	43 4	61 1

$$x_4(5)=1 \Rightarrow x_4=1, \ x_3(5-1) = x_3(4)$$

$$x_3(4)=3 \Rightarrow x_3=3, \ x_2(4-3) = x_2(1)$$

$$x_2(1)=0 \Rightarrow x_2=0, \ x_1(1-0) = x_1(1)$$

$$x_1(1)=1 \Rightarrow x_1=1$$

解: $x_1=1, x_2=0, x_3=3, x_4=1, F_4(5) = 61$

时间复杂度分析

备忘录表中有 m 行 n 列, 共计 mn 项

$$F_k(x) = \max_{0 \leq x_k \leq x} \{f_k(x_k) + F_{k-1}(x - x_k)\} \quad k > 1$$

$$F_1(x) = f_1(x)$$

x_k 有 $x+1$ 种可能的取值, 计算 $F_k(x)$ 项 ($2 \leq k \leq n, 1 \leq x \leq m$) 需要:

$x+1$ 次加法

x 次比较

时间复杂度分析

对备忘录中所有的项求和：

$$\text{加法次数} \sum_{k=2}^n \sum_{x=1}^m (x+1) = \frac{1}{2}(n-1)m(m+3)$$

$$\text{比较次数} \sum_{k=2}^n \sum_{x=1}^m x = \frac{1}{2}(n-1)m(m+1)$$

$$W(n) = O(nm^2)$$

小结

投资问题的动态规划算法

- 用两个不同类型的参数界定子问题
- 优化函数的递推方程及初值
- 根据备忘录中项的计算估计时间复杂度
- 时间复杂度为： $O(nm^2)$

动态规划算法 解背包问题

背包问题 (Knapsack Problem)

一个旅行者随身携带一个背包. 可以放入背包的物品有 n 种, 每种物品的重量和价值分别为 w_i, v_i . 如果背包的最大重量限制是 b , 每种物品可以放多个. 怎样选择放入背包的物品以使得背包的价值最大? 不妨设上述 w_i, v_i, b 都是正整数.

实例: $n = 4, b = 10$

$$v_1 = 1, \quad v_2 = 3, \quad v_3 = 5, \quad v_4 = 9,$$

$$w_1 = 2, \quad w_2 = 3, \quad w_3 = 4, \quad w_4 = 7, \quad 2$$

建模

解是 $\langle x_1, x_2, \dots, x_n \rangle$, 其中 x_i 是装入背包的第 i 种物品个数

$$\text{目标函数} \quad \max \sum_{i=1}^n v_i x_i$$

$$\text{约束条件} \quad \sum_{i=1}^n w_i x_i \leq b, \quad x_i \in \mathbb{N}$$

线性规划问题: 由线性条件约束的线性函数取最大或最小的问题

整数规划问题: 线性规划问题的变量 x_i 都是非负整数

子问题界定和计算顺序

子问题界定：由参数 k 和 y 界定

k : 考虑对物品 $1, 2, \dots, k$ 的选择

y : 背包总重量不超过 y

原始输入: $k = n, y = b$

子问题计算顺序:

$k = 1, 2, \dots, n$

对于给定的 $k, y = 1, 2, \dots, b$

优化函数的递推方程

$F_k(y)$: 装前 k 种物品, 总重不超过 y ,
背包达到的最大价值

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

$$F_0(y) = 0, \quad 0 \leq y \leq b, \quad F_k(0) = 0, \quad 0 \leq k \leq n$$

$$F_1(y) = \left\lfloor \frac{y}{w_1} \right\rfloor v_1, \quad F_k(y) = -\infty \quad y < 0$$



类似于投资问题, 如何写递推方程
这两种写法有什么区别

$$F_k(y) = \max_{0 \leq x_k \leq \lfloor y/w_k \rfloor} \{F_{k-1}(y - x_k w_k) + x_k v_k\}$$

标记函数

$i_k(y)$: 装前 k 种物品, 总重不超 y , 背包达到最大价值时装入物品的最大标号

$$i_k(y) = \begin{cases} i_{k-1}(y) & F_{k-1}(y) > F_k(y - w_k) + v_k \\ k & F_{k-1}(y) \leq F_k(y - w_k) + v_k \end{cases}$$

$$i_1(y) = \begin{cases} 0 & y < w_1 \\ 1 & y \geq w_1 \end{cases}$$

实例

输入: $v_1=1, v_2=3, v_3=5, v_4=9,$
 $w_1=2, w_2=3, w_3=4, w_4=7,$
 $b=10$

$F_k(y)$ 的计算表如下:

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	2	2	3	3	4	4	5
2	0	1	3	3	4	6	6	7	9	9
3	0	1	3	5	5	6	8	10	10	11
4	0	1	3	5	5	6	9	10	10	12

追踪解

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

$$i_4(10)=4 \Rightarrow x_4 \geq 1$$

$$i_4(10 - w_4) = i_4(3) = 2 \Rightarrow x_2 \geq 1, x_4 = 1, x_3 = 0$$

$$i_2(3 - w_2) = i_2(0) = 0 \Rightarrow x_2 = 1, x_1 = 0$$

解 $x_1=0, x_2=1, x_3=0, x_4=1$, 价值12

追踪算法

算法 Track Solution

输入: $i_k(y)$ 表, $k=1,2,\dots,n$, $y=1,2,\dots,b$

输出: x_1, x_2, \dots, x_n , n 种物品的装入量

1. for $k \leftarrow 1$ to n do $x_k \leftarrow 0$

2. $y \leftarrow b$, $k \leftarrow n$

3. $j \leftarrow i_k(y)$

初始追
踪位置

4. $x_k \leftarrow 1$

5. $y \leftarrow y - w_k$

6. while $i_k(y) = k$ do

继续放 k
种物品

7. $y \leftarrow y - w_k$

8. $x_k \leftarrow x_k + 1$

9. if $i_k(y) \neq 0$ then goto 4

继续追
下一种

时间复杂度 $O(nb)$

根据公式

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

备忘录需计算 nb 项，每项常数时间，
计算时间为 $O(nb)$ 。

伪多项式时间算法：时间为参数 b 和 n 的多项式，不是输入规模的多项式。
参数 b 是整数，表达 b 需要 $\log b$ 位，
输入规模是 $\log b$ 。

背包问题的推广

物品数受限背包：第 i 种物品最多用 n_i 个

0-1背包问题： $x_i = 0, 1, i = 1, 2, \dots, n$

多背包问题： m 个背包，背包 j 装入最大重量 $B_j, j = 1, 2, \dots, m$. 在满足所有背包重量约束条件下使装入物品价值最大.

二维背包问题：每件物品有重量 w_i 和体积 $t_i, i = 1, 2, \dots, n$, 背包总重不超过 b , 体积不超过 V , 如何选择物品以得到最大价值.

小结

- 划分子问题，确定子问题边界，将问题求解转变成多步判断的过程.
- 定义优化函数,以该函数极大(或极小)值作为依据,确定是否满足优化原则.
- 列优化函数的递推方程和边界条件
- 自底向上计算，设计备忘录 (表格)
- 考虑是否需要设立标记函数
- 时间复杂度估计

最长公共子序列

子序列

设序列 X, Z ,

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle$$

若存在 X 的元素构成的严格递增序列
使得

$$z_j = x_{i_j}, j = 1, 2, \dots, k$$

则称 Z 是 X 的子序列

X 与 Y 的公共子序列 Z : Z 是 X 和 Y
的子序列

子序列的长度：子序列的元素个数

最长公共子序列

问题：给定序列

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle$$

求 X 和 Y 的最长公共子序列

实例：

X : **A** **B** **C** **B** **D** **A** **B**

Y : **B** **D** **C** **A** **B** **A**

最长公共子序列: **B C B A**, 长度4

蛮力算法

不妨设 $m \leq n$, $|X| = m$, $|Y| = n$

算法：依次检查 X 的每个子序列在 Y 中是否出现

时间复杂度：

每个子序列 $O(n)$ 时间

X 有 2^m 个子序列

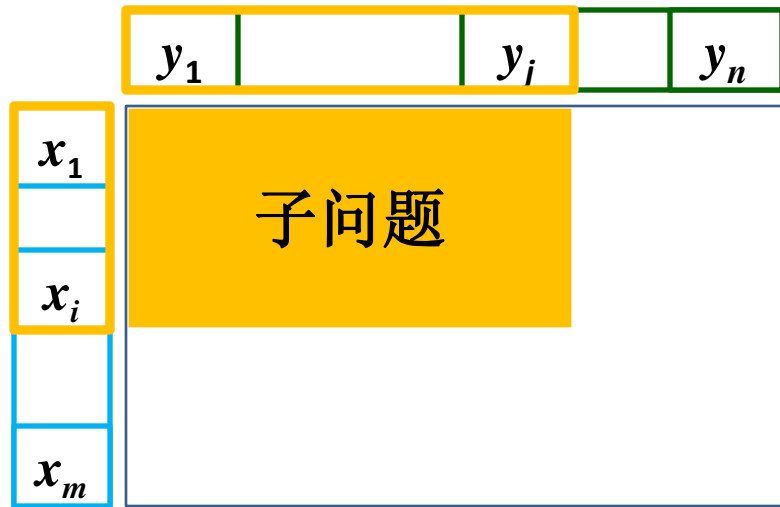
最坏情况下时间复杂度： $O(n 2^m)$

子问题界定

参数 i 和 j 界定子问题

X 的终止位置是 i , Y 的终止位置是 j

$X_i = \langle x_1, x_2, \dots, x_i \rangle$, $Y_j = \langle y_1, y_2, \dots, y_j \rangle$



子问题间的依赖关系

设 $X=\langle x_1, x_2, \dots, x_m \rangle$, $Y=\langle y_1, y_2, \dots, y_n \rangle$,
 $Z=\langle z_1, z_2, \dots, z_k \rangle$ 为 X 和 Y 的 LCS, 那么

(1) 若 $x_m = y_n \Rightarrow z_k = x_m = y_n$, 且

Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的 LCS;

(2) 若 $x_m \neq y_n$, $z_k \neq x_m \Rightarrow$

Z 是 X_{m-1} 与 Y 的 LCS;

(3) 若 $x_m \neq y_n$, $z_k \neq y_n \Rightarrow$

Z 是 X 与 Y_{n-1} 的 LCS.

满足优化原则和子问题重叠性

优化函数的递推方程

令 X 与 Y 的子序列

$$X_i = \langle x_1, x_2, \dots, x_i \rangle, \quad Y_j = \langle y_1, y_2, \dots, y_j \rangle$$

$C[i,j]$: X_i 与 Y_j 的 LCS 的长度

$C[i,j]$

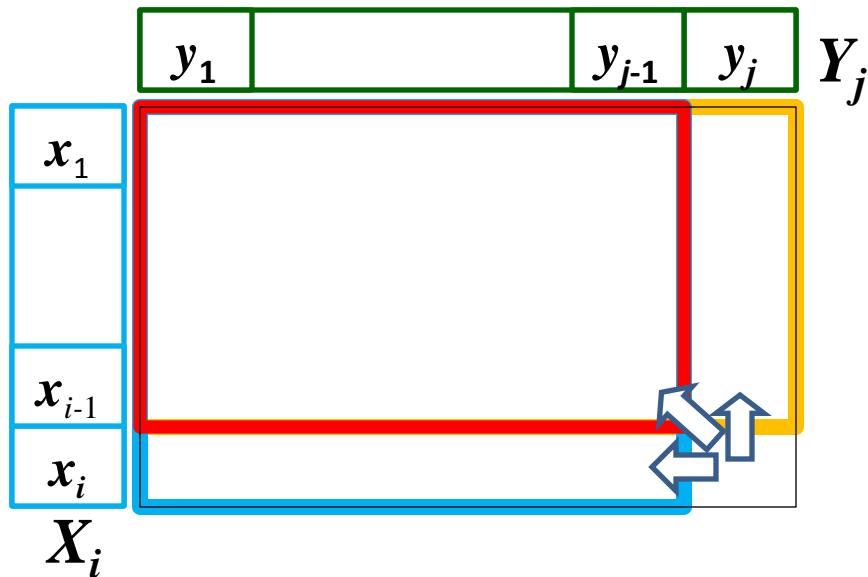
$$= \begin{cases} 0 & \text{若 } i=0 \text{ 或 } j=0 \\ C[i-1, j-1] + 1 & \text{若 } i, j > 0, x_i = y_j \\ \max\{C[i, j-1], C[i-1, j]\} & \text{若 } i, j > 0, x_i \neq y_j \end{cases}$$

标记函数

标记函数: $B[i, j]$, 值为 \nwarrow 、 \leftarrow 、 \uparrow

$C[i, j] = C[i-1, j-1] + 1$: \nwarrow $C[i, j] = C[i, j-1]$: \leftarrow

$C[i, j] = C[i-1, j]$: \uparrow



伪码

算法 LCS (X, Y, m, n)

1. for $i \leftarrow 1$ to m do $C[i, 0] \leftarrow 0$

初值

2. for $i \leftarrow 1$ to n do $C[0, i] \leftarrow 0$

3. for $i \leftarrow 1$ to m do

4. for $j \leftarrow 1$ to n do

子问题
 x_i, y_j

5. if $X[i] = Y[j]$

6. then $C[i, j] \leftarrow C[i-1, j-1] + 1$

7. $B[i, j] \leftarrow "\nwarrow"$

8. else if $C[i-1, j] \geq C[i, j-1]$

9. then $C[i, j] \leftarrow C[i-1, j]$

10. $B[i, j] \leftarrow "\uparrow"$

11. else $C[i, j] \leftarrow C[i, j-1]$

12. $B[i, j] \leftarrow "\leftarrow"$

追踪解

算法 Structure Sequence(B, i, j)

输入: $B[i, j]$

输出: X 与 Y 的最长公共子序列

1. if $i=0$ or $j=0$ then return //序列为空
2. if $B[i, j] = “\nwarrow”$
3. then 输出 $X[i]$
4. Structure Sequence($B, i-1, j-1$)
5. else if $B[i, j] = “\nearrow”$
6. then Structure Sequence ($B, i-1, j$)
7. else Structure Sequence ($B, i, j-1$)

标记函数的实例

输入: $X = \langle A, B, C, B, D, A, B \rangle$,
 $Y = \langle B, D, C, A, B, A \rangle$,

	1	2	3	4	5	6
1	$B[1,1]=\uparrow$	$B[1,2]=\uparrow$	$B[1,3]=\uparrow$	$B[1,4]=\nearrow$	$B[1,5]=\leftarrow$	$B[1,6]=\nwarrow$
2	$B[2,1]=\nwarrow$	$B[2,2]=\leftarrow$	$B[2,3]=\leftarrow$	$B[2,4]=\uparrow$	$B[2,5]=\nwarrow$	$B[2,6]=\leftarrow$
3	$B[3,1]=\uparrow$	$B[3,2]=\uparrow$	$B[3,3]=\nwarrow$	$B[3,4]=\leftarrow$	$B[3,5]=\uparrow$	$B[3,6]=\uparrow$
4	$B[4,1]=\uparrow$	$B[4,2]=\uparrow$	$B[4,3]=\uparrow$	$B[4,4]=\uparrow$	$B[4,5]=\nwarrow$	$B[4,6]=\leftarrow$
5	$B[5,1]=\uparrow$	$B[5,2]=\uparrow$	$B[5,3]=\uparrow$	$B[5,4]=\uparrow$	$B[5,5]=\uparrow$	$B[5,6]=\uparrow$
6	$B[6,1]=\uparrow$	$B[6,2]=\uparrow$	$B[6,3]=\uparrow$	$B[6,4]=\nwarrow$	$B[6,5]=\uparrow$	$B[6,6]=\nwarrow$
7	$B[7,1]=\uparrow$	$B[7,2]=\uparrow$	$B[7,3]=\uparrow$	$B[7,4]=\uparrow$	$B[7,5]=\uparrow$	$B[7,6]=\uparrow$

解: $X[2], X[3], X[4], X[6]$, 即 B, C, B, A

算法的时空复杂度

计算优化函数和标记函数：

赋初值，为 $O(m)+O(n)$

计算优化、标记函数迭代 $\Theta(mn)$ 次，

循环体内常数运算，时间为 $\Theta(mn)$

构造解：

每步缩小 X 或 Y 的长度，时间 $\Theta(m+n)$

算法时间复杂度： $\Theta(mn)$

空间复杂度： $\Theta(mn)$

小结

- 最长公共子序列问题的建模
- 子问题边界的界定
- 递推方程及初值，优化原则判定
- 伪码
- 标记函数与解的追踪
- 时间复杂度

图像压缩

黑白图像存储

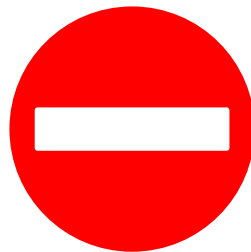
像素点灰度值：0~255，为8位二进制数

图像的灰度值序列： $\{p_1, p_2, \dots, p_n\}$ ， p_i 为第*i*个像素点灰度值

图像存储：每个像素的灰度值占8位，
总计空间为 $8n$



有没有更好的存储
方法



图像变位压缩的概念

变位压缩存储：将 $\{p_1, p_2, \dots, p_n\}$ 分成 m 段

$$S_1, S_2, \dots, S_m$$



同一段的像素占用位数相同

第 t 段有 $l[t]$ 个像素，每个占用 $b[t]$ 位

段头：记录 $l[t]$ (8位) 和 $b[t]$ (3位) 需要 11 位

总位数为

$$b[1] \cdot l[1] + b[2] \cdot l[2] + \dots + b[m] \cdot l[m] + 11m$$

图像压缩问题

约束条件：第 t 段像素个数 $l[t] \leq 256$

第 t 段占用空间： $b[t] \times l[t] + 11$

$$b[t] = \left\lceil \log(\max_{p_k \in S_t} p_k + 1) \right\rceil \leq 8$$

问题：给定像素序列 $\{p_1, p_2, \dots, p_n\}$ ，
确定最优分段，即

$$\min_T \left\{ \sum_{t=1}^m (b[t] \times l[t] + 11) \right\},$$

$T = \{S_1, S_2, \dots, S_m\}$ 为分段

实例

灰度值序列

$P=\{10,12,15,255,1,2,1,1,2,2,1,1\}$

分法1: $S_1=\{10, 12, 15\}$, $S_2=\{255\}$,
 $S_3=\{1, 2, 1, 1, 2, 2, 1, 1\}$

分法2: $S_1=\{10,12,15,255,1,2,1,1,2,2,1,1\}$

分法3: 分成12组, 每组一个数

存储空间

分法1: $11 \times 3 + 4 \times 3 + 8 \times 1 + 2 \times 8 = 69$

分法2: $11 \times 1 + 8 \times 12 = 107$

分法3: $11 \times 12 + 4 \times 3 + 8 \times 1 + 1 \times 5 + 2 \times 3 = 163$

子问题界定与计算顺序

子问题前边界为1，后边界为 i

对应像素序列为 $\langle p_1, p_2, \dots, p_i \rangle$

优化函数值 $S[i]$ 为最优分段存储位数

计算顺序

$i = 1$ 

$i = 2$ 

...

$i = n$ 

算法设计

递推方程：设 $S[i]$ 是 $\{p_1, p_2, \dots, p_i\}$ 的最优分段需要的存储位数， S_m 是最后分段

$$S[i] = \min_{1 \leq j \leq \min\{i, 256\}} \{ S[i-j] + j \times b[i-j+1, i] \} + 11$$

$$b[i-j+1, i] = \left\lceil \log(\max_{p_k \in S_m} p_k + 1) \right\rceil \leq 8$$



$S[i-j]$ 个位

j 个灰度

$j \times b[i-j+1, i]$ 位

伪码

Compress (P, n)

1. $Lmax \leftarrow 256; header \leftarrow 11; S[0] \leftarrow 0$

2. for $i \leftarrow 1$ to n do

3. $b[i] \leftarrow \text{length}(P[i])$

4. $bmax \leftarrow b[i]$

5. $S[i] \leftarrow S[i-1] + bmax$

6. $l[i] \leftarrow 1$

7. for $j \leftarrow 2$ to $\min\{i, Lmax\}$ do

8. if $bmax < b[i-j+1]$

9. then $bmax \leftarrow b[i-j+1]$

10. if $S[i] > S[i-j] + j * bmax$

11. then $S[i] \leftarrow S[i-j] + j * bmax$

12. $l[i] \leftarrow j$

13. $S[i] \leftarrow S[i] + header$

子问题后
边界 i

最后
段长 j

找到更
好分段

$P = \langle 10, 12, 15, 255, 1, 2 \rangle$.

$S[1]=15, S[2]=19, S[3]=23, S[4]=42, S[5]=50$

$l[1]=1, l[2]=2, l[3]=3, l[4]=1, l[5]=2$

10	12	15	255	1	2
----	----	----	-----	---	---

$S[5]=50$ $1 \times 2 + 11$ 63

10	12	15	255	1	2
----	----	----	-----	---	---

$S[4]=42$ $2 \times 2 + 11$ 57

10	12	15	255	1	2
----	----	----	-----	---	---

$S[3]=23$ $3 \times 8 + 11$ 58

10	12	15	255	1	2
----	----	----	-----	---	---

$S[2]=19$ $4 \times 8 + 11$ 62

10	12	15	255	1	2
----	----	----	-----	---	---

$S[1]=15$ $5 \times 8 + 11$ 66

10	12	15	255	1	2
----	----	----	-----	---	---


$6 \times 8 + 11$ 59

追踪解

算法 Traceback (n, l)

输入：数组 l

输出：数组 C

1. $j \leftarrow 1$ // j 为正在追踪的段数
2. while $n \neq 0$ do
3. $C[j] \leftarrow l[n]$ 
4. $n \leftarrow n - l[n]$
5. $j \leftarrow j + 1$

$C[j]$: 从后向前追踪的第 j 段的长度

时间复杂度: $O(n)$

小结

- 图像变位存储问题的建模
- 子问题边界的界定
- 递推方程及初值
- 伪码
- 标记函数与解的追踪
- 时间复杂度

最大子段和

最大子段和

问题： 给定 n 个数(可以为负数)的序列

$$(a_1, a_2, \dots, a_n)$$

$$\text{求 } \max\{0, \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k\}$$

实例

$$(-2, 11, -4, 13, -5, -2)$$

解： 最大子段和为 $a_2+a_3+a_4=20$

算法

算法1: 对所有的 (i, j) 对, 顺序求和 $a_i + \dots + a_j$ 并比较出最大的和

算法2: 分治策略, 将数组分成左右两半, 分别计算左边的最大和、右边的最大和、跨边界的最大和, 然后比较其中最大者

算法3: 动态规划

算法1

算法 Enumerate

输入：数组 $A[1..n]$,

输出： $sum, first, last$

1. $sum \leftarrow 0$
2. for $i \leftarrow 1$ to n do
3. for $j \leftarrow i$ to n do
4. $thisum \leftarrow 0$
5. for $k \leftarrow i$ to j do
6. $thisum \leftarrow thisum + A[k]$
7. if $thisum > sum$
8. then $sum \leftarrow thisum$
9. $first \leftarrow i$
10. $last \leftarrow j$

和的
边界 $i-j$

找到更
大和

时间复杂度： $O(n^3)$

算法2 分治策略

将序列分成左右两半，中间分点 $center$

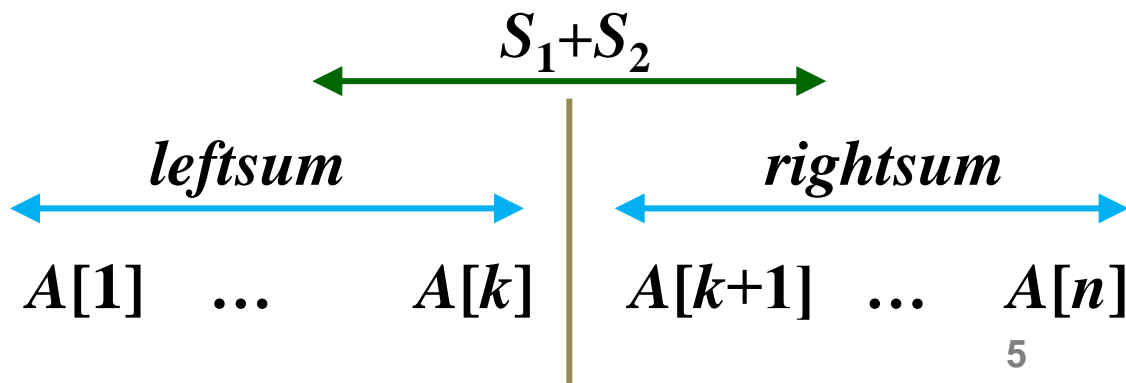
递归计算左段最大子段和 $leftsum$

递归计算右段最大子段和 $rightsum$

$center$ 到 a_1 的最大和 S_1 , $k=center$

$center + 1$ 到 a_n 的最大和 S_2

$\max \{ leftsum, rightsum, S_1+S_2 \}$



伪码

算法 MaxSubSum ($A, left, right$)

输入：数组 $A, left, right$ (左,右边界)

输出：最大子段和 sum 及子段边界

1. if $|A|=1$ then 输出元素(值为负输出0)
2. $center \leftarrow \lfloor (left+right)/2 \rfloor$
3. $leftsum \leftarrow \text{MaxSubSum}(A, left, center)$
4. $rightsum \leftarrow \text{MaxSubSum}(A, center+1, right)$
5. $S_1 \leftarrow A_1[center]$ //从 $center$ 向左
6. $S_2 \leftarrow A_2[center+1]$ //从 $center+1$ 向右
7. $sum \leftarrow S_1 + S_2$
8. if $leftsum > sum$ then $sum \leftarrow leftsum$
9. if $rightsum > sum$ then $sum \leftarrow rightsum$

时间复杂度

计算从 $k = center$ 开始到 a_1 方向的最大和，每次加1个元素，得到

$A[k]$, $A[k]+A[k-1]$, $A[k]+A[k-1]+A[k-2]$,
... , $A[k]+... +A[1]$

比较上述的最大和，时间为 $O(n)$,
右半边也是 $O(n)$

$$T(n) = 2T(n/2) + O(n)$$

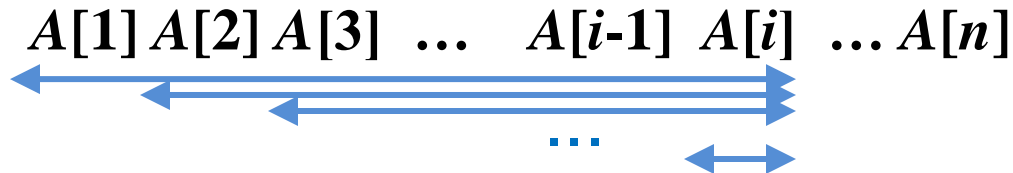
$$T(c) = O(1)$$

$$T(n) = O(n \log n)$$

算法3： 动态规划

子问题界定： 前边界为 1， 后边界 i ，
 $C[i]$ 是 $A[1..i]$ 中 **必须包含元素 $A[i]$** 的
向前连续延伸的最大子段和

$$C[i] = \max_{1 \leq k \leq i} \left\{ \sum_{j=k}^i A[j] \right\}$$



最大子段和 $C[i]$

优化函数的递推方程

递推方程:

$$\underline{C[i] = \max\{C[i-1] + A[i], A[i]\}}$$

$$i = 2, \dots, n$$

$$C[1] = A[1] \quad \text{若 } A[1] > 0$$

$$C[1] = 0 \quad \text{否则}$$

$$\text{解: } \text{OPT}(A) = \max_{1 \leq i \leq n} \{C[i]\}$$

伪码

算法 MaxSum (A, n)

输入：数组 A

输出：最大子段和 sum , 子段最后位置 c

1. $sum \leftarrow 0$
2. $b \leftarrow 0$
3. for $i \leftarrow 1$ to n do
4. if $b > 0$
5. then $b \leftarrow b + A[i]$
6. else $b \leftarrow A[i]$
7. if $b > sum$
8. then $sum \leftarrow b$
9. $c \leftarrow i$
10. return sum, c

子问题
后边界 i

找到更
大的和

时间复杂度： $O(n)$, 空间复杂度： $O(n)$

小结

- 几个算法：蛮力, 分治, 动态规划
- 动态归划算法：
 - 子问题界定
 - 列优化函数的递推方程和边界条件
 - (不一定是原问题的优化函数)
 - 自底向上计算, 设计备忘录 (表格)
 - 如何根据动态规划的解找原问题的解
 - 时间复杂度估计

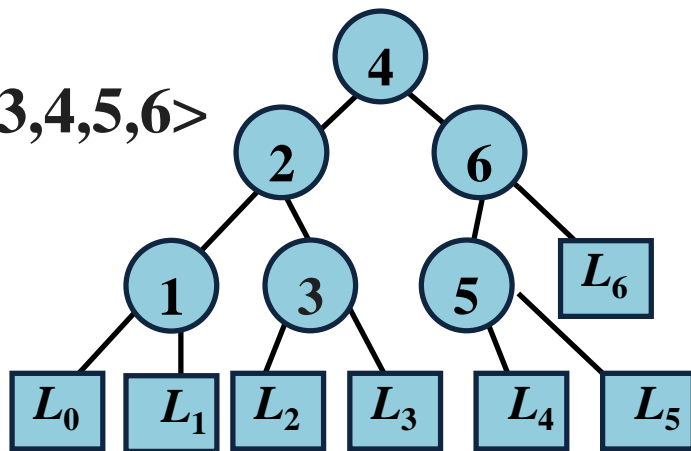
最优二叉检索树

二叉检索树

集合 S 为排序的 n 个元素, $x_1 < x_2 < \dots < x_n$, 将这些元素存储在一棵二叉树的结点上, 以查找 x 是否在这些数中. 如果 x 不在, 确定 x 在那个空隙 (方结点).

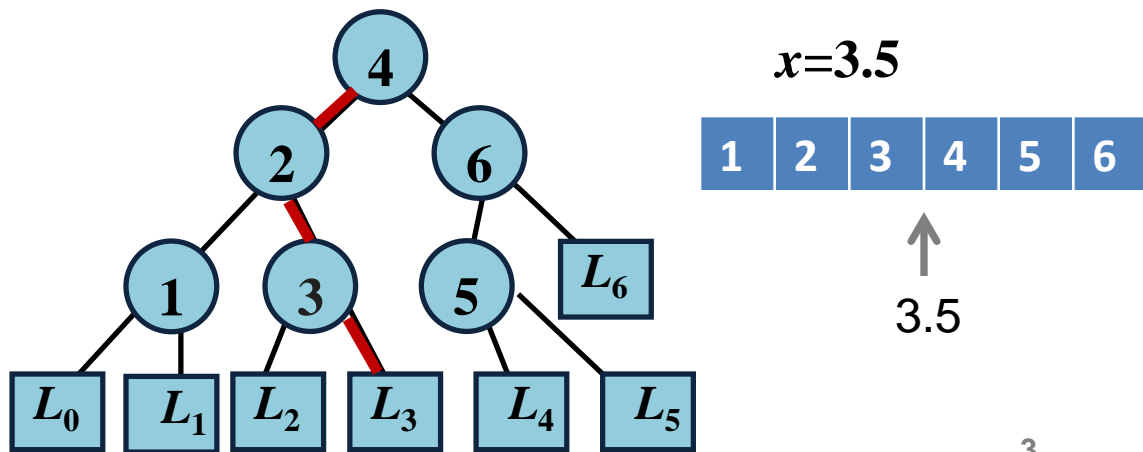
实例:

$S = \langle 1, 2, 3, 4, 5, 6 \rangle$



二叉树的检索方法

1. 初始, x 与根元素比较;
2. $x <$ 根元素, 递归进入左子树;
3. $x >$ 根元素, 递归进入右子树;
4. $x =$ 根元素, 算法停止, 输出 x ;
5. x 到叶结点算法停止, 输出 x 不在数组.



数据元素存取概率分布

空隙:

$$(x_0, x_1), (x_1, x_2), \dots, (x_{n-1}, x_n), (x_n, x_{n+1}),$$

$$x_0 = -\infty, \quad x_{n+1} = +\infty$$

给定序列 $S = \langle x_1, x_2, \dots, x_n \rangle$,

x 在 x_i 的概率为 b_i ,

x 在 (x_i, x_{i+1}) 的概率为 a_i ,

S 的存取概率分布如下:

$$P = \langle a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n \rangle$$

实例

实例: $S = \langle 1, 2, 3, 4, 5, 6 \rangle$

$P = \langle 0.04, \mathbf{0.1}, 0.01, \mathbf{0.2}, 0.05, \mathbf{0.2}, 0.02, \mathbf{0.1}, 0.02, \mathbf{0.1}, 0.07, \mathbf{0.05}, 0.04 \rangle$

1, 2, 3, 4, 5, 6 检索的概率分别为:

$\mathbf{0.1}, \mathbf{0.2}, \mathbf{0.2}, \mathbf{0.1}, \mathbf{0.1}, \mathbf{0.05}$

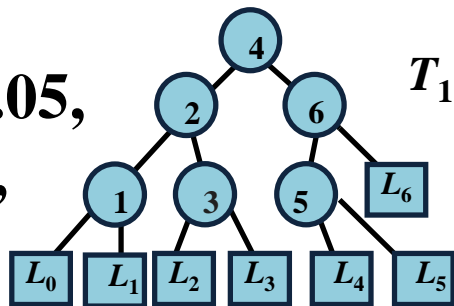
各个空隙的检索概率分别为:

$0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04$

检索数据的平均时间

$S = \langle 1, 2, 3, 4, 5, 6 \rangle$

$P = \langle 0.04, \mathbf{0.1}, 0.01, \mathbf{0.2}, 0.05, \mathbf{0.2}, 0.02, \mathbf{0.1}, 0.02, \mathbf{0.1}, \mathbf{0.07}, \mathbf{0.05}, 0.04 \rangle$

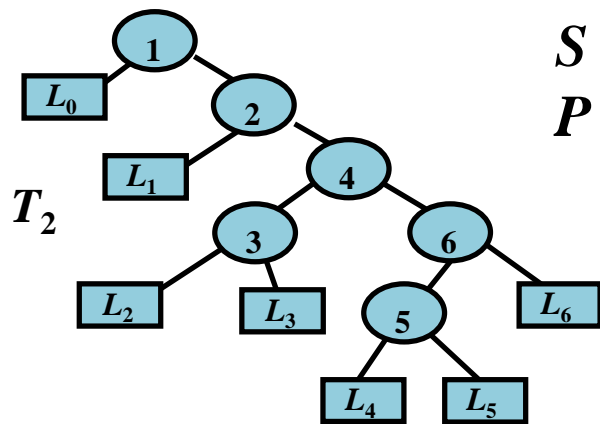


$m(T_1)$

$$\begin{aligned} &= [1 \times 0.1 + 2 \times (0.2 + 0.05) + 3 \times (0.1 + 0.2 + 0.1)] \\ &\quad + [3 \times (0.04 + 0.01 + 0.05 + 0.02 + 0.02 + 0.07) \\ &\quad + 2 \times 0.04] \end{aligned}$$

$$= 1.8 + \mathbf{0.71} = \mathbf{\underline{2.51}}$$

检索数据的平均时间



$S = \langle 1, 2, 3, 4, 5, 6 \rangle$

$P = \langle 0.04, \mathbf{0.1}, 0.01, \mathbf{0.2},$
 $0.05, \mathbf{0.2}, 0.02, \mathbf{0.1},$
 $0.02, \mathbf{0.1}, 0.07, \mathbf{0.05},$
 $0.04 \rangle$

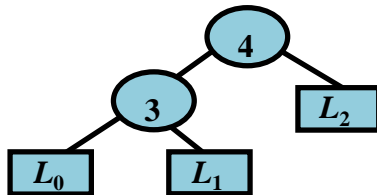
$m(T_2)$

$$\begin{aligned} &= [1 \times 0.1 + 2 \times 0.2 + 3 \times 0.1 + 4 \times (0.2 + 0.05) + 5 \times 0.1] \\ &\quad + [1 \times 0.04 + 2 \times 0.01 + 4 \times (0.05 + 0.02 + 0.04) \\ &\quad + 5 \times (0.02 + 0.07)] \\ &= 2.3 + 0.95 = \underline{\underline{3.25}} \end{aligned}$$

平均比较次数计算

数据集 $S = \langle x_1, x_2, \dots, x_n \rangle$

存取概率分布



$P = \langle a_0, b_1, a_1, b_2, \dots, a_i, b_{i+1}, \dots, b_n, a_n \rangle$

结点 x_i 在 T 中的深度是 $d(x_i)$, $i=1,2,\dots,n$,

空隙 L_j 的深度为 $d(L_j)$, $j=0,1,\dots,n$,

平均比较次数为:

$$t = \sum_{i=1}^n b_i (1 + d(x_i)) + \sum_{j=0}^n a_j d(L_j)$$

问题

给定数据集

$$S = \langle x_1, x_2, \dots, x_n \rangle,$$

及 S 的存取概率分布如下:

$$P = \langle a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n \rangle$$

求一棵最优的 (即平均比较次数最少的) 二分检索树.

小结

- 二叉检索树的构成
- 给定概率分布下，一棵二叉检索树的平均检索时间估计
- 什么是最优二叉检索树

最优二叉检索 树的算法

关键问题

子问题边界界定

如何将该问题归结为更小的子问题

优化函数的递推方程及初值

计算顺序

是否需要标记函数

时间复杂度分析

子问题划分

子问题边界为(i, j)

数据集: $S[i, j] = \langle x_i, x_{i+1}, \dots, x_j \rangle$

存取概率分布:

$$P[i, j] = \langle a_{i-1}, b_i, a_i, b_{i+1}, \dots, b_j, a_j \rangle$$

输入实例: $S = \langle A, \underline{B, C, D}, E \rangle$

$$P = \langle 0.04, \mathbf{0.1}, \underline{0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}}, \\ \underline{0.05, \mathbf{0.2}, 0.06, \mathbf{0.1}}, 0.01 \rangle$$

子问题: $S[2, 4] = \langle B, C, D \rangle$

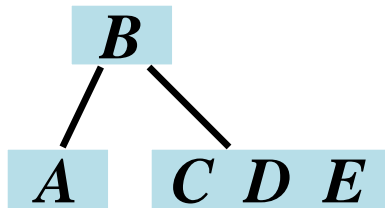
$$P[2, 4] = \langle 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06 \rangle$$

子问题归约

以 x_k 作为根归结为子问题:

$$S[i, k-1], P[i, k-1]$$

$$S[k+1, j], P[k+1, j]$$



$$S[1,5] = \langle A, B, C, D, E \rangle$$

$$P[1,5] = \langle 0.04, \mathbf{0.1}, 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, \\ 0.05, \mathbf{0.2}, 0.06, \mathbf{0.1}, 0.01 \rangle$$

$$S[1,1] = \langle A \rangle,$$

$$P[1,1] = \langle 0.04, \mathbf{0.1}, 0.02 \rangle$$

$$S[3,5] = \langle C, D, E \rangle,$$

$$P[3,5] = \langle 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06, \mathbf{0.1}, 0.01 \rangle$$

子问题的概率之和

子问题界定 $S[i,j]$ 和 $P[i,j]$, 令

$$w[i,j] = \sum_{p=i-1}^j a_p + \sum_{q=i}^j b_q$$

是 $P[i,j]$ 中所有概率(数据与空隙)之和

实例: $S[2,4]=\langle B,C,D \rangle$

$$P[2,4]=\langle 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06 \rangle$$

$$w[2,4]=(\mathbf{0.3}+\mathbf{0.1}+\mathbf{0.2})$$

$$+(\mathbf{0.02}+\mathbf{0.02}+\mathbf{0.05}+\mathbf{0.06})$$

$$= \mathbf{0.75}$$

优化函数的递推方程

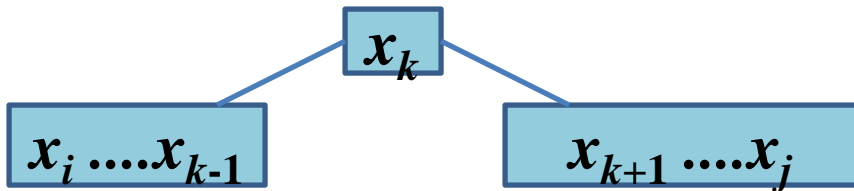
设 $m[i,j]$ 是相对于输入 $S[i,j]$ 和 $P[i,j]$ 的最优二叉搜索树的平均比较次数

递推方程:

$$m[i,j] = \min_{i \leq k \leq j} \{ \underline{m[i, k-1]} + \underline{m[k+1, j]} + w[i, j] \},$$

$$1 \leq i \leq j \leq n$$

$$m[i, i-1] = 0, \quad i = 1, 2, \dots, n$$



$m[i, j]_k$ 公式的证明

$m[i, j]_k$: 根为 x_k 时平均比较次数的最小值

作为子
树增加
次数

$$m[i, j]_k$$

$$= (m[i, k-1] + \underline{w[i, k-1]}) + (m[k+1, j] + \underline{w[k+1, j]}) + 1 \times b_k$$

$$= (m[i, k-1] + m[k+1, j]) + (w[i, k-1] + b_k + w[k+1, j])$$

$$= (m[i, k-1] + m[k+1, j]) + (\underbrace{\sum_{p=i-1}^{k-1} a_p + \sum_{q=i}^{k-1} b_q}_{\text{代入概率公式}}) + b_k + (\underbrace{\sum_{p=k}^j a_p + \sum_{q=k+1}^j b_q}_{\text{代入概率公式}})$$

代入概
率公式

$$= (m[i, k-1] + m[k+1, j]) + \sum_{p=i-1}^j a_p + \sum_{q=i}^j b_q$$

化简

$$= \boxed{m[i, k-1] + m[k+1, j] + w[i, j]}$$

递推方程

$$m[i, j] = \min_{i \leq k \leq j} \{m[i, k-1] + m[k+1, j] + w[i, j]\},$$

平均比较次数：在所有 k 的情况下

$m[i, j]_k$ 的最小值

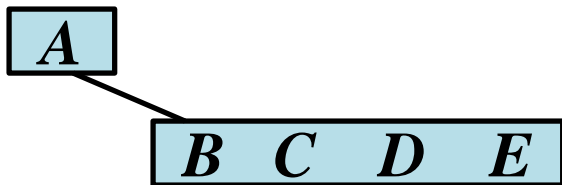
$$m[i, j] = \min \{ m[i, j]_k \mid i \leq k \leq j \}$$

初值 $m[i, i-1]=0$ 对应于空的子问题，

例如 $S = \langle A, B, C, D, E \rangle$ ，取 A 作根，

$i = 1, k=1$ ，左边子问题为空树，

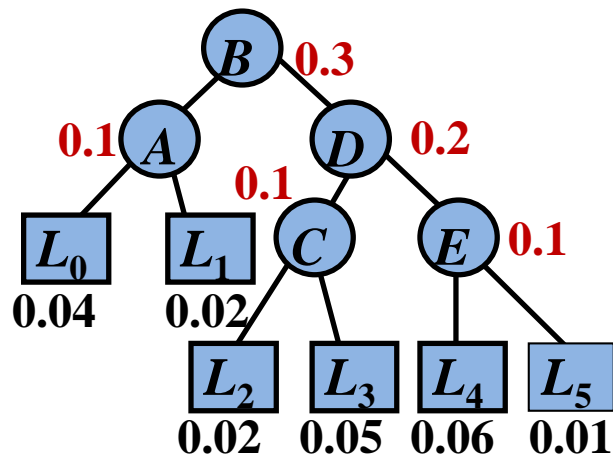
对应于： $S[1,0]$ ， $m[1,0]=0$ 的情况。



实例

$$m[i, j] = \min_{i \leq k \leq j} \{ m[i, k-1] + m[k+1, j] + w[i, j] \}$$

$$m[i, i-1] = 0$$



以B为根

$k=2$

$$m[1, 1] = 0.16$$

$$m[3, 5] = 0.88$$

$$m[1, 5] = 1 + \min_{k=2,3,4} \{ m[1, k-1] + m[k+1, 5] \}$$

$$= 1 + \{ m[1, 1] + m[3, 5] \} = 1 + \{ 0.16 + 0.88 \} = 2.04$$

计算复杂性估计

$$m[i, j] = \min_{i \leq k \leq j} \{m[i, k-1] + m[k+1, j] + w[i, j]\}$$

$$1 \leq i \leq j \leq n$$

$$m[i, i-1] = 0, \quad i = 1, 2, \dots, n$$

i, j 的所有组合 $O(n^2)$ 种

每种要对不同的 k 进行计算, $k = O(n)$

每次计算为常数时间

时间复杂性: $T(n) = O(n^3)$

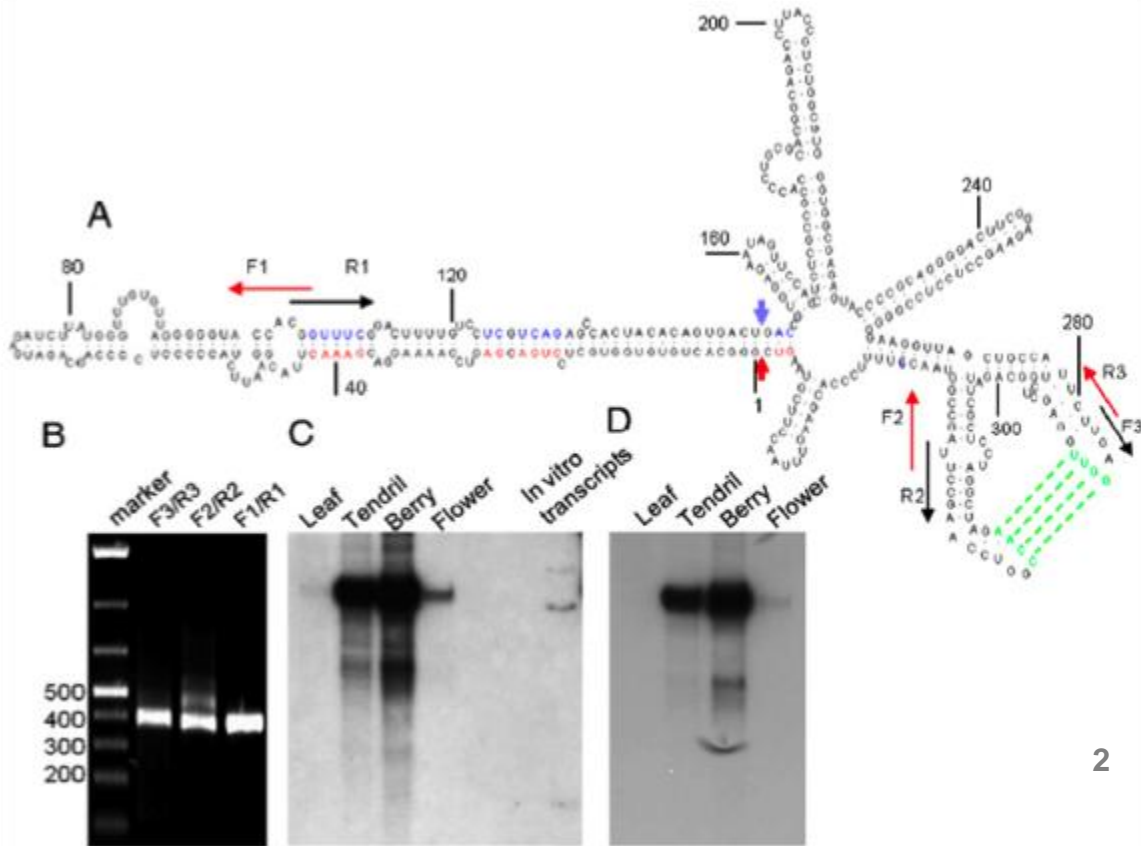
空间复杂度: $S(n) = O(n^2)$

小结

- 划分子问题,以数据结点作为树根
- 定义优化函数,列出递推方程与边界条件
- 自底向上计算,设计备忘录(表格)
- 设立标记函数记录构成最优二叉搜索树或子树时根的位置.
- 时间复杂度估计

RNA二级结构预测

375nt 的环形类病毒GHVd 的 RNA二级结构预测和RT-PCR、Northern blot检测结果

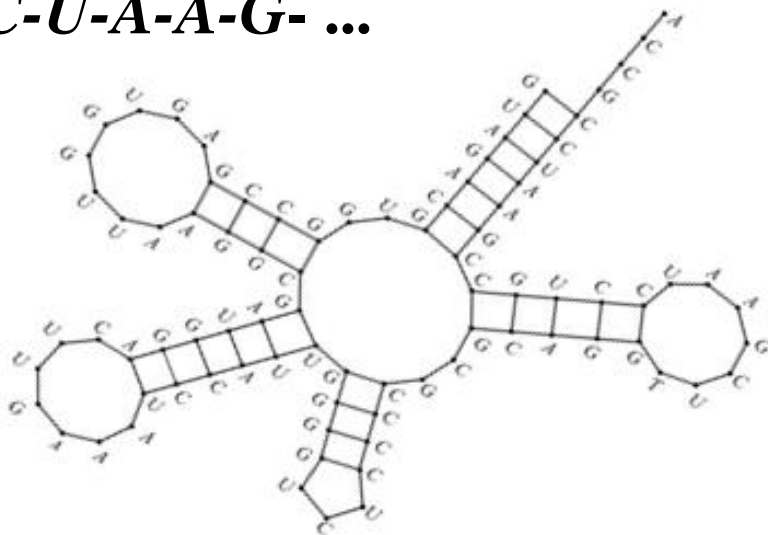


RNA二级结构

一级结构： 由字母 *A, C, G, U* 标记的核苷酸构成的一条链.

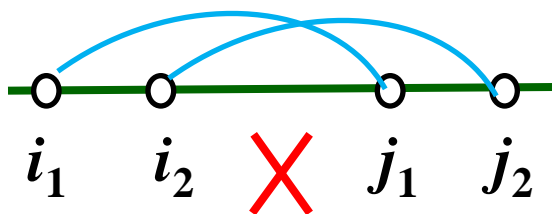
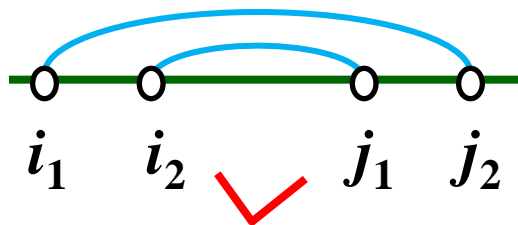
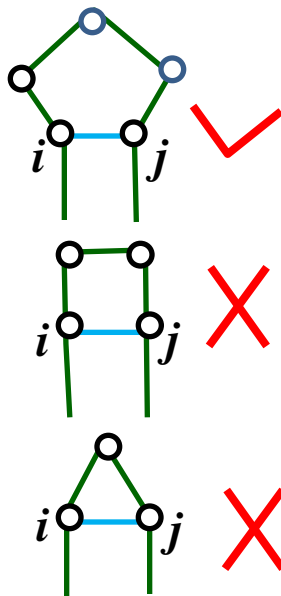
实例： *A-C-C-G-C-C-U-A-A-G-C-C-G-U-C-C-U-A-A-G- ...*

二级结构：
核苷酸相互
匹配构成的
平面结构

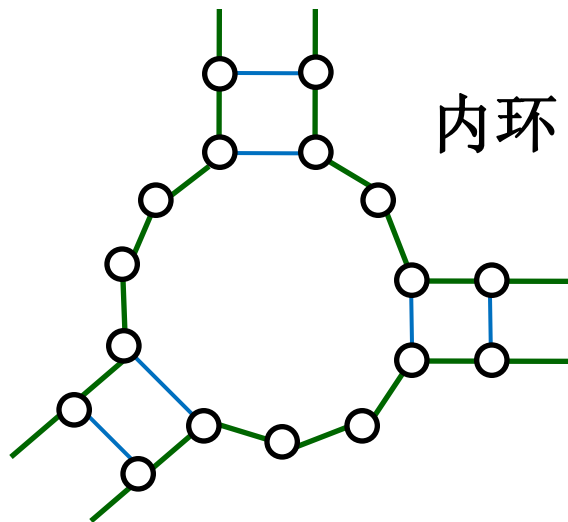
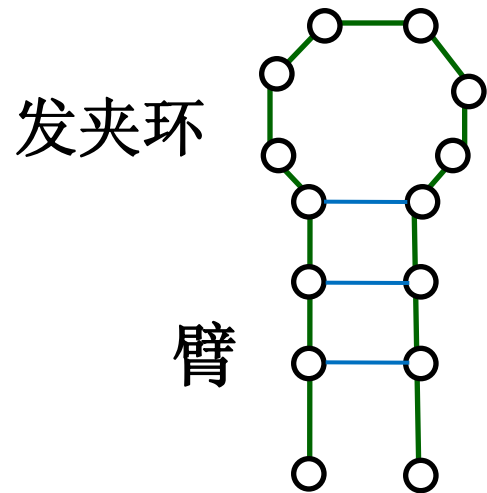


匹配原则

- 配对 $U-A$, $C-G$
- 末端不出现“尖角”，位置 $i-j$ 配对，则 $i \leq j-4$
- 每个核苷酸只能参加一个配对
- 不允许交叉，即如果位置 i_1, i_2, j_1, j_2 满足 $i_1 < i_2 < j_1 < j_2$ ，不允许 i_1-j_1, i_2-j_2 配对，但可以允许 i_1-j_2, i_2-j_1 配对。



匹配的结构



RNA二级结构问题

给定RNA的一条链（一级结构），预测它的可能的稳定的二级结构

稳定二级结构满足的条件

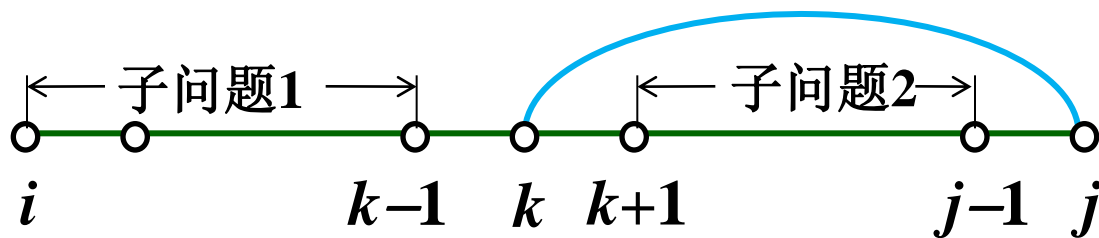
生物学条件：具有最小自由能

简化条件：具有最多的匹配对数

问题：给定RNA链，求具有最多匹配对数的二级结构，即最优结构。

建模

- 子问题界定：前边界 i , 后边界 j
- 若 j 与 k (所有可能) 位置匹配, 归约为
子问题 1: i 到 $k-1$ 的链
子问题 2: $k+1$ 到 $j-1$ 的链



- 若 j 不参与匹配, 则原问题归约为 i 到 $j-1$ 的子问题

优化函数的递推方程

令 $C[i,j]$ 是序列 $S[i..j]$ 的最大匹配对数

$$C[i,j] = \max\{C[i,j-1],$$
$$\max_{i \leq k \leq j-4} \{1 + C[i,k-1] + C[k+1,j-1]\}\}$$

$$1 \leq i, \quad j \leq n, \quad j - i \geq 4$$

$$C[i,j] = 0 \quad j - i < 4$$

满足优化原则

计算顺序：按照子问题长度计算

计算复杂度分析

子问题个数: i, j 对的组合有 $O(n^2)$ 个

对于给定的 i 和 j , j 需要考察与所有可能的 k 是否匹配, 其中 $i \leq k \leq j-4$, 需要 $O(n)$ 时间.

算法时间复杂度是 $O(n^3)$.

小结

- 划分子问题，确定子问题边界 i, j 与归约方法.
- 定义优化函数,列递推方程和初值.
- 自底向上计算，设计备忘录 (表格)
- 设立标记函数，记下最优划分位置
- 时间复杂度估计

序列比对

序列比对

- 为确定两个序列之间的相似性或同源性，将它们按照一定的规律排列，进行比对.
- 应用：
生物信息学中用于研究同源性，如蛋白质序列或 **DNA** 序列. 在比对中，错配与突变相对应，空位与插入或缺失相对应.
计算语言学中用于语言进化或文本相似性的研究.

序列之间的编辑距离

编辑距离：

给定两个序列 S_1 和 S_2 ,

通过一系列字符编辑（插入、删除、替换）等操作，将 S_1 转变成 S_2 .

完成这种转换所需要的最少的编辑操作个数称为 S_1 和 S_2 的编辑距离.

实例

`vintner` 转变成 `writers`,
编辑距离 ≤ 6

	<code>v i n t n e r</code>
删除 v:	<code>- i n t n e r</code>
插入 w:	<code>w i n t n e r</code>
插入 r:	<code>w r i n t n e r</code>
删除 n:	<code>w r i - t n e r</code>
删除 n:	<code>w r i t - e r</code>
插入 s:	<code>w r i t e r s</code>

子问题界定和归约

$S_1[1..n]$ 和 $S_2[1..m]$ 表示两个序列

子问题: $S_1[1..i]$ 和 $S_2[1..j]$, 边界 (i, j)

操作	归约子问题	编辑距离
删除 $S_1[i]$	$(i-1, j)$	+1
$S_1[i]$ 后插入 $S_2[j]$	$(i, j-1)$	+1
$S_1[i]$ 替换为 $S_2[j]$	$(i-1, j-1)$	+1
$S_1[i]=S_2[j]$	$(i-1, j-1)$	+0

优化函数的递推方程

$C[i,j]$: $S_1[1..i]$ 和 $S_2[1..j]$ 的编辑距离

$$C[i,j] = \min\{C[i-1,j]+1, C[i,j-1]+1, \\ C[i-1,j-1]+t[i,j]\}$$

$$t[i,j] = \begin{cases} 0 & S_1[i] = S_2[j] \\ 1 & S_1[i] \neq S_2[j] \end{cases}$$

$$C[0,j] = j,$$

$$C[i,0] = i$$

计算复杂度分析

- 子问题 由 i, j 界定,
有 $O(mn)$ 个子问题
- 每个子问题的计算
为常数时间
- 算法的时间复杂度是 $O(nm)$

动态规划算法设计要点

- (1) 引入参数来界定子问题的边界. 注意子问题的重叠程度.
- (2) 给出带边界参数的优化函数定义与优化函数的递推关系, 找到递推关系的初值.
- (3) 判断该优化问题是否满足优化原则.
- (4) 考虑是否需要标记函数.

动态规划算法设计要点(续)

- (5) 采用自底向上的实现技术，从最小的子问题开始迭代计算，计算中用备忘录保留优化函数和标记函数的值。
- (6) 动态规划算法的时间复杂度是对所有子问题(备忘录)的计算工作量求和(可能需要追踪解的工作量)
- (7) 动态规划算法一般使用较多的存储空间，这往往成为限制动态规划算法使用的瓶颈因素。

贪心法的例子： 活动选择问题

活动选择问题

输入： $S = \{1, 2, \dots, n\}$ 为 n 项活动的集合， s_i, f_i 分别为活动 i 的开始和结束时间。

活动 i 与 j 相容 $\Leftrightarrow s_i \geq f_j$ 或 $s_j \geq f_i$ 。

求：最大的两两相容的活动集 A

输入实例：

i	1	2	3	4	5	6	7	8	9	10
s_i	1	3	2	5	4	5	6	8	8	2
f_i	4	5	6	7	9	9	10	11	12	13

解： $\{1, 4, 8\}$

贪心算法

挑选过程是多步判断，每步依据某种“短视”的策略进行活动选择，选择时注意满足相容性条件。

策略1： 开始时间早的优先

排序使 $s_1 \leq s_2 \leq \dots \leq s_n$ ，从前向后挑选

策略2： 占用时间少的优先

排序使得 $f_1 - s_1 \leq f_2 - s_2 \leq \dots \leq f_n - s_n$ ，
从前向后挑选

策略3： 结束早的优先

排序使 $f_1 \leq f_2 \leq \dots \leq f_n$ ，从前向后挑选

策略1的反例

策略1：开始早的优先

反例： $S = \{1, 2, 3\}$

$s_1=0, f_1=20, s_2=2, f_2=5, s_3=8, f_3=15$

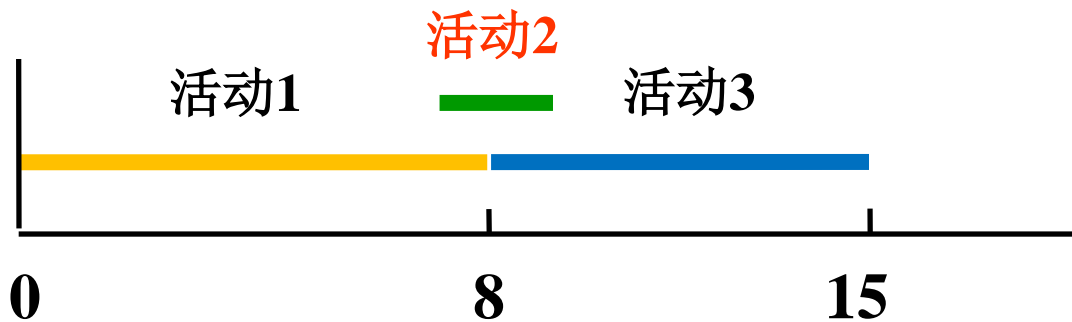


策略2的反例

策略2：占时少的优先

反例： $S = \{ 1, 2, 3 \}$

$s_1=0, f_1=8, s_2=7, f_2=9, s_3=8, f_3=15$



策略3伪码

算法 Greedy Select

输入：活动集 S , s_i, f_i ,

$i = 1, 2, \dots, n, f_1 \leq \dots \leq f_n$

输出： $A \subseteq S$, 选中的活动子集

1. $n \leftarrow \text{length}[S]$

2. $A \leftarrow \{1\}$

已选入的
最后标号

3. $j \leftarrow 1$

4. for $i \leftarrow 2$ to n do

5. if $s_i \geq f_j$

判相容

6. then $A \leftarrow A \cup \{i\}$

7. $j \leftarrow i$

8. return A

完成时间 $t = \max \{f_k : k \in A\}$

运行实例

输入: $S = \{ 1, 2, \dots, 10 \}$

i	1	2	3	4	5	6	7	8	9	10
s_i	1	3	0	5	3	5	6	8	8	2
f_i	4	5	6	7	8	9	10	11	12	13

解: $A = \{1, 4, 8\}$, $t = 11$

时间复杂度

$$O(n \log n) + O(n) = O(n \log n)$$



如何证明该算法对所有的实例
都得到正确的解?

贪心算法的特点

设计要素：

- (1) 贪心法适用于组合优化问题.
- (2) 求解过程是多步判断过程，最终的判断序列对应于问题的最优解.
- (3) 依据某种“短视的”贪心选择性质判断，性质好坏决定算法的成败.
- (4) 贪心法必须进行正确性证明.
- (5) 证明贪心法不正确的技巧：举反例.

贪心法的优势：算法简单，时间和空间复杂性低

贪心法正确性 证明：活动选择

一个数学归纳法的例子

例：证明对于任何自然数 n ,

$$1+2+\dots+n = n(n+1)/2$$

证 $n=1$, 左边=1, 右边= $1 \times (1+1)/2=1$

假设对任意自然数 n 等式成立, 则

$$1+2+\dots+(n+1)$$

$$= (1+2+\dots+n) + (n+1)$$

$$= n(n+1)/2 + (n+1)$$

$$= (n+1)(n/2+1)$$

$$= (n+1)(n+2)/2$$

归纳假设代入

第一数学归纳法

适合证明涉及自然数的命题 $P(n)$

归纳基础： 证明 $P(1)$ 为真 (或 $P(0)$ 为真).

归纳步骤： 若对所有 n 有 $P(n)$ 为真，证明
 $P(n+1)$ 为真

$$\forall n, P(n) \rightarrow P(n+1)$$

$$P(1)$$

$$n=1, P(1) \Rightarrow P(2)$$

$$n=2, P(2) \Rightarrow P(3)$$

...

第二数学归纳法

适合证明涉及自然数的命题 $P(n)$

归纳基础： 证明 $P(1)$ 为真 (或 $P(0)$ 为真).

归纳步骤： 若对所有小于 n 的 k 有 $P(k)$ 真,
证明 $P(n)$ 为真

$$\forall k (k < n \wedge P(k)) \rightarrow P(n)$$

$$P(1)$$

$$n=2, \quad P(1) \Rightarrow P(2)$$

$$n=3, \quad P(1) \wedge P(2) \Rightarrow P(3)$$

...

两种归纳法的区别

归纳基础一样 $P(1)$ 为真

归纳步骤不同

证明逻辑

归纳法1: $P(1) \Rightarrow P(2) \Rightarrow P(3) \dots$

归纳法2:

$$\left. \begin{array}{l} P(1) \\ P(1) \Rightarrow P(2) \end{array} \right\} \Rightarrow \left. \begin{array}{l} P(2) \\ P(3) \end{array} \right\} \Rightarrow P(4) \dots$$

算法正确性归纳证明

证明步骤:

1. 叙述一个有关自然数 n 的命题, 该命题断定该贪心策略的执行最终将导致最优解. 其中自然数 n 可以代表算法步数或者问题规模.
2. 证明命题对所有的自然数为真.
归纳基础(从最小实例规模开始)
归纳步骤(第一或第二数学归纳法)

活动选择算法的命题

命题

算法 Select 执行到第 k 步, 选择 k 项活动

$$i_1 = 1, i_2, \dots, i_k$$

则存在最优解 A 包含活动 $i_1=1, i_2, \dots, i_k$.

根据上述命题: 对于任何 k , 算法前 k 步的选择都将导致最优解, 至多到第 n 步将得到问题实例的最优解

归纳证明： 归纳基础

令 $S=\{1,2,\dots,n\}$ 是活动集, 且 $f_1 \leq \dots \leq f_n$

归纳基础: $k=1$, 证明存在最优解包含活动 1

证 任取最优解 A , A 中活动按截止时间递增排列. 如果 A 的第一个活动为 j , $j \neq 1$, 用 1 替换 A 的活动 j 得到解 A' , 即

$$A' = (A - \{j\}) \cup \{1\},$$

由于 $f_1 \leq f_j$, A' 也是最优解, 且含有 1.

$$f_1 \leq f_j$$



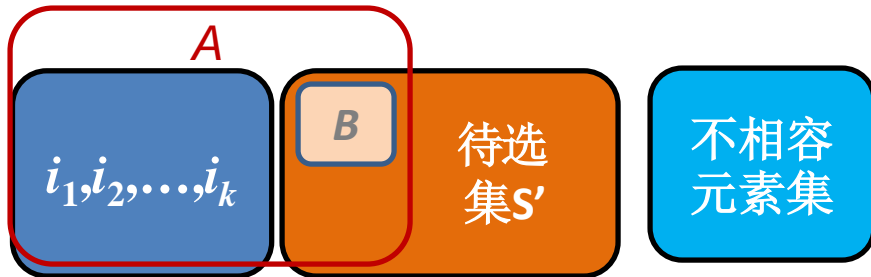
归纳步骤

假设命题对 k 为真, 证明对 $k+1$ 也为真.

证 算法执行到第 k 步, 选择了活动 $i_1=1, i_2, \dots, i_k$, 根据归纳假设存在最优解 A 包含 $i_1=1, i_2, \dots, i_k$, A 中剩下活动选自集合 S'

$$S' = \{ i \mid i \in S, s_i \geq f_k \}$$

$$A = \{ i_1, i_2, \dots, i_k \} \cup B$$

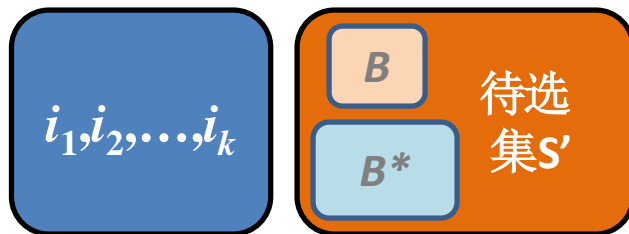


归纳步骤（续）

B 是 S' 的最优解.（若不然, S' 的最优解为 B^* , B^* 的活动比 B 多, 那么

$$B^* \cup \{1, i_2, \dots, i_k\}$$

是 S 的最优解, 且比 A 的活动多, 与 A 的最优性矛盾.)

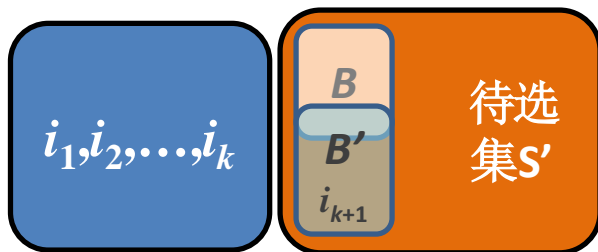


归纳步骤 (续)

将 S' 看成子问题，根据归纳基础，
存在 S' 的最优解 B' 有 S' 中的第一个
活动 i_{k+1} ，且 $|B'| = |B|$ ，于是

$$\begin{aligned} & \{i_1, i_2, \dots, i_k\} \cup B' \\ &= \{i_1, i_2, \dots, i_k, i_{k+1}\} \cup (B' - \{i_{k+1}\}) \end{aligned}$$

也是原问题的最优解.



小结

- 贪心法正确性证明方法：数学归纳法
第一数学归纳法、第二数学归纳法
- 活动选择问题的贪心法证明：
叙述一个涉及步数的算法正确性命题
证明归纳基础
证明归纳步骤

最优装载问题

最优装载问题

问题:

n 个集装箱 $1, 2, \dots, n$ 装上轮船, 集装箱 i 的重量 w_i , 轮船装载重量限制为 C , 无体积限制. 问如何装使得上船的集装箱最多? 不妨设每个箱子的重量 $w_i \leq C$.

该问题是0-1背包问题的子问题. 集装箱相当于物品, 物品重量是 w_i , 价值 v_i 都等于1, 轮船载重限制 C 相当于背包重量限制 b .

建模

设 $\langle x_1, x_2, \dots, x_n \rangle$ 表示解向量, $x_i = 0, 1$,
 $x_i = 1$ 当且仅当第 i 个集装箱装上船

目标函数 $\max \sum_{i=1}^n x_i$

约束条件 $\sum_{i=1}^n w_i x_i \leq C$

$$x_i = 0, 1 \quad i = 1, 2, \dots, n$$

算法设计

- 贪心策略：轻者优先
- 算法设计：
将集装箱排序，使得

$$w_1 \leq w_2 \leq \dots \leq w_n$$

按照标号从小到大装箱，直到装入下一个箱子将使得集装箱总重超过轮船装载重量限制，则停止.

正确性证明思路

- **命题：**对装载问题任何规模为 n 的输入实例，算法得到最优解。
- 设集装箱从轻到重记为 $1, 2, \dots, n$.

归纳基础 证明对任何只含 1 个箱子的输入实例，贪心法得到最优解。
显然正确。

- **归纳步骤** 证明：假设对于任何 n 个箱子的输入实例贪心法都能得到最优解，那么对于任何 $n+1$ 个箱子的输入实例贪心法也得到最优解。

归纳步骤证明思路

$N=\{1,2,\dots,n+1\}, w_1\leq w_2\leq\dots\leq w_{n+1}$



去掉箱子1, 令 $C' = C - \{w_1\}$,
得到规模 n 的输入 $N' = \{2,3,\dots,n+1\}$



关于输入 N' 和 C' 的最优解 I'



在 I' 加入箱子1, 得到 I



证明 I 是关于输入 N 的最优解

正确性证明

假设对于 n 个集装箱的输入，贪心法都可以得到最优解，考虑 输入

$$N = \{ 1, 2, \dots, n+1 \}$$

其中 $w_1 \leq w_2 \leq \dots \leq w_{n+1}$.

由归纳假设，对于

$$N' = \{ 2, 3, \dots, n+1 \}, \quad C' = C - w_1,$$

贪心法得到最优解 I' . 令

$$I = I' \cup \{1\}$$

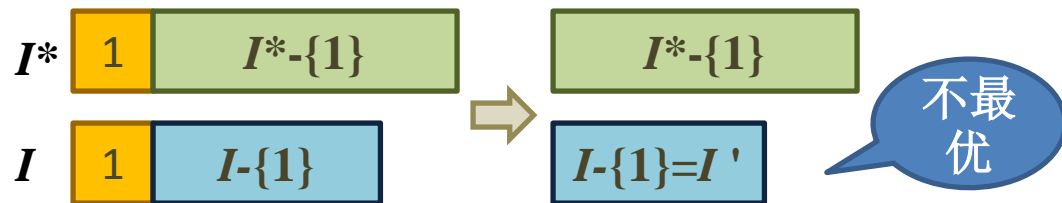
正确性证明 (续)

I (算法解) 是关于 N 的最优解.

若不然, 存在包含 1 的关于 N 的最优解 I^* (如果 I^* 中没有 1, 用 1 替换 I^* 中的第一个元素得到的解也是最优解), 且 $|I^*| > |I|$; 那么 $I^* - \{1\}$ 是 N' 和 C' 的解且

$$|I^* - \{1\}| > |I - \{1\}| = |I'|$$

与 I' 是关于 N' 和 C' 的最优解矛盾.



小结

- 装载问题是0-1背包的子问题 (每件物品重量为1)，NP难的问题存在多项式时间可解的子问题.
- 贪心法证明：对规模归纳

最小延迟调度问题

最小延迟调度

问题:

客户集合 A , $\forall i \in A$, t_i 为服务时间, d_i 为要求完成时间, t_i, d_i 为正整数. 一个调度 $f: A \rightarrow \mathbb{N}$, $f(i)$ 为客户 i 的开始时间. 求最大延迟达到最小的调度, 即求 f 使得

$$\min_f \{ \max_{i \in A} \{ f(i) + t_i - d_i \} \}$$

$$\forall i, j \in A, i \neq j, f(i) + t_i \leq f(j)$$

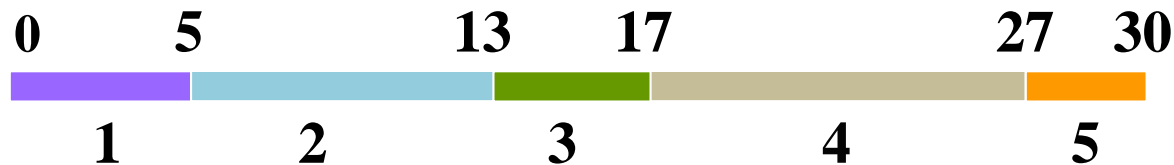
$$\text{or } f(j) + t_j \leq f(i)$$

实例：调度1

$A = \{1, 2, 3, 4, 5\}$, $T = \langle 5, 8, 4, 10, 3 \rangle$,
 $D = \langle 10, 12, 15, 11, 20 \rangle$

调度1：顺序安排

$f_1(1)=0, f_1(2)=5, f_1(3)=13, f_1(4)=17, f_1(5)=27$



各任务延迟：0, 1, 2, **16**, 10;

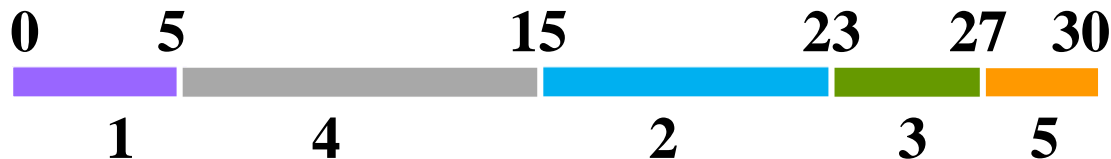
最大延迟：16

更优的调度2

$A = \{1, 2, 3, 4, 5\}$, $T = \langle 5, 8, 4, 10, 3 \rangle$,
 $D = \langle 10, 12, 15, 11, 20 \rangle$

调度2: 按截止时间从前到后安排

$f_2(1)=0, f_2(2)=15, f_2(3)=23, f_2(4)=5, f_2(5)=27$



各任务延迟: 0, 11, 12, 4, 10;

最大延迟: 12

贪心策略

贪心策略1: 按照 t_i 从小到大安排

贪心策略2: 按照 $d_i - t_i$ 从小到大安排

贪心策略3: 按照 d_i 从小到大安排

策略1 对某些实例得不到最优解.

反例: $t_1=1, d_1=100, t_2=10, d_2=10$

策略2 对某些实例得不到最优解.

反例: $t_1=1, d_1=2, t_2=10, d_2=10$

策略3伪码

算法 Schedule

输入: A, T, D

输出: f

1. 排序 A 使得 $d_1 \leq d_2 \leq \dots \leq d_n$

2. $f(1) \leftarrow 0$

从0时
刻起

3. $i \leftarrow 2$

4. while $i \leq n$ do

5. $f(i) \leftarrow f(i-1) + t_{i-1}$

没有
空闲

6. $i \leftarrow i + 1$

设计思想: 按完成时间从早到晚安排任务, 没有空闲.

交换论证：正确性证明

证明思路：

- 分析一般最优解与算法解的区别（成分,排列顺序不同）
- 设计一种转换操作（替换成分或交换次序），可以在有限步将任意一个普通最优解逐步转换成算法的解
- 上述每步转换都不降低解的最优性质

贪心算法的解的性质：

没有空闲时间，没有逆序。

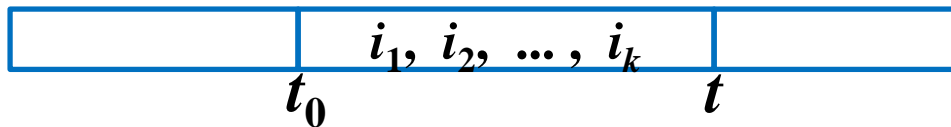
逆序 (i, j) : $f(i) < f(j)$ 且 $d_i > d_j$

引理

引理1 所有没有逆序、没有空闲时间的调度具有相同的最大延迟.

证: 设 f 没有逆序, 在 f 中具有相同完成时间 d 的客户 i_1, i_2, \dots, i_k 连续安排, 其开始时刻为 t_0 , 完成这些任务的时刻是 t , 最大延迟为最后任务延迟 $t-d$, 与 i_1, i_2, \dots, i_k 的排列次序无关.

$$t = t_0 + (t_{i_1} + t_{i_2} + \dots + t_{i_k})$$



证明要点

从一个没有空闲的最优解出发，逐步转变成没有逆序的解。根据引理 1，这个解和算法解具有相同的最大延迟。

- (1) 如果一个最优调度存在逆序，那么存在 $i < n$ 使得 $(i, i+1)$ 构成一个逆序，称为相邻的逆序。
- (2) 交换相邻逆序 i 和 j ，得到的解仍旧最优。
- (3) 每次交换后逆序数减 1，至多经过 $n(n-1)/2$ 次交换得到一个没有逆序的最优调度——等价于算法的解。

交换相邻逆序仍旧最优

设 f_1 是一个任意最优解，存在相邻逆序 (i, j) 。交换 i 和 j 的顺序，得到解 f_2 。那么 f_2 的最大延迟不超过 f_1 的最大延迟。

理由：

- (1) 交换 i, j 与其他客户延迟时间无关
- (2) 交换后不增加 j 的延迟，但可能增加 i 的延迟
- (3) i 在 f_2 的延迟小于 j 在 f_1 的延迟，因此小于 f_1 的最大延迟 r

i 在 f_2 的延迟不超过
 j 在 f_1 的延迟



$$\text{delay}(f_2, i) = \underline{s+t_j+t_i} - d_i$$

$$\text{delay}(f_1, j) = \underline{s+t_i+t_j} - d_j$$

$$d_j < d_i$$

$$\text{delay}(f_2, i) < \text{delay}(f_1, j) \leq r$$

小结

贪心法正确性证明方法：交换论证

- 分析算法解与一般最优解的区别, 找到把一般解改造成算法解的一系列操作(替换成份、交换次序)
- 证明操作步数有限
- 证明每步操作后的得到解仍旧保持最优

得不到最优解 的处理方法

得不到最优解的处理

- 输入参数分析：
考虑输入参数在什么取值范围内使用贪心法可以得到最优解
- 误差分析：
估计贪心法——近似算法所得到的解与最优解的误差（对所有的输入实例在最坏情况下误差的上界）

找零钱问题

问题 设有 n 种零钱，重量分别为 w_1, w_2, \dots, w_n ，价值分别为 $v_1 = 1, v_2, \dots, v_n$ 。需要付的总钱数是 y 。不妨设币值和钱数都为正整数。问：如何付钱使得所付钱的总重最轻？

实例

$v_1=1, v_2=5, v_3=14, v_4=18, w_i=1, i=1,2,3,4.$
 $y=28$

最优解： $x_3=2, x_1=x_2=x_4=0$ ，总重2

建模

令选用第 i 种硬币的数目是 x_i ,

$$i = 1, 2, \dots, n$$

$$\min\left\{\sum_{i=1}^n w_i x_i\right\}$$

$$\sum_{i=1}^n v_i x_i = y, \quad x_i \in \mathbf{N}, \quad i = 1, 2, \dots, n$$

动态规划算法

设 $F_k(y)$ 表示用前 k 种零钱，总钱数为 y 的最小重量

$$F_k(y) = \min_{0 \leq x_k \leq \left\lfloor \frac{y}{v_k} \right\rfloor} \{F_{k-1}(y - v_k x_k) + w_k x_k\}$$

$$F_1(y) = w_1 \left\lfloor \frac{y}{v_1} \right\rfloor = w_1 y$$

贪心法

单位价值重量轻的货币优先，设

$$\frac{w_1}{v_1} \geq \frac{w_2}{v_2} \geq \dots \geq \frac{w_n}{v_n}$$

使用前 k 种零钱，总钱数为 y
贪心法的总重为 $G_k(y)$,

$$G_k(y) = w_k \left\lfloor \frac{y}{v_k} \right\rfloor + G_{k-1}(y \bmod v_k) \quad k > 1$$

$$G_1(y) = w_1 \left\lfloor \frac{y}{v_1} \right\rfloor = w_1 y$$

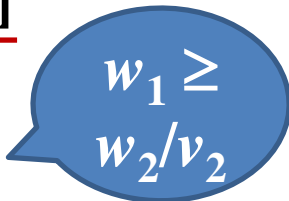
$n=1,2$ 贪心法是最优解

$n = 1$ 只有一种零钱, $F_1(y) = G_1(y)$

$n = 2$, x_2 越大, 得到的解越好

$$F_2(y) = \min_{0 \leq x_2 \leq \lfloor y/v_2 \rfloor} \{F_1(y - v_2 x_2) + w_2 x_2\}$$

$$\begin{aligned} & F_1(y - v_2(\underline{x_2 + \delta})) + w_2(\underline{x_2 + \delta}) \\ & - [F_1(y - v_2 \underline{x_2}) + w_2 \underline{x_2}] \\ = & [w_1(\underline{y - v_2 x_2 - v_2 \delta}) + \underline{w_2 x_2 + w_2 \delta}] \\ & - [w_1(\underline{y - v_2 x_2}) + \underline{w_2 x_2}] \\ = & -w_1 v_2 \delta + w_2 \delta \\ = & \delta(-w_1 v_2 + w_2) \leq 0 \end{aligned}$$


$$w_1 \geq w_2/v_2$$

判别条件

定理 对每个正整数 k , 假设对所有非负整数 y 有 $G_k(y) = F_k(y)$, 且存在 p 和 δ 满足

$$v_{k+1} = pv_k - \delta,$$

其中 $0 \leq \delta < v_k$, $v_k \leq v_{k+1}$, p 为正整数,

则下面的命题等价:

- (1) $G_{k+1}(y) = F_{k+1}(y)$ 对一切正整数 y ;
- (2) $G_{k+1}(pv_k) = F_{k+1}(pv_k)$;
- (3) $w_{k+1} + G_k(\delta) \leq pw_k$.

几点说明

- 根据条件(1)与(3)的等价性，可以对 $k = 3, 4, \dots, n$ ，依次利用条件(3)对贪心法是否得到最优解做出判别.
- 条件(3)验证 1 次需 $O(k)$ 时间， $k = O(n)$ ，整个验证时间 $O(n^2)$
- 条件(2)是条件(1) 在 $y = pv_k$ 时的特殊情况. 若条件(1)成立，显然有条件(2)成立. 反之，若条件(2)不成立，则条件(1)不成立，钱数 $y = pv_k$ 恰好提供了一个贪心法不正确的反例.

验证实例

$$v_{k+1} = pv_k - \delta, \quad 0 \leq \delta < v_k, \quad p \in \mathbb{Z}^+$$
$$w_{k+1} + G_k(\delta) \leq pw_k$$

例 $v_1=1, v_2=5, v_3=14, v_4=18, w_i=1, i=1,2,3,4$. 对一切 y 有

$$G_1(y)=F_1(y), \quad G_2(y)=F_2(y).$$

验证 $G_3(y) = F_3(y)$

$$v_3 = pv_2 - \delta \Rightarrow p = 3, \delta = 1.$$

$$w_3 + G_2(\delta) = 1 + 1 = 2$$

$$pw_2 = 3 \times 1 = 3$$

$$w_3 + G_2(\delta) \leq pw_2$$

贪心法对于 $n=3$ 的实例得到最优解

验证实例

$$\begin{aligned} v_{k+1} &= pv_k - \delta, \quad 0 \leq \delta < v_k, \quad p \in \mathbb{Z}^+ \\ w_{k+1} + G_k(\delta) &\leq pw_k \end{aligned}$$

例 $v_1=1, v_2=5, v_3=14, v_4=18, w_i=1,$

$i=1,2,3,4$. 对一切 y 有

$$G_1(y)=F_1(y), G_2(y)=F_2(y), G_3(y)=F_3(y)$$

验证 $G_4(y) = F_4(y)$

$$v_4 = pv_3 - \delta \Rightarrow p=2, \delta=10$$

$$w_4 + G_3(\delta) = 1 + 2 = 3$$

$$pw_3 = 2 \times 1 = 2$$

$$w_4 + G_3(\delta) > pw_3$$

$n=4, y=pv_3=28,$

最优解: $x_3=2$, 贪心法: $x_4=1, x_2=2$ 11

小结

- 贪心策略不一定得到最优解，在这种情况下可以有两种处理方法：
 - (1) 参数化分析：分析参数取什么值可得到最优解
 - (2) 估计贪心法得到的解在最坏情况下与最优解的误差
- 一个参数化分析的例子:找零钱问题

最优前缀码

二元前缀码

- 二元前缀码

用0-1字符串作为代码表示字符，要求任何字符的代码都不能作为其它字符代码的前缀

- 非前缀码的例子

$a: 001, b: 00, c: 010, d: 01$

- 解码的歧义，例如字符串 0100001

解码1: 01, 00, 001 d, b, a

解码2: 010, 00, 01 c, b, d

前缀码的二叉树表示

前缀码：

$$\{00000, 00001, 0001, 001, 01, 100, 101, 11\}$$

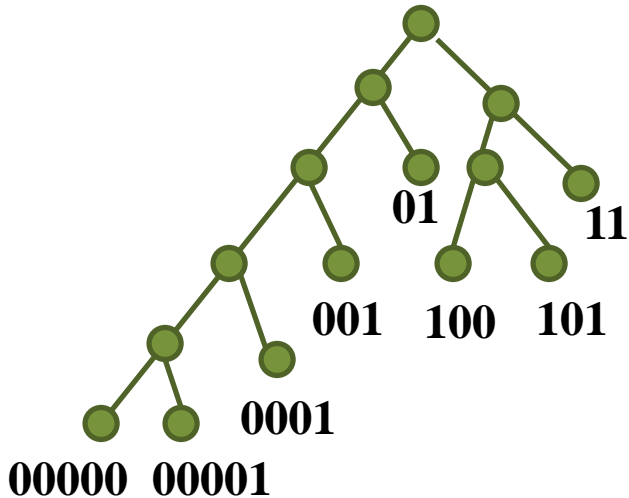
构造树:

0-左子树

1-右子树

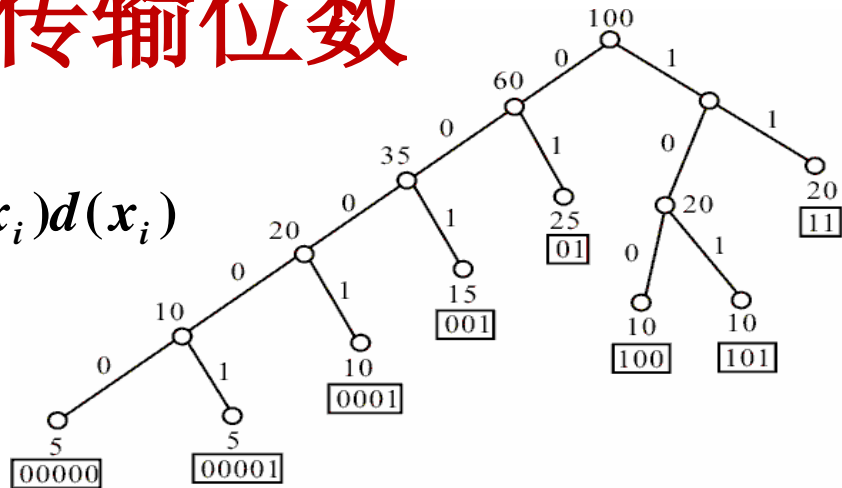
码对应一片树叶

最大位数为树深



平均传输位数

$$B = \sum_{i=1}^n f(x_i) d(x_i)$$



$$B = [(5+5) \times 5 + 10 \times 4 + (15+10+10) \times 3 + (25+20) \times 2] / 100 = 2.85$$

问题： 给定字符集 $C = \{x_1, x_2, \dots, x_n\}$ 和每个字符的频率 $f(x_i)$, $i=1, 2, \dots, n$. 求关于 C 的一个**最优前缀码**(平均传输位数最小).

哈夫曼树算法伪码

算法 Huffman(C)

输入: $C = \{x_1, x_2, \dots, x_n\}, f(x_i), i=1, 2, \dots, n.$

输出: Q //队列

1. $n \leftarrow |C|$
2. $Q \leftarrow C$ //频率递增队列 Q
3. for $i \leftarrow 1$ to $n-1$ do
4. $z \leftarrow \text{Allocate-Node}()$ //生成结点 z
5. $z.\text{left} \leftarrow Q$ 中最小元 //最小作 z 左儿子
6. $z.\text{right} \leftarrow Q$ 中最小元 //最小作 z 右儿子
7. $f(z) \leftarrow f(x) + f(y)$
8. Insert(Q, z) // 将 z 插入 Q
9. return Q

实例

输入 $a:45; b:13; c:12; d:16; e:9; f:5$

编码:

f --0000,

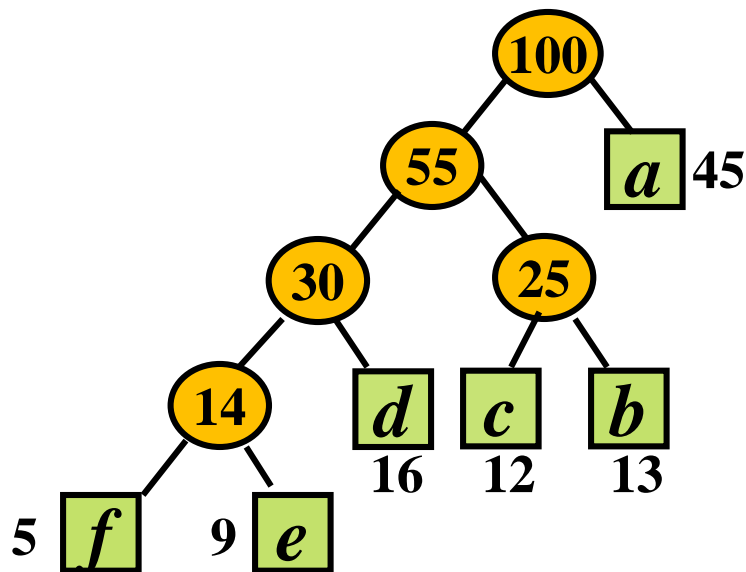
e --0001,

d --001,

c --010,

b —011,

a --1

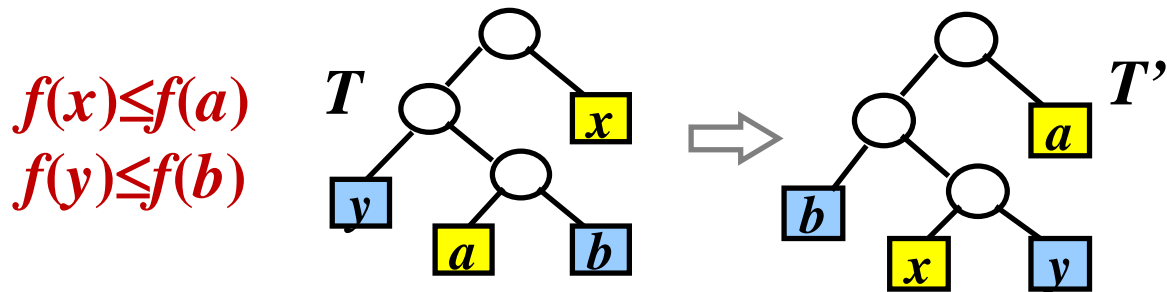


平均位数:

$$4 \times (0.05 + 0.09) + 3 \times (0.16 + 0.12 + 0.13) + 1 \times 0.45 = 2.24$$

最优前缀码性质：引理1

引理1: C 是字符集, $\forall c \in C, f(c)$ 为频率, $x, y \in C$, $f(x), f(y)$ 频率最小, 那么存在最优二元前缀码使得 x, y 码字等长且仅在最后一位不同.



$$B(T) - B(T') = \sum_{i \in C} f[i]d_T(i) - \sum_{i \in C} f[i]d_{T'}(i) \geq 0$$

其中 $d_T(i)$ 为 i 在 T 中的层数 (i 到根的距离)

引理2

引理 设 T 是二元前缀码的二叉树, $\forall x, y \in T$, x, y 是树叶兄弟, z 是 x, y 的父亲, 令

$$T' = T - \{x, y\}$$

且令 z 的频率

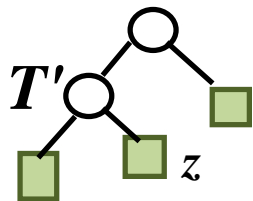
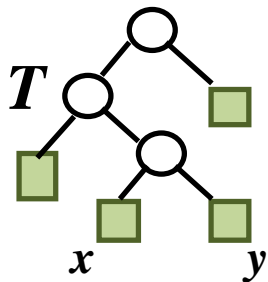
$$f(z) = f(x) + f(y)$$

T' 是对应二元前缀码

$$C' = (C - \{x, y\}) \cup \{z\}$$

的二叉树, 那么

$$B(T) = B(T') + f(x) + f(y)$$



引理2证明

证 $\forall c \in C - \{x, y\}$, 有

$$d_T(c) = d_{T'}(c) \Rightarrow f(c)d_T(c) = f(c)d_{T'}(c)$$

$$d_T(x) = d_{T'}(y) = d_{T'}(z) + 1$$

$$B(T) = \sum_{i \in T} f(i)d_T(i)$$

$$= \sum_{i \in T, i \neq x, y} f(i)d_T(i) + f(x)d_T(x) + f(y)d_T(y)$$

$$= \sum_{i \in T', i \neq z} f(i)d_{T'}(i) + f(z)d_{T'}(z) + (f(x) + f(y))$$

$$= B(T') + f(x) + f(y)$$

小结

- 二元前缀码及其二叉树表示
- 给定频率下的平均传输位数计算公式
- 最优前缀码——平均传输位数最少
- 哈夫曼算法
- 前缀码的性质

哈夫曼算法 的证明及应用

两个引理

引理1: 设 C 是字符集, $\forall c \in C, f(c)$ 为频率, $x, y \in C, f(x), f(y)$ 频率最小, 那么存在最优二元前缀码使得 x, y 码字等长, 且仅在最后一位不同.

引理2 设 T 是二元前缀码所对应的二叉树, $\forall x, y \in T, x, y$ 是树叶兄弟, z 是 x, y 的父亲, 令 $T' = T - \{x, y\}$, 且令 z 的频率 $f(z) = f(x) + f(y)$, T' 是对应于二元前缀码 $C' = (C - \{x, y\}) \cup \{z\}$ 的二叉树, 那么 $B(T) = B(T') + f(x) + f(y)$.

算法正确性证明思路

定理 Huffman 算法对任意规模为 n ($n \geq 2$) 的字符集 C 都得到关于 C 的最优前缀码的二叉树.

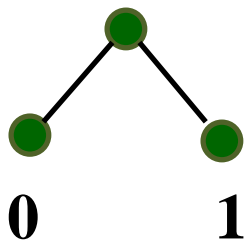
归纳基础 证明: 对于 $n=2$ 的字符集, Huffman算法得到最优前缀码.

归纳步骤 证明: 假设Huffman算法对于规模为 k 的字符集都得到最优前缀码, 那么对于规模为 $k+1$ 的字符集也得到最优前缀码.

归纳基础

$n=2$, 字符集 $C=\{x_1, x_2\}$,

对任何代码的字符至少都需要1位二进制数字. **Huffman**算法得到的代码是 0 和 1, 是最优前缀码.



归纳步骤

假设Huffman算法对于规模为 k 的字符集都得到最优前缀码. 考虑规模为 $k+1$ 的字符集

$$C = \{x_1, x_2, \dots, x_{k+1}\},$$

其中 $x_1, x_2 \in C$ 是频率最小的两个字符.

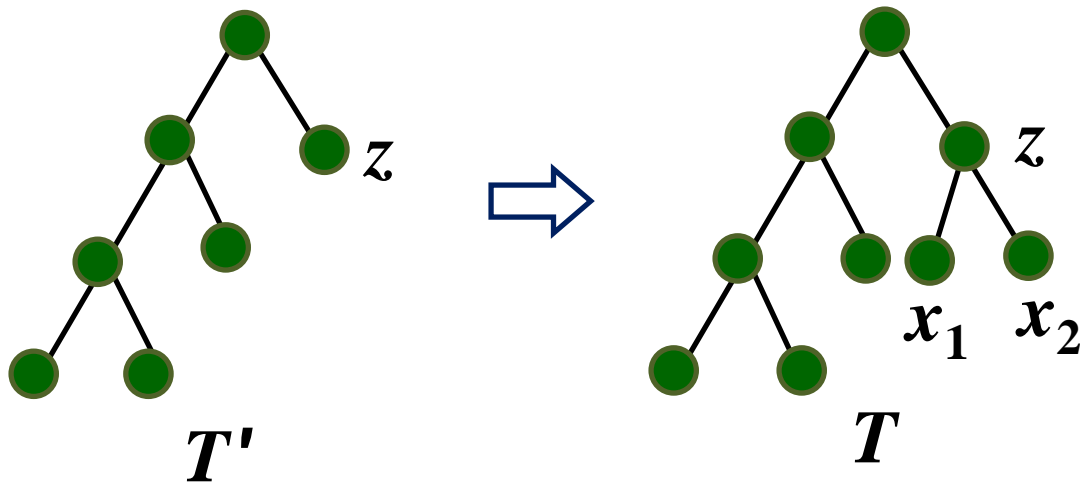
令

$$C' = (C - \{x_1, x_2\}) \cup \{z\},$$
$$f(z) = f(x_1) + f(x_2)$$

根据归纳假设, 算法得到一棵关于字符集 C' , 频率 $f(z)$ 和 $f(x_i)$ ($i=3, 4, \dots, k+1$) 的最优前缀码的二叉树 T' .

归纳步骤(续)

把 x_1, x_2 作为 z 的儿子附到 T' 上, 得到树 T , 那么 T 是关于 $C=(C'-\{z\})\cup\{x_1, x_2\}$ 的最优前缀码的二叉树.



归纳步骤 (续)

如若不然, 存在更优树 T^* , $B(T^*) < B(T)$,
且由引理1, 其树叶兄弟是 x_1 和 x_2 .

去掉 T^* 中 x_1 和 x_2 , 得到 $T^{*'}$. 根据引理2

$$\begin{aligned} B(T^{*'}) &= B(T^*) - \underline{(f(x_1) + f(x_2))} \\ &< B(T) - \underline{(f(x_1) + f(x_2))} \\ &= B(T') \end{aligned}$$

与 T' 是一棵关于 C' 的最优前缀码的二叉树矛盾.

应用：文件归并

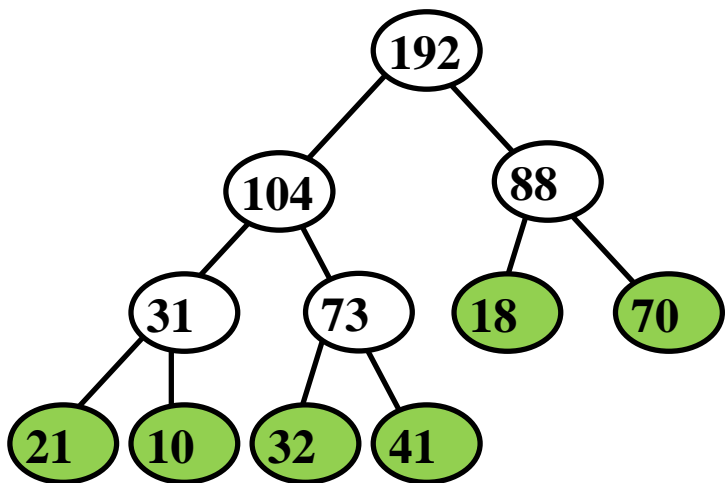
问题：给定一组不同长度的排好序文件构成的集合 $S = \{f_1, \dots, f_n\}$, 其中 f_i 表示第 i 个文件含有的项数. 使用二分归并将这些文件归并成一个有序文件.

归并过程对应于二叉树：

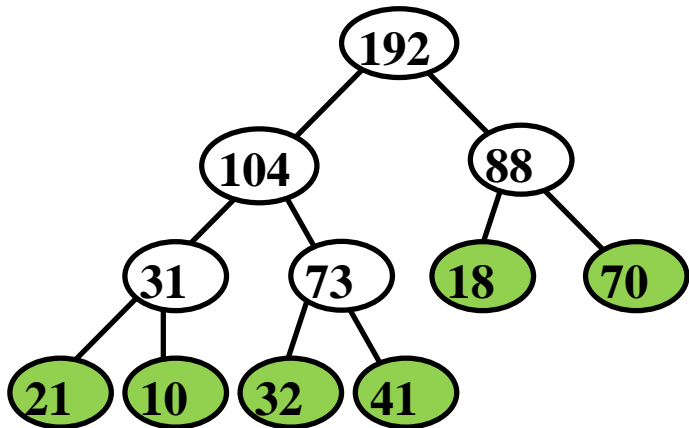
文件为树叶. f_i 与 f_j 归并的文件是它们的父结点.

两两顺序归并

实例： $S = \{ 21, 10, 32, 41, 18, 70 \}$



归并代价



$$(1) (21+10-1)+(32+41-1)+(18+70-1)+ \\ (31+73-1)+(104+88-1) = 483$$

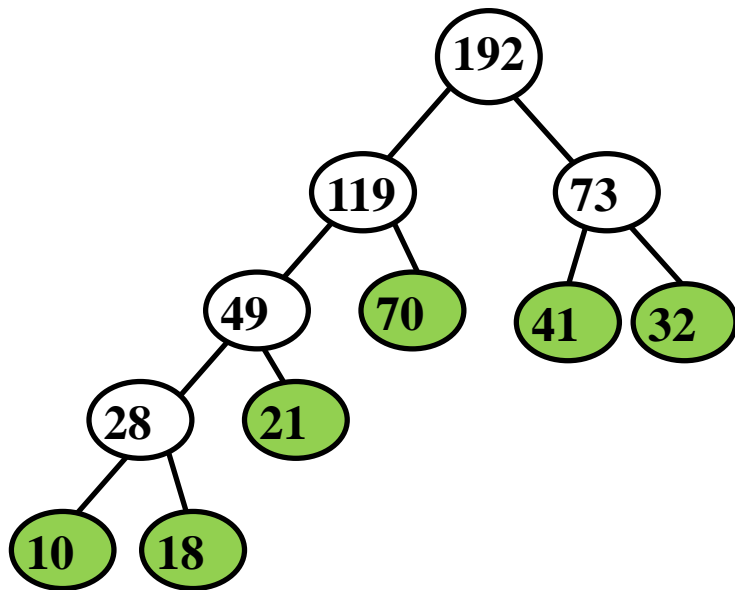
$$(2) (21+10+32+41) \times 3 + (18+70) \times 2 - 5 = 483$$

代价计算公式

$$\sum_{i \in S} d(i) f_i - (n - 1)$$

实例：Huffman树归并

输入： $S=\{21,10,32,41,18,70\}$



代价: $(10+18) \times 4 + 21 \times 3 + (70+41+32) \times 2 - 5 = 456$

小结

- 哈夫曼算法的正确性证明：
对规模归纳
- 哈夫曼算法的应用：
文件归并

最小生成树

最小生成树

无向连通带权图

$$G = (V, E, W),$$

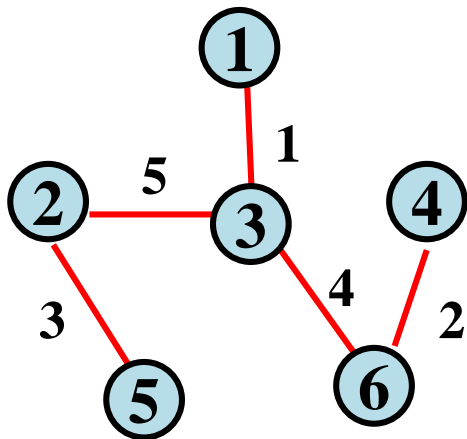
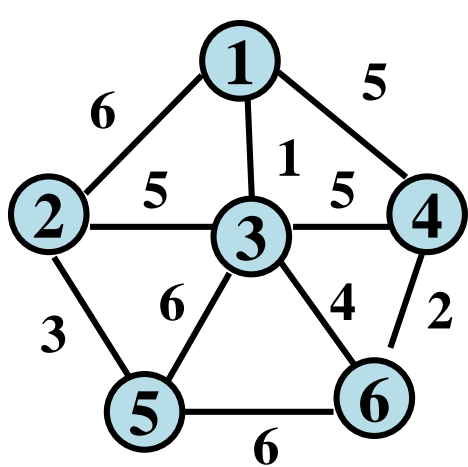
其中 $w(e) \in W$ 是边 e 的权.

G 的一棵生成树 T 是包含了 G 的所有顶点的树, 树中各边的权之和 $W(T)$ 称为树的权, 具有最小权的生成树称为 G 的最小生成树.

最小生成树的实例

$G=(V,E,W), V=\{1,2,3,4,5,6\}, W$ 如图所示.

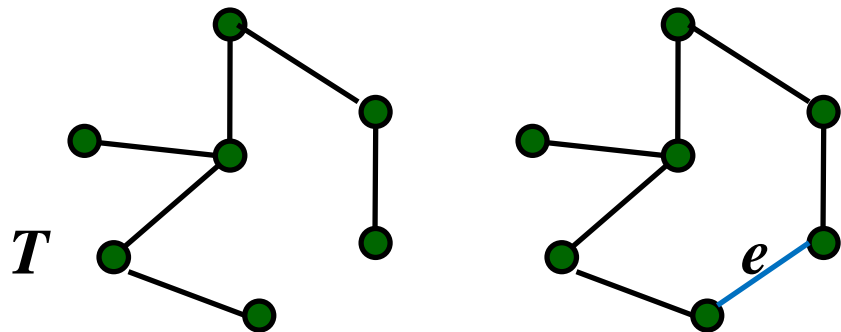
$E = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,5\},$
 $\{3,4\},\{3,5\},\{3,6\},\{4,6\},\{5,6\}\}$



生成树的性质

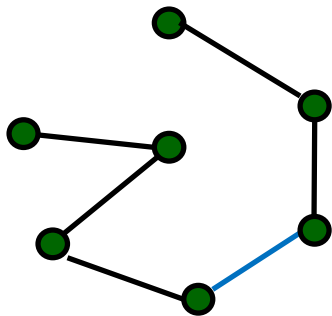
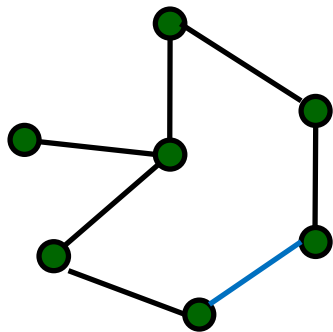
命题1 设 G 是 n 阶连通图, 那么

- (1) T 是 G 的生成树当且仅当 T 无圈且有 $n-1$ 条边.
- (2) 如果 T 是 G 的生成树, $e \notin T$, 那么 $T \cup \{e\}$ 含有一个圈 C (回路).



生成树的性质（续）

(3) 去掉圈 C 的任意一条边，就得到 G 的另外一棵生成树 T' 。



T'

生成树性质的应用

- 算法步骤：选择边。
约束条件：不形成回路
截止条件：边数达到 $n-1$.

- 改进生成树 T 的方法

在 T 中加一条非树边 e , 形成回路 C , 在 C 中去掉一条树边 e_i , 形成一棵新的生成树 T'

$$W(T') - W(T) = W(e) - W(e_i)$$

若 $W(e) \leq W(e_i)$, 则 $W(T') \leq W(T)$

求最小生成树

问题:

给定连通带权图 $G = (V, E, W)$,
 $w(e) \in W$ 是边 e 的权. 求 G 的一棵最小生成树.

贪心法:

Prim 算法,
Kruskal 算法

生成树在网络中有着重要应用

小结

- 生成树与生成树的权
- 最小生成树
- 生成树的性质

Prim算法

设计思想

输入：图 $G=(V,E,W)$, $V=\{1,2,\dots,n\}$

输出：最小生成树 T

设计思想：

初始 $S = \{1\}$,

选择连接 S 与 $V-S$ 集合的最短边 $e = \{i,j\}$, 其中 $i \in S, j \in V-S$. 将 e 加入树 T , j 加入 S .

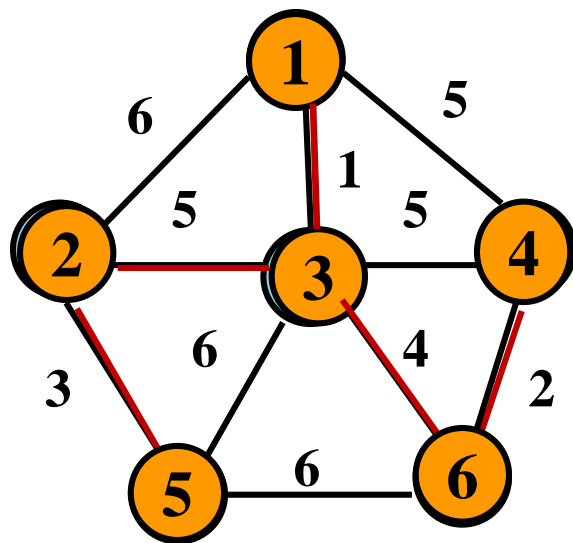
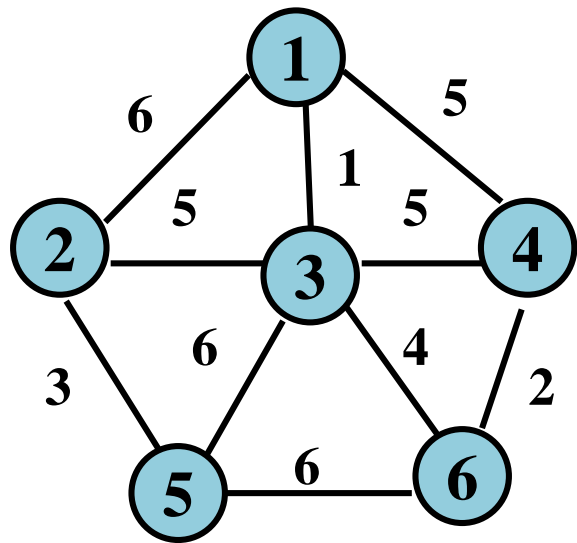
继续执行上述过程, 直到 $S=V$ 为止.

伪码

算法 Prim (G, E, W)

1. $S \leftarrow \{ 1 \}$
2. while $V - S \neq \emptyset$ do
3. 从 $V - S$ 中选择 j 使得 j 到 S
 中顶点的边权最小
4. $S \leftarrow S \cup \{ j \}$

实例



正确性证明: 归纳法

命题: 对于任意 $k < n$, 存在一棵最小生成树包含算法前 k 步选择的边.

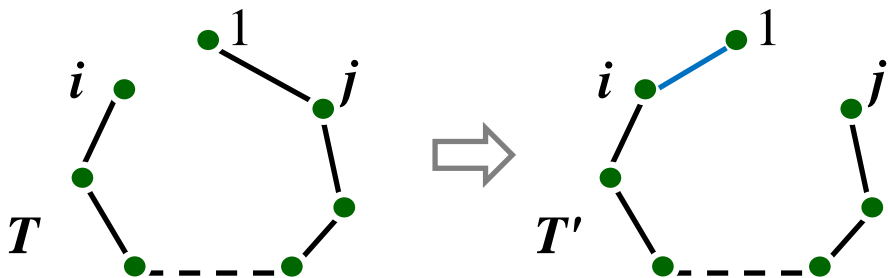
归纳基础: $k = 1$, 存在一棵最小生成树 T 包含边 $e = \{1, i\}$, 其中 $\{1, i\}$ 是所有关联 1 的边中权最小的.

归纳步骤: 假设算法前 k 步选择的边构成一棵最小生成树的边, 则算法前 $k+1$ 步选择的边也构成一棵最小生成树的边.

归纳基础

证明： 存在一棵最小生成树 T 包含关联结点1的最小权的边 $e=\{1,i\}$.

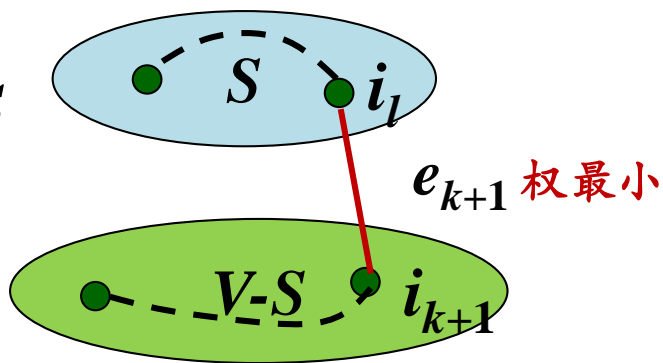
证 设 T 为一棵最小生成树，假设 T 不包含 $\{1,i\}$ ，则 $T \cup \{\{1,i\}\}$ 含有一条回路，回路中关联1的另一条边 $\{1,j\}$. 用 $\{1,i\}$ 替换 $\{1,j\}$ 得到树 T' ，则 T' 也是生成树，且 $W(T') \leq W(T)$.



归纳步骤

假设算法进行了 k 步，生成树的边为 e_1, e_2, \dots, e_k ，这些边的端点构成集合 S 。由归纳假设存在 G 的一棵最小生成树 T 包含这些边。

算法第 $k+1$ 步选择顶点 i_{k+1} ，则 i_{k+1} 到 S 中顶点边权最小，设此边 $e_{k+1}=\{i_{k+1}, i_l\}$ 。若 $e_{k+1} \in T$ ，算法 $k+1$ 步显然正确。



归纳步骤（续）

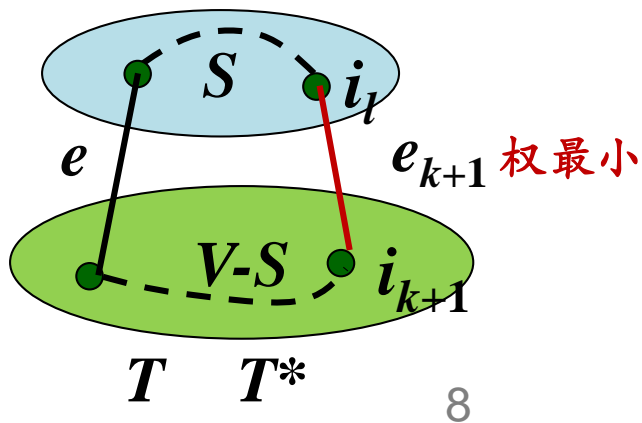
假设 T 不含有 e_{k+1} ，则将 e_{k+1} 加到 T 中形成一条回路。这条回路有另外一条连接 S 与 $V-S$ 中顶点的边 e ，

令 $T^* = (T - \{e\}) \cup \{e_{k+1}\}$

则 T^* 是 G 的一棵生成树，包含 e_1, e_2, \dots, e_{k+1} ，且

$$W(T^*) \leq W(T)$$

算法到 $k+1$ 步仍得到最小生成树。



时间复杂度

算法步骤执行 $O(n)$ 次

每次执行 $O(n)$ 时间：

找连接 S 与 $V-S$ 的最短边

算法时间： $T(n) = O(n^2)$

小结

- **Prim算法的设计**
贪心策略：连接 S 与 $V-S$ 的最短边
正确性证明：对步数归纳
伪码
- 时间复杂度： $O(n^2)$

Kruskal算法

设计思想

输入：图 $G=(V,E,W)$, $V=\{1,2,\dots,n\}$

输出： G 的最小生成树 T

设计思想：

- (1) 按照长度从小到大对边排序.
- (2) 依次考察当前最短边 e , 如果 e 与 T 的边不构成回路, 则把 e 加入树 T , 否则跳过 e . 直到选择了 $n-1$ 条边为止.

伪码

算法 Kruskal

输入：连通图 G // 顶点数 n ，边数 m

输出： G 的最小生成树

1. 权从小到大排序 E 的边, $E=\{e_1, e_2, \dots, e_m\}$

2. $T \leftarrow \emptyset$

3. repeat

4. $e \leftarrow E$ 中的最短边

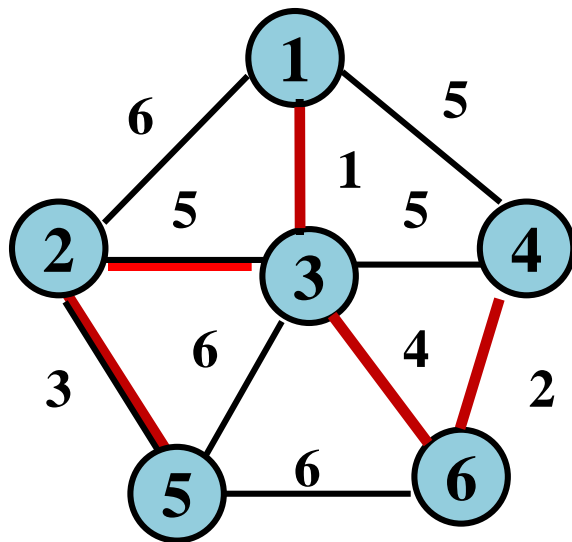
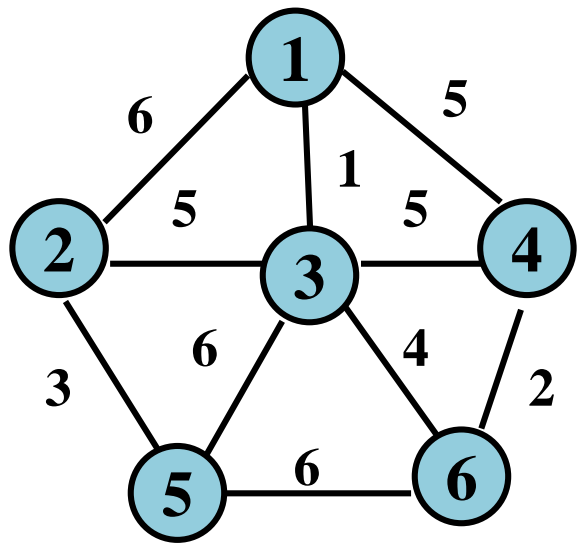
5. if e 的两端点不在同一连通分支

6. then $T \leftarrow T \cup \{e\}$

7. $E \leftarrow E - \{e\}$

8. until T 包含了 $n-1$ 条边

实例



正确性证明思路

命题：对于任意 n , 算法对 n 阶图找到一棵最小生成树.

证明思路：

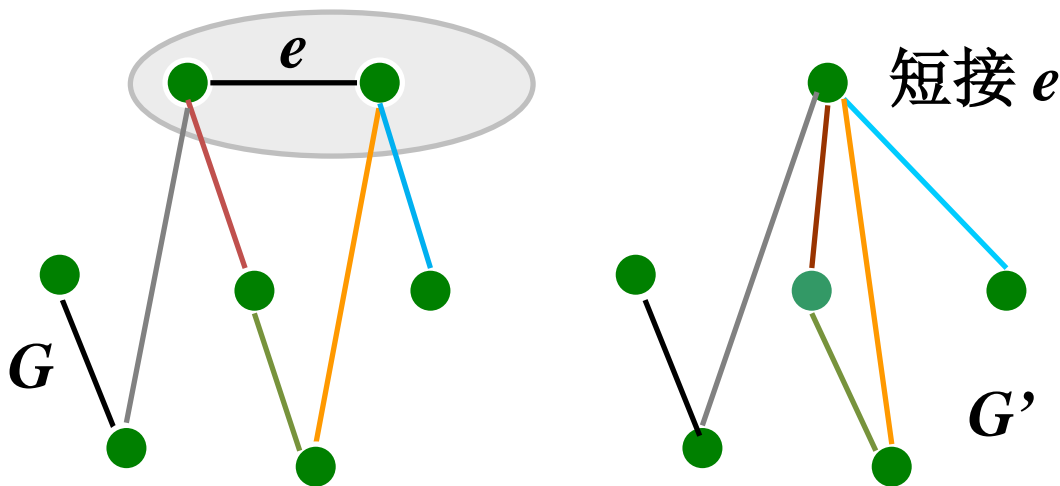
归纳基础 证明： $n = 2$, 算法正确.

G 只有一条边，最小生成树就是 G .

归纳步骤 证明：假设算法对于 n 阶图是正确的，其中 $n > 1$ ，则对于任何 $n+1$ 阶图算法也得到一棵最小生成树.

短接操作

任给 $n+1$ 个顶点的图 G , G 中最小权边 $e = \{i, j\}$, 从 G 中短接 i 和 j , 得到图 G' .



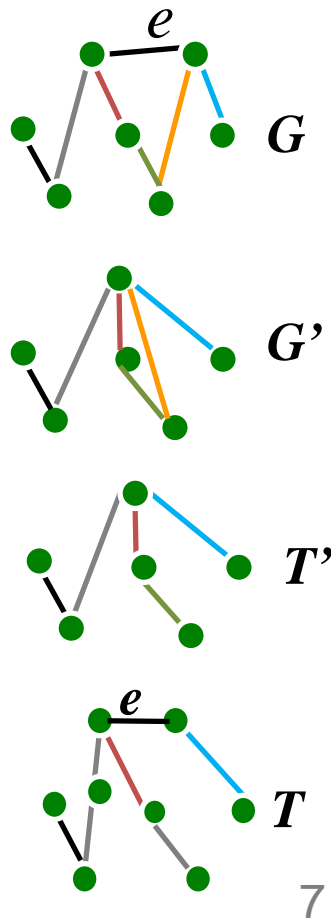
归纳步骤证明

对于任意 $n+1$ 阶图 G 短接最短边 e , 得到 n 阶图 G'

根据归纳假设算法得到 G' 的最小生成树 T'

将被短接的边 e “拉伸”回到原来长度, 得到树 T

证明 T 是 G 的最小生成树



T 是 G 的最小生成树

$T = T' \cup \{e\}$ 是关于 G 的最小生成树.

否则存在 G 的含边 e 的最小生成树 T^* , $W(T^*) < W(T)$. (如果 $e \notin T^*$, 在 T^* 中加边 e , 形成回路. 去掉回路中任意别的边所得生成树的权仍旧最小).

在 T^* 短接 e 得到 G' 的生成树 $T^* - \{e\}$,

$$\begin{aligned} W(T^* - \{e\}) &= W(T^*) - w(e) \\ &< W(T) - w(e) = W(T') \end{aligned}$$

与 T' 的最优性矛盾.

算法实现与时间复杂度

建立FIND数组， $\text{FIND}[i]$ 是结点 i 的连通分支标记。

(1) 初始 $\text{FIND}[i] = i$.

(2) 连通分支合并，较小分支标记更新为较大分支标记

每个结点至多更新 $\log n$ 次，
建立和更新 FIND 数组: $O(n \log n)$

时间: $O(m \log m) + O(n \log n) + O(m)$
 $= O(m \log n)$

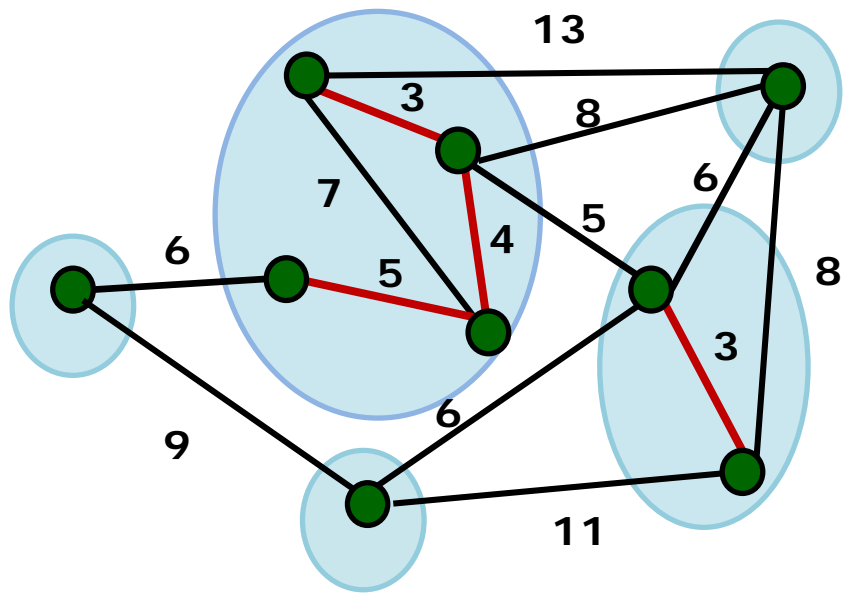
应用：数据分组问题

一组数据(照片, 文件, 生物标本)要把它们按照相关性进行分类.

用相似度函数或“距离”来描述个体之间的差距.

如果分成5类, 使得每类内部的个体尽可能相近, 不同类之间的个体尽可能地“远离”. 如何划分?

应用：数据分组问题



单链聚类

类似于Kruskal算法

- (1) 按照边长从小到大对边排序
- (2) 依次考察当前最短边 e , 如果 e 与 已经选中的边不构成回路, 则把 e 加入集合, 否则跳过 e . 计数图的连通分支个数.
- (3) 直到保留了 k 个连通分支为止.

小结

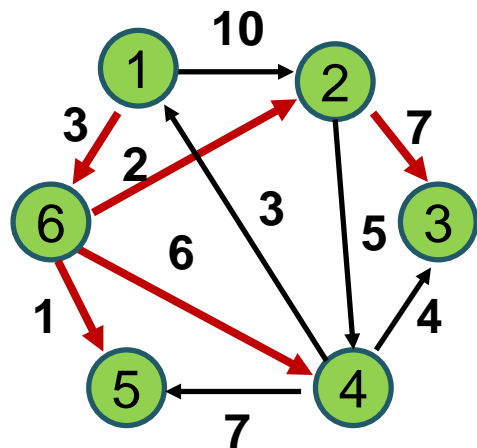
- **Kruskal**算法的贪心策略：在不构成回路条件下选当前最短边
- 正确性证明：对规模归纳
- 时间复杂度： $O(m\log n)$
- 应用：单链聚类

单源最短路径

单源最短路问题

给定带权有向网络 $G=(V,E,W)$, 每条边 $e=\langle i,j \rangle$ 的权 $w(e)$ 为非负实数, 表示从 i 到 j 的距离. 源点 $s \in V$.

求: 从 s 出发到达其它结点的最短路径.



源点: 1

$1 \rightarrow 6 \rightarrow 2$: $short[2] = 5$

$1 \rightarrow 6 \rightarrow 2 \rightarrow 3$: $short[3] = 12$

$1 \rightarrow 6 \rightarrow 4$: $short[4] = 9$

$1 \rightarrow 6 \rightarrow 5$: $short[5] = 4$

$1 \rightarrow 6$: $short[6] = 3$

Dijkstra算法有关概念

$x \in S \Leftrightarrow x \in V$ 且从 s 到 x 的最短路径已经找到

初始: $S = \{ s \}$, $S = V$ 时算法结束

从 s 到 u 相对于 S 的最短路径: 从 s 到 u 且仅经过 S 中顶点的最短路径

$dist[u]$: 从 s 到 u 相对 S 最短路径的长度

$short[u]$: 从 s 到 u 的最短路径的长度

$dist[u] \geq short[u]$

算法设计思想

输入：有向图 $G = (V, E, W)$,

$V = \{ 1, 2, \dots, n \}$, $s = 1$

输出：从 s 到每个顶点的最短路径

1. 初始 $S = \{1\}$
2. 对于 $i \in V - S$, 计算1到 i 的相对 S 的最短路, 长度 $dist[i]$
3. 选择 $V - S$ 中 $dist$ 值最小的 j , 将 j 加入 S , 修改 $V - S$ 中顶点的 $dist$ 值.
4. 继续上述过程, 直到 $S = V$ 为止.

伪码

算法 Dijkstra

1. $S \leftarrow \{ s \}$
2. $dist[s] \leftarrow 0$
3. for $i \in V - \{ s \}$ do
4. $dist[i] \leftarrow w(s,i)$ // s 到 i 没边, $w(s,i)=\infty$
5. while $V - S \neq \emptyset$ do
6. 从 $V - S$ 取相对 S 的最短路径顶点 j
7. $S \leftarrow S \cup \{ j \}$
8. for $i \in V - S$ do
9. if $dist[j] + w(j,i) < dist[i]$
10. then $dist[i] \leftarrow dist[j] + w(j,i)$

更新
dist值

运行实例

输入: $G=\langle V,E,W\rangle$, 源点 1
 $V=\{1,2,3,4,5,6\}$

$S=\{1\}$,

$dist[1]=0$

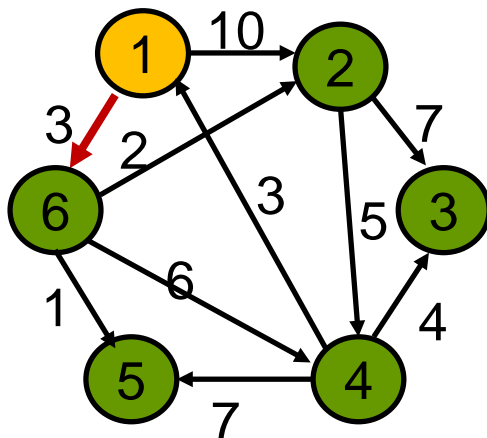
$dist[2]=10$

$dist[6]=3$

$dist[3]=\infty$

$dist[4]=\infty$

$dist[5]=\infty$



实例（续）

$S=\{1,6\}$

$dist[1] = 0$

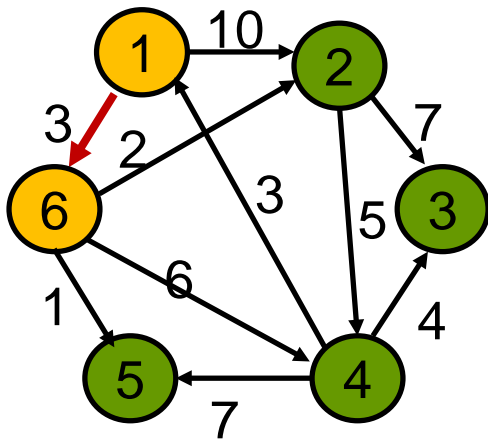
$dist[6] = 3$

$dist[2] = 5$

$dist[4] = 9$

$dist[5] = 4$

$dist[3] = \infty$



运行实例（续）

$S=\{1,6,5\}$

$dist[1]=0$

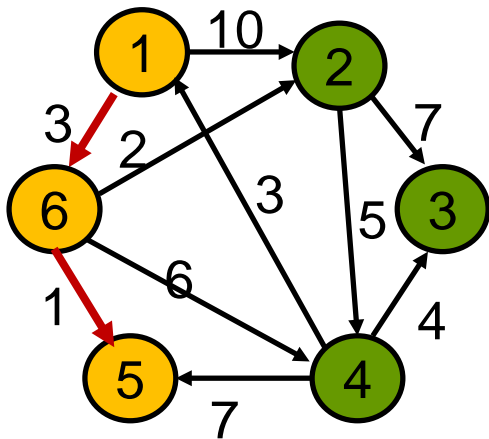
$dist[6]=3$

$dist[5]=4$

$dist[2]=5$

$dist[4]=9$

$dist[3]=\infty$



运行实例（续）

$S = \{1, 6, 5, 2\}$

$dist[1]=0$

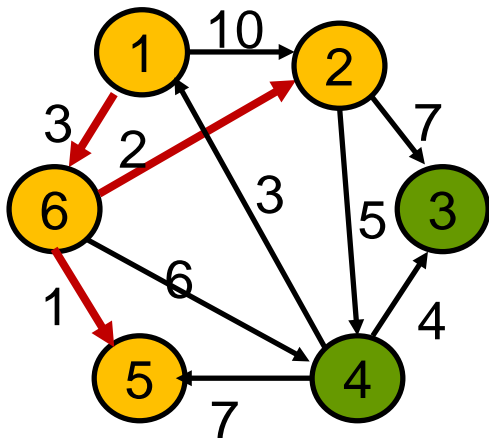
$dist[6]=3$

$dist[5]=4$

$dist[2]=5$

$dist[4]=9$

$dist[3]=12$



运行实例（续）

$S=\{1,6,5,2,4\}$

$dist[1]=0$

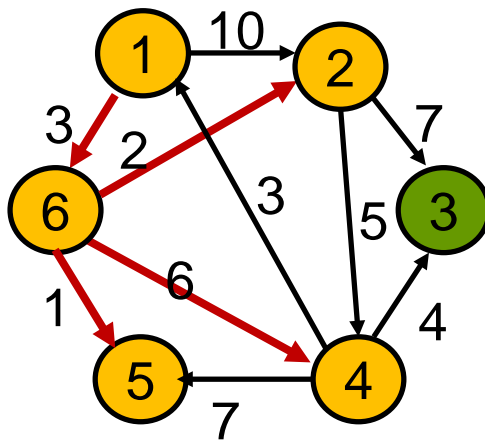
$dist[6]=3$

$dist[5]=4$

$dist[2]=5$

$dist[4]=9$

$dist[3]=12$



运行实例（续）

$S=\{1,6,5,2,4,3\}$

dist [1]=0

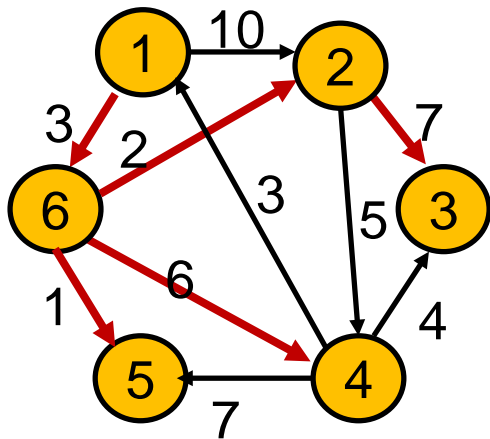
dist [6]=3

dist [5]=4

dist [2]=5

dist [4]=9

dist [3]=12



short[1]=0, *short*[2]=5, *short*[3]=12,

short[4]=9, *short*[5]=4, *short*[6]=3.

小结

- 单源最短路径问题
- Dijkstra算法设计思想及伪码
- 运行实例

Dijkstra算法 的正确性

归纳证明思路

命题：当算法进行到第 k 步时，对于 S 中每个结点 i ,

$$\textit{dist}[i] = \textit{short}[i]$$

归纳基础

$$k = 1, S = \{s\}, \textit{dist}[s] = \textit{short}[s] = 0.$$

归纳步骤

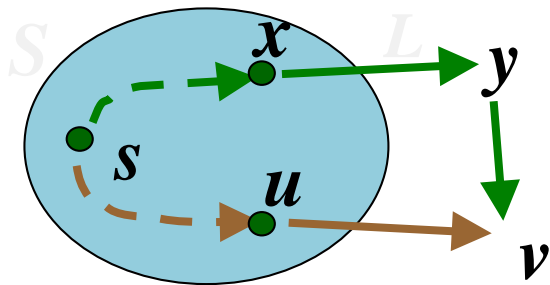
证明：假设命题对 k 为真，则对 $k+1$ 命题也为真.

归纳步骤证明

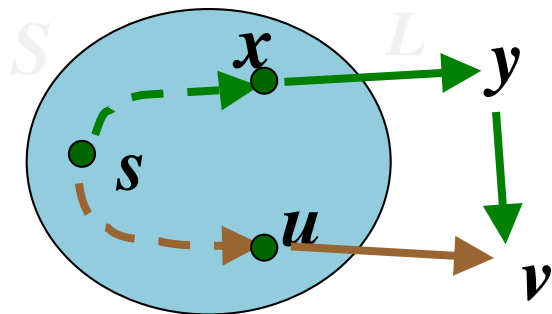
假设命题对 k 为真，考虑 $k+1$ 步算法
选择顶点 v (边 $\langle u, v \rangle$). 需要证明

$$\text{dist}[v] = \text{short}[v]$$

若存在另一条 s - v 路径 L (绿色), 最后一次出 S 的顶点为 x , 经过 $V-S$ 的第一个顶点 y , 再由 y 经过一段在 $V-S$ 中的路径到达 v .



归纳步骤证明（续）



在 $k+1$ 步算法选择顶点 v ，而不是 y ，

$$\text{dist}[v] \leq \text{dist}[y]$$

令 y 到 v 的路径长度为 $d(y, v)$

$$\text{dist}[y] + d(y, v) \leq L$$

于是 $\text{dist}[v] \leq L$ ，即 $\text{dist}[v] = \text{short}[v]$

时间复杂度

- 时间复杂度: $O(nm)$
算法进行 $n-1$ 步
每步挑选1个具有最小 $dist$ 函数值的
结点进入 S , 需要 $O(m)$ 时间
- 选用基于堆实现的优先队列的数据结构, 可以将算法时间复杂度降到 $O(m\log n)$

贪心法小结

- 贪心法适用于组合优化问题.
- 求解过程是多步判断过程，最终的判断序列对应于问题的最优解.
- 判断依据某种“短视的”贪心选择性质，性质的好坏决定了算法的正确性. 贪心性质的选择往往依赖于直觉或者经验.

贪心法小结（续）

- 贪心法正确性证明方法：
 - (1) 直接计算优化函数，贪心法的解恰好取得最优值
 - (2) 数学归纳法（对算法步数或者问题规模归纳）
 - (3) 交换论证
- 证明贪心策略不对：举反例

贪心法小结（续）

- 对于某些不能保证对所有的实例都得到最优解的贪心算法（近似算法），可做参数化分析或者误差分析.
- 贪心法的优势：算法简单，时间和空间复杂性低

贪心法小结（续）

- 几个著名的贪心算法
最小生成树的Prim算法
最小生成树的Kruskal算法
单源最短路的Dijkstra算法

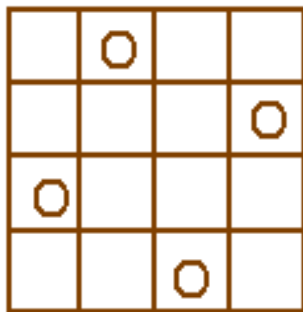
几个回溯算法 的例子

4后问题

4后问题：在 4×4 的方格棋盘上放置4个皇后，使得没有两个皇后在同一行、同一列、也不在同一条45度的斜线上。问有多少种可能的布局？

解是 4 维向量 $\langle x_1, x_2, x_3, x_4 \rangle$

解： $\langle 2, 4, 1, 3 \rangle$, $\langle 3, 1, 4, 2 \rangle$

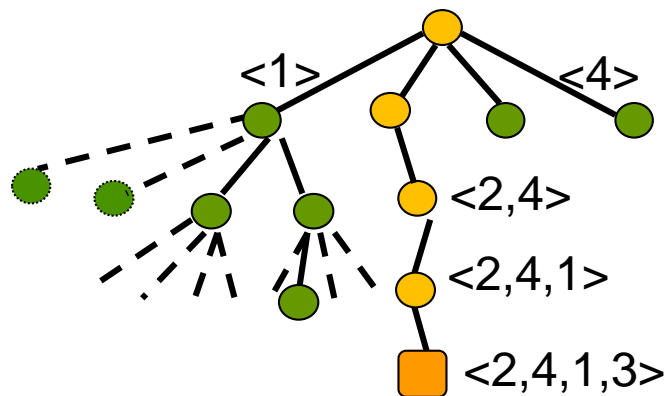


推广到8后问题

解：8维向量，有 92个.

例如： $\langle 1, 5, 8, 6, 3, 7, 2, 4 \rangle$ 是解.

搜索空间：4叉树



每个结点有4个儿子，分别代表选择
1,2,3,4列位置

第 i 层选择解向量中第 i 个分量的值

最深层的树叶是解

按深度优先次序遍历树，找到所有解

0-1背包问题

问题:

有 n 种物品，每种物品只有 1 个. 第 i 种物品价值为 v_i , 重量为 w_i , $i=1,2,\dots,n$.
问如何选择放入背包的物品，使得总重量不超过 B , 而价值达到最大?

实例:

$$V=\{12,11,9,8\}, W=\{8,6,4,3\}, B=13$$

最优解:

$\langle 0,1,1,1 \rangle$, 价值: 28, 重量: 13

算法设计

解： n 维0-1向量 $\langle x_1, x_2, \dots, x_n \rangle$,
 $x_i=1 \Leftrightarrow$ 物品 i 选入背包

结点： $\langle x_1, x_2, \dots, x_k \rangle$ （部分向量）

搜索空间： 0-1取值的二叉树，称为子集树，有 2^n 片树叶.

可行解：满足约束条件 $\sum_{i=1}^n w_i x_i \leq B$ 的解

最优解：可行解中价值达到最大的解

实例

输入:

$V=\{12,11,9,8\}$, $W=\{8,6,4,3\}$, $B=13$

2个可行解:

$\langle 0,1,1,1 \rangle$, 选入物品2,3,4, 价值为28,
重量为13

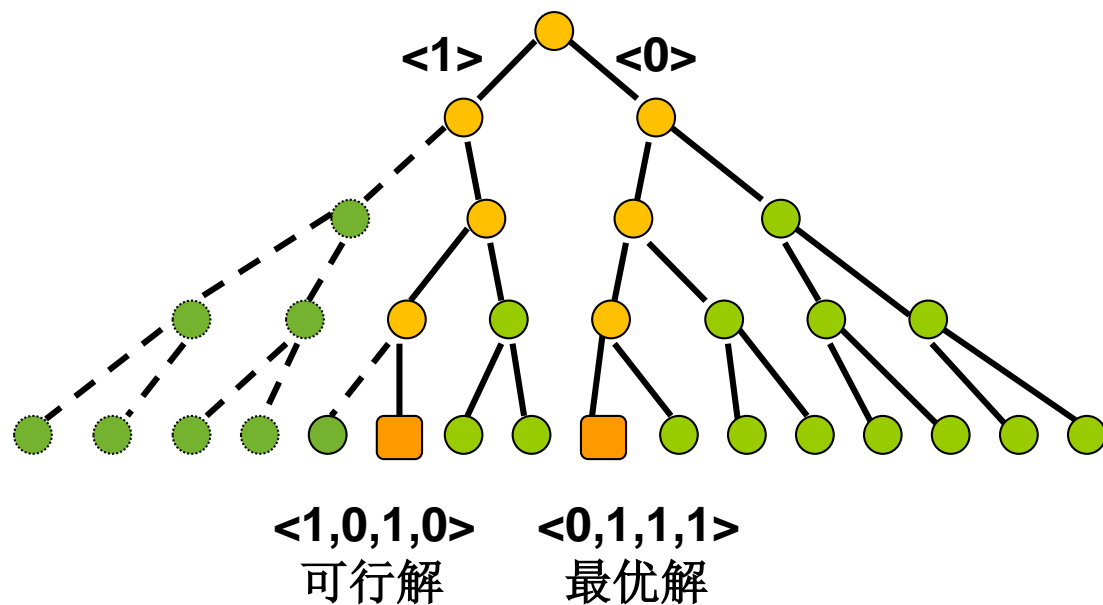
$\langle 1,0,1,0 \rangle$, 选入物品1,3, 价值为21,
重量为12

最优解: $\langle 0,1,1,1 \rangle$

搜索空间

实例: $V=\{12,11,9,8\}$, $W=\{8,6,4,3\}$, $B=13$

搜索空间: 子集树, 2^n 片树叶



货郎问题

问题：有 n 个城市，已知任两个城市之间的距离，求一条每个城市恰好经过一次的回路，使得总长度最小。

建模：城市集 $C=\{c_1, c_2, \dots, c_n\}$ ，距离

$$d(c_i, c_j) = d(c_j, c_i) \in \mathbb{Z}^+, 1 \leq i < j \leq n$$

求： $1, 2, \dots, n$ 的排列 k_1, k_2, \dots, k_n 使得

$$\min\left\{\sum_{i=1}^{n-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_n}, c_{k_1})\right\}$$

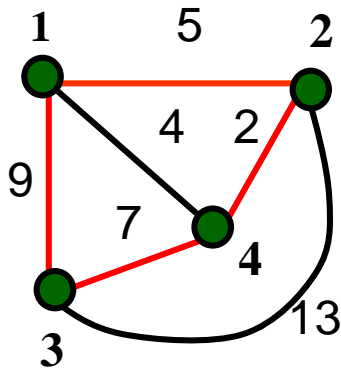
实例

$$C = \{1, 2, 3, 4\}$$

$$d(1, 2) = 5, d(1, 3) = 9,$$

$$d(1, 4) = 4, d(2, 3) = 13,$$

$$d(2, 4) = 2, d(3, 4) = 7$$

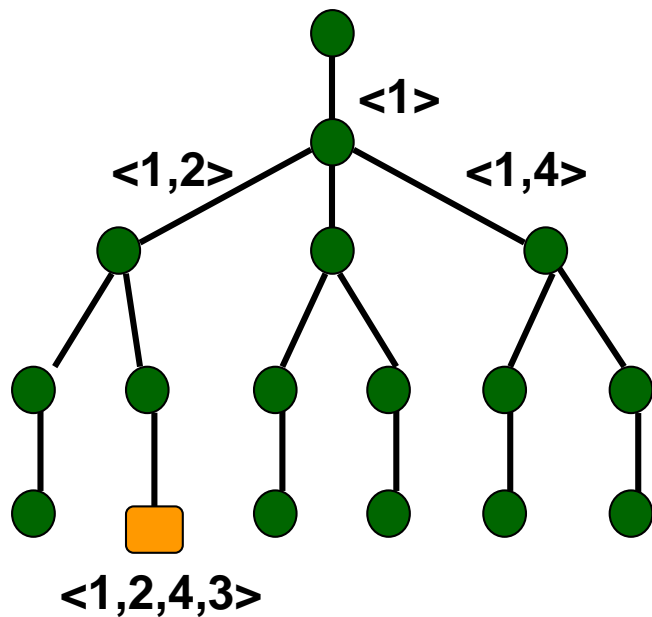


解: $\langle 1, 2, 4, 3 \rangle$,

$$\text{长度} = 5 + 2 + 7 + 9 = 23$$

搜索空间

排列树, 有 $(n-1)!$ 片树叶



小结

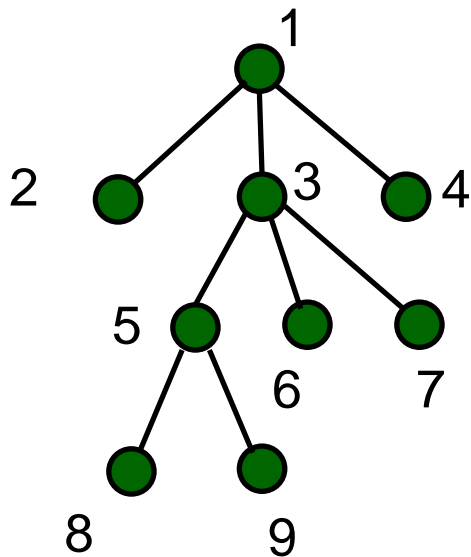
- 回溯算法的例子： n 后问题，0-1背包问题，货郎问题
- 解：向量
- 搜索空间：树，可能是 n 叉树、子集树、排列树等等，树的结点对应于部分向量，可行解在叶结点
- 搜索方法：深度优先，宽度优先，...
跳越式遍历搜索树，找到解

回溯算法的设计 思想和适用条件

问题分析

问题	解性质	解描述 向量	搜索 空间	搜索 方式	约束 条件
n 后	可行解	$\langle x_1, x_2, \dots, x_n \rangle$ x_i : 第 i 行列号	n 叉树	深度, 宽度优先	彼此不攻击
0-1背包	最优解	$\langle x_1, x_2, \dots, x_n \rangle$ $x_i = 0, 1$, $x_i = 1 \Leftrightarrow$ 选 i	子集树	深度, 宽度优先	不超背包重量
货郎	最优解	$\langle i_1=1, i_2, \dots, i_n \rangle$ $1, 2, \dots, n$ 的排列	排列树	深度, 宽度优先	选没有经过的城市
特点	搜索解	向量, 不断扩张部分向量	树	跳跃式遍历	约束条件回溯判定

深度与宽度优先搜索



深度优先访问顺序:

1→2→3→5→8→9
→6→7→4

宽度优先访问顺序:

1→2→3→4→5→6
→7→8→9

回溯算法基本思想

- (1) **适用**：求解搜索问题和优化问题.
- (2) **搜索空间**：树，结点对应部分解向量，可行解在树叶上.
- (3) **搜索过程**：采用系统的方法隐含遍历搜索树.
- (4) **搜索策略**：深度优先，宽度优先，函数优先，宽深结合等.

回溯算法基本思想(续)

(5) 结点分支判定条件:

满足约束条件---分支扩张解向量

不满足约束条件, 回溯到该结点的父结点.

(6) 结点状态: 动态生成

白结点(尚未访问)

灰结点(正在访问该结点为根的子树)

黑结点(该结点为根的子树遍历完成)

(7) 存储: 当前路径

结点状态

深度优先

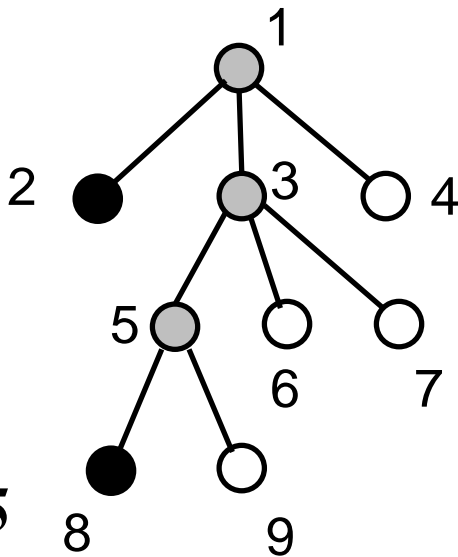
访问次序:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$

已完成访问: 2, 8

已访问但未结束: 1, 3, 5

尚未访问: 9, 6, 7, 4



回溯算法的适用条件

在结点 $\langle x_1, x_2, \dots, x_k \rangle$ 处

$P(x_1, x_2, \dots, x_k)$ 为真

\Leftrightarrow 向量 $\langle x_1, x_2, \dots, x_k \rangle$ 满足某个性质
(n 后中 k 个皇后放在彼此不攻击的位置)

多米诺性质

$$P(x_1, x_2, \dots, x_{k+1}) \rightarrow P(x_1, x_2, \dots, x_k) \quad 0 < k < n$$

$$\neg P(x_1, x_2, \dots, x_k) \rightarrow \neg P(x_1, x_2, \dots, x_{k+1}) \quad 0 < k < n$$

k 维向量不满足约束条件, 扩张向量到
 $k+1$ 维仍旧不满足, 可以回溯.

一个反例

例 求不等式的整数解

$$5x_1 + 4x_2 - x_3 \leq 10, \quad 1 \leq x_k \leq 3, \quad k=1,2,3$$

$P(x_1, \dots, x_k)$: 将 x_1, x_2, \dots, x_k 代入原不等式的相应部分, 部分和小于等于10

不满足多米诺性质:

$$5x_1 + 4x_2 - x_3 \leq 10 \not\Rightarrow 5x_1 + 4x_2 \leq 10$$

变换使得问题满足多米诺性质:

令 $x_3 = 3 - x_3'$,

$$5x_1 + 4x_2 + x_3' \leq 13$$

$$1 \leq x_1, \quad x_2 \leq 3, \quad 0 \leq x_3' \leq 2$$

小结

- 回溯算法的适用条件：
多米诺性质
- 回溯算法的设计步骤
 - (1) 定义解向量和每个分量的取值范围
解向量 为 $\langle x_1, x_2, \dots, x_n \rangle$
确定 x_i 的取值集合为 $X_i, i = 1, 2, \dots, n$.

小结（续）

- (2) 在 $\langle x_1, x_2, \dots, x_{k-1} \rangle$ 确定如何计算 x_k 取值集合 S_k , $S_k \subseteq X_k$
- (3) 确定结点儿子的排列规则
- (4) 判断是否满足多米诺性质
- (5) 确定每个结点分支的约束条件
- (6) 确定搜索策略: 深度优先, 宽度优先等
- (7) 确定存储搜索路径的数据结构

回溯算法的 实现及实例

回溯算法递归实现

算法 **ReBack** (k)

1. if $k > n$ then $\langle x_1, x_2, \dots, x_n \rangle$ 是解
2. else while $S_k \neq \emptyset$ do
3. $x_k \leftarrow S_k$ 中最小值
4. $S_k \leftarrow S_k - \{x_k\}$
5. 计算 S_{k+1}
6. **ReBack** ($k+1$)

算法 **ReBacktrack** (n)

输入: n

输出: 所有的解

1. for $k \leftarrow 1$ to n 计算 X_k 且 $S_k \leftarrow X_k$
2. **ReBack** (1)

迭代实现

迭代算法 Backtrack

输入: n

输出: 所有的解

确定初
始取值

1. 对于 $i=1, 2, \dots, n$ 确定 X_i

2. $k \leftarrow 1$

3. 计算 S_k

满足约束
分支搜索

4. while $S_k \neq \emptyset$ do

5. $x_k \leftarrow S_k$ 中最小值; $S_k \leftarrow S_k - \{x_k\}$

6. if $k < n$ then

7. $k \leftarrow k+1$; 计算 S_k

8. else $\langle x_1, x_2, \dots, x_n \rangle$ 是解

回溯

9. if $k > 1$ then $k \leftarrow k-1$; goto 4

装载问题

问题：有 n 个集装箱，需要装上两艘载重分别为 c_1 和 c_2 的轮船. w_i 为第 i 个集装箱的重量，且 $w_1+w_2+\dots+w_n \leq c_1+c_2$
问：是否存在一种合理的装载方案把这 n 个集装箱装上船？如果有，请给出一种方案.

实例：

$W = < \underline{90}, 80, \underline{40}, 30, 20, \underline{12}, 10 >$
 $c_1=152, c_2=130$

解：1,3,6,7装第一艘船，其余第2艘船 4

求解思路

输入： $W = \langle w_1, w_2, \dots, w_n \rangle$ 为集装箱重量
 c_1 和 c_2 为船的最大载重量

算法思想： 令第一艘船的装入量为 W_1 ,

1. 用回溯算法求使得 $c_1 - W_1$ 达到最小的装载方案.
2. 若满足

$$w_1 + w_2 + \dots + w_n - W_1 \leq c_2$$

则回答 “Yes”, 否则回答 “No”

伪码

算法 Loading (W, c_1),

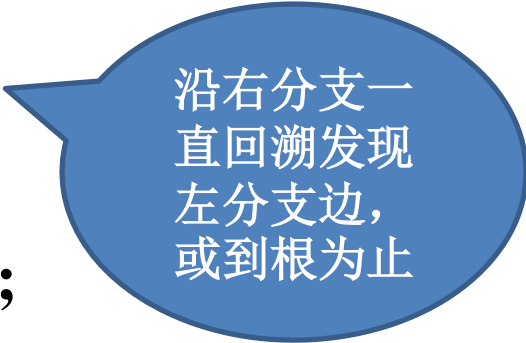
B为当前空隙
best 最小空隙

1. Sort(W);
2. $B \leftarrow c_1$; $best \leftarrow c_1$; $i \leftarrow 1$;
3. while $i \leq n$ do
4. if 装入 i 后重量不超过 c_1
5. then $B \leftarrow B - w_i$; $x[i] \leftarrow 1$; $i \leftarrow i + 1$;
6. else $x[i] \leftarrow 0$; $i \leftarrow i + 1$;
7. if $B < best$ then 记录解; $best \leftarrow B$;
8. Backtrack(i); 回溯
9. if $i = 1$ then return 最优解
10. else goto 3.

子过程 Backtrack

算法 Backtrack(i)

1. while $i > 1$ and $x[i] = 0$ do
2. $i \leftarrow i - 1$;
3. if $x[i] = 1$
4. then $x[i] \leftarrow 0$;
5. $B \leftarrow B + w_i$;
6. $i \leftarrow i + 1$.



沿右分支一直回溯发现左分支边，或到根为止

实例

$W = \langle 90, 80, 40, 30, 20, 12, 10 \rangle$

$c_1=152, c_2=130$

解:

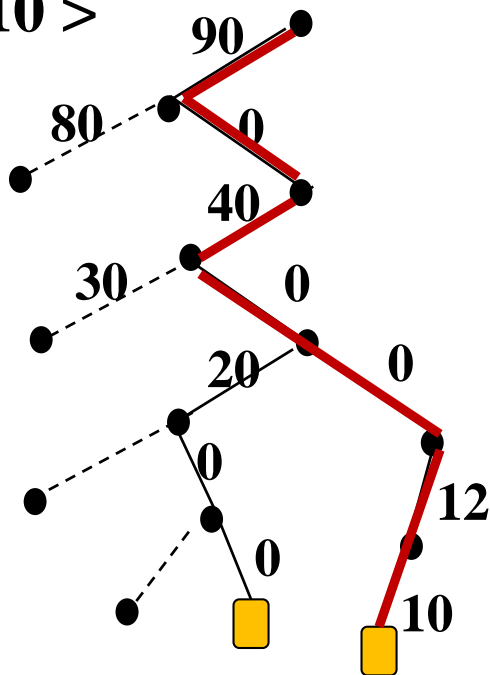
可以装, 方案如下:

1,3,6,7 装第一艘船

2,4,5 装第二艘船

时间复杂性:

$$W(n)=O(2^n)$$



小结

- 回溯算法的实现：
递归实现、迭代实现
- 装载问题
问题描述
算法伪码
最坏情况下时间复杂度 $O(2^n)$

图的着色

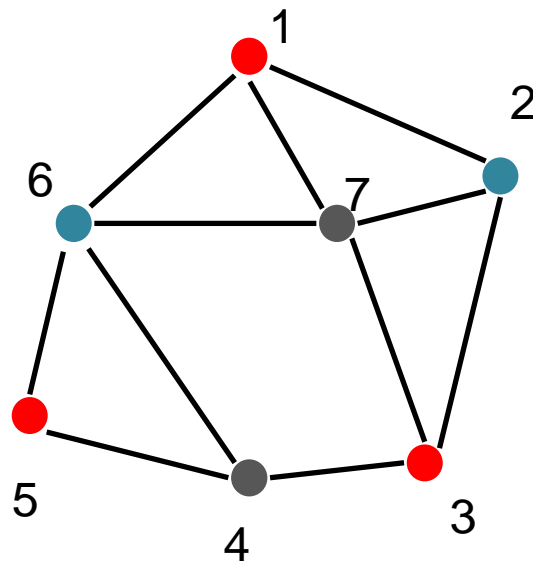
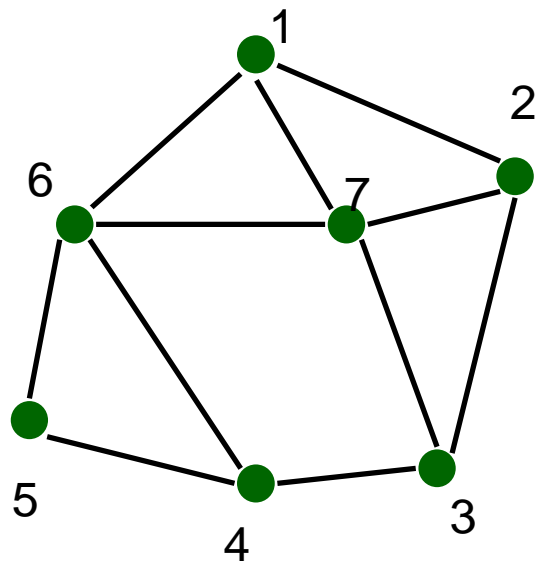
着色问题

输入:

无向连通图 G 和 m 种颜色的集合
用这些颜色给图的顶点着色, 每个
顶点一种颜色. 要求是: G 的每条
边的两个顶点着不同颜色.

输出: 所有可能的着色方案.
如果不存在着色方案, 回答 “No”.

实例



$n=7, m=3$

解向量

设 $G=(V,E)$, $V=\{1,2, \dots, n\}$

颜色编号: $1, 2, \dots, m$

解向量: $\langle x_1, x_2, \dots, x_n \rangle$,

$$x_1, x_2, \dots, x_n \in \{1, 2, \dots, m\}$$

结点的部分向量 $\langle x_1, x_2, \dots, x_k \rangle$

$$x_1, x_2, \dots, x_k, 1 \leq k \leq n,$$

表示只给顶点 $1, 2, \dots, k$ 着色的部分方案

算法设计

搜索树： m 叉树

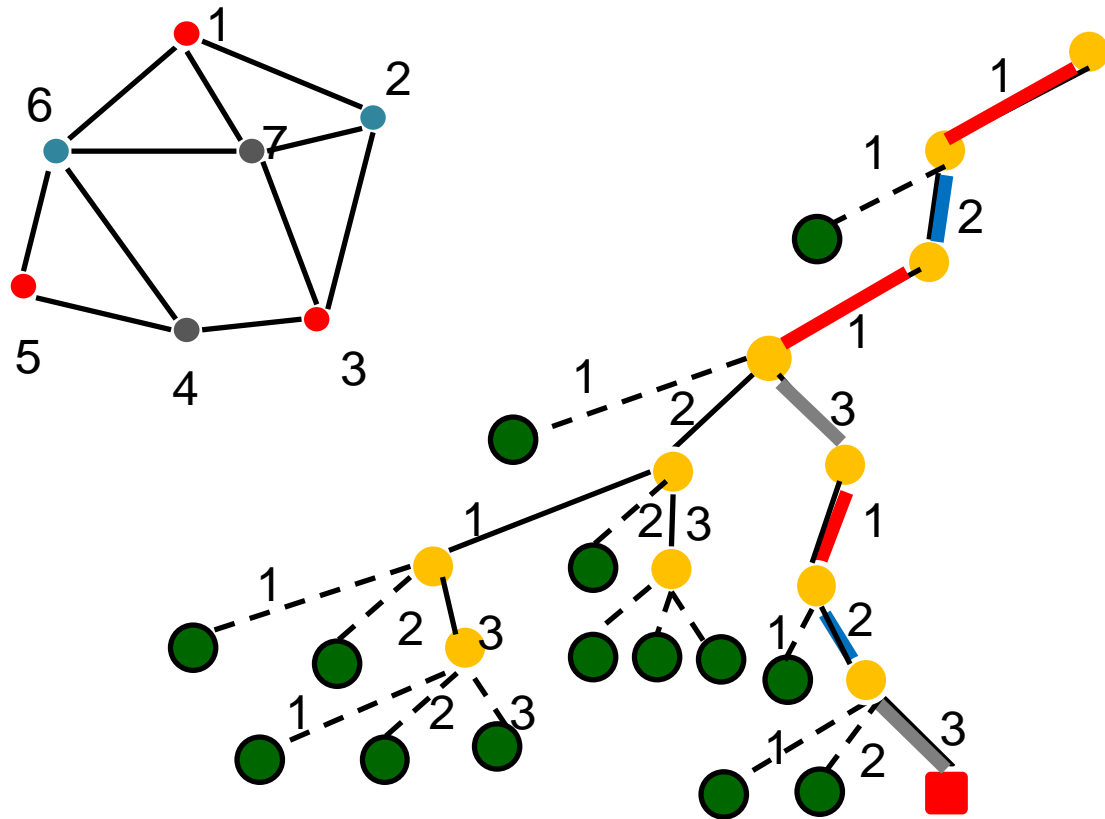
约束条件： 在结点 $\langle x_1, x_2, \dots, x_k \rangle$ 处，
顶点 $k+1$ 的邻接表中结点已用过的颜色
不能再使用

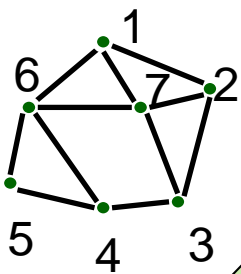
如果邻接表中结点已用过 m 种颜色，
则结点 $k+1$ 没法着色，从该结点回溯
到其父结点。满足多米诺性质

搜索策略： 深度优先

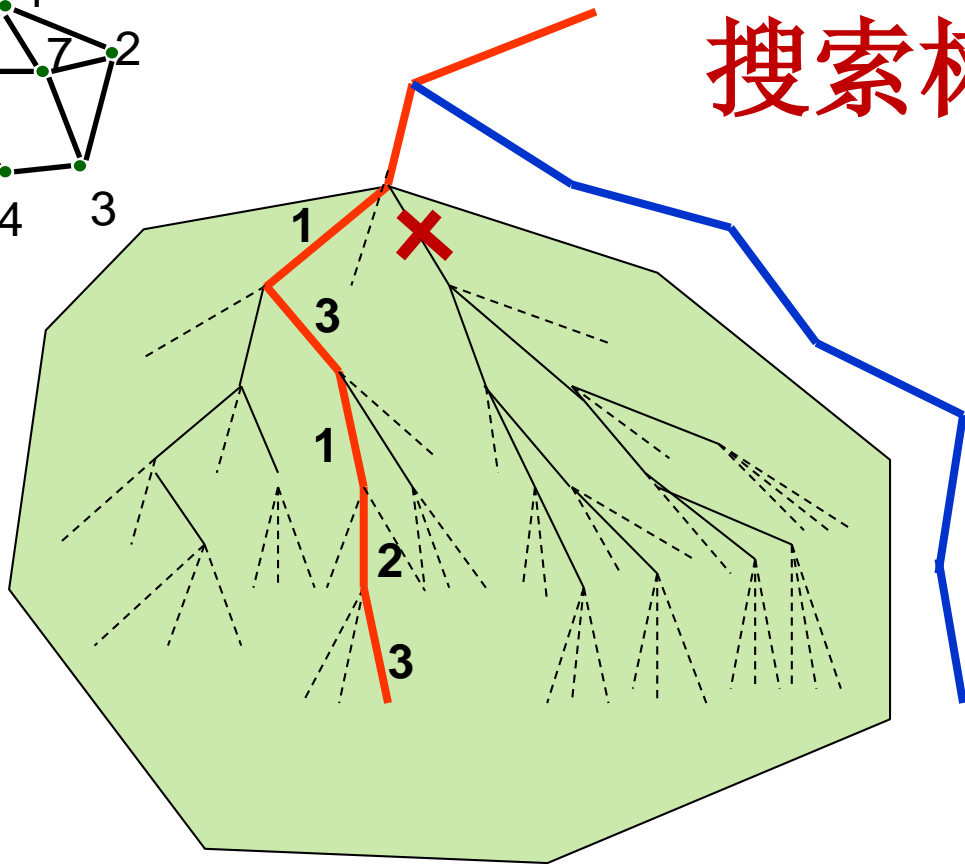
时间复杂度： $O(n m^n)$

运行实例





搜索树



第一个解向量: $\langle 1, 2, 1, 3, 1, 2, 3 \rangle_7$

时间复杂度与 改进途径

时间复杂度: $O(nm^n)$

根据对称性, 只需搜索 $1/3$ 的解空间. 当 1 和 2 确定, 即 $\langle 1, 2 \rangle$ 以后, 只有 1 个解, 在 $\langle 1, 3 \rangle$ 为根的子树中也只有 1 个解. 由于 3 个子树的对称性, 总共 6 个解.

在取定 $\langle 1, 2 \rangle$ 后, 不可扩张成 $\langle 1, 2, 3 \rangle$, 因为 7 和 1, 2, 3 都相邻. 7 没法着色. 可以从打叉的结点回溯, 而不必搜索其子树.

着色问题的应用

会场分配问题:

有 n 项活动需要安排, 对于活动 i, j , 如果 i, j 时间冲突, 就说 i 与 j 不相容. 如何分配这些活动, 使得每个会场的活动相容且占用会场数最少?

建模:

活动作为图的顶点, 如果 i, j 不相容, 则在 i 与 j 之间加一条边, 会场标号作为颜色标号. 求图的一种着色方案, 使得使用的颜色数最少.

小结

- 着色问题的描述
- 着色问题的算法设计
- 时间复杂度及改进途径
- 着色问题的应用

搜索树结点数 的估计

搜索树结点数估计

Monte Carlo方法

1. 从根开始，随机选择一条路径，直到不能分支为止，即从 x_1, x_2, \dots ，依次对 x_i 赋值，每个 x_i 的值是从当时的 S_i 中随机选取，直到向量不能扩张为止.
2. 假定搜索树的其他 $|S_i|-1$ 个分支与以上随机选出的路径一样，计数搜索树的点数.
3. 重复步骤 1 和 2，将结点数进行概率平均.


伪码

Monte Carlo

输入: n 为皇后数, t 为抽样次数

输出: sum , 即 t 次抽样路长平均值

1. $sum \leftarrow 0$
2. for $i \leftarrow 1$ to t do // 取样次数 t
3. $m \leftarrow \text{Estimate}(n)$ // m 为结点数
4. $sum \leftarrow sum + m$
5. $sum \leftarrow sum / t$



一次抽样结果

一次抽样

m 为本次取样得到的树结点总数

k 为层数

r_2 为上层结点数

r_1 为本层结点数

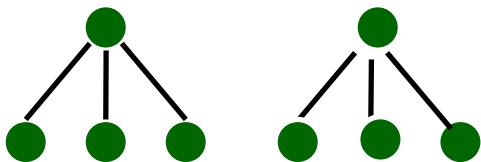
$r_1 = r_2 \cdot \text{分支数}$

n 为树的层数

从树根向下计算,随机选择,直到树叶.

$$r_2 = 2$$

$$r_1 = r_2 \cdot 3 = 6$$



子过程的伪码

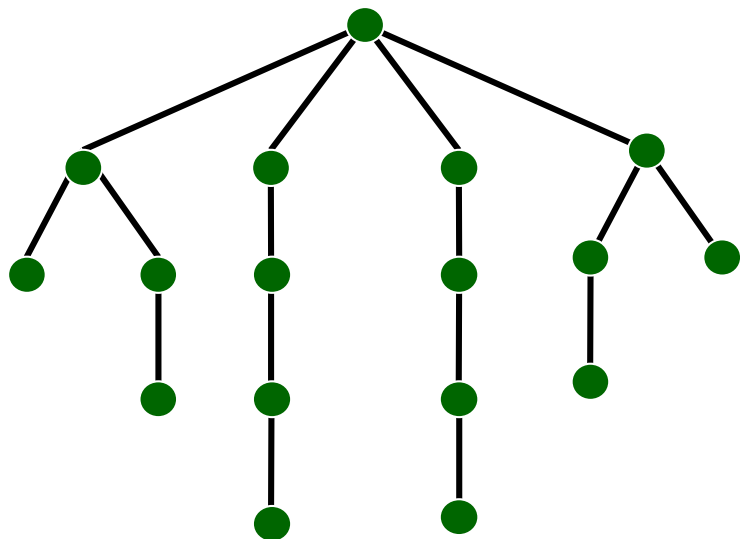
算法Estimate(n)

1. $m \leftarrow 1$; $r_2 \leftarrow 1$; $k \leftarrow 1$ // m 为结点总数
2. while $k \leq n$ do
3. if $S_k = \emptyset$ then return m
4. $r_1 \leftarrow |S_k| * r_2$ // r_1 为扩张后结点总数
5. $m \leftarrow m + r_1$ // r_2 为扩张前结点总数
6. $x_k \leftarrow$ 随机选择 S_k 的元素
7. $r_2 \leftarrow r_1$
8. $k \leftarrow k + 1$

不能
分支

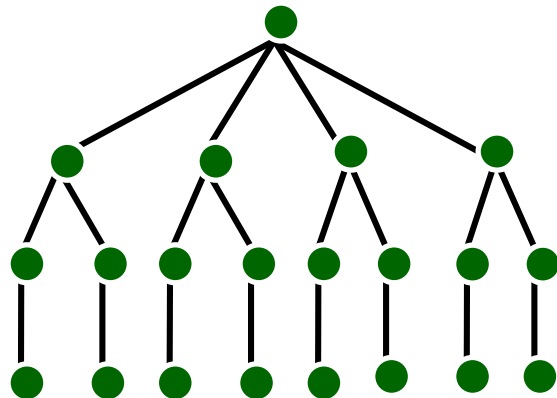
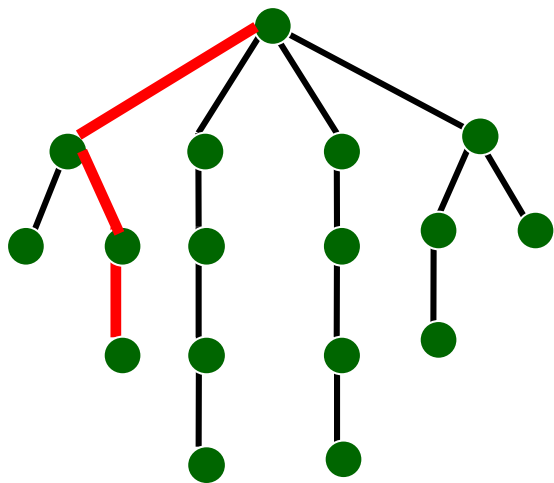
随机选
择一步

4后搜索树遍历的结点



结点数=17

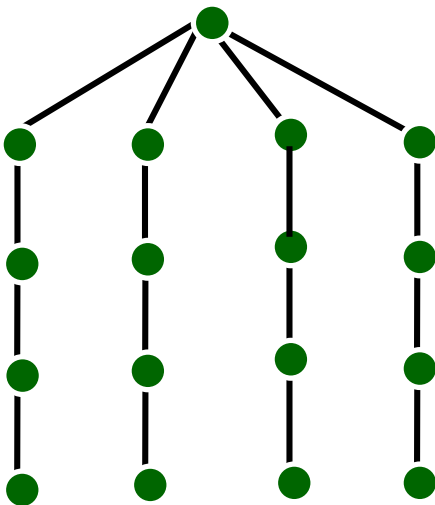
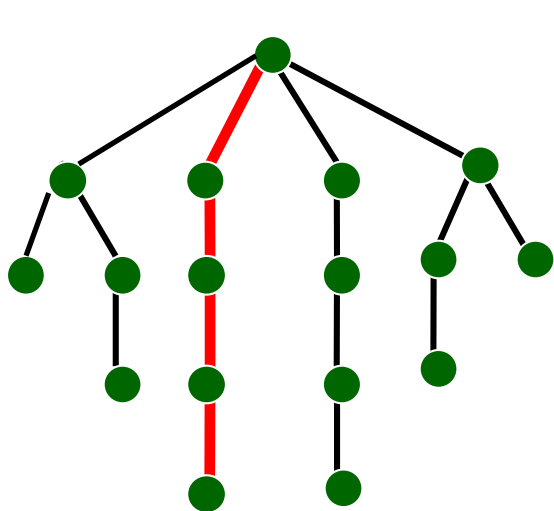
随机选择路径1



Case1: $\langle 1, 4, 2 \rangle$,

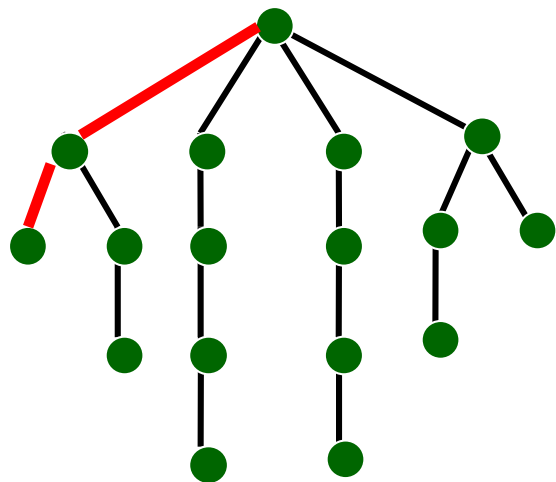
结点数 21

随机选择路径2

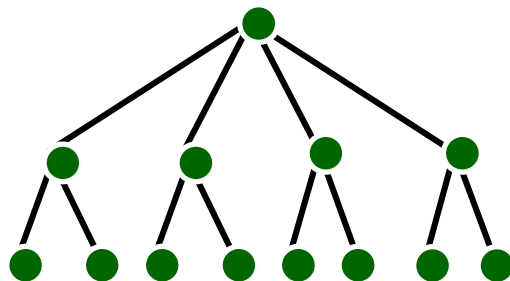


Case1: $\langle 2, 4, 1, 3 \rangle$, 结点数 17

随机选择路径3



Case3: $\langle 1, 3 \rangle$,



结点数 13

估计结果

假设 4 次抽样测试:

case1: 1次,

case2: 1次,

case3: 2次,

平均结点数 $= (21 \times 1 + 17 \times 1 + 13 \times 2) / 4 = 16$

搜索空间访问的结点数为 17

小结

- **Monte Carlo 方法**

目的： 估计搜索树真正访问结点数

步骤：

随机抽样，选择一条路径

用这条路径代替其他路径

逐层累加树的结点数

多次选择，取结点数的平均值

分支限界 及其应用

组合优化问题

- 组合优化问题的相关概念

目标函数（极大化或极小化）

约束条件（解满足的条件）

可行解: 搜索空间满足约束条件的解

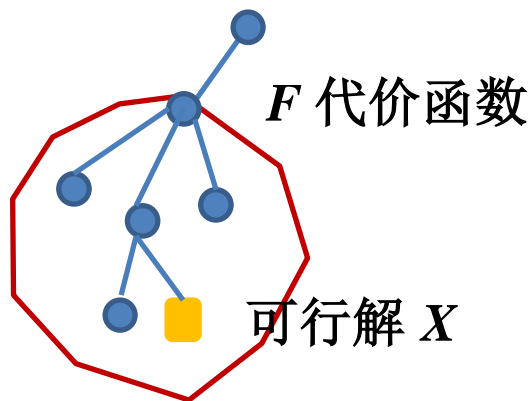
最优解: 使得目标函数达到极大（或极小）的可行解

- **背包问题**

$$\begin{aligned} \max \quad & x_1 + 3x_2 + 5x_3 + 9x_4 \\ & 2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10 \\ & x_i \in \mathbb{N}, \quad i = 1, 2, 3, 4 \end{aligned}$$

代价函数

- **计算位置**: 搜索树的结点
- **值**: 极大化问题是以该点为根的子树所有可行解的值的上界（极小化问题为下界）
- **性质**: 对极大化问题父结点代价不小于子结点的代价（极小化问题相反）

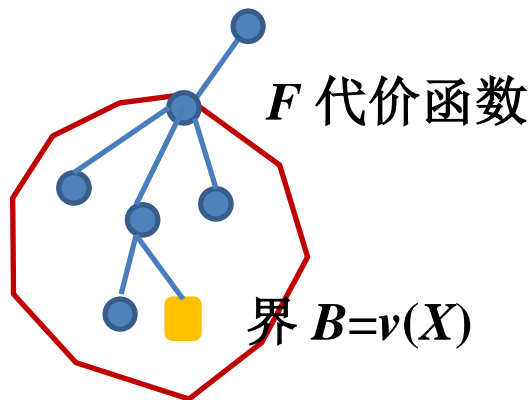


$$F \geq v(X)$$

极大化

界

- **含义**: 当前得到可行解的目标函数的最大值(极小化问题相反)
- **初值**: 极大化问题初值为 0 (极小化问题为最大值)
- **更新**: 得到更好的可行解时



分支限界

停止分支回溯父结点的依据:

1. 不满足约束条件
2. 对于极大化问题, 代价函数值小于当前界 (对于极小化问题是大于界)

界的更新

对极大化问题, 如果一个新的可行解的优化函数值大于 (极小化问题为小于) 当前的界, 则把界更新为该可行解的值

实例

背包问题:

4 种物品, 重量 w_i 与价值 v_i 分别为

$$v_1=1, v_2=3, v_3=5, v_4=9$$

$$w_1=2, w_2=3, w_3=4, w_4=7$$

背包重量限制为10

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10$$

$$x_i \in \mathbf{N}, i = 1, 2, 3, 4$$

代价函数的设定

- 对结点 $\langle x_1, x_2, \dots, x_k \rangle$ ，估计以该结点为根的子树中可行解的上界。
- 按 v_i/w_i 从大到小排序， $i = 1, 2, \dots, n$

- 代价函数 = 已装入价值 + Δ

Δ : 还可继续装入最大价值的上界

$\Delta = \text{背包剩余重量} \times v_{k+1}/w_{k+1}$ (可装)

$\Delta = 0$ (不可装)

实例：背包问题

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10$$

$$x_i \in \mathbf{N}, i = 1, 2, 3, 4$$

对变元重新排序使得 $\frac{v_i}{w_i} \geq \frac{v_{i+1}}{w_{i+1}}$

排序后

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10$$

$$x_i \in \mathbf{N}, i = 1, 2, 3, 4$$

代价函数与分支策略

结点 $\langle x_1, x_2, \dots, x_k \rangle$ 的代价函数

$$\sum_{i=1}^k v_i x_i + (b - \sum_{i=1}^k w_i x_i) \cdot v_{k+1} / w_{k+1}$$

若对某个 $j > k \square b - \sum_{i=1}^k w_i x_i \geq w_j$

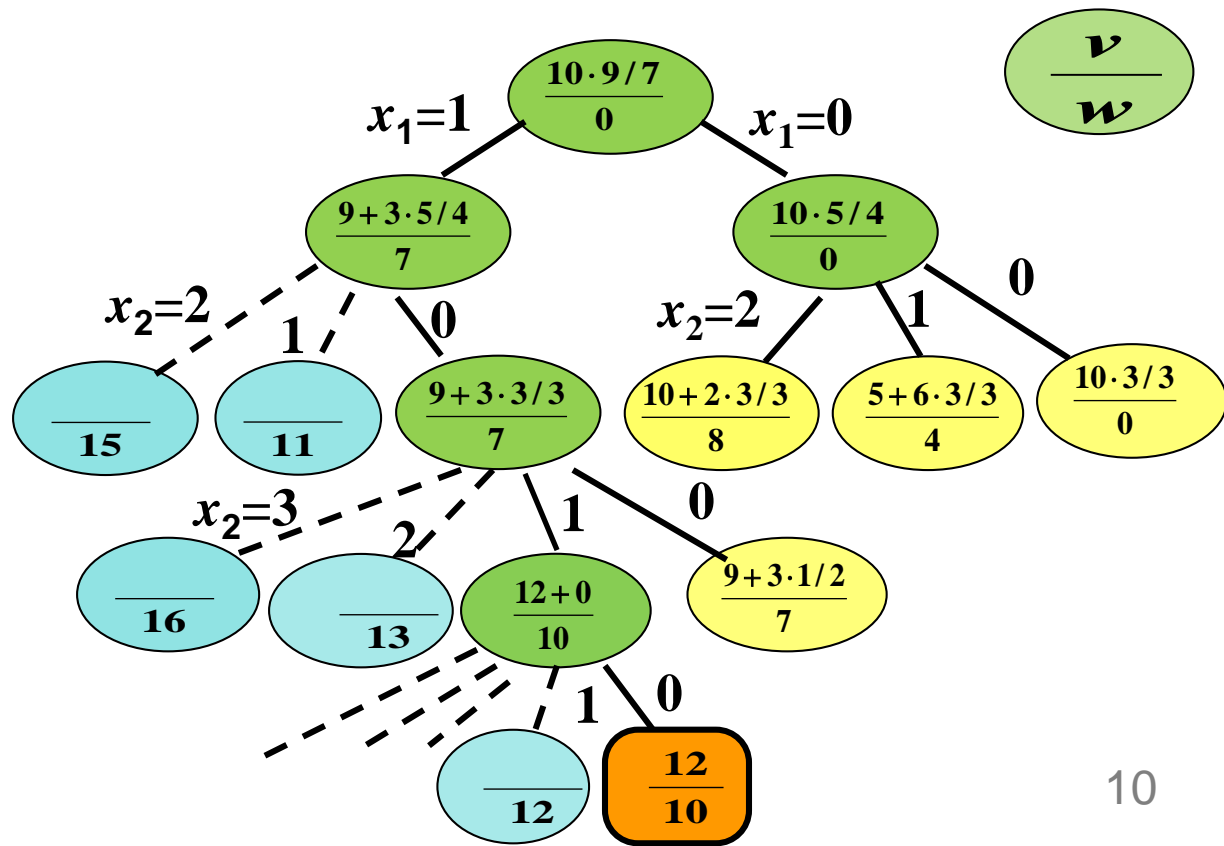
$\sum_{i=1}^k v_i x_i$ 否则

分支策略----深度优先

实例

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, \quad x_i \in \mathbb{N}, i=1,2,3,4$$



小结

- 分支限界适用于组合优化问题
- 对结点 $\langle x_1, \dots, x_k \rangle$ 定义代价函数
当前结点为根子树的可行解的上界
或下界
极大化问题与极小化问题的区别
- 定义界的初值
得到新的更好的可行解就更新界

最大团问题

最大团问题

问题: 无向图 $G=\langle V,E\rangle$, 求 G 的最大团.

G 的**子图**: $G'=\langle V',E'\rangle$, 其中 $V'\subseteq V, E'\subseteq E$,

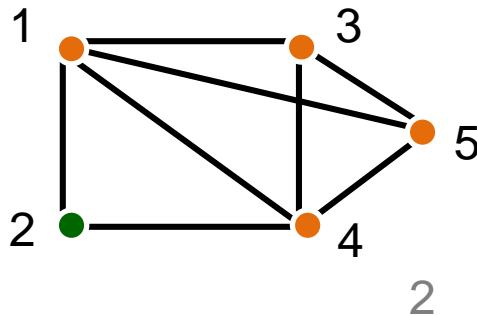
G 的**补图**: $\overline{G}=\langle V,E'\rangle$, E' 是 E 关于完全图
边集的补集

G 中的**团**: G 的完全子图

G 的**最大团**: 顶点数最多的团

实例

最大团: $\{1, 3, 4, 5\}$,



独立集与团

G 的**点独立集**: G 的顶点子集 A , 且

$$\forall u, v \in A, \{u, v\} \notin E.$$

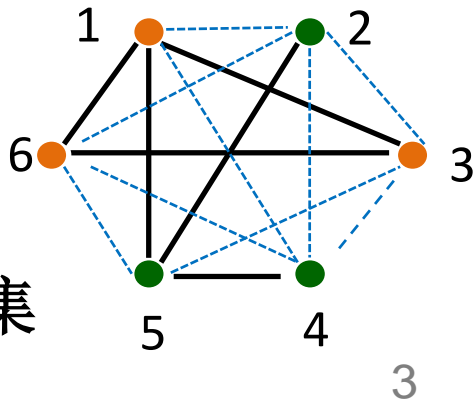
最大点独立集: 顶点最多的点独立集

命题: U 是 G 的最大团当且仅当 U 是 \overline{G} 的最大点独立集.

G 的最大团:

$$U = \{1, 3, 6\}$$

补图 \overline{G} 的最大点独立集



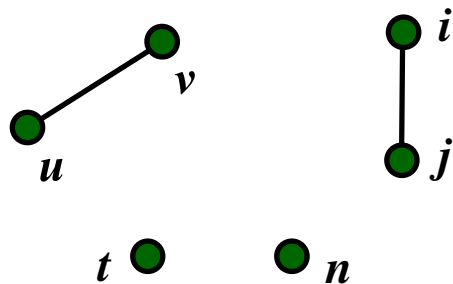
应用

编码, 故障诊断, 计算机视觉, 聚类分析,
经济学, 移动通信, VLSI 电路设计, ...

例子: 噪音使信道传输字符发生混淆

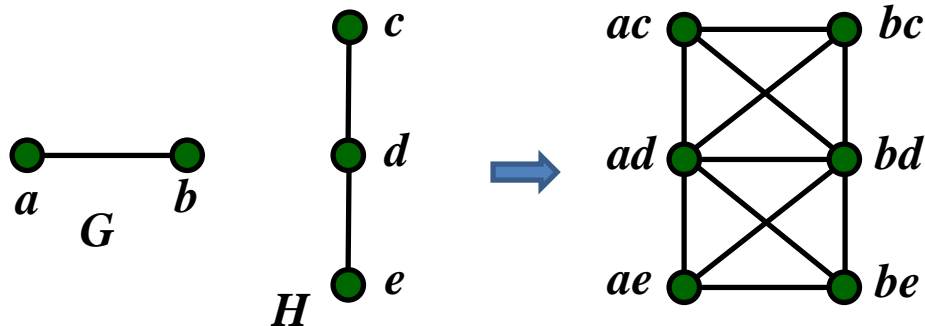
混淆图 $G = \langle V, E \rangle$, V 为有穷字符集,

$\{u, v\} \in E \Leftrightarrow u$ 和 v 易混淆



编码设计

xy 与 uv 混淆 $\Leftrightarrow x$ 与 u 混淆且 y 与 v 混淆
 $\vee x=u$ 且 y 与 v 混淆 $\vee x$ 与 u 混淆且 $y=v$



G 与 H 的正规积

为减少噪音干扰，设计代码应该找
混淆图中的最大点独立集

最大团问题

问题： 给定无向图 $G = \langle V, E \rangle$, 其中顶点集 $V = \{ 1, 2, \dots, n \}$, 边集为 E . 求 G 中的最大团.

解： $\langle x_1, x_2, \dots, x_n \rangle$ 为 0-1 向量,
 $x_k=1$ 当且仅当顶点 k 属于最大团.

蛮力算法： 对每个顶点子集, 检查是否构成团, 即其中每对顶点之间是否都有边. 有 2^n 个子集, 至少需要指数时间.

分支限界算法设计

搜索树为子集树.

结点 $\langle x_1, x_2, \dots, x_k \rangle$ 的含义:

已检索顶点 $1, 2, \dots, k$, 其中 $x_i=1$ 表示
顶点 i 在当前的团内, $i = 1, 2, \dots, k$

约束条件: 该顶点与当前团内每个顶
点都有边相连

界: 当前已检索到的极大团的顶点数

代价函数

代价函数：目前的团可能扩张为极大团的顶点数上界

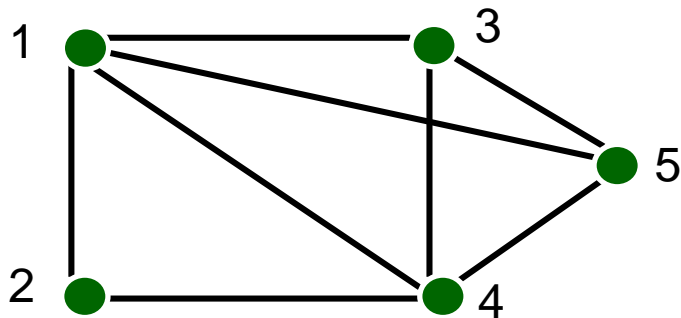
$$F = C_n + n - k$$

其中 C_n 为目前团的顶点数（初始为0）

k 为结点层数

最坏情况下时间： $O(n2^n)$

实例



顶点编号顺序为 1, 2, 3, 4, 5,

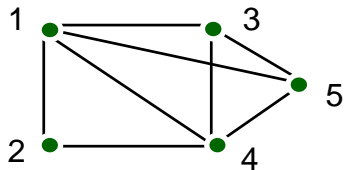
对应 x_1, x_2, x_3, x_4, x_5 ,

$x_i = 1$ 当且仅当 i 在团内

分支规定左子树为 1, 右子树为 0.

B 为界, F 为代价函数值.

搜索树



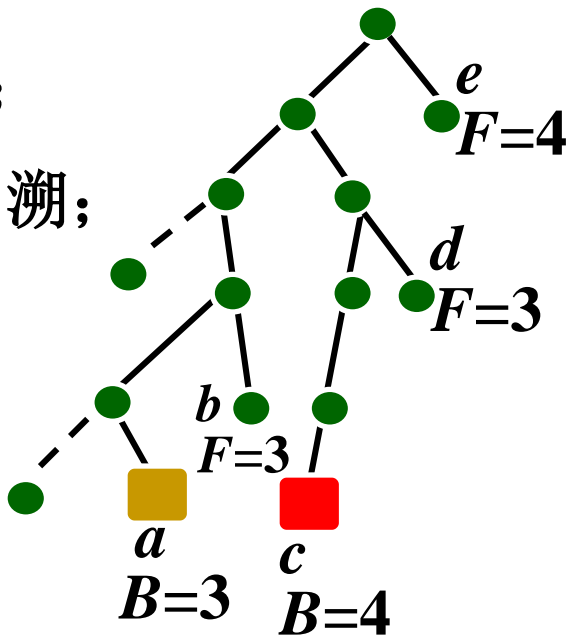
a: 极大团 $\{1,2,4\}$,
顶点数为 3, 界 $B=3$;

b: 代价函数值 $F=3$, 回溯;

c: 极大团 $\{1,3,4,5\}$,
顶点数为 4,
修改界 $B=4$;

d: $F=3$, 不必搜索;

e: $F=4$, 不必搜索.



输出最大团 $\{1,3,4,5\}$, 顶点数为 4.

小结

- 最大团问题的定义
与点独立集的关系
- 分支限界算法的设计
树的结构：子集树
分支约束条件
代价函数与界的设定
- 最坏情况的时间复杂度： $O(n2^n)$

货郎问题

货郎问题的定义

- 输入

有穷个城市的集合 $C=\{c_1, c_2, \dots, c_n\}$,

距离 $d(c_i, c_j)=d(c_j, c_i) \in \mathbb{Z}^+$, $1 \leq i < j \leq n$

- 解: $1, 2, \dots, n$ 的排列 k_1, k_2, \dots, k_n 使得:

$$\min \left\{ \sum_{i=1}^{n-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_n}, c_{k_1}) \right\}$$

算法设计

解向量为 $\langle 1, i_1, i_2, \dots, i_{n-1} \rangle$, 其中 i_1, i_2, \dots, i_{n-1} 为 $\{2, 3, \dots, n\}$ 的排列.

搜索空间为排列树, 结点 $\langle i_1, i_2, \dots, i_k \rangle$ 表示得到 k 步路线.

约束条件: 令 $B = \{ i_1, i_2, \dots, i_k \}$, 则

$$i_{k+1} \in \{ 2, \dots, n \} - B$$

即每个结点只能访问一次.

代价函数与界

界：当前得到的最短巡回路线长度

代价函数：设顶点 c_i 出发的最短边长度为 l_i ， d_j 为选定巡回路线中第 j 段的长度

$$L = \sum_{j=1}^k d_j + l_{i_k} + \sum_{i_j \notin B} l_{i_j}$$

已走过
路径长

剩余长
度下界

代价函数

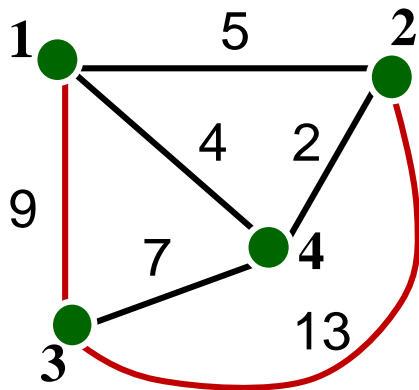
$$L = \sum_{j=1}^k d_j + l_{i_k} + \sum_{i_j \notin B} l_{i_j}$$

部分路线<1,3,2>

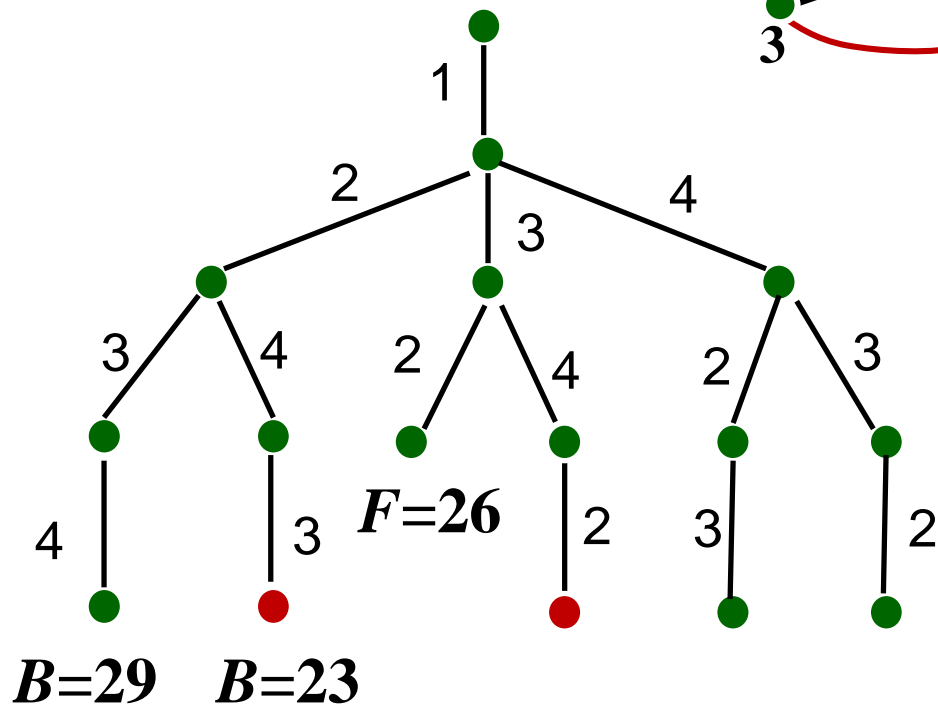
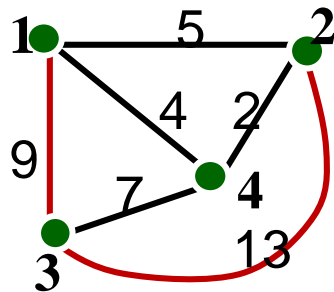
$$L=9+13+2+2=26$$

9+13为走过的路径长度

后两项分别为从结点2及4出发的最短边长



搜索树



实例运行

深度优先遍历搜索树

- 第一个界: $\langle 1, 2, 3, 4 \rangle$, 长度为 29
- 第二个界: $\langle 1, 2, 4, 3 \rangle$, 长度为 23
- 结点 $\langle 1, 3, 2 \rangle$: 代价函数值 $26 > 23$, 不再搜索, 返回 $\langle 1, 3 \rangle$, 右子树向下
- 结点 $\langle 1, 3, 4 \rangle$, 代价函数值 $9 + 7 + 2 + 2 = 20$, 继续, 得到可行解 $\langle 1, 3, 4, 2 \rangle$, 长度 23.
- 回溯到结点 $\langle 1 \rangle$, 沿 $\langle 1, 4 \rangle$ 向下...

最优解 $\langle 1, 2, 4, 3 \rangle$ 或 $\langle 1, 3, 4, 2 \rangle$, 长度 23

算法分析

- 搜索树的树叶个数： $O((n-1)!)$ ，
每片树叶对应 1 条路径，每条路径有 n 个结点.
- 每个结点代价函数计算时间 $O(1)$ ，
每条路径的计算时间为 $O(n)$
- 最坏情况下算法的时间 $O(n!)$

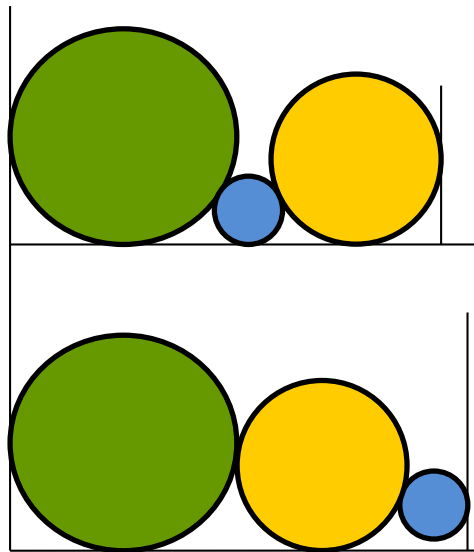
小结

- 货郎问题的分支限界算法：
约束条件：只能选没有走过的结点
代价函数：走过长度+后续长度的
下界
- 时间复杂度： $O(n!)$

园排列问题

圆排列问题

给定 n 个圆的半径序列, 各圆与底线相切排列, 假定每个圆占大于1的长度, 求具有最小长度 l_n 的圆的排列顺序.



三个圆,半径给定
两种排列具有不同排列长度, 第一种方式具有更小的排列长度

算法设计：分支限界

解： $\langle i_1, i_2, \dots, i_n \rangle$ 为 $1, 2, \dots, n$ 的排列

搜索空间为排列树

部分解向量 $\langle i_1, i_2, \dots, i_k \rangle$ ：表示前 k 个圆已排好. 令

$$B = \{i_1, i_2, \dots, i_k\}$$

下一个圆选择 i_{k+1}

约束条件： $i_{k+1} \in \{1, 2, \dots, n\} - B$

界： 当前得到的最小圆排列长度 3

符号说明

k : 算法已经选择了第1— k 个圆

r_k : 第 k 个圆的半径

d_k : 第 $k-1$ 个圆到第 k 个圆的圆心水平距离, $k>1$

x_k : 第 k 个圆的圆心坐标, 规定 $x_1=0$,

l_k : 第 1— k 个圆的排列长度

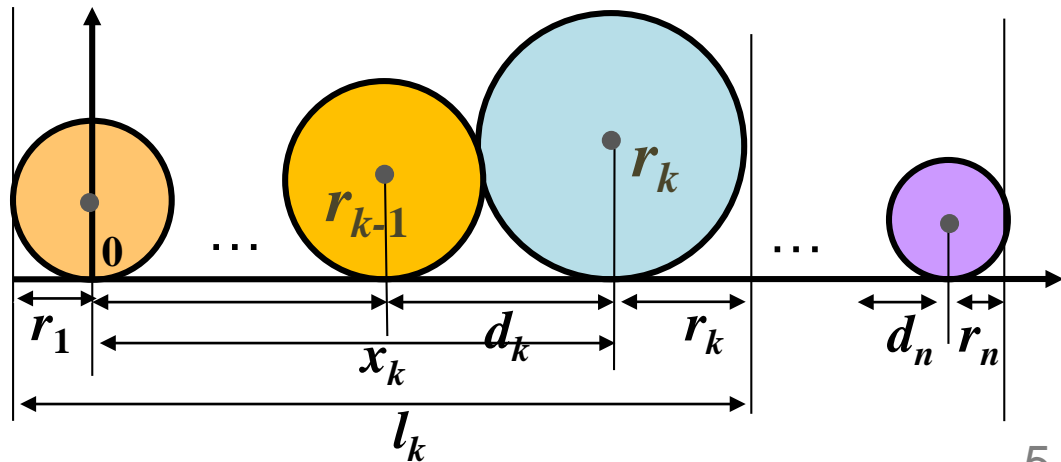
L_k : 放好 1— k 个圆以后, 对应结点的代价函数值 $L_k \leq l_n$

圆排列长度估计

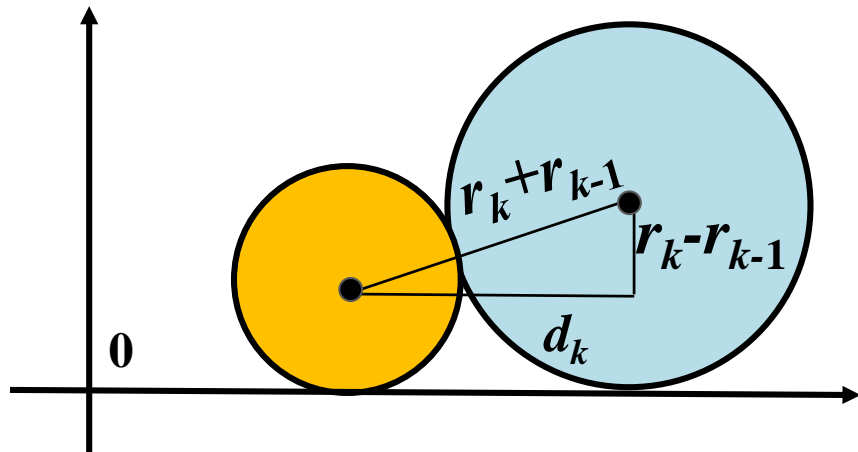
$$x_k = x_{k-1} + d_k$$

部分排列长度 $l_k = x_k + r_k + r_1$

排列长度 $l_n = x_k + d_{k+1} + d_{k+2} + \dots + d_n + r_n + r_1$



有关量的计算



$$\begin{aligned} d_k &= \sqrt{(r_k + r_{k-1})^2 - (r_k - r_{k-1})^2} \\ &= 2\sqrt{r_{k-1}r_k} \end{aligned}$$

代价函数

$$d_k = 2\sqrt{r_{k-1}r_k}$$

排列
长度

$$l_n = x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + \dots + 2\sqrt{r_{n-1} r_n} + r_n + r_1$$

$$\geq x_k + 2(n-k)r + r + r_1$$

下界

$$L_k = x_k + (2n - 2k + 1)r + r_1$$

$$r = \min(r_{i_j}, r_k)$$

$$i_j \in \{1, 2, \dots, n\} - B$$

时间复杂度

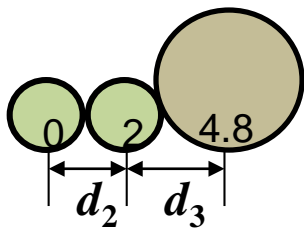
- 搜索树树叶数为 $O(n!)$
- 每条路径代价函数的计算为 $O(n)$
- 最坏情况下算法时间复杂度为
 $O(n \cdot n!) = O((n+1)!)$

实例

输入: $R=\{1,1,2,2,3,5\}$

排列 $\langle 1,2,3,4,5,6 \rangle$

半径排列: $1,1,2,2,3,5$



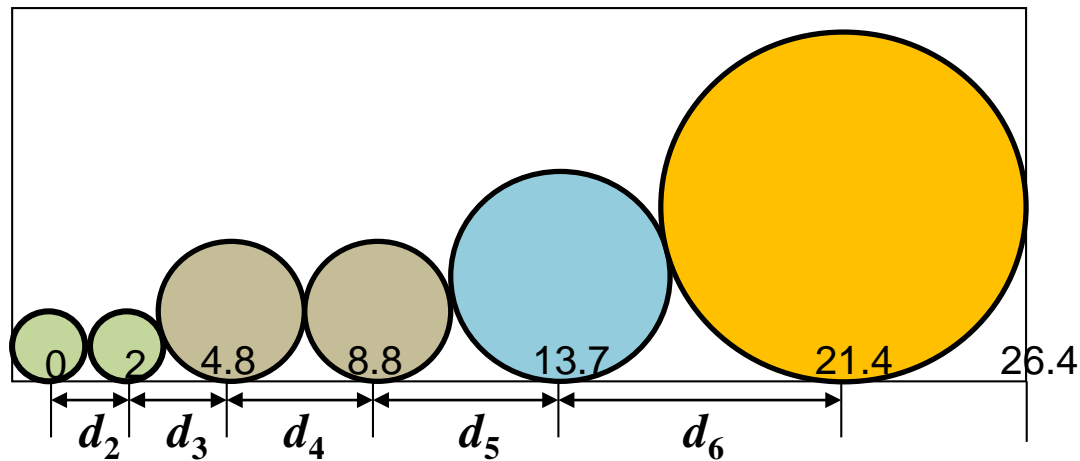
$$L_3=7.8+12=19.8$$

k	r_k	d_k	x_k	l_k	L_k
1	1	0	0	2	12
2	1	2	2	4	12
3	2	2.8	4.8	7.8	19.8
4	2	4	8.8	11.8	19.8
5	3	4.9	13.7	17.7	23.7
6	5	7.7	21.4	27.4	27.4

实例计算

半径排列: 1,1,2,2,3,5, 可行解 $l_6=27.4$

最优解: $\langle 1,3,5,6,4,2 \rangle$, 最短长度26.5



小结


- 园排列问题的定义
- 如何设计代价函数？
已有的排列长度+后续的圆排列
长度的下界
- 时间复杂度估计： $O((n+1)!)$

连续邮资问题

连续邮资问题

问题： 给定 n 种不同面值的邮票, 每个信封至多贴 m 张, 试给出邮票的最佳设计, 使得从 1 开始, 增量为 1 的连续邮资区间达到最大?

实例： $n = 5, m = 4$.

设计1: 面值 $X_1 = \langle 1, 3, 11, 15, 32 \rangle$,
 邮资连续区间 $\{1, 2, \dots, 70\}$

设计2: 面值 $X_2 = \langle 1, 6, 10, 20, 30 \rangle$,
邮资连续区间 $\{1, 2, 3, 4\}$

算法设计

可行解 : $\langle x_1, x_2, \dots, x_n \rangle$, $x_1 = 1$,
 $x_1 < x_2 < \dots < x_n$

搜索策略: 深度优先

约束条件: 在结点 $\langle x_1, x_2, \dots, x_i \rangle$ 处,
邮资最大连续区间为 $\{1, \dots, r_i\}$, x_{i+1}
的取值范围是

$$\{x_i + 1, \dots, r_i + 1\}$$

若 $x_{i+1} > r_i + 1$, $r_i + 1$ 的邮资将没法支付.

r_i 的计算

$y_i(j)$: 用至多 m 张面值 x_i 的邮票加上 x_1, x_2, \dots, x_{i-1} 面值的邮票贴 j 邮资时的最少邮票数, 则

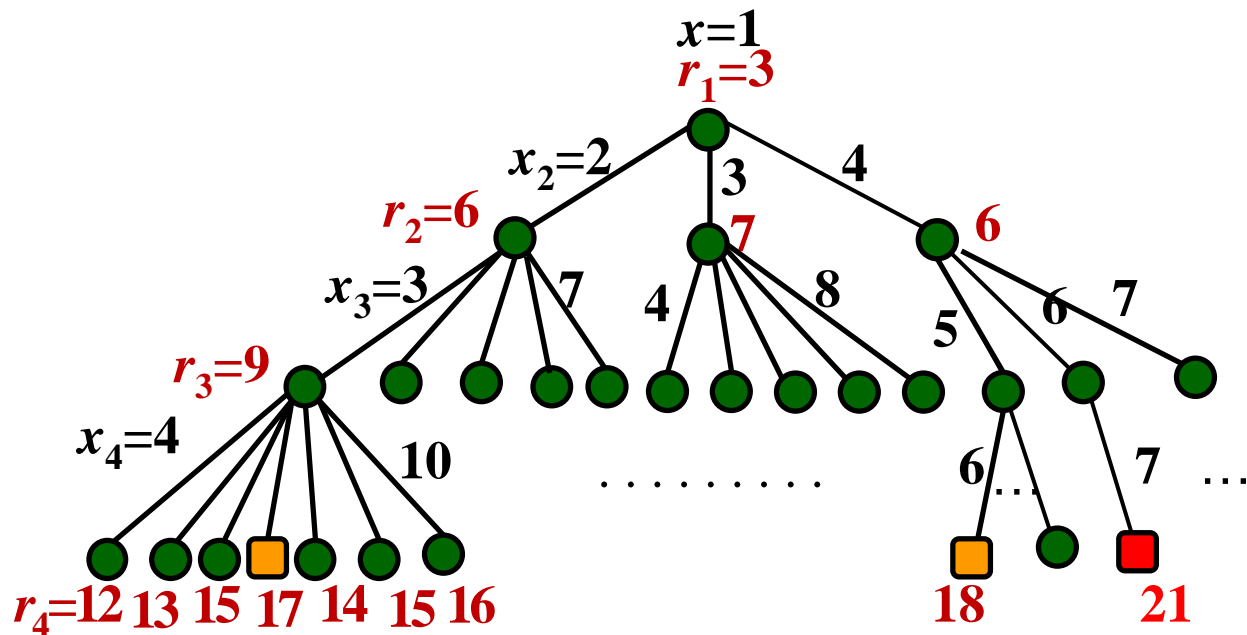
$$y_i(j) = \min_{1 \leq t \leq m} \{t + y_{i-1}(j - t x_i)\}$$

$$y_1(j) = j$$

$$r_i = \min\{j \mid y_i(j) \leq m, y_i(j+1) > m\}$$

界: max , m 张邮票连续付的最大邮资

部分搜索树 $n=4, m=3$



解: $X=\langle 1,4,6,7 \rangle$, 最大连续区间 $\{1,\dots,21\}$ 5

回溯算法小结

- (1) 适于求解组合搜索问题及优化问题
- (2) 求解条件：满足多米诺性质
- (3) 解的表示：解向量，求解是不断扩充解向量的过程
- (4) 回溯条件：
 - 搜索问题——约束条件
 - 优化问题——约束条件 + 代价函数
- (5) 分支策略：深度优先、宽度优先、宽深结合、函数优先

小结（续）

(6) 结点状态：白结点,黑结点,灰结点

(7) 算法时间复杂度：

$$W(n) = (p(n) f(n))$$

其中 $p(n)$ 为每个结点的工作量

$f(n)$ 为结点个数

最坏情况下时间通常为指数级

平均情况下比蛮力算法好

空间代价小

小结（续）

(8) 降低时间复杂性的主要途径：

- 根据树的分支设计优先策略
结点少分支优先，解多分支优先
- 利用搜索树的对称性裁减子树
- 分解为子问题，若求解时间 $f(n)=c2^n$ ，
组合时间 $O(2^{n/k})$ ，分解为 k 个 n/k 规模
子问题，则该算法时间为
$$T(n)=kc2^{n/k}+O(2^{n/k})=O(2^{n/k})=o(2^n)$$

算法设计与分析

课程总结

算法课程的知识框架

分治策略

动态规划

贪心法

回溯与分支限界

使用条件
主要步骤
分析方法
改进途径
典型例子

算法的数学基础：
序列求和、递推方程求解

算法的基本概念：
算法的伪码表示
算法的两种时间复杂度
复杂度函数阶的表示

函数的阶

- 阶的符号: $O, \Omega, \Theta, o, \omega$
- 阶的高低
 - 至少指数级: $2^n, 3^n, n!, \dots$
 - 多项式级: $n, n^2, n \log n, n^{1/2}, \dots$
 - $\log n$ 的多项式级: $\log n, \log^2 n, \dots$

- 注意

阶反映的是大的 n ($n > n_0$) 的情况
可以忽略前面的有限项

序列求和

- 基本求和公式

等比数列

等差数列

调和数级数

- 估计和式的阶

放大，然后估计上界

用积分估计上下界

递推方程求解

- 主要的求解方法

迭代 + 进行序列求和

递归树 + 求和

主定理：注意条件验证

- 一些常见的递推方程的解

$$f(n) = af\left(\frac{n}{b}\right) + d(n)$$

算法设计技术

- 设计技术

分治策略

动态规划

贪心法

回溯和分支限界

- 关注问题

使用条件

主要设计步骤

时间复杂度分析方法

改进途径

典型例子

分治策略

- **适用条件**: 归约为独立求解子问题
- **设计步骤**: 归约方法, 初始子问题的计算, 子问题解的综合方法. 注意子问题划分均衡, 类型相同
- **递归算法分析**: 求解递推方程
- **改进途径**: 减少子问题数, 预处理
- **典型问题**: 二分检索, 归并排序, 芯片测试, 幂乘, 矩阵乘法, 最临近点对, 多项式求值

动态规划

- **适用条件**: 优化问题, 多步判断求解, 满足优化原则, 子问题重叠
- **设计步骤**: 确定子问题边界, 列关于目标函数的递推方程及初值; 自底向上, 备忘录存储; 标记函数及解的追踪方法
- **复杂度分析**: 备忘录, 递推方程
- **典型问题**: 矩阵链相乘, 投资, 背包, 最长公共子序列, 图像压缩, 最大子段和, 最优二分检索树, 生物信息学应用

贪心法

- **适用条件**: 组合优化问题, 多步判断求解, 有贪心选择性质
- **设计步骤**: 局部优化策略的确定及算法正确性证明 (直接证明, 数学归纳法, 交换论证)
- **复杂度分析**
- **典型问题**: 活动选择, 装载问题, 最小延迟调度, 最优前缀码, 最小生成树, 单源最短路

回溯和分支限界

- **适用条件**: 搜索或优化问题, 多步判断求解, 满足多米诺性质
- **设计步骤**: 确定解向量, 搜索树结构, 搜索顺序, 结点分支搜索的约束条件与代价函数, 路径存储
- **搜索树结点数估计**
- **复杂度分析**
- **典型问题**: n 后问题, 背包问题, 货郎问题, 装载问题, 最大团问题, 圆排列问题, 连续邮资问题

算法设计

- 设计思想：尽量选复杂度低的算法
- 算法实现依赖于数据结构，选择合适的数据结构
- 实际问题中的综合考虑：时空权衡，实现成本的权衡, ...