Algorithmen und Wahrscheinlichkeit

Bsc. Informatik ETH Zürich

Laurin Seeholzer

June 1, 2026

Contents

1 Connectivness of Graphs

Definition 1.1. Let $G = (V, E)$ be a graph, $u, v \in V, u \neq v$. u and v are called **connected** if there is a $u - v$-path in G.

Definition 1.2. A graph $G = (V, E)$ is called **connected** if $\forall u, v \in V, u \neq v$ a $u - v$-path in G exists.

Definition 1.3. Let $G = (V, E)$ be a graph, $u, v \in V, u \neq v$. u and v are called k**-connected** if $|V| \geq k + 1$ and

$$\forall X \subseteq V \setminus \{u, v\} \text{ with } |X| < k, u \text{ and } v \text{ are connected in } G[V \setminus X]$$

Definition 1.4. A graph $G = (V, E)$ is called k**-connected** if $|V| > k$ and

$$\forall X \subseteq V \text{ with } |X| < k, G[V \setminus X] \text{ is connected}$$

Definition 1.5. Let $G = (V, E)$ be a graph, $u, v \in V$ and $X \subseteq V \setminus \{u, v\}$. X is called a $u - v$**-separator** if u and v are in different connected components of $G[V \setminus X]$.

Remark 1.6. More generally one can say that if a Graph G contians a vertex v stuch that $deg(v) < k$, then G can not be k-connected.

Theorem 1.7. Menger's Theorem (Vertex Version): Let $G = (V, E)$ be a graph and $u, v \in V$. Every $u - v$-separator X has size $\mid X \mid \geq k$ iff there are k inernally vertexdisjoint $u - v$-paths.

Theorem 1.8. Menger's Theorem (All Vertices Version): A graph G is k-connected iff. $\forall u, v \in V, u \neq v$ there are k internally vertex-disjoint $u - v$-paths.

Definition 1.9. A graph $G = (V, E)$ is called k**-edge-connected** iff

$$\forall X \subseteq E \text{ with } |X| < k, (V, E \setminus X) \text{ is connected}$$

Theorem 1.10. Menger's Theorem (Edge Version): A graph G is k-edge-connected iff. $\forall u, v \in V, u \neq v$ there are k edge-disjoint $u - v$-paths.

Remark 1.11. The following always holds:

$$\text{Vertex Connectivity} \leq \text{Edge Connectivity} \leq \text{Minimum Degree } \delta(G)$$

1.1 Specialcase $k = 1$

Definition 1.12. Let $G = (V, E)$ be a connected graph.

- A vertex $v \in V$ is called a **cut vertex** (Artikulationsknoten) iff. $G[V \setminus \{v\}]$ is disconnected.
- An edge $e \in E$ is called a **cut edge** (Brücke) iff. $(V, E \setminus \{e\})$ is disconnected.

Lemma 1.13. If $e = \{x, y\} \in E$ is a cut edge, then $\deg(x) = 1$ or x is a cut vertex (analogously for y).

Definition 1.14. Blocks: We define an equivalence relation $\sim$ on E by:

$$e \sim f \iff e = f \vee \exists \text{ cycle containing } e \text{ and } f$$

The equivalence classes are called **blocks**.

Remark 1.15. Note that two blocks can share at most one vertex, more over this vertex has to be a cut vertex.

Definition 1.16. Block Graph: The block graph of a connected graph G is a bipartite graph $T = (A \cup B, E_T)$ where:

- $A = \{\text{cut vertices of } G\}$
- $B = \{\text{blocks of } G\}$
- Edge $\{a, b\} \in E_T \iff a$ is incident to an edge in block b.

Theorem 1.17. If G is connected, the block graph of G is a tree.

1.2 Finding Cut Vertices and Bridges (DFS & Low Values)

To find cut vertices and bridges efficiently in linear time $O(|V| + |E|)$, we use a modified **Depth-First Search (DFS)** to compute a spanning forest (or tree T if G is connected) and assign two numbers to each vertex: a DFS number $\text{dfs}[v]$ and a low-value $\text{low}[v]$.

Definition 1.18. During DFS, the edges of G are classified into:

- **Tree edges**: Edges in the DFS spanning tree T traversed to discover new vertices.
- **Back edges** (Restkanten): Edges in $G \setminus T$ that connect a vertex to one of its ancestors in the DFS tree.

Definition 1.19. For every vertex $v \in V$, the low-value $\text{low}[v]$ is defined as:

$$\text{low}[v] := \min \Big\{ \text{dfs}[u] \;\Big|\; u \text{ is reachable from } v \text{ by a path consisting of any number of tree edges followed by at most one back edge} \Big\}$$

It can be recursively computed as:

$$\text{low}[v] = \min \left\{ \text{dfs}[v], \min_{\{v,w\} \in E_{\text{back}}} \text{dfs}[w], \min_{\{v,w\} \in E_{\text{tree}}, w \text{ child of } v} \text{low}[w] \right\}$$

Theorem 1.20. Characterization of Cut Vertices and Bridges: Let T be a DFS tree of a connected graph G started at root s.

- The root s is a **cut vertex** iff s has at least two children in T.
- A vertex $v \neq s$ is a **cut vertex** iff there exists a child w of v in T such that:
$$\text{low}[w] \geq \text{dfs}[v]$$
- A tree edge $\{v, w\}$ (where w is a child of v) is a **bridge** (cut edge) iff:
$$\text{low}[w] > \text{dfs}[v]$$

Algorithm 1.21

```
1     #DFS-Visit(G, v)
2     #Computes low values and identifies cut vertices recursively.
3
4     1. num := num + 1
5     2. dfs[v] := num
6     3. low[v] := dfs[v]
7     4. isArtVert[v] := false
8     5. for each neighbor w of v do:
9     if dfs[w] = 0 then:
```

```
10      T := T U {(v, w)}
11      val := DFS-Visit(G, w)
12      if val >= dfs[v] then:
13      isArtVert[v] := true
14      low[v] := min(low[v], val)
15      else if {v, w} not in T then:
16      low[v] := min(low[v], dfs[w])
17      6. return low[v]
```

Algorithm 1.22

```
1       #DFS(G, s)
2       #Main driver to find all cut vertices and bridges of a
            connected graph.
3
4       1. for each vertex v in V:
5       dfs[v] := 0
6       2. num := 0
7       3. T := empty set
8       4. DFS-Visit(G, s)
9       5. if s has >= 2 children in T then:
10      isArtVert[s] := true
11      else:
12      isArtVert[s] := false
```

Theorem 1.23. For any connected graph $G = (V, E)$ stored as adjacency lists, all cut vertices and bridges can be computed in time $O(|V| + |E|)$ and space $O(|V|)$.

2 Hamiltonian Cycles and Eulertours

2.1 Eulertours

Definition 2.1. An **Eulertour** is a closed walk in a graph G that contains every edge exactly once.

Theorem 2.2. A connected graph $G = (V, E)$ contains an Eulertour iff the degree of every vertex is even.

Remark 2.3. We can find an Eulertour in time $O(|E|)$. Using an algorithm illustrated by a Runner and a Turtle. The Runner is set lose and traverses the graph until he reaches a vertex he has already visited (we found a cycle). Then the Turtle is set lose and traverses the graph along that path until it reaches a vertex that has unvisited neighbours, from where on the Runner is set lose again (we expand the cycle by findin a new one adjacent to it). This process expands the tour by a cycle at a time until all edges have been traversed.

2.2 Hamiltonian Cycles

Definition 2.4. A **Hamiltonian Cycle** is a cycle in $G = (V, E)$ that contains every vertex exactly once.

Theorem 2.5. Let $m, n \geq 2$. An $n \times m$ grid contains a Hamiltonian cycle iff $n \cdot m$ is even.

Theorem 2.6. A bipartite graph $G = (A \cup B, E)$ with $|A| \neq |B|$ cannot contain a Hamiltonian cycle.

Example 2.7. hamiltonian cycles in Hypercubes Let Q_n be the n-dimensional hypercube. Q_n has 2^n vertices and $n \cdot 2^{n-1}$ edges. Q_n is bipartite and has a Hamiltonian cycle iff $n \geq 2$ therfore we can find a Hamiltonian cycle in Q_n for all $n \geq 2$.

Theorem 2.8. Dirac's Theorem (1952): Every graph $G = (V, E)$ with $|V| \geq 3$ and minimum degree $\delta(G) \geq |V|/2$ contains a Hamiltonian cycle.

Proof. We proof by induction on the number of vertices in a cycle.

1. Lemma: Observe that because $\delta(G) \geq |V|/2$ we have that the neighbour-hoods of any two vertices intersect and therfore there is a $u - v$ path for every $u, v \in V$.

2. k-cycle $\implies$ $k+1$-path: Let C be a cycle of length $k \leq |V|$. As G is connected there exists some edge $(u, v) \in E$ with $u \in C$ and $v \notin C$. Hence there exists a path of length $k+1$

3. k-path $\implies$ $k+1$-path or k-cycle: Let $P = \{v_1, \dots v_k\}$ be a path in G. If $N(v_1)\text{or}N(v_k) \not\subseteq \{v_1, \dots, v_k\}$, then we can extend P to a cycle of length $k+1$. If $N(v_1)\text{and}N(v_k) \subseteq \{v_1, \dots v_k\}$, then we define $N_k^+ := \{v_{i+1} \mid v_i \in N(v_k)\} \subseteq \{v_2, \dots, v_k\}$. By the lemma we know that $N(v_1) \cap N_k^+ \neq \emptyset$ and therfore there exists some $v_j \in N(v_1) \cap N_k^+$. This means that v_j is adjacent to v_1 and v_{j-1} is adjacent to v_k. Hence we can extend P to a cycle of length $k+1$ by taking the path $v_1, v_{j-1}, v_{j-2}, \dots, v_2, v_k, v_{k-1}, \dots, v_j, v_1$.

Via induction on the length of the longest path in G we can show that for $k =\mid V \mid$ there has to be a $k+1$ path or a k-cycle and as a k cylce is not possible there must exist a k-cycle. □

Algorithm 2.9

```
#Dynamic Programming for Hamiltonian Cycle
#Time complexity O(n^2 * 2^n).
#Define P[S, x] = 1 if there is a 1-x path containing exactly
    the vertices in S.

P[S, x] = max { P[S \ {x}, y] | y in N(x) intersect S, y != 1 }
```

2.3 Counting Hamiltonian Cycles

Finding if a graph has a Hamiltonian cycle is NP-complete. However, we can efficiently *count* the number of Hamiltonian cycles in a graph using the **Principle of Inclusion-Exclusion (PIE)** in time $O(n^{2.81} \log n \cdot 2^n)$ and polynomial space $O(n^2)$, which is much more space-efficient than the $O(n \cdot 2^n)$ space required by dynamic programming.

Definition 2.10. Let $G = (V, E)$ be a graph with $V = [n]$. Choose an arbitrary start vertex $s \in V$. For any subset $S \subseteq V \setminus \{s\}$, we define:

$W_S :=$ {Walks of length n in G with start and end vertex s that do not visit any vertices in S}

Remark 2.11. The size of the set W_S can be computed efficiently: let A_S be the adjacency matrix of the induced subgraph $G[V \setminus S]$. Then $|W_S|$ is exactly the (s, s)-entry of the matrix $(A_S)^n$. Using matrix exponentiation with Strassen's algorithm, this takes $O(n^{2.81} \log n)$ time and $O(n^2)$ space.

Theorem 2.12. The number of Hamiltonian cycles in $G = (V, E)$ is given by:

$$\text{Number of Hamiltonian Cycles} = \frac{1}{2} \sum_{S \subseteq V \setminus \{s\}} (-1)^{|S|} |W_S|$$

Proof. A walk of length n starting and ending at s is a Hamiltonian cycle if and only if it visits every vertex of G exactly once (which means it visits all n vertices). The set $W_\emptyset$ contains all walks of length n from s to s in G. However, this also includes walks that do not visit all vertices. The set of "invalid" walks is $\bigcup_{i \in V \setminus \{s\}} W_{\{i\}}$, since any walk that misses at least one vertex $i \neq s$ is in this union. Thus, the number of walks of length n starting and ending at s that visit all vertices is:

$$\left| W_\emptyset \setminus \bigcup_{i \in V \setminus \{s\}} W_{\{i\}} \right|$$

Using the Principle of Inclusion-Exclusion, and noting that $\bigcap_{i \in S} W_{\{i\}} = W_S$, we get:

$$\left| W_\emptyset \setminus \bigcup_{i \in V \setminus \{s\}} W_{\{i\}} \right| = \sum_{S \subseteq V \setminus \{s\}} (-1)^{|S|} |W_S|$$

Since every Hamiltonian cycle can be traversed in two directions, each cycle is counted exactly twice as a walk from s to s. Dividing by 2 yields the total number of Hamiltonian cycles. □

Algorithm 2.13

```
#Zaehle_Hamiltonkreise (Count Hamiltonian Cycles)
#Time complexity: O(n^2.81 * log n * 2^n), Space complexity: O(
    n^2)

1. Choose arbitrary start vertex s := 1
2. Initialize Z := |W_\emptyset|
3. For all non-empty subsets S \subseteq V \setminus {s}:
Determine |W_S| using the adjacency matrix of G[V \ S]
Z := Z + (-1)^|S| * |W_S|
4. Z := Z / 2
5. Return Z
```

Theorem 2.14. The algorithm Zaehle_Hamiltonkreise correctly computes the number of Hamiltonian cycles in G and requires $O(n^2)$ space and $O(n^{2.81} \log n \cdot 2^n)$ time, where $n = |V|$.

Theorem 2.15. Hamiltonian decision is NP-Complete. (Karp 1972)

2.4 Travelling Salesman Problem (TSP)

The travelling salesman problem is the problem of finding a Hamiltonian cycle in a complete graph K_n with minimal weight. In other words, the shortes roundtrip/roundtour.

Theorem 2.16. The travelling salesman problem is NP-Complete.

Proof. This is easy to prove by reduction from Hamiltonian Cycle. Assume we have a Graph G and want to check if it is Hamiltionian. Now lets extend the graph to be complete and set the weight w_e of every edge $e = (u, v) \in G$ to 0 and the weight of all the newly added edges to 1. Now we can check if the TSP has a solution with weight 0. If it does, then G has a Hamiltonian cycle, otherwise it does not. Therfore TSP has to be NP-Complete as Hamiltionian decision is to. □

2.4.1 Metric TSP

Assume we have a graph G for which the triangle inequality holds, i.e.:

$$w(u, v) \leq w(u, w) + w(w, v) \quad \forall u, v, w \in V$$

Theorem 2.17. For Metric TSP there is a 2 and even a 1.5 approximation algorithm in polynomial time.

Algorithm 2.18

```
#2-Approximation for Metric TSP
#This Hamiltonian cycle has length at most 2 * |E(T)| = 2 * (|V|-1).

1. Determine a minimal-spanning tree T
2. Double all the edges of T to get an Eulerian multigraph T'
3. Go along that Eulertour and skip every vertex that has already been visited
```

Theorem 2.19. Christofides Algorithm: Using MST, Matchings and Eulertours we can find a 1.5 approximation algorithm for Metric TSP in polynomial time.

Algorithm 2.20

```
#Christofides 1.5-Approximation for Metric TSP

1.  Determine a minimal-spanning tree T
Observe that |E(T)| <= |E(H)| for any Hamiltonian cycle H.
2.  Find a minimal matching for the odd-degree vertices of T
Observe that |M| <= |E(H)|/2.
3.  Determine Eulertour W in T with the edges of M added
Observe that |W| <= |E(T)| + |M| <= |E(H)| + |E(H)|/2 = 1.5 * |E(H)|.
```

3 Matchings

Definition 3.1. An edge set $M \subseteq E$ is called a **matching** in $G = (V, E)$ if no vertex is incident to more than one edge from M.

Definition 3.2. A vertex v is **covered** by M if it is incident to an edge in M.

Definition 3.3. A matching M is

- **perfect** if every vertex is covered ($|M| = |V|/2$).
- **maximal** if it cannot be extended by adding edges.
- **maximum** if there is no matching with more edges.

Theorem 3.4. The geedy algorithm leads to a maximal matching M_{greedy} with $\mid M_{greedy} \mid \geq \frac{1}{2} \mid M_{max} \mid$.

Proof. Observe that every edge in M_{max} covers at least one edge of M_{greedy} else we would be able to add it to M_{greedy}. Further more every edge in M_{greedy} covers at most two edges of M_{max}. □

Algorithm 3.5

```
1     M = empty set
2     while E is not empty:
3     choose an arbitrary edge e from E
4     add e to M
5     remove all edges incident to the endpoints of e from E
```

Definition 3.6. An **M-augmenting path** is a path that alternates between edges in M and not in M, starting and ending in uncovered vertices.

Remark 3.7. If we fing an M_{greedy}-augmenting path we can switch the edges along that path to increase the size of the matching by 1.

Theorem 3.8. Hall's Marriage Theorem: A bipartite graph $G = (A \cup B, E)$ contains a matching of cardinality $|M| = |A|$ iff:

$$\forall X \subseteq A : |X| \leq |N(X)|$$

Proof.
$\Longrightarrow$ This is trivial as if such a matchin exists then of course for every subset $X \subseteq A$ we have $|X| \leq |N(X)|$ as we can match every vertex in X with a distinct vertex in $N(X)$.
$\Longleftarrow$ We show this using induction over the size of A. The BC is $\mid A \mid = 1$ is trivially true. If $\forall |X| \leq |N(X)|$ holds then we chose a random edge (u, v) and delete all incident edges and show that the theorem still holds. If $\exists |X_0| = |N(X_0)|$ then we show that the graphs $G_1 = (A \setminus X_0 \cup B \setminus N(X_0))$ and $G_2 = (X_0 \cup N(X_0))$ the theorem still holds. □

Theorem 3.9. Frobenius Corollary: Every k-regular bipartite graph contains a perfect matching.

Remark 3.10. It even holds that every k-regular bipartite graph is a union of perfect matchings.

Theorem 3.11. In a k- and 2^k-regular bipartite graph we can find a perfect matching in $O(|E|)$ time.

Theorem 3.12. In bipartite grapahs we can find a perfect mathcin in time $O(|V||E|)$.

Remark 3.13. Important **matching-algorithms**:

- $O(|V|^{\frac{1}{2}}|E|)$ Hopcroft-Karp (bipartite unweighted)
- $O(|E|^{1+O(1)})$ Hpcroft-Karp (bipartite polynomial-weighted)
- $O(|V|^{\frac{1}{2}}|E|)$ Micali-Vazirani (general unweighted)
- $O(|V|^{\frac{1}{2}}|E|)$ Gabow-Tarjan (general polynomial-weighted)
- $O(|V|^{2.373})$ Mucha-Sankowski (general unweighted)

Theorem 3.14. Berge's Theorem (1957): A matching M is maximum iff there is no M-augmenting path.

Proof. Else we change the edges along the augmenting path to increase the size of the matching by 1 which contradicts that it was a maximum matching. □

Algorithm 3.15

```
1      #BFS to find augmenting paths in bipartite graphs
2
```

```
    L(0) = {uncovered vertices in A} (mark all as visited)

    for i: 1 - n:
    if i odd:
    L(i) = {unvisited neighbours of L(i-1) via edges in E \ M}
    else:
    L(i) = {unvisited partners of L(i-1) via edges in M}
    Mark L(i) as visited

    if vertex v in L(i) not covered
    return path to v
```

Algorithm 3.16

```
    #Hopcroft-Karp Algorithm

    while there exists an augmenting path P:
    find a maximal set of vertex-disjoint shortest augmenting paths
    augment M along all these paths
```

4 Coloring

Definition 4.1. A **k-coloring** of $G = (V, E)$ is a mapping $c : V \to [k]$ such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$.

Definition 4.2. The **chromatic number** $\chi(G)$ is the minimum number of colors.

Definition 4.3. A **stable set** is a set of vertices where no two vertices are adjacent.

Theorem 4.4. $\chi(G) \leq 2 \iff G$ is bipartite.

Theorem 4.5. For $k \geq 3$, the problem "Is $\chi(G) \leq k$?" is NP-complete.

Remark 4.6. For general graphs all ϵ-approximations are NP-hard.

Algorithm 4.7

```
1      #Greedy Coloring
2      #Requires at most maximum degree + 1 colors.
3
4      choose an arbitrary order of vertices v1...vn
5
6      c[v1] = 1
7      for i = 2 to n:
8      c[vi] = min { j in [k] | j != c[vj] for all j < i and (vi, vj)
           in E }
```

Lemma 4.8. For any graph G with maximum degree $\Delta(G)$, the greedy algorithm yields a coloring with at most $\Delta(G) + 1$ colors.

Proof. Let v be a vertex in G. The greedy algorithm assigns to v the smallest color that is not used by any of its neighbors. Since v has at most $\Delta(G)$ neighbors, there are at most $\Delta(G)$ colors that are used by its neighbors. Therefore, there is always a color available for v in the set $[\Delta(G) + 1]$. □

Lemma 4.9. There is an order of vertices $v_1, \ldots, v_n$ such that the greedy algorithm yields a coloring with $\chi(G)$ colors.

Proof. **Intuition**: Let c be an optimal coloring of G. We can order the vertices such that $c(v_1) \leq c(v_2) \leq \cdots \leq c(v_n)$. Therfore each color (stable set) is processed at once and the greedy algorithm assigns the smallest color to each vertex. □

Lemma 4.10. If for the coloring order we have $|N(v_i) \cap \{v_1, \ldots, v_{i-1}\}| \leq k \forall 2 \leq i \leq n$ then the greedy algorithm uses at most $k+1$ colors.

Remark 4.11. Heuristik: v_n has the smallest degree, v_{n-1} has the smallest degree after removing v_n, etc.

Remark 4.12. If every subgraph has a vertex of degree at most k, then the greedy algorithm including the Heuristik yields a coloring with at most $k+1$ colors.

Example 4.13. For trees the greedy algorithm with Heuristik achieves a 2-coloring.

Lemma 4.14. If every block of a block-graph can be colored in k colors so can the block-graph itself.

Proof. **Intuition**: We can rotate the colors in each block such that the colors of the cut vertices match. □

Theorem 4.15. Brooks' Theorem: For a connected graph G that is neither a complete graph K_n nor an odd cycle C_{2n+1} we can find a $\Delta(G)$-coloring in $O(|E|)$ time.

Algorithm 4.16

```
#Algorithm for Brooks' Theorem

if (maxDeg == 2) color in two colors
else if (exists v with deg(v) < maxDeg)
color with greedy and Heuristik
else if (exists cut vertex)
color all blocks with heuristik and color changing
else (find v,x,y, with x,y in N(v) and x,y not adjacent)
remove x,y and color recursively
```

5 Probability Theory

Definition 5.1. A **discrete probability space** is a set of elementary events often denoted as

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$$

Every **elementary event** ω_i has a probability $P[\omega_i] \in [0, 1]$ such that

$$\sum_{\omega_i \in \Omega} P[\omega_i] = 1$$

Remark 5.2. A finite probability space, in which every elementary event has the same probability is called a **Laplace-Space**. Observe that for a Laplace-Space Ω and an arbitrary event E we have:

$$P[E] = \frac{|E|}{|\Omega|}$$

Definition 5.3. An **Event** $E \subseteq \Omega$ is a set of elementary events. The probability of an event is defined as:

$$P[E] = \sum_{\omega_i \in E} P[\omega_i]$$

The complementary event $\bar{E} = \Omega \setminus E$ has probability:

$$P[\bar{E}] = 1 - P[E]$$

Theorem 5.4. For two events A, B in a probability space Ω we have:

- $P[\emptyset] = 0$ and $P[\Omega] = 1$
- $P[A] \in [0, 1]$
- $P[\bar{A}] = 1 - P[A]$
- If $A \subset B$ then $P[A] \leq P[B]$

Theorem 5.5. For events $A_1, A_2, ...A_n$ it holds

$$P[A_1 \cup A_2 \cup \cdots \cup A_n] \leq \sum_{i=1}^{n} P[A_i]$$

with equality iff $A_1, A_2, ...A_n$ are pairwise disjoint.

Proof. For $i \in \{1, \ldots, n\}$ let $B_i = A_i \setminus \bigcup_{j=1}^{i-1} A_j$. Then $P[B_i] \leq P[A_i]$ and for $i \neq j$ we

have $B_i \cap B_j = \emptyset$.

$$P[\bigcup_{i=1}^{n} A_i] = P[\bigcup_{i=1}^{n} B_i] = \sum_{i=1}^{n} P[B_i] \leq \sum_{i=1}^{n} P[A_i]$$

□

Theorem 5.6. Inclusion-Exclusion Principle: For events $A_1, A_2, ...A_n$ with $n \geq 2$ it holds

$$P[\bigcup_{i=1}^{n} A_i] = \sum_{i=1}^{n} (-1)^{i+1} P[\bigcap_{j=1}^{i} A_j]$$

Proof. To proof this theorem we look at an arbitrary $\omega \in \bigcup_{i=1}^{n} A_i$. If ω is in exactly k of the events A_i, then it is counted N_k times where

$$N_k = k - \binom{k}{2} + \binom{k}{3} - \cdots + (-1)^{k+1} \binom{k}{k}$$

We show that $N_k = 1$ for all $k \in \{1, 2, \ldots, n\}$. For that we use the binomial theorem:

$$(x - y)^k = \sum_{i=0}^{k} \binom{k}{i} x^{k-i} (-y)^i$$

Now let $x = 1$ and $y = 1$:

$$(1 - 1)^k = \sum_{i=0}^{k} \binom{k}{i} (-1)^i = \binom{k}{0} - \binom{k}{1} + \binom{k}{2} - \cdots + (-1)^k \binom{k}{k} = 1 - N_k$$

Therfore:

$$N_k = 1 - (1 - 1)^k = 1$$

□

Example 5.7. We shuffle a deck of cards and want to know the probability that any of the cards is at the same place as it was before. Let A_i be the event that the i-th card is at the same place as it was before. Then we want to find

$$P[A_1 \cup A_2 \cup \cdots \cup A_{52}]$$

We define A_S as the event that all cards $i \in S \subseteq [n]$ are at the same place as they were before.

$$P[A_S] = \frac{(n - |S|)!}{n!}$$

Using the inclusion-exclusion principle we get

$$\begin{aligned} P[\bigcup_{i=1}^{n} A_i] &= \sum_{i=1}^{n} A_i - \sum_{S \subset [n], |S|=2} A_S + \sum_{S \subset [n], |S|=3} A_S - \ldots \\ &= n \frac{(n-1)!}{n!} - \binom{n}{2} \frac{(n-2)!}{n!} + \binom{n}{3} \frac{(n-3)!}{n!} - \ldots \\ &= 1 - \frac{1}{2!} + \frac{1}{3!} - \cdots + (-1)^{n+1} \frac{1}{n!} \approx 1 - \frac{1}{e} \approx 0.632 \end{aligned}$$

5.1 Conditional Probability

Definition 5.8. Let A and B be events with $P[B] > 0$. The **conditional probability** of A given B is defined as:

$$P[A \mid B] = \frac{P[A \cap B]}{P[B]}$$

Remark 5.9. Let $A \subseteq \Omega$ be an event, then the following always hold

- $P[A \mid \Omega] = P[A]$
- $P[A \mid A] = 1$
- $P[A \mid \bar{A}] = 0$
- $P[A \cap B] = P[A \mid B]P[B] = P[B \mid A]P[A]$

Example 5.10. Two-Siblings Problem (Zweikinderproblem): Assume a family has two children, and both genders are equally likely at birth. We define the sample space as:

$$\Omega = \{\text{gg}, \text{gb}, \text{bg}, \text{bb}\}$$

where the letters represent the gender (g for girl, b for boy) in order of birth. Each elementary event has probability 1/4. We want to find the probability that both children are girls ($A = \{\text{gg}\}$).

1. **Scenario 1**: We know that the family has at least one girl ($B_{\text{at least one}} = \{\text{gg}, \text{gb}, \text{bg}\}$). Then:
$$P[A \mid B_{\text{at least one}}] = \frac{P[A \cap B_{\text{at least one}}]}{P[B_{\text{at least one}}]} = \frac{1/4}{3/4} = \frac{1}{3}$$
2. **Scenario 2**: We know that the older child is a girl ($B_{\text{older}} = \{\text{gg}, \text{gb}\}$). Then:
$$P[A \mid B_{\text{older}}] = \frac{P[A \cap B_{\text{older}}]}{P[B_{\text{older}}]} = \frac{1/4}{2/4} = \frac{1}{2}$$

This shows how critical the exact conditioning information is, as the gender of the older sibling is independent of the younger one, whereas the global statement of "at least one girl" couples the two events.

Theorem 5.11. Multiplication-Theorem: Let $A_1, A_2, ..A_n$ be events. If $P[A_1 \cap A_2 \cap \cdots \cap A_{n-1}] > 0$ then it holds:

$$P[A_1 \cap A_2 \cap \cdots \cap A_n] = P[A_1] \cdot P[A_2 \mid A_1] \cdot P[A_3 \mid A_1 \cap A_2] \ldots P[A_n \mid A_1 \cap A_2 \cap \cdots \cap A_{n-1}]$$

Proof. We can prove the correctness of this theorem by using the definition of conditional probability:

$$P[A_1] \cdot P[A_2 \mid A_1] \cdot P[A_3 \mid A_1 \cap A_2] \ldots P[A_n \mid A_1 \cap A_2 \cap \cdots \cap A_{n-1}]$$

$$= P[A_1] \cdot \frac{P[A_1 \cap A_2]}{P[A_1]} \cdot \frac{P[A_1 \cap A_2 \cap A_3]}{P[A_1 \cap A_2]} \cdots \frac{P[A_1 \cap A_2 \cap \cdots \cap A_n]}{P[A_1 \cap A_2 \cap \cdots \cap A_{n-1}]}$$
$$= P[A_1 \cap A_2 \cap \cdots \cap A_n]$$

□

Example 5.12. Birthday Problem: What is the probability of two people in a room of m having the same birthday? We find the probability that none of them have the same birthday this is

$$\frac{364}{365} \cdot \frac{363}{365} \cdot \cdots \cdot \frac{365 - m + 1}{365}$$

Theorem 5.13. Theorem of total probability: Let $A_1, ...A_n$ be pairwise disjoint events and let $B \subseteq A_1 \cup ... \cup A_n$ then it holds:

$$P[B] = \sum_{i=1}^{n} P[B \mid A_i]P[A_i]$$

Example 5.14. Goat Problem: Assume three doors of which we know that one hides a sportscar and the other two a goat. You pick one arbitrary door, then one of the other two opens and you see a goat. Should you now switch to the other door that is left or should you stay with the one you are? Let A_1 be the event that you picked the door with the car and A_2 the event that you are infront of the door with a goat. Further let B be the event that you win if you change doors. Now observe

$$Pr[B \mid A_1] = 0$$
$$Pr[B \mid A_2] = 1$$
$$Pr[A_1] = 1/3$$
$$Pr[A_2] = 2/3$$

Using the theorem above we get:

$$Pr[B] = Pr[B \mid A_1]Pr[A_1] + Pr[B \mid A_2]Pr[A_2] = 0 \cdot 1/3 + 1 \cdot 2/3 = 2/3$$

Theorem 5.15. Bayes' Theorem: Let $A_1, A_2...A_n$ be pariwise disjoint events and $B \subseteq A_1 \cup A_2... \cup A_n$ with $P[B] > 0$. Then:

$$P[A_i \mid B] = \frac{P[B \mid A_i]P[A_i]}{\sum_{j=1}^{n} P[B \mid A_j]P[A_j]}$$

Proof. To prove bayes theorem we use the theorem of total probability:

$$P[A_i \mid B] = \frac{P[B \cap A_i]}{P[B]} = \frac{P[B \mid A_i]P[A_i]}{\sum_{j=1}^{n} P[B \mid A_j]P[A_j]}$$

□

Example 5.16. Desease Test: We test a person for a desease that has a probability of $P[A] = 0.01$ to occur. Let A be the event that the tested person has the desease. Further let B be the event that the test was positive. Assume we know the sensitivity of the test $P[B \mid A] = 0.72$ and the specificity $P[\bar{B} \mid \bar{A}] = 0.98$. We want to find the probability that a person has the desease under the condition that the test is positive:

$$P[A \mid B] = \frac{P[B \mid A]P[A]}{P[B \mid A]P[A] + P[B \mid \bar{A}]P[\bar{A}]} = \frac{0.72 \cdot 0.01}{0.72 \cdot 0.01 + 0.02 \cdot 0.99} = \frac{0.0072}{0.027} \approx 0.267$$

Here we see that even if the test is quite accurate, the probability that a person has the desease is still quite low. This is because the probability of the desease is very low. If the probability of the desease was higher, the probability of the test being positive would be higher.

5.2 Independence

Definition 5.17. Two events A and B are **independent** if $P[A \cap B] = P[A]P[B]$ or equivalently $P[A \mid B] = P[A]$.

Definition 5.18. Events $A_1, A_2, \ldots, A_n$ are independent if

$$\forall I \subseteq \{1, 2, \ldots, n\} \quad P[\bigcap_{i \in I} A_i] = \prod_{i \in I} P[A_i]$$

Remark 5.19. For infinite families of events it is not enough to check the condition for all finite subsets.

Lemma 5.20. Let $A_1, A_2 \ldots A_n$ be events and let $A_i^0 \coloneqq \bar{A}_i$ and $A_i^1 \coloneqq A_i$ for $i \in \{1, 2, \ldots, n\}$. Then $A_1, A_2 \ldots A_n$ are independent iff for all $(s_1, \ldots s_n) \in \{0, 1\}^n$ it holds that:

$$P[\bigcap_{i=1}^{n} A_i^{s_i}] = \prod_{i=1}^{n} P[A_i^{s_i}]$$

Proof. $\Longleftarrow$: We proof this direction by induction over the number of events with exponents $s_i = 0$. For the basecase $s_1 = s_2 \cdots = 1$ it holds trivially.

$$P[\bar{A}_1 \cap A_2^{s_2} \cap A_3^{s_3} \ldots] = P[A_2^{s_2} \cap A_3^{s_3} \ldots] - P[A_1 \cap A_2^{s_2} \cap A_3^{s_3} \ldots]$$

Using the incuction hypothesis we get:

$$= \prod_{i=2}^{n} P[A_i^{s_i}] - P[A_1] \prod_{i=2}^{n} P[A_i^{s_i}] = (1 - P[A_1]) \prod_{i=2}^{n} P[A_i^{s_i}] = P[\bar{A}_1] \prod_{i=2}^{n} P[A_i^{s_i}]$$

$\Longrightarrow$: ... □

Lemma 5.21. Let A, B and C be independent events then so are $A \cap B$ and C and also $A \cup B$ and C.

5.3 Random Variables

Definition 5.22. A **random variable** is a map $X : \Omega \to \mathbb{R}$ where Ω is the set of all events of a probability space.

Example 5.23. The event that the random variable X takes a value less than or equal to 5 is denoted as:

$$X \leq 5 \coloneqq \{\omega \in \Omega \mid X(\omega) \leq 5\}$$

Definition 5.24. The **probability mass function** of a discrete random variable X is defined as:

$$f_X(x) = P(X = x)$$

Definition 5.25. The **distribution function** of a discrete random variable X is defined as:

$$F_X(x) = P(X \leq x)$$

5.3.1 Expectation

Definition 5.26. Let X be a random variable. The expectation of X is defined as:

$$\mathbb{E}[X] \coloneqq \sum_{\alpha \in W_x} \alpha P(X = \alpha)$$

aslong as the sum is absolutely convergent, else we say the expectation is undefined.

Lemma 5.27. For any random variable X we have:

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega)P(\omega) = \sum_{k=1}^{\infty} P(X \geq k) - \sum_{k=1}^{\infty} P(X \leq -k)$$

Proof.

$$\mathbb{E}[X] = \sum_{\alpha \in W_X} \alpha P[X = \alpha] = \sum_{\alpha \in W_X} \alpha \sum_{\omega \in \Omega, X(\omega) = \alpha} P[\omega] = \sum_{\omega \in \Omega} X(\omega)P[\omega]$$

□

Theorem 5.28. Let X be a random variable with values in $W_X \subseteq \mathbb{N}_0$. Then:

$$\mathbb{E}[X] = \sum_{k=1}^{\infty} P[X \geq k]$$

Proof.

$$\mathbb{E}[X] = \sum_{k=0}^{\infty} kP[X = k] = \sum_{k=0}^{\infty} \sum_{j=1}^{k} P[X = k] = \sum_{j=1}^{\infty} \sum_{k=j}^{\infty} P[X = k] = \sum_{j=1}^{\infty} P[X \geq j]$$

□

Example 5.29. We throw a coin and we know $P[\text{head}] = p$. We want to know $\mathbb{E}[X]$ with $X :=$ number of throws until we get head. Using the above theorem we get:

$$\mathbb{E}[X] = \sum_{k=1}^{\infty} P[X \geq k] = \sum_{k=1}^{\infty} (1-p)^{k-1} = \frac{1}{1-(1-p)} = \frac{1}{p}$$

Definition 5.30. For an event $E \subseteq \Omega$ the corresponding indicator-variable X_E is defined as:

$$X_E(\omega) := \begin{cases} 1 & \text{if } \omega \in E \\ 0 & \text{if } \omega \notin E \end{cases}$$

Remark 5.31. For an indicator variable X_E we have:

$$\mathbb{E}[X_E] = P[E]$$

Theorem 5.32. The linearity of expectation says that for random variables $X_1, \ldots, X_n$ and $a_1, \ldots a_n, b \in \mathbb{R}$ we have:

$$X := \sum_{i=1}^{n} a_i X_i + b$$

$$\mathbb{E}[X] = \sum_{i=1}^{n} a_i \mathbb{E}[X_i] + b$$

Remark 5.33. We can intuitively describe this as "the expectation of a sum is the sum of the expectations".

Proof.

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) P[\omega] = \sum_{\omega \in \Omega} (a_1 X_1(\omega) + \cdots + a_n X_n(\omega)) P[\omega]$$

$$= a_1 \sum_{\omega \in \Omega} X_1(\omega) P[\omega] + \cdots + a_n \sum_{\omega \in \Omega} X_n(\omega) P[\omega] = a_1 \mathbb{E}[X_1] + \cdots + a_n \mathbb{E}[X_n]$$

□

Example 5.34. We throw a coin m times. Let $X :=$ number of partial sequences with threetimes heads. What is the expectation of X?

$$X = X_1 + X_2 + \cdots + X_{m-2}$$

where X_i is the indicator variable for the event that the i-th partial sequence is three heads.

$$\mathbb{E}[X_i] = \frac{1}{8}$$

By the linearity of the expectation we get:

$$\mathbb{E}[X] = \sum_{i=1}^{m-2} \mathbb{E}[X_i] = \frac{m-2}{8}$$

Remark 5.35. Recall Stable sets in a graph is a set of vertices that are not adjacent to each other. We will now use a randomized algorithm to determin a maximal stable set.

Algorithm 5.36

```
1      S[v] = 1/0 with probability p
2      for all u,v in E:
3      if S[u] = 1 and S[v] = 1:
4      S[v] = 0
5      return S
```

this results in a stable set but what is the expected size of S?

Theorem 5.37. The algorithm produces a stable set of expected size at least $n * P - m * P^2$

Proof. Let X be the random variable for the size of the stable set S after the first step. Further let Y be the random variable for the amount of edges we looked at in step 2. Observe

$$\mid S \mid \geq X - Y$$

By the linearity of expectation we get:

$$\mathbb{E}[\mid S \mid] \geq \mathbb{E}[X] - \mathbb{E}[Y]$$

We know that $\mathbb{E}[X] = nP$. Further we know that $\mathbb{E}[Y] = mP^2$. Thus we get:

$$\mathbb{E}[\mid S \mid] \geq nP - mP^2$$

□

5.3.2 Variance

Definition 5.38. For a random variable X with $\mathbb{E}[X] = \mu$ the variance is defined as:

$$Var[X] \coloneqq \mathbb{E}[(X - \mu)^2]$$

The standard deviation of X is defined as $\sigma \coloneqq \sqrt{Var[X]}$.

Example 5.39. Let $P[X = -10^{10}] = 1/2$ and $P[X = 10^{10}] = 1/2$. Then $\mathbb{E}[X] = 0$ and $Var[X] = 10^{20}$ and $\sigma = 10^{10}$.

Theorem 5.40. For a random variable X we have $Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

Proof.

$$Var[X] = \mathbb{E}[(X - \mu)^2] = \mathbb{E}[X^2 - 2\mu X + \mu^2] = \mathbb{E}[X^2] - 2\mu\mathbb{E}[X] + \mu^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

□

Lemma 5.41. For a random variable X and $a, b \in \mathbb{R}$ we have $Var[aX + b] = a^2 Var[X]$

Proof.

1. $Var[X + b] = \mathbb{E}[(X + b - \mathbb{E}[X + b])^2] = \mathbb{E}[(X + b - (\mathbb{E}[X] + b))^2] = \mathbb{E}[(X - \mathbb{E}[X])^2] = Var[X]$
2. $Var[aX] = \mathbb{E}[(aX - \mathbb{E}[aX])^2] = \mathbb{E}[(aX - a\mathbb{E}[X])^2] = \mathbb{E}[a^2(X - \mathbb{E}[X])^2] = a^2\mathbb{E}[(X - \mathbb{E}[X])^2] = a^2 Var[X]$

□

Definition 5.42. For a random variable X we call $\mathbb{E}[X^k]$ the k-th moment of X and $\mathbb{E}[(X - \mathbb{E}[X])^k]$ the k-th central moment of X.

Remark 5.43. Observe that the first moment is the expectation itself. Further observe that the second central moment is the variance.

5.3.3 Multiple Random Variables

Example 5.44. Throw a coin and a dice. Let X be the outcome of the coin and Y be the outcome of the dice. What is the expectation of $X + Y$?

Definition 5.45. Random variables $X_1, X_2, \ldots, X_n$ are called independent iff for all $a_1, \ldots, a_n \in W_{X_1} \times \cdots \times W_{X_n}$ we have:

$$P[X_1 = a_1, \ldots, X_n = a_n] = P[X_1 = a_1] \ldots P[X_n = a_n]$$

Lemma 5.46. Observe that events $A_1, \ldots, A_n$ are independent iff their indicator variables $X_{A_1}, \ldots, X_{A_n}$ are independent.

Proof. Note that the event "$I_{A_j} = \alpha_j$" is equivalent to the event "A_j" if $\alpha_j = 1$ and to the event $\bar{A}_j$ if $\alpha_j = 0$. Same for $A_j^{\alpha_j}$. If now $A_1, \ldots, A_n$ are independent then for all $\alpha_1, \ldots, \alpha_n \in \{0, 1\}$ we have:

$$P[I_{A_1} = \alpha_1, \ldots, I_{A_n} = \alpha_n] = P[I_{A_1} = \alpha_1] \ldots P[I_{A_n} = \alpha_n]$$

Using the above note we get:

$$P[A_1^{\alpha_1} \cap \cdots \cap A_n^{\alpha_n}] = P[A_1^{\alpha_1}] \ldots P[A_n^{\alpha_n}]$$

□

Lemma 5.47. Let $X_1, \ldots, X_n$ be independent random variables and $S_1, \ldots, S_n \subseteq \mathbb{R}$ be arbitrary sets then:

$$P[X_1 \in S_1, \ldots, X_n \in S_n] = P[X_1 \in S_1] \ldots P[X_n \in S_n]$$

(Further this implies that for $X_1, \ldots, X_n$ independent random variables and $I \subseteq \{1, \ldots, n\}$ we have $(X_i)_{i \in I}$ are independent.)

Proof.

$$P[X_1 \in S_1, \ldots, X_n \in S_n] = \sum_{\alpha_1 \in S_1} \cdots \sum_{\alpha_n \in S_n} P[X_1 = \alpha_1, \ldots, X_n = \alpha_n]$$

by indepencence of X_i we get:

$$= \sum_{\alpha_1 \in S_1} \cdots \sum_{\alpha_n \in S_n} P[X_1 = \alpha_1] \ldots P[X_n = \alpha_n]$$

$$= (\sum_{\alpha_1 \in S_1} P[X_1 = \alpha_1]) \ldots (\sum_{\alpha_n \in S_n} P[X_n = \alpha_n]) = P[X_1 \in S_1] \ldots P[X_n \in S_n]$$

□

Theorem 5.48. Let $f_1, \ldots, f_n : \mathbb{R} \to \mathbb{R}$ be functions in $\mathbb{R}$ and $X_1, \ldots, X_n$ be independent random variables. Then $f_1(X_1), \ldots, f_n(X_n)$ are independent too.

Proof. Let $S_i = \{\beta \mid f_i(\beta) = \alpha_i\}$. Then we have:

$$P[f_1(X_1) = \alpha_1, \ldots, f_n(X_n) = \alpha_n] = P[X_1 \in S_1, \ldots, X_n \in S_n]$$

by the previous lemma we get:

$$= P[X_1 \in S_1] \ldots P[X_n \in S_n] = P[f_1(X_1) = \alpha_1] \ldots P[f_n(X_n) = \alpha_n]$$

□

5.3.4 Combined Random Variables

Theorem 5.49. For two independent random variable X, Y and $Z = X + Y$ we have:

$$f_Z(\alpha) = \sum_{\beta \in W_X} f_X(\beta) f_Y(\alpha - \beta)$$

Remark 5.50. This can intuitivaly be seen by looking at every possibility for $X = \beta$: for $X + Y = \alpha$ to hold we need $Y = \alpha - \beta$. Since X and Y are independent we can multiply their probabilities and sum that over all β

Theorem 5.51. For random variables $X_1, \ldots, X_n$ we have:

$$\mathbb{E}[X_1 + \cdots + X_n] = \mathbb{E}[X_1] + \cdots + \mathbb{E}[X_n]$$

Remark 5.52. Calculation Rules for combined random variables

1. $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
2. $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ if X, Y are independent
3. $Var[X + Y] = Var[X] + Var[Y]$ if X, Y are independent

Theorem 5.53. For independent random variablex $X_1, \ldots, X_n$ we have:

$$\mathbb{E}[X_1 \ldots X_n] = \mathbb{E}[X_1] \ldots \mathbb{E}[X_n]$$

$$Var[X_1 + \cdots + X_n] = Var[X_1] + \cdots + Var[X_n]$$

Proof. We prove for $n = 2$ general case can be proved inductively.

$$\mathbb{E}[(X + Y)^2] = \mathbb{E}[X^2 + 2XY + Y^2] = \mathbb{E}[X^2] + 2\mathbb{E}[XY] + \mathbb{E}[Y^2]$$

$$(\mathbb{E}[X + Y])^2 = (\mathbb{E}[X] + \mathbb{E}[Y])^2 = \mathbb{E}[X]^2 + 2\mathbb{E}[X]\mathbb{E}[Y] + \mathbb{E}[Y]^2$$

By subtracting both equations we get:

$$\mathbb{E}[(X + Y)^2] - (\mathbb{E}[X + Y])^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2 + \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$$

Which is equivalent to $Var[X + Y] = Var[X] + Var[Y]$. □

Important 5.54. Note that $Var[X \cdot Y] = Var[X] \cdot Var[Y]$ does not hold in general.

Theorem 5.55. Let X be a random variable. For pariwise disjoint events $A_1, \ldots, A_n$ with $A_1 \cup \cdots \cup A_n = \Omega$ and $P[A_i] > 0 \forall i$ we have:

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X \mid A_i] P[A_i]$$

Proof.

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} P[X = \omega]\omega = \sum_{i=1}^{n} \omega \sum_{\omega \in A_i} P[X = \omega | A_i] P[A_i]$$

$$= \sum_{i=1}^{n} A_i \sum_{\omega \in W_X} \omega P[X = \omega | A_i] = \sum_{i=1}^{n} \mathbb{E}[X \mid A_i] P[A_i]$$

□

Theorem 5.56. The waldsche identity Let N and X be independent random variabeles further let

$$Z \coloneqq \sum_{i=1}^{N} X_i$$

where X_i are independent copies of X then:

$$\mathbb{E}[Z] = \mathbb{E}[N]\mathbb{E}[X]$$

Proof.

$$\mathbb{E}[Z] = \sum_{k=1}^{n} \mathbb{E}[Z \mid N = k] P[N = k] = \sum_{k=1}^{n} \mathbb{E}[X_1 + \cdots + X_k] P[N = k]$$

$$= \sum_{k=1}^{n} k\mathbb{E}[X] P[N = k] = \mathbb{E}[X] \sum_{k=1}^{n} k P[N = k] = \mathbb{E}[X]\mathbb{E}[N]$$

□

Example 5.57. Throw a dice where N is the face of the dice we get. Afterwards throw a coin N-times and let Z be the number of heads. What is the expectation of Z?

$$\mathbb{E}[Z] = \sum_{k=1}^{6} P[N = k]\mathbb{E}[Z \mid N = k] = \sum_{k=1}^{6} \frac{1}{6}\frac{k}{2} = \frac{1}{12}\sum_{k=1}^{6} k = \frac{1}{12}\frac{6 \cdot 7}{2} = \frac{7}{4}$$

Example 5.58. Throw a coin until we get heads. Let N be the number of throws. Then throw a coin N-times and let Z be the number of heads. What is the expectation of Z?

$$\mathbb{E}[N] = 2 (\text{geometric distribution } p = 1/2)$$

$$Z = \sum_{i=1}^{N} X_i \quad \text{where Xi are bernulli distributed with} \quad p = 1/2 \text{ hence } \mathbb{E}[X] = 1/2$$

$$\mathbb{E}[Z] = \mathbb{E}[N]\mathbb{E}[X] = 2 \cdot \frac{1}{2} = 1$$

Remark 5.59. The Waldsche identity can be used to analyse the expected runtime of randomized algorithms. Where N is the number of calls of the subroutine and X the runtime of a single call. We can caluclate the total expected runtime using Z.

5.4 Distributions

Definition 5.60. Let X be a random variable with values in $\{0, 1\}$ and density

$$P[X = 1] = p \quad \text{and} \quad P[X = 0] = 1 - p$$

Then X follows a **Bernoulli distribution** with parameter p, denoted as $X \sim \text{Bernoulli}(p)$.

Example 5.61. Throwing a coin, indicatorvariable for heads.

Remark 5.62. Observe that for a bernoulli-distribution we have:

$$\mathbb{E}[X] = p \quad \text{and} \quad Var[X] = p(1 - p)$$

Definition 5.63. Let X be a random variable with values in $\{0, 1, \ldots, n\}$ and density

$$P[X = k] = \binom{n}{k} p^k (1 - p)^{n-k}$$

Then X follows a **Binomial distribution** with parameters n and p, denoted as $X \sim \text{Bin}(n, p)$.

Example 5.64. Number of heads after throwing a coin n times.

Remark 5.65. Observe that for a binomial-distribution we have:

$$\mathbb{E}[X] = np \quad \text{and} \quad Var[X] = np(1 - p)$$

Definition 5.66. Let X be a random variable with density

$$P[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}$$

Then X follows a **Poisson distribution** with parameter λ, denoted as $X \sim \text{Po}(\lambda)$.

Remark 5.67. Observe that for a poisson-distribution we have:

$$\mathbb{E}[X] = \lambda \quad \text{and} \quad Var[X] = \lambda$$

Remark 5.68. This is often used to model events that do not happen often. For example the number of emails you get in a day. (λ would be the average num of emails you get a day).

Definition 5.69. Let X be a random variable with density

$$P[X = k] = p(1-p)^{k-1}$$

Then X follows a **Geometric distribution** with parameter p, denoted as $X \sim \text{Geo}(p)$.

Example 5.70. Assume we throw a coin with $P[\text{head}] = p$ until we get n times heads. Let X be the number of throws until we get n times heads.

$$P[X = k] = \binom{k-1}{n-1} p^n (1-p)^{k-n}$$

Remark 5.71. Observe that for a geometric-distribution we have:

$$\mathbb{E}[X] = \frac{1}{p} \quad \text{and} \quad Var[X] = \frac{1-p}{p^2}$$

Definition 5.72. For $n \geq 2$ a random variable X with density

$$P[X = k] = \binom{k-1}{n-1} p^n (1-p)^{k-n}$$

is called negative binomial distributed with order n.

Example 5.73. Coupon Collector's Problem (Sammelbilder-Problem): There are n different types of collectible cards. In every round we uniformly get one of them. Let X be the number of rounds until we have all n types of cards. What is $\mathbb{E}[X]$?

We divide the process into phases: Phase i denotes the steps from acquiring the $(i-1)$-th unique card (exclusive) to acquiring the i-th unique card (inclusive). Let X_i be the number of rounds spent in phase i. Clearly, the total number of rounds is $X = \sum_{i=1}^{n} X_i$.

In phase i, we already have $i-1$ unique cards. The probability of obtaining a new unique card in any round is:

$$p_i = \frac{n-(i-1)}{n} = \frac{n-i+1}{n}$$

Thus, X_i is geometrically distributed with parameter p_i, meaning its expectation is:

$$\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$$

By the linearity of expectation, we have:

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{j=1}^{n} \frac{1}{j} = nH_n$$

where H_n is the n-th harmonic number. Since $H_n = \ln n + \mathcal{O}(1)$, we get:

$$\mathbb{E}[X] = n \ln n + \mathcal{O}(n).$$

5.5 Bounding Probability

Theorem 5.74. Markov's Inequality: Let X be a **non-negative** random variable. For any $t \geq 0$:

$$P[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$$

or equivalently:

$$P[X \geq t\mathbb{E}[X]] \leq \frac{1}{t}$$

Proof.

$$\mathbb{E}[X] = \sum_{\alpha \in W_X} \alpha P[X = \alpha] = \sum_{\alpha \leq t} \alpha P[X = \alpha] + \sum_{\alpha > t} \alpha P[X = \alpha]$$

$$\geq \sum_{\alpha > t} \alpha P[X = \alpha] > \sum_{\alpha > t} tP[X = \alpha] = tP[X > t]$$

$$\implies P[X > t] < \frac{\mathbb{E}[X]}{t}$$

□

Theorem 5.75. Chebyshev's Inequality: Let X be a random variable. For any $t \geq 0$:

$$P[|X - \mathbb{E}[X]| \geq t] \leq \frac{Var[X]}{t^2}$$

Proof. We want to show $P[|X - \mathbb{E}[X]| \geq t] \leq \frac{Var[X]}{t^2}$. Let $Y = (X - \mathbb{E}[X])^2$. By Markov's inequality:

$$P[Y \geq t^2] \leq \frac{\mathbb{E}[Y]}{t^2} \implies P[(X - \mathbb{E}[X])^2 \geq t^2] \leq \frac{\mathbb{E}[(X - \mathbb{E}[X])^2]}{t^2}$$

$$\implies P[|X - \mathbb{E}[X]| \geq t] \leq \frac{Var[X]}{t^2}$$

□

Example 5.76. Let there be n diffrent collectable items. We buy items X times until we have a full set. We have already shown that $\mathbb{E}[X] = nH_n$. Now what is the variance of X? We know:

$$X = \sum_{i=1}^{n} X_i \text{independent, geometric distributed with } p_i = \frac{n-i+1}{n}$$

Hence

$$Var[X_i] = \frac{1-p_i}{p_i^2} \leq \frac{1}{p_i^2} = \frac{n^2}{(n-i+1)^2}$$

Further we show

$$Var[X] = \sum_{i=1}^{n} Var[X_i] \leq \sum_{i=1}^{n} \frac{n^2}{(n-i+1)^2} = n^2 \sum_{i=1}^{n} \frac{1}{i^2} \leq n^2 \frac{\pi^2}{6}$$

Using chebyshev's inequality we can bound the probability of X being far from its mean. Let $t = n\sqrt{\ln n}$:

$$P[|X - \mathbb{E}[X]| \geq n\sqrt{\ln n}] \leq \frac{Var[X]}{t^2} \leq \frac{n^2 \frac{\pi^2}{6}}{n^2 \ln n} = \frac{\pi^2}{6 \ln n}$$

Example 5.77. Special case Binomial distribution: Let $X \sim Bin(n, 1/2)$. Then $\mathbb{E}[X] = n/2$ and $Var[X] = n/4$. Using Markov's inequality:

$$P[X \geq n] \leq \frac{\mathbb{E}[X]}{n} = \frac{n/2}{n} = 1/2$$

Using Chebyshev's inequality:

$$P[X \geq n] = P[|X - \mathbb{E}[X]| \geq n/2] \leq \frac{Var[X]}{(n/2)^2} = \frac{n/4}{n^2/4} = 1/n$$

5.5.1 Chernoff Bounds

Remark 5.78. Chernoff Bounds can be used for binomial distribution $X \sim Bin(n, p)$.

Theorem 5.79. Let $X_1, \ldots, X_n$ be independent Bernoulli random variables with $P[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{n} X_i$.

$$P[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2 \mathbb{E}[X]} \quad \forall 0 \leq \delta \leq 1$$

$$P[X \leq (1-\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{2}\delta^2 \mathbb{E}[X]} \quad \forall 0 \leq \delta \leq 1$$

$$P[X \geq t] \leq 2^{-t} \quad \forall t \geq 2e\mathbb{E}[X]$$

Proof. We prove the third inequality. We use Markov's inequality on $Y = 4^X$. We have

$$X \geq t \iff 4^X \geq 4^t$$

Therfore it holds that

$$P[X \geq t] = P[4^X \geq 4^t] \leq \frac{\mathbb{E}[4^X]}{4^t}$$

Now what is $\mathbb{E}[4^X]$? Observe that as $X_1, X_2, \ldots, X_n$ are independent, $4^{X_1}, 4^{X_2}, \ldots, 4^{X_n}$ are also independent. Hence

$$\mathbb{E}[4^X] = \mathbb{E}[4^{\sum X_i}] = \mathbb{E}[\prod_{i=1}^{n} 4^{X_i}] = \prod_{i=1}^{n} \mathbb{E}[4^{X_i}]$$

Further

$$\mathbb{E}[4^{X_i}] = \sum_{x=0}^{n} 4^x P[X_i = x] = (1-p_i)4^0 + p_i 4^1 = 1 + 3p_i \leq e^{3p_i} \leq 2^t$$

$$\implies \mathbb{E}[4^X] \leq \prod_{i=1}^{n} e^{3p_i} = e^{3\sum p_i} = e^{3\mathbb{E}[X]}$$

$$\implies P[X \geq t] \leq \frac{\mathbb{E}[4^X]}{4^t} \leq \frac{2^t}{4^t} = \frac{1}{2^t}$$

□

Remark 5.80. To prove the other bounds we use the same technique on $Y = (1+\delta)^X$ and $Y = (1-\delta)^X$.

5.5.2 Target Shooting

Given finite sets S, U with $S \subseteq U$ we want to know $\frac{|S|}{|U|}$.

Example 5.81. Let U be the population of Switzerland and S the set of people that have the flu.

We can't compute this directly, so we sample k elements from U and count how many of them are in S.

Algorithm 5.82

```
1    Choose u_1 ... u_n independently and uniformly at random from U
2    Return 1/n * |{u_i in S}|
```

Let Y be the number of elements in S. We want to bound the approximation error $|Y - \frac{|S|}{|U|}|$. Observe that for every element u_i we have an indipendent bernoulli variable $Y_i \in \{0,1\}$ such that $Y = \frac{1}{n}\sum_{i=1}^{n} Y_i$ and $P[Y_i = 1] = \frac{|S|}{|U|}$.

Theorem 5.83. Let $\delta, \epsilon \geq 0$ be given. If we choose $n \geq 3\frac{|U|}{|S|}\frac{1}{\epsilon^2}\ln\frac{2}{\delta}$, then with probability at least $1-\delta$ we have that Y is in

$$[(1-\epsilon)\frac{|S|}{|U|}, (1+\epsilon)\frac{|S|}{|U|}]$$

Proof. We show that the probability of being outside that intervall is less than δ.

$$P\left[\left|Y - \frac{|S|}{|U|}\right| \geq \epsilon\frac{|S|}{|U|}\right] = P\left[|Y - \mathbb{E}[Y]| \geq \epsilon\mathbb{E}[Y]\right]$$

$$= P[|NY - \mathbb{E}[NY]| \geq n\epsilon\mathbb{E}[NY]]$$

Now observe that $NY = Y_1 + \cdots + Y_n$ where Y_i are independent bernoulli variables with $P[Y_i = 1] = \frac{|S|}{|U|}$. Hence we can use chernoff bounds.

$$P[|NY - \mathbb{E}[NY]| \geq n\epsilon\mathbb{E}[NY]] \leq 2e^{-\frac{1}{3}\epsilon^2\mathbb{E}[NY]} = 2e^{-\frac{1}{3}\epsilon^2 N\frac{|S|}{|U|}}$$

With our assumption of $N \geq 3\frac{|U|}{|S|}\frac{1}{\epsilon^2}\ln\frac{2}{\delta}$ we get exactly:

$$P\left[\left|Y - \frac{|S|}{|U|}\right| \geq \epsilon\frac{|S|}{|U|}\right] \leq \delta$$

□

6 Randomized Algorithms

Definition 6.1. A **Randomized Algorithm** is an algorithm that instead of being a function $A(I)$ with input I is a function $A(I, R)$ With a source of randomness R.

Remark 6.2. Sources of randomness R can be random bits, random integers, etc. obtained from physical phenomena or pseudo-random number generators.

Definition 6.3. A **Monte Carlo Algorithm** is a randomized algorithm that is always fast, but the result is not always correct (with a certain probability).

Definition 6.4. A **Las Vegas Algorithm** is a randomized algorithm that always returns the correct result, but the running time might be large (with a certain probability).

Remark 6.5. If we consider the las vegas algorithm that returns ??? after a certain time limit T then we have a version of Las Vegas algorithm that always terminates with a bounded runtime T and can be convertet to both standard Las Vegas or Monte Carlo algorithms.

Remark 6.6. Observe that if we run the above algorithm until we get an actual result and not ???, we have a standard Las Vegas Algorithm. Further note that if we just return a random result when we get ???, we have a Monte Carlo Algorithm.

6.1 Reduction of Error probability

Theorem 6.7. Let A be a randomized algorithm which never returns a false result but might return ??? where

$$P[A(I) \text{ is correct}] \geq \epsilon$$

Then $\forall \delta > 0$ the algorithm A_δ, which calls A until we get a correct value or until we get $N = \epsilon^{-1} \ln(\delta^{-1})$ times ???, offers

$$P[A_\delta(I) \text{ is correct}] \geq 1 - \delta$$

Remark 6.8. To reduce the probability of error to a constant factor we only need a constant amount of extra iterations.

Important 6.9. Generally it is **not possible to reduce the probability of error for monte-carlo** algorithm, except for algorithms that only have **one sided error** or an **error probability of** $< \frac{1}{2}$.

Theorem 6.10. Let A be a randomized algorithm which always returns $true$ or $false$ where

$$P[A(I)\, true] = 1, \text{if I is a true-instance}$$

$$P[A(I)\, false] \geq \epsilon, \text{if I is a false-instance}$$

Then $\forall \delta > 0$ the algorithm A_δ, which calls A until we $false$ or until we get $N = \epsilon^{-1} \ln(\delta^{-1})$ times $true$, offers

$$P[A_\delta(I) \text{ is correct}] \geq 1 - \delta$$

Theorem 6.11. Let $\epsilon > 0$ and A a randomized algorithm, which always returns $true$ or $false$ with

$$P[A(I) \text{ is correct}] \geq \frac{1}{2} + \epsilon$$

Then $\forall \delta > 0$ the algorithm A_δ, which calls A $N = 4\epsilon^{-2} \ln(\delta^{-1})$ times and returns the majority answer, offers

$$P[A_\delta(I) \text{ is correct}] \geq 1 - \delta$$

6.2 Optimization problems

Remark 6.12. We assume we have an algorithm that always returns a valid solution but not necessarily an optimal one.

Example 6.13. We want to find an algorithm that finds a solution which is at least as good as the expected value of the solution of the given algorithm.

Theorem 6.14. Let $\epsilon > 0$ and A be a randomized algorithm for an optimization problem where

$$P[A(I) \geq f(I)] \geq \epsilon$$

then $\forall \delta > 0$ the algorithm A_δ, which calls A $N = \frac{1}{\epsilon^2} \ln(\delta^{-1})$ times and returns the

best solution found, offers

$$P[A_\delta(I) \geq f(I)] \geq 1 - \delta$$

6.3 Primality Testing

Definition 6.15. Primality Testing Problem: Given $n \in \mathbb{N}$, decide if n is prime (i.e., has no divisors in $\{2, \ldots, n-1\}$). This is heavily used in cryptography (e.g. RSA) to generate large random prime numbers securely.

Remark 6.16. History of Primality Tests:

- **Sieve of Eratosthenes** (200BC): Finds all primes $\leq n$ (not polynomial time).
- **Miller-Rabin** (1976): Monte-Carlo algorithm with one-sided error (always correct for primes).
- **Solovay-Strassen** (1977): Alternative Monte-Carlo algorithm.
- **Adleman-Huang** (1987): Monte-Carlo algorithm, always correct for non-primes.
- **AKS** (Agrawal-Kayal-Saxena, 2002): First deterministic algorithm in polynomial time.

Algorithm 6.17

```
1    //Euclid Primality Test
2    Choose a [1...n-1] uniformly at random.
3    If ggT(a, n) = 1:
4    return ‘‘Prime’’
5    else
6    return ‘‘Not Prime’’
```

Remark 6.18. This is an extremely naive algorithm:

- If n is prime: Output is always correct.
- If n is not prime: False output "Prime" with probability $\frac{|Z_n^*|}{n-1} \approx 1 - \frac{1}{\sqrt{n}}$, which is unfortunately very high.

Proof. For a composite number n, the worst-case scenario (which maximizes the number of coprimes) occurs when $n = p^2$ for some prime p.

In this case, the number of coprimes is:

$$\varphi(n) = p(p-1) = n - \sqrt{n}$$

The probability of falsely picking an a that is coprime to n is therefore:

$$P(\text{False "Prime"}) = \frac{\varphi(n)}{n-1} \approx \frac{n-\sqrt{n}}{n} = 1 - \frac{1}{\sqrt{n}}$$

□

Definition 6.19. Euler's Totient Function $\varphi(n) = |Z_n^*|$: Number of integers in $\{1, \ldots, n-1\}$ that are coprime to n. If n is prime, $\varphi(n) = n - 1$.

Theorem 6.20. Theorem of Lagrange: If H is a subgroup of G, then $|H|$ divides $|G|$.
Fermat's Little Theorem: If n is prime, then for all $a \in \{1, \ldots, n-1\}$, $a^{n-1} \equiv 1 \pmod{n}$.

Algorithm 6.21

```
1     \textbf{Fermat Test}: Choose $a \in \{1, \dots, n-1\}$
         uniformly at random. If $a^{n-1} \bmod n = 1$, return ``
         Prime'', else ``Not Prime''.
```

Remark 6.22.
- If n is prime: Output is always correct.
- If n is not prime: False output "Prime" with probability $\leq \frac{1}{2}$, *unless* n is a **Carmichael Number**.

Definition 6.23. A **Carmichael Number** is a composite number n where $a^{n-1} \equiv 1 \pmod{n}$ holds for all a coprime to n, causing the Fermat test to almost always improperly return "Prime" (e.g. $561 = 3 \times 11 \times 17$).

Proof. Let PB_n be the set of "false witnesses" (numbers that incorrectly pass the test):

$$PB_n = \{a \in Z_n^* \mid a^{n-1} \equiv 1 \pmod{n}\}$$

This set forms a subgroup of Z_n^* because it is closed under multiplication:

$$\text{If } a, b \in PB_n, \text{ then } (ab)^{n-1} = a^{n-1}b^{n-1} \equiv 1 \pmod{n}$$

By Lagrange's theorem, if n is not a Carmichael number (meaning $PB_n \neq Z_n^*$), then PB_n is a strict subgroup. The theorem states that its size $|PB_n|$ must cleanly divide the size of the parent group $|Z_n^*|$.

Because PB_n is a strict subgroup, the maximum possible size of this subgroup is exactly half the group:

$$|PB_n| \leq \frac{|Z_n^*|}{2} \leq \frac{n-1}{2}$$

Meaning at most half of the elements in the test space will incorrectly evaluate to true. □

Definition 6.24. Miller-Rabin Certificate: Let $n > 2$ be odd. Write $n - 1 = 2^k d$ with d odd. If n is prime and $a \in \{1, \ldots, n-1\}$, then either:

$$a^d \equiv 1 \pmod{n} \quad \text{or} \quad \exists i \in \{0, \ldots, k-1\} : a^{2^i d} \equiv n - 1 \pmod{n}$$

If this condition **fails**, a is a *Miller-Rabin Certificate* proving n is strictly not prime.

Algorithm 6.25

```
1    //Miller-Rabin Test
2    Choose a at random.
3    Check if the certificate condition holds.
```

Remark 6.26.

- If n is prime: Output is always correct.
- If n is not prime: False output "Prime" with probability $\leq \frac{1}{4}$ (even for Carmichael numbers). This means repeating the test k times drops error probability to $\leq 4^{-k}$.

Proof. If n is prime, $\mathbb{Z}_n$ forms a mathematical field. In a field, the equation $x^2 = 1$ has exactly two roots (1 and -1):

$$x = 1 \quad \text{and} \quad x = -1 \equiv (n-1) \pmod{n}$$

By Fermat's little theorem $a^{n-1} \equiv 1 \pmod{n}$. We can rewrite this exponent using $n-1 = 2^k d$ as:

$$a^{2^k d} \equiv 1 \pmod{n}$$

If we repeatedly take square roots of this value, the sequence must hit -1 before it hits 1, unless it was 1 from the very beginning ($a^d \equiv 1$). If the sequence does not follow this rule, the field properties are violated, and thus n cannot be prime.

While it rigorously holds that the false output probability is tightly bounded at $\leq \frac{1}{4}$, the full mathematical proof leverages the Chinese Remainder Theorem to show the non-witnesses form a strict subgroup bounded to one-quarter the size. □

6.4 Sorting and Selecting

Theorem 6.27. The expected runtime of randomized quicksort is $O(n \log n)$ or more specifically

$$\mathbb{E}[T_{1,n}] \leq 2(n+1)\ln(n) + O(n)$$

Proof. Let $T_{\ell,r}$ denote the (random) number of comparisons performed during a call to $QuickSort(A, \ell, r)$. We want to show that

$$E[T_{1,n}] \leq 2(n+1)\ln n + O(n)$$

For $\ell \geq r$, $T_{\ell,r} = 0$. For $\ell < r$, using Theorem 2.32 and the linearity of expectation, we have

$$E(T_{\ell,r}) = \sum_{i=\ell}^{r} \Pr[t = i] \cdot (r - \ell + E[T_{\ell,i-1}] + E[T_{i+1,r}])$$

$$= \frac{1}{r-\ell+1} \cdot \sum_{i=0}^{r-\ell} (r - \ell + E[T_{\ell,\ell+i-1}] + E[T_{\ell+i+1,r}])$$

Here, we used the fact that t is uniformly distributed over $\{\ell, \ldots, r\}$—this follows directly from the assumption that the elements of A are distinct. Looking more closely at the recursion shown above, we see that the expected value $E[T_{\ell,r}]$ does not depend on r and ℓ individually, but only on the number of elements to be sorted, $n = r - \ell + 1$. This observation motivates the following recursive definition of the sequence t_n:

$$t_n = \begin{cases} 0, & \text{if } n \leq 1 \\ \frac{1}{n} \sum_{i=0}^{n-1} (n - 1 + t_i + t_{n-i-1}), & \text{if } n \geq 2 \end{cases}$$

By induction on $r - \ell$, it is easy to see that for all ℓ, r, the equation $E[T_{\ell,r}] = t_{r-\ell+1}$ holds. In the following, we derive an upper bound for t_n. For all $n \geq 3$, by definition:

$$n \cdot t_n = \sum_{i=0}^{n-1} (n - 1 + t_i + t_{n-i-1})$$

and similarly:

$$(n-1) \cdot t_{n-1} = \sum_{i=0}^{n-2} (n - 2 + t_i + t_{n-i-2})$$

Subtracting the second equation from the first, we obtain:

$$n \cdot t_n - (n-1) \cdot t_{n-1} = 2(n-1) + 2t_{n-1}$$

and therefore:

$$t_n = \frac{n+1}{n} \cdot t_{n-1} + \frac{2(n-1)}{n} \leq \frac{n+1}{n} \cdot t_{n-1} + 2$$

We now use induction again (this time on n) to show that for all $n \geq 2$:

$$t_n \leq 2 \sum_{i=3}^{n+1} \frac{n+1}{i}$$

For $n = 2$, it follows from the definition of t_n that $t_2 = 1 \leq 2 = \sum_{i=3}^{3} 2$. For $n \geq 3$, we can use the inequality $t_n \leq \frac{n+1}{n} \cdot t_{n-1} + 2$ derived above along with the induction hypothesis to verify the validity of the inequality for all $n \geq 3$. Since $\sum_{i=1}^{n} \frac{1}{i} = H_n = \ln n + O(1)$, it follows in particular that:

$$E[T_{1,n}] = t_n \leq 2(n+1) \ln n + O(n)$$

□

Theorem 6.28. The expected runtime of randomized quickselect is $O(n)$.

Proof. When we execute $QuickSelect(A, 1, n, k)$, it results in a random number N of calls of the form $Partition(A, \ell_i, r_i, p)$ for a (likewise random) sequence

$$(\ell_0, r_0, k_0), (\ell_1, r_1, k_1), \ldots, (\ell_N, r_N, k_N)$$

with $(\ell_0, r_0, k_0) = (1, n, k)$. In this process, either $(\ell_{i+1}, r_{i+1}) = (\ell_i, t-1)$ or $(\ell_{i+1}, r_{i+1}) = (t+1, r_i)$, where t is a random variable uniformly distributed on $\{\ell_i, \ldots, r_i\}$ (because the elements of A are pairwise distinct). The runtime of a call to $QuickSelect(A, 1, n, k)$, measured by the number of comparisons of elements in A, is:

$$T = \sum_{i=0}^{N} (r_i - \ell_i)$$

because each call $Partition(A, \ell_i, r_i, p)$ requires exactly $r_i - \ell_i$ comparisons. We now define N_j as the number of calls to $QuickSelect$ for which $(3/4)^j n < r_i - \ell_i + 1 \leq (3/4)^{j-1} n$ holds. Then it certainly follows that:

$$T \leq \sum_{j=1}^{\infty} N_j \cdot (3/4)^{j-1} n$$

due to the linearity of expectation, we have:

$$E[T] \leq n \cdot \sum_{j=1}^{\infty} E[N_j] \cdot (3/4)^{j-1}$$

What can we say about $E[N_j]$? To determine this, we consider that the number of elements to be processed is reduced from $r_i - \ell_i + 1$ to at most $\frac{3}{4}(r_i - \ell_i + 1)$ whenever we choose one of the middle $\frac{1}{2}(r_i - \ell_i + 1)$ elements as the pivot. In other words: with every choice of a pivot element, there is a probability of at least $1/2$ that the number of elements is reduced by at least a factor of $3/4$. Therefore, $E[N_j] \leq 2$ for all $j \geq 1$, and thus:

$$E[T] \leq 2n \sum_{j=1}^{\infty} (3/4)^{j-1} = 2n \cdot \frac{1}{1 - 3/4} = 8n$$

The expected runtime of $QuickSelect(A, 1, n, k)$ is indeed linear, as claimed. □

6.5 Duplicate Detection

Definition 6.29. We are given a sequence of data-elements $A_i \in \Sigma^*$ where Σ is a set of size n. Determine if there are any duplicates in A.

Remark 6.30. We could sort the sequence in $O(nlogn)$ and check for duplicates in $O(n)$. But what if the data elements are too large to efficiently sort them?

Algorithm 6.31

```
S = [A1, A2 ... An];
X = [(hash(A1), 1), (hash(A2), 2), ... (hash(An), n)];
Duplicates = [];
X.sortByHashValue();

for every block of elements with the same hash value do:
if block.size > 1:
For every pair (Ai, Aj) in block do:
if Ai.data == Aj.data:
Duplicates.add((Ai.data, Aj.data));
return Duplicates;
```

Remark 6.32. Note that we have to check each pair in their block as our hash function guarantees that same datavalues recieve same hash values but not necessarily uniquely.

Theorem 6.33. The expected number of actual data comparisons is $O(\#duplicates)+O(1)$.

Proof. □

6.6 Floyd-Cycle Detection

Definition 6.34. Given an array $A[1,\ldots,n]$ where $A[i] \in \{1,\ldots,n\}$ find two indices $i \neq j$ such that $A[i] = A[j]$.

Remark 6.35. We want to solve this problem wiht $O(n)$ expected runtime and $O(1)$ extra space.

Remark 6.36. Idea: We can model this as a directed graph with nodes $1 \ldots n$ and edges $i \to A[i]$.

Lemma 6.37. There exists a $t \in \mathbb{N}$ such that $X_t = X_{2t}$ if there are duplicates in A.

Proof. Assume there are duplicates in A hence there is a cycle in the graph. Let $r = \lfloor \frac{k}{l} \rfloor \cdot l - k \geq 0$ with l the length of a cycle and k the length of the path before the cycle starts (tail). Observe that $X_{k+r} = X_{2(k+r)}$ furhter

$$k + r = \lfloor \frac{k}{l} \rfloor l < (\frac{k}{l} + 1)l = k + l \leq n$$

Thus $t = k + r$ is an itteration index where the sequence repeats itself. Therfore if we find such t then $X_t = X_{2t}$ and the two corresponding array entries must be equal. □

Algorithm 6.38

```
    tortoise = A[n]
    hare = A[A[n]]
    t = 1

    while tortoise != hare:
    tortoise = A[tortoise]
    hare = A[A[hare]]
    t += 1

    hare = n
    while hare != tortoise:
    i = hare;
    j = tortoise;
    hare = A[hare]
    tortoise = A[tortoise]

    return i, j
```

Remark 6.39. Observe that this algorithm only uses $O(1)$ extra space.

Lemma 6.40. Let $t \leq n$ with $X_t = X_{2t}$ then $X_{t+k} = X_k$ holds for all $k \geq 0$.

Proof. We know $X_t = X_{2t}$ and thus $2t = t + p * l$ for some $p \in \mathbb{N}$ hence $t = p * l$ holds and finally $X_{t+k} = X_{k+l*p} = X_k$ holds by induction. □

6.7 Long-Path Problem

Definition 6.41. Given a graph $G = (V, E)$ and some $B \in \mathbb{N}$. Decide whether there exists a path of length B in G.

Theorem 6.42. If there exists an algorithm for the long-path problem with $t(n)$ time, then we can decide in $t(2n-2) + O(n^2)$ time if a given graph is Hamiltonian.

Proof. We construct a new graph G' with $n' \leq 2n - 2$ such that

$$G \text{ hamiltonian} \iff G' \text{ has a path of length } n$$

Let $v \in V$ be an arbitrary vertex of G. We construct G' by replacing edges $e = (v, w) \in E$ with edges (w, w') where w' is a copy of w and removing v. Observe that if G has a Hamiltonian cycle, this cycle includes v hence in G' this corresponds to a path of length

n namely $w_i', w_i, \dots w_j, w_j'$ for some w_i, w_j adjacent to v. Further observe that if G' has a path of length n then all vertices in that path are distinct and have degree ≥ 2 hence they must be the n vertices in G (as the added vertices have degree 1). Hence the end and start vertices of that path correspond to two distinct vertices adjacent to v in G and removing these two vertices and the edges from that path yields a Hamiltonian cycle in G. Thus we have shown

$$G \text{ hamiltonian} \iff G' \text{ has a path of length } n$$

□

Definition 6.43. A **colorful path** in a k-edge-colored graph G is a path where every edge has a different color.

Remark 6.44. Instead of finding a path of length k we look for a colorful path with at least k colors. For this we define

$P_i(v) \coloneqq \{S \subset [k] || S| = i + 1 \text{ and } \exists \text{ path } v_0 = v, e_1, \dots e_{i+1}, v_{i+1} \text{ with edge colors } S\}$

if for a $v \in V$ we have $P_{k-1}(v) \neq \emptyset$ we found a colorful path with k colors. We compute for all $v \in V$ the set $P_i(v)$ for $i = 0 \dots k - 1$.

Lemma 6.45. We can compute P_i dynamically using base case

$$P_0(v) = \{\{c(v)\}\}$$

and reccurence

$$P_i(v) = \bigcup_{u \in N(v)} \{S \cup \{c(v)\} | S \in P_{i-1}(u) \wedge c(v) \notin S\}$$

Algorithm 6.46

```
1       colorful(G, i):
2
3       for all v in V do:
4       P[i][v] = {}
5       for all u in N(v) do:
6       for all S in P[i-1][u] do:
7       if c(v,u) not in S then
8       P[i][v].add(S union {c(v,u)})
```

Theorem 6.47. The runtime of the above algorithm is bounded by $O(2^k \cdot k \cdot m)$.

Proof.

$$\sum_{i=1}^{k-1} O(\text{colorful(G, i)})$$

$$= \sum_{i=1}^{k-1} O(\sum_{v \in V} deg(v) \cdot \max_{u \in N(v)} |P_{i-1}(v)|)$$

$$= \sum_{i=1}^{k-1} O(m \cdot \binom{k}{i} \cdot i)$$

$$= O(2^k \cdot k \cdot m)$$

□

Remark 6.48. Assume G contains a $k-1$ long path P

$$P[\exists \text{coloful path of length of length k-1 }]$$

$$\geq P[\text{P is colorful}]$$

$$\geq \frac{k!}{k^k} \geq e^{-k}$$

Theorem 6.49. The monte carlo algorithm (Repeated λe^k times)has a runtime of $O(\lambda \cdot (2e)^k \cdot k \cdot m)$ and a probability of error $e^{-\lambda}$

6.8 Minimum cut in Multigraphs

Definition 6.50. Minimum Edge Cut Problem: Given a multigraph $G = (V, E)$ determine the cardinality $\mu(G)$ of a minimum edge cut in G.

Definition 6.51. A **multigraph** G is an undirected, unweighted graph whith possibly multiple edges between two vertices.

Definition 6.52. An **Edge Cut** is a set of edges $C \subseteq E$ such that $G \setminus C$ is disconnected.

Remark 6.53. This problem is equivalent to finding a Partition $S \cup T = V$ such that the number of edges between S and T is minimized.

Lemma 6.54. Observe that the following holds:

$$2|E| = \sum_{v \in V} deg(v) \quad \text{and} \quad \mu(G) \leq mindeg(v)$$

Theorem 6.55. Using maxflow we can dtermine the global min cut in $O(n^4 \log n)$ time.

Proof. We can find the global min cut by selecting an arbitrary vertes s which must be on one side of the min cut. Then we iterate over all $t \in V \setminus \{s\}$ and determine the min cut between s and t in time $O(nm \log n)$. This will lead us to an alogrithm of $O(n^2 m \log n) \leq O(n^4 \log n)$ □

Definition 6.56. Edge Contraction: Let $G = (V, E)$ be a graph. Edge contraction operates on an edge $(u, v) \in E$ and merges the two vertices u and v into a single vertex w. All edges incident to u or v are now incident to w.

Theorem 6.57. Using edge contractions we can find a global min cut in $O(n^4 \log n)$ with a success probability of $\frac{1}{n}$.

Proof. Observe that contracting an edge $(u, v) \in E$ only increses the min cut if the edge is part of the min cut, else the min cut remains the same. Hence we can contract $n-2$ times and get a graph with 2 vertices where the min cut is exactly the degree of the vertices. The probability of choosing an edge which is part of the min cut is at most

$$P[e \in C_{min}] = \frac{\mu(G)}{|E|} \leq \frac{2|E|}{n|E|} = \frac{2}{n}$$

Now observe that for $n \geq 3$ we have

$$P[\mu(G_n) = \mu(G_{n-1})] \geq (1 - \frac{2}{n}) P[\mu(G_{n-1}) = \mu(G_{n-2})]$$

Hence

$$P[\mu(G) = \mu(G_2)] \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \dots \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} \cdot 1 = \frac{2}{n(n-1)} = \binom{n}{2}^{-1}$$

□

Theorem 6.58. By repeating the above algorithm $\lambda \binom{n}{2}$ times and returning the smalles value we get a success probability of $1 - e^{-\lambda}$. Let $\lambda = \ln n$ we get a failure prob of $\frac{1}{n}$.

6.8.1 Bootstrapping

Remark 6.59. Observe that the probability of picking the wrong edge grows with each step hence the last few steps are critical and the probability of failure is dominated by the last few steps.

Theorem 6.60. We can stop the contraction process early at $|G| = t$ in $O(n(n-t))$ and compute the rest using $\lambda = 1$ in $O(t^4)$ to reduce error in the last few steps. By repeting this $\lambda \frac{n(n-1)}{t(t-1)} \frac{e}{1-e}$ and setting $t = \sqrt{n}$ we get a total runtime of $O(\lambda n^3)$ with an error probability of $e^{-\lambda}$.

6.9 Small Enclosing Disk Problem

Definition 6.61. Given a finite set $P \subset \mathbb{R}^2$ find the smallest disk/circle C_P such that $P \subset C_P$

Lemma 6.62. For every set P there exists a certificate $C(P)$ of three points such that the disk defined by the points in $C(P)$ is C_P.

Remark 6.63. The brutforce approach tries every combination Q of three points in $O(n^3)$ time and checks in $O(n)$ if $P \subseteq Q(P)$.

Remark 6.64. Another idea would be to choose Q randomly and check if $P \subseteq Q(P)$. This still leads to a $O(n^4)$ randomized algorithm.

Theorem 6.65. By doubling the number of copies of points $p_i \notin Q$ every iteration we get a randomized algorithm which finds the smallest enclosing disk in $O(n \log n)$ expected time.

Proof. The Sampling Lemma states that for a multiset P' of size N and a random subset R of size r, the expected number of points outside the disk $C(R)$ is bounded by:

$$\mathbb{E}[|P' \setminus C^{\bullet}(R)|] \leq 3\frac{N-r}{r+1} \leq 3\frac{N}{r+1}$$

Since we use $r = 11$ in every round, the total number of points X_k in the multiset after k rounds grows only moderately in expectation:

$$\mathbb{E}[X_k] \leq (1 + \frac{3}{11+1})^k \cdot n = (\frac{5}{4})^k \cdot n$$

At the same time, there exists a small basis $B \subseteq P$ with $|B| \leq 3$ that defines the optimal disk $C(P)$. As long as the algorithm has not terminated ($T \geq k$), at least one point from B must be outside $C(Q)$ and thus be doubled in each round. By the pigeonhole principle, after k rounds, at least one point in B must have been doubled at least $k/3$ times:

$$X_k \geq 2^{k/3}$$

By combining these two bounds, we can bound the probability that the algorithm is still running after k rounds:

$$2^{k/3} \cdot Pr[T \geq k] \leq \mathbb{E}[X_k] \leq (\frac{5}{4})^k \cdot n$$

$$Pr[T \geq k] \leq \left(\frac{5}{4 \cdot 2^{1/3}}\right)^k \cdot n \approx 0.993^k \cdot n$$

Since this probability decreases exponentially as k increases, the expected number of rounds is $E[T] = O(\log n)$. Given that each round takes $O(n)$ time to check all points, the total expected runtime is:

$$\mathbb{E}[\text{runtime}] = O(n \log n)$$

□

6.10 Convex Hull

Definition 6.66. The **Convex Hull** $\mathcal{CH}(P)$ of a finite set of points $P \subset \mathbb{R}^2$ is the intersection of all convex sets containing P. Equivalently, it is the set of all convex combinations of points in P:

$$\mathcal{CH}(P) = \left\{ \sum_{i=1}^{|P|} \lambda_i p_i \;\middle|\; p_i \in P, \lambda_i \geq 0, \sum_{i=1}^{|P|} \lambda_i = 1 \right\}$$

Definition 6.67. A directed segment $\vec{pq}$ for $p, q \in P$ is a **border edge** of $\mathcal{CH}(P)$ if and only if all other points in $P \setminus \{p, q\}$ lie on the same side (e.g., strictly to the left) of the directed line passing through p and q.

Definition 6.68. For a directed line passing through points $p = (x_p, y_p)$ and $q = (x_q, y_q)$, a point $r = (x_r, y_r)$ is located to the **left** or **right** of $\vec{pq}$ according to the sign of the 2D cross product:

$$\text{CCW}(p, q, r) = (x_q - x_p)(y_r - y_p) - (y_q - y_p)(x_r - x_p)$$

If $\text{CCW}(p, q, r) > 0$, r lies strictly to the left of $\vec{pq}$ (counter-clockwise orientation). If $\text{CCW}(p, q, r) < 0$, r lies strictly to the right (clockwise orientation). If $\text{CCW}(p, q, r) =$

0, the points are collinear.

Theorem 6.69. Given a vertex $p \in P$ that is guaranteed to be on the boundary of $\mathcal{CH}(P)$, the next vertex $q \in P$ on the convex hull can be found in $\mathcal{O}(n)$ time. This is achieved by finding a point q such that for all $r \in P \setminus \{p, q\}$, we have $\text{CCW}(p, q, r) > 0$ (all points lie to the left).

Remark 6.70. Initialization: The point $p_0 \in P$ with the minimum y-coordinate (and minimum x-coordinate in case of a tie) is an extremal point of P and is therefore guaranteed to be a vertex of $\mathcal{CH}(P)$.

Theorem 6.71. The **Gift Wrapping Algorithm (Jarvis March)** iteratively finds the next border edge, computing the convex hull of P in $\mathcal{O}(nh)$ time, where h is the number of vertices on the convex hull.

Proof. The algorithm starts with an extremal point p_0. In each step, it finds the point q that forms the smallest angle with the current edge (i.e., the point "most to the right"). By the geometric definition of a convex hull, the segment $\vec{pq}$ must be a border edge because all other points lie to the left of it. The algorithm terminates when it wraps back around to p_0, successfully tracing the entire boundary.

We must perform h iterations to find all vertices of the convex hull. In each iteration, we scan through all n points in P to find the one that forms the smallest angle with the current edge. The angle comparison (via cross product) takes $\mathcal{O}(1)$ time. Thus, the total time complexity is $\mathcal{O}(h \cdot n)$. □

Code Example 6.72

```
    p = min_x(min_y(P))
    hull = empty_list()

    do {
            hull.add(p)
            q = any_point_in(P) != p
            for r in P {
                    if r on the right side of p-q:
                    q = r
                }
            p = q
        } while p != hull[0]
```

6.11 Reachable Vertices

Definition 6.73. Given a directed graph $G = (V, E)$, for each vertex $v \in V$, calculate the number of reachable vertices $n_v = |R(v)|$, where $R(v) = \{u \in V \mid u \text{ is reachable from } v\}$.

Remark 6.74. For an undirected graph, this is simply the size of the connected component. It can be solved in $O(n + m)$ by first identifying components via DFS and then counting their sizes.

Remark 6.75. The problem is not efficiently solvable for directed graphs in the worst case, as an exact solution takes $O(n(n+m))$. Improving this to sub-quadratic time likely contradicts the Strong Exponential Time Hypothesis. Thus, we seek an approximate solution in near-linear time.

Algorithm 6.76

```
\\Subroutine: Minimum Reachable Value
\\To enable approximation, we use a subroutine that finds
\\the smallest value r_u among all reachable vertices u for
   every v

V = V.sort(ascending, r_v)
min[v] = \infty
for v_i in V:
if min[v_i] == \infty:
DFSREV(v_i)
min[u] = r_{v_i} \\for all u in R(v_i)
```

6.11.1 Approximation Algorithm

Remark 6.77. The core idea is to assign each vertex v a random variable $r_v \sim \text{Uniform}([0, 1])$. The minimum value $x_v = \min_{u \in R(v)} r_u$ among n_v such variables is roughly $1/n_v$ in expectation.

Theorem 6.78. Algorithm: Repeat the process $l = \lceil 2 \log_2(2n/\delta) \rceil$ times. In each iteration, compute $x_{i,v}$ using the subroutine. The estimate $\tilde{n}_v$ is the reciprocal of the median of these $x_{i,v}$ values, which achieves an approximation in $O((n \log n + m) \log(n/\delta))$ time.

Theorem 6.79. With probability $\geq 1 - \delta$, the estimate satisfies $n_v/20 \leq \tilde{n}_v \leq 20n_v$ for all v.

Theorem 6.80. By increasing the iterations l to $O(\frac{1}{\epsilon^2}\log(\frac{n}{\delta}))$, a $(1\pm\epsilon)$-approximation can be achieved.

Remark 6.81. This algorithmic approach (using minima of random variables) is also used in the *count-distinct* problem (e.g., HyperLogLog) to estimate the number of unique elements in data streams.

7 Flow Networks

Definition 7.1. A **Flow Network** is a directed graph $G = (V, E)$ with a **source** $s \in V$, a **sink** $t \in V$, and a **capacity function** $c : E \to \mathbb{R}_{\geq 0}$ assigning a non-negative capacity to each edge.

Definition 7.2. A **flow** in a network $G = (V, E, c, s, t)$ is a function $f : E \to \mathbb{R}_{\geq 0}$ satisfying:

- **Capacity constraint**: $\forall e \in E : 0 \leq f(e) \leq c(e)$
- **Flow conservation**: $\forall v \in V \setminus \{s, t\} : \sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$

Definition 7.3. The **value** of a flow f is defined as the total flow leaving the source:

$$|f| := \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ into } s} f(e)$$

Remark 7.4. By flow conservation, the net flow out of s equals the net flow into t:

$$|f| = \sum_{e \text{ into } t} f(e) - \sum_{e \text{ out of } t} f(e)$$

Definition 7.5. For a flow f and a vertex v, the **net outflow** of v is:

$$\text{netout}(v) := \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e)$$

Flow conservation requires $\text{netout}(v) = 0$ for all $v \in V \setminus \{s, t\}$.

7.1 Cuts

Definition 7.6. An s-t **cut** is a partition (S, T) of V with $s \in S$ and $t \in T$. The **capacity** of the cut is:

$$c(S, T) := \sum_{\substack{e=(u,v) \\ u \in S,\, v \in T}} c(e)$$

Lemma 7.7. For any flow f and any s-t cut (S, T):

$$|f| = \sum_{\substack{e=(u,v) \\ u \in S,\, v \in T}} f(e) \; - \sum_{\substack{e=(u,v) \\ u \in T,\, v \in S}} f(e)$$

Proof. Sum the net outflow equation over all $v \in S$. By flow conservation, interior vertices contribute 0, and only the source s has net outflow $|f|$. Collecting edge terms gives exactly the net flow across the cut. □

Theorem 7.8. For any flow f and any s-t cut (S, T):

$$|f| \leq c(S, T)$$

Proof. By the lemma above and the capacity constraint $f(e) \leq c(e)$:

$$|f| = \sum_{e \in (S,T)} f(e) - \sum_{e \in (T,S)} f(e) \leq \sum_{e \in (S,T)} c(e) = c(S, T)$$

□

7.2 Max-Flow Min-Cut

Theorem 7.9. Max-Flow Min-Cut Theorem: The maximum value of an s-t flow equals the minimum capacity of an s-t cut:

$$\max_f |f| = \min_{(S,T)} c(S, T)$$

Remark 7.10. The theorem has three equivalent characterizations. The following are equivalent:

- f is a maximum flow.
- There is no augmenting path from s to t in the residual network G_f.
- There exists an s-t cut (S, T) with $|f| = c(S, T)$.

7.3 Residual Networks and Augmenting Paths

Definition 7.11. Given a flow f in network G, the **residual network** $G_f = (V, E_f)$ has residual capacity:

$$c_f(u, v) := \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E \\ f(v, u) & \text{if } (v, u) \in E \\ 0 & \text{otherwise} \end{cases}$$

The edge set E_f contains all pairs (u, v) with $c_f(u, v) > 0$.

Remark 7.12. Each original edge $(u, v) \in E$ with $f(u, v) > 0$ introduces a **back edge** (v, u) in G_f with residual capacity $f(u, v)$. This allows flow to be "cancelled" or rerouted.

Definition 7.13. An **augmenting path** is a directed path from s to t in the residual network G_f. The **bottleneck capacity** of such a path P is:

$$\Delta(f, P) := \min_{e \in P} c_f(e)$$

Lemma 7.14. If P is an augmenting path with bottleneck $\Delta = \Delta(f, P)$, then augmenting f along P (adding Δ on forward edges, subtracting Δ on backward edges) yields a valid flow f' with $|f'| = |f| + \Delta$.

7.4 Ford-Fulkerson Algorithm

Algorithm 7.15

```
1     //Ford-Fulkerson Algorithm
2     Initialize f(e) = 0 for all e in E.
3
4     while there exists an augmenting path P from s to t in G_f:
5     delta = bottleneck capacity of P
6     for each edge (u, v) in P:
7     if (u, v) in E:
8     f(u, v) += delta
9     else:    // back edge (u, v), i.e. (v, u) in E
10    f(v, u) -= delta
11
12    return f
```

Theorem 7.16. If all capacities are integers, Ford-Fulkerson terminates in $O(|f^*| \cdot |E|)$ time, where $|f^*|$ is the value of the maximum flow.

Proof. Each augmentation increases $|f|$ by at least 1 (since capacities are integers, $\Delta \geq 1$). The flow value is bounded by $|f^*|$, so there are at most $|f^*|$ augmentations. Each augmenting path search via DFS/BFS takes $O(|E|)$. □

Important 7.17. Ford-Fulkerson is **not guaranteed to terminate** for irrational capacities. For rational capacities it always terminates, but can be exponentially slow with bad path choices.

7.5 Maximum Matchings using MaxFlow

Definition 7.18. Maxium Bipartite Matching: Given a bipartite graph $G = (S \cup T, E)$ find a maximum matching in G.

Remark 7.19. We can define a flow network $G' = (V', E', c')$ with

$$V' = S \quad \cup \quad T \quad \cup \quad \{s, t\}$$

$$E' = s \times S \quad \cup \quad E \quad \cup \quad T \times t$$

$$c' = 1$$

Then the maximum integer flow f in G' equals to $val(f) = \frac{|M|}{3}$, where M is the maximum matching in G.

Remark 7.20. This can be solved in $O(mn)$ using Ford-Fulkerson.

7.6 Matchings in Bipartite Graphs

Definition 7.21. Maximum Bipartite Matching: Given a bipartite graph $G = (A \cup B, E)$ find a maximum matching in G.

Remark 7.22. We can construct a flow network $G' = (V', E', c')$ with

$$V' = S \quad \cup \quad T \quad \cup \quad \{s, t\}$$

$$E' = s \times S \quad \cup \quad E \quad \cup \quad T \times t$$

$$c' = 1$$

Lemma 7.23. The maximum flow in G' equals to $\frac{|M|}{2}$, where M is the maximum matching in G.

7.7 Edge-disjoint Paths

Definition 7.24. Edge-disjoint Paths: Given a graph and vertices s, t find a maximum set of edge-disjoint paths from s to t.

Remark 7.25. We can construct a flow network $N_G = (V, E', c')$ with

$$E' = \{(u, v), (v, u) | \forall \{u, v\} \in E\}$$

$$c'(e) = 1 \quad \forall e \in E'$$

We can find k edge disjoint paths in G and by calculating the max flow of N_G and greedily selecting consecutive edges until we reach the sink. Observe that this is possible because every vertex has the same amount of incomming as outgoing edges.

Lemma 7.26. As we can find k edge-disjoint paths in G we have $k \leq maxFlow(N_G)$. Further if we had k edge disjoint paths we would be able to construct a flow of value k in N_G by sending 1 unit of flow along each path. Thus $k \leq maxFlow(N_G)$. By this we geometrically

$$k = maxFlow(N_G)$$

Lemma 7.27. Observe that the number of edges k' that separate s from t in G form a minimal $s - t$ cut in N_G and thus

$$k' = minCut(S, T)$$

Theorem 7.28. By the above lemmas and the max-flow min-cut theorem we have (Menger's Theorem)

$$k = maxFlow(N_G) = minCut(S, T) = k'$$

Which means that the maximum number of edge-disjoint paths equals to the minimum number of edges that separate s from t in G.

7.8 Image Segmentation using MaxFlow

Definition 7.29. Image Segmentation: Given a n-dimensional image with pixels P and edges between neighboring pixels E. We are given weights

$$\alpha_p, \beta_p \geq 0 \quad \forall p \in P$$

expressing the preference of pixel p to be in the foreground A and background B

respectively and confidence values

$$\gamma_e \geq 0 \quad \forall e \in E$$

describing the confidence that neighboring pixels are in the same partition. Find a partition (A, B) of P with maximum quality:

$$q(A,B) = \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e=(u,v) \in E'} \gamma_e$$

where E' are edges between pixels in P that are in different partitions.

Remark 7.30. The intuition beind the quality function is that we want to maximize the sum of the confidences in the partitions while penaltizing neighboring pixels that are in different partitions based on the confidence that they are in the same partition.

Lemma 7.31. Let $S = \sum_{p \in P}(\alpha_p + \beta_p)$. We have

$$q(A,B) = S - \sum_{p \in A} \beta_p - \sum_{p \in B} \alpha_p - \sum_{e=(u,v) \in E'} \gamma_e$$

obsere that instead of maximizing q we can try to minimize

$$q'(A,B) = \sum_{p \in A} \beta_p - \sum_{p \in B} \alpha_p + 1 \sum_{e=(u,v) \in E'} \gamma_e$$

Theorem 7.32. To solve the problem we can create a flow network where there are

- edges from s to pixels with capacity α_p and from pixels to t with capacity β_p
- edges with capacity γ_e in both directions between neighboring pixels

The min-cut in this network then corresponds to the maximum quality.

Proof. Obsere that in any S, T cut the capacity of the cut is made of β_p for $p \in A$ and α_p for $p \in B$ and γ_e for $e \in E'$. Hence y minimizing the cut S, T we minimize q' therfore maximize our partition quality q. □