

Algorithms and Probability Summary

April 30, 2020

Chapter 1

Graphentheorie

1.1 Basics and Definitions

Graph: A graph is a tuple (V, E) , where V is a finite non empty set of vertices and E is a set of vertex pairs indicating the edges $V \subseteq E$
 $(E \subseteq \binom{V}{2} := \{(x, y) | x, y \in V, x \neq y\})$

Complete (Vollständig): There is an edge between each pair of vertices (den. K_n)

Walk (Weg): A sequence of vertices $\langle v_1, v_2, \dots, v_n \rangle$ if $\forall i$ there exists an edge from v_i to v_{i+1} . The length of the walk is given by the number of steps, i.e. $n-1$.

Path (Pfad): A walk which doesn't contain any vertex more than once (den. P_n)

Closed Walk (Zyklus): A walk in which $v_1 = v_n$ (den. C_n)

Cycle (Kreis): A closed walk with length of at least three and the vertices v_1, \dots, v_{k-1} are pairwise distinct (in a directed graph it must have length of at least two)

Loops (Schlingen): An edge from a vertex to itself

Multiple edges (Mehrfachkanten): When vertex pairs are connected by multiple edges

Multigraph (Multigraph): A Graph which contains loops and multiple edges (In this lecture we assume that a graph is not a multigraph unless stated otherwise)

Neighbourhood (Nachbarschaft): All outgoing and incoming edges to/from a vertex v denoted $N_G(v) := \{u \in V | \{v, u\} \in E\}$

Degree (Grad): Indicates the size of the neighbourhood $deg_G(v) := |N_G(v)|$

k-regular (k-regulär): If every vertex $v \in V$ has degree $deg(v) = k$
◦ A complete graph K_n is $n-1$ -regular

Adjacent (Adjazent): Two vertices u and v if there is an edge u, v

Satz 1.2 For any Graph $G=(V, E)$ we have $\sum_{v \in V} deg(v) = 2|E|$

Korollar 1.3 For any Graph $G = (V, E)$ the number of vertices with uneven degree is even. (Direct Proof by splitting V into even and odd degree sets then use S1.2)

Subgraph (Teilgraph): A Graph $H = (V_H, E_H)$ is a subgraph of a graph $G = (V_G, E_G)$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G$ denoted $H \subseteq G$

Induced Subgraph (Induzierte Teilgraph): If $E_H = E_G \cap \binom{V_H}{2}$ denoted $H = G[V_H]$. If there is an edge (u, v) in G and u, v are also vertices in H then there must be an edge (u, v) in H as well

1.1.1 Connectivity and Trees

Connected (Zusammenhängend): if for any Vertices $s, t \in V$ there is a s - t path. A subgraph $C \subseteq G$ for which this trait is maximal is called a connected component (hence for all subgraphs $H \neq C$ with $C \subseteq H \subseteq G$ is not connected)

Cycle Free (Kreisfrei): A Graph which doesn't contain a cycle

Tree (Baum): A Graph which is Cycle free and connected

Leaf (Blatt): $T=(V,E)$ a tree and $v \in V$ a vertex with $\deg(v) = 1$

Lemma 1.5: $T = (V,E)$ a tree with $|V| \geq 2$, it follows:

a): T contains atleast 2 leafs. (Proof: If there was only one leaf $2|E| = \sum_{v \in V} \deg(v) \geq 1 + 2(|V| - 1)$ which is a contradiction to S1.6)

b): if $v \in V$ is a leaf, the graph $T-v$ is also a tree.

Satz 1.6 $G=(V,E)$ a Graph with $|V| \geq 1$ vertices, the following is equivalent:

- G is a tree
- G is connected and cycle free
- G is connected and $|E| = |V| - 1$
- G is cycle free and $|E| = |V| - 1$
- for any $x,y \in V$, G contains exactly one x - y path.

Forrest (Wald): $W = (V,E)$ a graph which is cycle free, every component of a forrest is a tree

Lemma 1.7 A forrest $G = (V,E)$ contains $|V| - |E|$ connected components (Proof by induction)

Directed Graph (Gerichteter Graph): A graph where the edges are represented by ordered pairs, i.e The directed graph D is given by the tuple (V,A) where V is the set of vertices and $A \subseteq V \times V$ a set of directed edges. Compared to an undirected graph, between two vertices there can be two edges (x,y) and (y,x)

Out-Degree(Aus-Grad): $\deg^+(v) := |\{(x,y) \in A | x = v\}|$

In-Degree(In-Grad): $\deg^-(v) := |\{(x,y) \in A | y = v\}|$

Satz 1.8 For any directed graph $D = (V,A)$ the following is true.
 $\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |A|$.

Acyclic (Azyklisch): A directed graph which doesn't contain a cycle (DAG). DAG's have a topological ordering.

Satz 1.9 For any DAG $D=(V,A)$ we can find a topological ordering in $\mathcal{O}(|V| + |A|)$

Strongly Connected (start zusammenhängend): For a DAG $D=(V,A)$, if for every pair of vertices $u,v \in V$ a directed u - v -Path exists

Weakly Connected (schwach zusammenhängend): When the underlying graph(i.e ignoring the direction of the edges) is connected

1.1.2 Datastructures

The two main ways of storing graphs, is with Adjacency matrices and Adjacency lists.

1.2 Trees

1.3 Paths

Shortest Paths Given: A connected Graph $G = (V,E)$, two vertices $s,t \in V$ and a costfunction $c : E \rightarrow \mathbb{R}$

Goal: Find an s - t -path P in G with $\sum_{e \in P} c(e) = \min$

Algorithms which can solve shortest path problems:

1. Dijkstras Algorithm
2. Floyd-Warshall
3. Bellman-Ford
4. Johnson's Algorithm

1.4 Connection

Definition 1.23 A Graph $G = (V,E)$ is **k-connected (k-zusammenhängend)** if $|V| \geq k+1$ and for all subsets $X \subseteq V$ with $|X| < k$ the following is true: The Graph $G[V \setminus X]$ is connected. (Hence you would need to remove atleast k vertices to destroy the connectivity of the graph, the only exception is a complete graph which is by definition $k-1$ connected)

Definition 1.24 A Graph $G = (V, E)$ is **k-edge-connected (k-kanten-zusammenhängend)**, if for all subsets $X \subseteq E$ with $|X| < k$ the following is true: $(V, E \setminus X)$ is connected. (Hence at least k edges must be removed to destroy the connectivity of the Graph)

Satz 1.25 Menger $G = (V, E)$ the following applies:

1. G is k -connected iff for all pairs of vertices $u, v \in V$, $u \neq v$, at least k internal-vertex disjoint u - v paths exist
2. G is k -edge-connected iff for all pairs of vertices $u, v \in V$, $u \neq v$, at least k edge-disjoint u - v -paths exist

1.4.1 Articulation vertex (Artikulationsknoten)

Articulation vertex (Artikulationsknoten): If a graph is connected but not 2-connected, then there exists a vertex v with the attribute that $G[V \setminus \{v\}]$ is not connected. Articulation vertices can be detected using a modified DFS.

Forward Edge (Vorwärtskante): An edge starting from a vertex with a lower dfs number than the destination vertex

Backwards Edge (Rückwärtskante): An edge starting from a vertex with a higher dfs number than the destination vertex

we assign $\in V$ a number $\text{low}[v] :=$ the smallest dfs-Number, that can be reached from the vertex v using any number of forward edges and at most one backward edge. It follows for all $v \in V$: $\text{low}[v] \leq \text{dfs}[v]$

v is an Articulation vertex $\Leftrightarrow v = s$ and s has degree of at least 2 or $v \neq s$ and there exists a $w \in V$ with $\{v, w\} \in E(T)$ and $\text{low}[w] \geq \text{dfs}[v]$

TODO: Implement DFS-Visit which finds the articulation vertices for a given graph

Satz 1.27 For a connected graph $G = (V, E)$, implemented with an adjacency list Articulation vertices can be found in $\mathcal{O}(|E|)$

1.4.2 Bridges (Brücken)

Bridge (Brücke): An edge $e \in E$ such that $(V, E \setminus \{e\})$ is not connected

From the definition of the bridge it follows that a spanning tree must contain all bridges of a graph and that the vertices at the end of the bridge are either Articulation vertices or vertices with degree 1.

An edge (v, w) of the depth-first-search tree is a bridge iff $\text{low}[w] \geq \text{dfs}[v]$

Satz 1.28 For a connected graph $G = (V, E)$ implemented with an adjacency list, articulation vertices and bridges can be found in $\mathcal{O}(|E|)$

1.5 Cycles

1.5.1 Eulerwalk (Eulertour)

Definition 1.29 A Eulerwalk in a graph $G = (V, E)$ is a cycle which contains each edge exactly once. If G contains a Eulerwalk then $\deg(v)$ of all $v \in V$ is even. (Proof by contradiction assume G has vertices of even degrees pick a starting node v and an arbitrary node u and show that the path cannot end in u arguing with the parity of the degree)

In a connected graph eulerian graph a Eulerwalk can be found in $\mathcal{O}(|E|)$

TODO: Implement an Algorithm which can find a Eulerwalk in $\mathcal{O}(|E|)$

1.5.2 Hamilton Cycles (Hamiltonkreise)

Definition 1.31 A Hamilton Cycle in a graph $G = (V, E)$ is a cycle in which all vertices of V are visited exactly once. If a graph contains a Hamilton Cycle it is called hamiltonian (hamiltonisch). Whether or not a graph contains a hamilton cycle is NP-complete.

Satz 1.33 The Algorithm HAMILTONKREIS to find a Hamilton cycle of a given graph G needs $\mathcal{O}(n * 2^n)$ memory and has a runtime of $\mathcal{O}(n^2 * 2^n)$, where $n = |V|$

TODO: Implement the Algorithm HAMILTONKREIS

1.5.3 Special Cases

Lattice (Gittergraph) An $m \times n$ lattice is hamiltonian if m or n is even (Proof using parity argument)

Lemma 1.35 $G = (A \uplus B, E)$ a bipartite Graph with $|A| \neq |B|$, then G cannot contain a Hamilton cycle

Hypercube (Hyperwürfel): The set of edges of a Hypercube H_d is $\{0, 1\}^d$ hence the set of all 0-1 sequences of length d . Two vertices are connected if their sequences differ at exactly one spot. A d -dimensional Hypercube contains a Hamilton cycle for all $d \geq 2$ (Proof by induction)

Satz 1.37(Dirac) If $G = (V, E)$ is a graph with $|V| \geq 3$ vertices and each vertex has at least $|V|/2$ neighbours, then G is hamiltonian.

1.5.4 The Travelling Salesman Problem

Given: A complete Graph K_n and a function $l : \binom{[n]}{2} \rightarrow \mathbb{N}$ which gives each edge a length

Goal: Find a Hamilton cycle C in K_n with $\sum_{e \in C} l(e) = \min\{\sum_{e \in C'} l(e) \mid C' \text{ is a Hamilton cycle in } K_n\}$

For a graph $G = (V, E)$ with $V = [n]$ vertices we define a weight function l as $l(\{u, v\}) = \begin{cases} 0, & \text{if } \{u, v\} \in E \\ 1, & \text{else} \end{cases}$

hence the length of a minimal Hamilton cycle in K_n when using l is 0 iff G contains a Hamilton cycle, hence this allows us evaluate the efficiency of a solution. We define the optimum solution as $opt(K_n, l) := \min\{\sum_{e \in C'} l(e) \mid C' \text{ is a Hamilton cycle in } K_n\}$. An algorithm which always finds a Hamilton cycle with $\sum_{e \in C} l(e) \leq \alpha * opt(K_n, l)$ is known as an α -Approximation algorithm.

Satz 1.39 If there exists an α -Approximation algorithm for an $\alpha > 1$ for the Traveling Salesman Problem with a runtime of $\mathcal{O}(f(n))$, then an algorithm exists for all graphs with n vertices which decides whether it is hamiltonian or not in the same time complexity

Metric Travelling Salesman Problem (Metrisches TSP)

Given: A complete Graph K_n and a function $l : \binom{[n]}{2} \rightarrow \mathbb{N}$ with the condition $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$ for all $x, y, z \in [n]$

Goal: Find a Hamilton cycle C in K_n with $\sum_{e \in C} l(e) = \min\{\sum_{e \in C'} l(e) \mid C' \text{ is a Hamilton cycle in } K_n\}$ The condition is called the triangle inequality (Dreiecksungleichung) and states that the direct connection between two vertices x and z cannot be longer than the detour over the vertex y .

Satz 1.40 The Metric Travelling Salesman Problem has a 2-Approximation algorithm with a runtime of $\mathcal{O}(n^2)$

1.6 Matchings

Matching A set of edges $M \subseteq E$ is called a Matching of a graph $G = (V, E)$, if no vertex of the graph is incident to more than one edge from M i.e. $e \cap f = \emptyset$ for all $e, f \in M$ with $e \neq f$. A vertex v is "covered" (überdeckt) if there is an edge $e \in M$ which contains v .

Perfect Matching If each vertex is covered by exactly one edge of the Matching M i.e. $|M| = |V|/2$. Not all graphs contain a perfect matching for example a star graph.

Maximal matching (Inklusionsmaximal): $G = (V, E)$, for a Matching M if $M \cup \{e\}$ is not a Matching for all edges $e \in E \setminus M$.

Maximum matching (Kardinalitätsmaximal): $G = (V, E)$ for a Matching M if $|M| \geq |M'|$ for all Matchings M' in G

Example: A path consisting of three edges. Creating a Matching using the middle edge would create a Maximal matching but not a Maximum matching. A Maximum and a Maximal Matching can be created by taking the two outer edges

1.6.1 Algorithms

GREEDY MATCHING This algorithm picks random edges from E and adds it to the matching at the same time it deletes all incident edges from E . The algorithm stops when $E = \emptyset$. This algorithm can find a Maximal matching in time $\mathcal{O}(|E|)$ for which the following applies: $|M_{Greedy}| \geq \frac{1}{2} |M_{max}|$ where M_{max} is a Maximum matching

Union of two Matchings: Let M_1 and M_2 be arbitrary Matchings. $G_M = (V; M_1 \cup M_2)$. Every vertex in G_M has degree at most 2, hence all components of the graph are paths and/or cycles (cycles having even length). If we assume $|M_1| < |M_2|$, then every cycle/path of even length will have the same amount of edges from M_1 as M_2 . From our assumption we know that there must be a path P which contains more edges from M_2 than M_1 the two outer edges belonging to M_2 . Hence we can create a new Matching M'_1 using P which will contain one more edge. We achieve this by switching the edges in P i.e. $M'_1 := (M_1 \cup (P \cap M_2)) \setminus (P \cap M_1)$. We say P is a M_1 -augmented path.

M-augmented Path (augmentierender Pfad): Let M be an arbitrary Matching, an M augmented path is a path where the last two edges of P are not covered by M and P consists of edges alternating between edges belonging to M and not belonging to M

AUGMENTED MATCHING This algorithm finds a Maximum matching. We start by finding a Matching which consists of only one arbitrary edge. As long as the matching is not a maximum matching we repeat the following: We take an augmented path and we increase the size of the Matching. We know that after $|V|/2 - 1$ times the matching will be maximum because a matching can't have more than $|V|/2$ edges. We can easily find augmented paths in a bipartite Graph using a modified BFS. The total runtime of our algorithm is $\mathcal{O}(|V| \cdot |E|)$

Satz 1.45 If n is even and $l : \binom{[n]}{2} \rightarrow \mathbb{N}$ a weight function of the complete graph K_n then we can find a minimal perfect Matching (i.e a matching where the sum of edge weights are minimal) in $\mathcal{O}(n^3)$

Satz 1.46 From S1.45 it follows that for the Metric Travelling Salesman Problem there is a 3/2-Approximation algorithm with a runtime of $\mathcal{O}(n^3)$

1.6.2 Der Satz von Hall

Bipartite A graph $G=(V,E)$ is bipartite if we can partition the set of vertices V into two sets A and B such that all edges in E contain a vertex from A and a vertex from B (denoted: $G = (A \uplus B, E)$)

Satz von Hall/Heiratssatz: For a bipartite graph $G = (A \uplus B, E)$ there is a Matching M of cardinality $|M| = |A|$ iff $|N(X)| \geq |X|$ for all $X \subseteq A$. From the Satz von Hall it follows that a k -regular graph always has a perfect matching.

Satz 1.48 Let $G = (A \uplus B, E)$ be a k -regular bipartite graph. There exists an M_1, \dots, M_k such that $E = M_1 \uplus \dots \uplus M_k$ and all $M_i, 1 \leq i \leq k$ are perfect matchings in G . The perfect matching can be found in $\mathcal{O}(|E|)$.

Satz 1.49 Let $G=(V,E)$ a 2^k -regular bipartite graph. We can find a perfect matching in $\mathcal{O}(|E|)$

1.7 Colouring (Färbungen)

Vertex colouring (Knoten Färbung): The vertex colouring of a graph $G = (V,E)$ with k colours is a mapping $c : V \rightarrow [k]$ such that the following holds: $c(u) \neq c(v)$ for all edges $u,v \in E$

Chromatic number (chromatische Zahl): denoted $\chi(G)$ is the minimal number of colours needed to color the vertices of G . A complete graph has the chromatic number n . Cycles of even length have chromatic number 2, uneven length have a chromatic number 3. Trees with atleast two vertices have a chromatic number 2. Graphs with chromatic number k are also called k -partite. To decide whether or not a graph G is bipartite can be done in $\mathcal{O}(|E|)$ with a DFS or BFS.

Satz 1.53 A graph $G=(V,E)$ is bipartite iff it does not contain a cycle of odd length as a subgraph

Satz 1.54 (Vierfarbensatz): Any map can be coloured using 4 colours.

GREEDY FARBUNG This algorithm calculates the colouring of a graph by picking vertices at random and giving it the lowest colour not used by its neighbours. There exists a order of vertices for which the GREEDY algorithm needs $\chi(G)$ colors.

Satz 1.55 Let G be a connected graph. For the number $C(G)$ of colours needed by GREEDY FARBUNG to color the graph G the following applies: $\chi(G) \leq C(G) \leq \Delta(G) + 1$ ($\Delta(G) := \max_{v \in V} \deg(v)$, hence the max degree of a vertex in G). If the graph is saved in an adjacency list then we can find the colouring in $\mathcal{O}(|E|)$

Satz 1.59 (Satz von Brooks) Let $G=(V,E)$ be a connected graph which is not complete nor a cycle with odd degree i.e $G \neq K_n$ and $G \neq C_{2n+1}$ then the following holds $\chi(G) \leq \Delta(G)$ and there exists an Algorithm which can colour the graph in $\mathcal{O}(|E|)$ with $\Delta(G)$ colours.

Satz 1.60 Let $G=(V,E)$ be a graph and $k \in \mathbb{N}$ a natural number such that every induced subgraph of G has a vertex with degree at most k . It follows that $\chi(G) \leq k + 1$ and we can find a $(k+1)$ -colouring in $\mathcal{O}(|E|)$

Satz 1.61 (Mycielski-Konstruktion): For all $k \geq 2$ there is a triangle free graph G_k with $\chi(G_k) \geq k$ (Proof by induction)

Satz 1.62 Every 3-colourable graph $G=(V,E)$ can be coloured in $\mathcal{O}(|E|)$ with $\mathcal{O}(\sqrt{|V|})$ colours. Given a graph $G=(V,E)$, is $\chi(G) \leq 3$ is NP-Complete.

Chapter 2

Probability Theory and Randomised Algorithms

2.1 Definitions and Notations

Definition 2.1 A discrete Probabilitiespace(diskreter Wahrscheinlichkeitsraum) is defined by a Set of **outcomes** (Ergebnismenge) denoted $\Omega = \{\omega_1, \omega_2, \dots\}$ of **elementary outcomes**(Elementarereignissen). Each elementary outcome ω_i is assigned a **(elementary)probability**(Elementar – Wahrscheinlichkeit) denoted $Pr[\omega_i]$ where $0 \leq Pr[\omega_i] \leq 1$ and $\sum_{\omega \in \Omega} Pr[\omega] = 1$. A set $E \subseteq \Omega$ is called an **outcome**(Ereignis) The probability of $Pr[E]$ of an outcome is defined by: $Pr[E] := \sum_{\omega \in E} Pr[\omega]$

If E is an outcome, we define $\bar{E} := \Omega \setminus E$ the **compliment outcome**(Komplementarereignis)

Finite probabilitiespace (endlicher Wahrscheinlichkeitsraum): A Probability space $\Omega = \{\omega_1, \dots, \omega_n\}$ (Assumption for infinite probability spaces $\Omega = \mathbb{N}_0$)

Lemma 2.2 For outcomes A,B the following applies:

1. $Pr[\emptyset] = 0, Pr[\Omega] = 1$
2. $0 \leq Pr[A] \leq 1$
3. $Pr[\bar{A}] = 1 - Pr[A]$
4. if $A \subseteq B, Pr[A] \leq Pr[B]$
5. (Additionssatz) Wenn die Ereignisse A_1, \dots, A_n paarweise disjunkt sind (also wenn für alle Paare $i \neq j$ gilt, dass $A_i \cap A_j = \emptyset$) so folgt:

$$Pr\left[\bigcup_{i=1}^n A_i\right] = \sum_{i=1}^n Pr[A_i]$$

Union Bound (Boolesche Ungleichung): Für beliebige Ereignisse A_1, \dots, A_n gilt:

$$Pr\left[\bigcup_{i=1}^n A_i\right] \leq \sum_{i=1}^n Pr[A_i]$$

Siebformel, Prinzip Inklusion/Exklusion: Für Ereignisse $A_1, \dots, A_n (n \geq 2)$ gilt:

$$\begin{aligned} Pr\left[\bigcup_{i=1}^n A_i\right] &= \sum_{i=1}^n Pr[A_i] - \sum_{1 \leq i_1 < i_2 \leq n} Pr[A_{i_1} \cap A_{i_2}] + \dots \\ &\quad + (-1)^{l+1} \sum_{1 \leq i_1 < \dots < i_l \leq n} Pr[A_{i_1} \cap \dots \cap A_{i_l}] + \dots \\ &\quad + (-1)^{n+1} \cdot Pr[A_1 \cap \dots \cap A_n] \end{aligned}$$

Laplace Raum: endlicher Wahrscheinlichkeitsraum, in dem alle Elementarereignisse gleich wahrscheinlich sind. In einem Laplace-Raum gilt für jedes Ereignis E:

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

Möglichkeiten k Elemente aus einer n-elementigen Menge zu ziehen : $(n^k := n(n-1)(n-2) \dots (n-k-1) = \frac{n!}{(n-k)!})$

Bedingte Wahrscheinlichkeit: A und B seien Ereignisse mit $Pr[B] > 0$. Die bedingte Wahrscheinlichkeit $Pr[A|B]$ (Die W'keit, dass Ereignis A eintritt, wenn wir schon wissen dass Ereignis B eingetreten ist) von A gegeben B ist t definiert durch:

$$Pr[A|B] := \frac{Pr[A \cap B]}{Pr[B]}$$

	geordnet	ungeordnet
mit Zurücklegen	n^k	$\binom{n+k-1}{k}$
ohne Zurücklegen	$n^{\underline{k}}$	$\binom{n}{k}$

(a)

Beispiel: $k=2$ Elemente aus $S=\{1,2,3\}$ ziehen ($n=3$)

	geordnet	ungeordnet
mit Zurücklegen	$(1,1), (1,2), (1,3)$ $(2,1), (2,2), (2,3)$ $(3,1), (3,2), (3,3)$	$\{1,1\}, \{1,2\}, \{1,3\}$ $\{2,2\}, \{2,3\}, \{3,3\}$
ohne Zurücklegen	$(1,2), (1,3), (2,1)$ $(2,3), (3,1), (3,2)$	$\{1,2\}, \{1,3\}, \{2,3\}$

(b)

Satz 2.12 Satz der totalen W'keit: Die Ereignisse A_1, \dots, A_n seien paarweise disjunkt und es gelte $B \subseteq A_1 \cup \dots \cup A_n$ dann folgt:

$$Pr[B] = \sum_{i=1}^n Pr[A_i \cap B] = \sum_{i=1}^n Pr[B|A_i] \cdot Pr[A_i]$$

Satz 2.10 Multiplikationsatz: Seien die Ereignisse A_1, \dots, A_n gegeben. Falls $Pr[A_1 \cap \dots \cap A_n] > 0$ ist, gilt:

$$Pr[A_1 \cap \dots \cap A_n] = Pr[A_1] \cdot Pr[A_2|A_1] \cdot Pr[A_3|A_1 \cap A_2] \dots Pr[A_n|A_1 \cap \dots \cap A_{n-1}]$$

Satz von Bayes: Die Ereigniss A_1, \dots, A_n seien paarweise disjunkt. Ferner $B \subseteq A_1 \cup \dots \cup A_n$ ein Ereignis mit $Pr[B] > 0$ Dann gilt für ein beliebiges $i = 1, \dots, n$:

$$Pr[A_i|B] = \frac{Pr[A_i \cap B]}{Pr[B]} = \frac{Pr[B|A_i] \cdot Pr[A_i]}{\sum_{j=1}^n Pr[B|A_j] \cdot Pr[A_j]}$$

Unabhängigkeit Zwei Ereignisse: Die Ereignisse A und B heissen unabhängig, wenn gilt:

$$Pr[A \cap B] = Pr[A] \cdot Pr[B]$$

Unabhängigkeit: Die Ereignisse A_1, \dots, A_n heissen unabhängig, wenn für alle Teilmengen $I \subseteq \{1, \dots, n\}$ mit $I = \{i_1, \dots, i_k\}$ gilt, dass

$$Pr[A_{i_1} \cap \dots \cap A_{i_k}] = Pr[A_{i_1}] \dots Pr[A_{i_k}] \quad (2.2)$$

Eine unendliche Familie von Ereignissen A_i mit $i \in \mathbb{N}$ heisst unabhängig, wenn (2.2) für jede endliche Teilmenge $I \subseteq \mathbb{N}$ erfüllt ist. Dies ist offensichtlich erfüllt, wenn die Ereignisse physikalisch unabhängig sind (e.g wenn jedes A_i einem unabhängigen Münzwurf entspricht) aber ist nicht unbedingt erforderlich.

Die Ereignisse A_1, \dots, A_n sind genau dann unabhängig wenn für alle $(s_1, \dots, s_n) \in \{0, 1\}^n$ gilt dass

$$Pr[A_1^{s_1} \cap \dots \cap A_n^{s_n}] = Pr[A_1^{s_1}] \dots Pr[A_n^{s_n}]$$

wobei $A_i^0 = \overline{A_i}$ und $A_i^1 = A_i$

Seien A, B und C unabhängige Ereignisee. Dann sind auch $A \cap B$ und C bzw. $A \cup B$ und C unabhängig.

2.2 Zufallsvariablen

Eine funktion welche jede element unser Wahrscheinlichkeitsraum ein Reellezahl zuordnet.

$$X : \Omega \rightarrow \mathbb{R}$$

Beispiel:

" $X \leq 5$ steht für das Ereignis, dass die Zufallsvariable einen Wert kleiner gleich 5 annimmt also:

$$\Rightarrow X \leq 5 \triangleq \{\omega \in \Omega : X(\omega) \leq 5\}$$

Dichtefunktion:

$$f_X : \mathbb{R} \rightarrow [0, 1], \quad x \mapsto Pr[X = x]$$

Verteilungsfunktion:

$$F_X : \mathbb{R} \rightarrow [0, 1], \quad x \mapsto Pr[X \leq x] = \sum_{x' \in W_X : x' \leq x} Pr[X = x']$$

Beispiele:

Wir werfen eine Münze drei Mal:

$X := \text{Anzahl „Kopf“}$

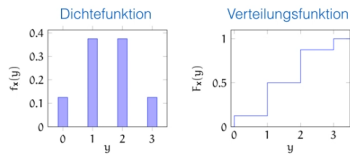
Wahrscheinlichkeitsraum: $\Omega = \{KKK, KKZ, \dots, ZZZ\}$

$$\text{Prob}[X=0] = \text{Pr}[\{ZZZ\}] = 1/8$$

$$\text{Prob}[X=1] = \text{Pr}[\{ZZK, ZKZ, KZZ\}] = 3/8$$

$$\text{Prob}[X=2] = \text{Pr}[\{ZKK, KKZ, KZK\}] = 3/8$$

$$\text{Prob}[X=3] = \text{Pr}[\{KKK\}] = 1/8$$



(a)

Wir werfen einen Würfel drei Mal:

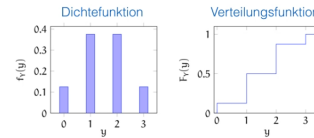
$Y := \text{Anzahl ungerader Augenzahlen}$

Wahrscheinlichkeitsraum: $\Omega = \{1,2,3,4,5,6\}^3$

$$\text{Prob}[Y=0] = \text{Pr}[\text{erster Wurf gerade}] \cdot \text{Pr}[\text{zweiter Wurf gerade}] \cdot \text{Pr}[\text{dritter Wurf gerade}] = 1/8$$

$$\begin{aligned} \text{Prob}[Y=1] &= \text{Pr}[\text{erster Wurf gerade}] \cdot \text{Pr}[\text{zweiter Wurf gerade}] \cdot \text{Pr}[\text{dritter Wurf ungerade}] + \\ &\quad \text{Pr}[\text{erster Wurf gerade}] \cdot \text{Pr}[\text{zweiter Wurf ungerade}] \cdot \text{Pr}[\text{dritter Wurf gerade}] + \\ &\quad \text{Pr}[\text{erster Wurf ungerade}] \cdot \text{Pr}[\text{zweiter Wurf gerade}] \cdot \text{Pr}[\text{dritter Wurf gerade}] = 3/8 \end{aligned}$$

$$\text{Prob}[Y=2] = 3/8, \quad \text{Prob}[Y=3] = 1/8$$



(b)

Bernoulli-Verteilung:

$$X \sim \text{Bernoulli}(p)$$

$$f_X(x) = \begin{cases} p & \text{für } x = 1, \\ 1 - p & \text{für } x = 0 \\ 0 & \text{sonst} \end{cases}$$

$$\mathbb{E}[X] = p$$

Binomial Verteilung:

$$X \sim \text{Bin}(n, p)$$

$$f_X(x) = \begin{cases} \binom{n}{x} p^x (1-p)^{n-x}, & x \in \{0, 1, \dots, n\} \\ 0, & \text{sonst.} \end{cases}$$

$$\mathbb{E}[X] = np \quad \text{Var}[X] = np(1-p) \quad (\text{gleichung für Varianz gilt nur wenn die } X_i\text{'s unabhängig sind})$$

$\text{Bin}(n, \frac{\lambda}{n})$ konvergiert für $n \rightarrow \infty$ gegen $\text{Po}(\lambda)$ Beispiel: Werfen einer Münze n mal, $X = \text{Anzahl Kopf}$

Negative Binomialverteilung:

$$X \sim \text{NegativeBinomial}(n)$$

$$f_X(k) = \begin{cases} \binom{k-1}{n-1} (1-p)^{k-n} p^n, & \text{für } k = 1, 2, \dots \\ 0, & \text{sonst} \end{cases}$$

$$\mathbb{E}[X] = \frac{n}{p}$$

Beispiel: Warten auf den n-ten Erfolg

Geometrische Verteilung:

$$X \sim \text{Geo}(p)$$

$$f_X(i) = \begin{cases} p(1-p)^{i-1} & \text{für } i \in \mathbb{N} \\ 0 & \text{sonst.} \end{cases}$$

$$F_X(n) = 1 - (1-p)^n \quad \text{für alle } n=1,2,\dots$$

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

Beispiel: Wiederholtes Werfen einer Münze, $X = \# \text{ Würfe bis zum ersten Mal Kopf}$

Gedächtnislosigkeit: Ist $X \sim \text{Geo}(p)$, so gilt für alle $s, t \in \mathbb{N}$:

$$\text{Pr}[X \geq s+t | X > s] = \text{Pr}[X \geq t]$$

Beispiel: Wahrscheinlichkeit im ersten Wurf Kopf zu bekommen ist identisch zur Wahrscheinlichkeit nach 1000 Fehlversuchen im 1001ten Wurf Kopf zu bekommen.

Poisson-Verteilung:

$$X \sim \text{Po}(\lambda)$$

$$f_X(i) = \begin{cases} \frac{e^{-\lambda} \lambda^i}{i!} & \text{für } i \in \mathbb{N}_0 \\ 0 & \text{sonst} \end{cases}$$

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

Beispiel: Modellierung seltener Ereignisse e.g. $X := \# \text{ Herzinfarkte in der Schweiz in der nächsten Stunde}$

Erwartungswert: Den Zufallsvariablen X definieren wir den Erwartungswert $\mathbb{E}[X]$ durch:

$$\mathbb{E}[X] := \sum_{x \in W_X} x \cdot Pr[X = x]$$

sofern die Summe konvergiert. Ansonsten sagen wir, dass der Erwartungswert undefiniert ist. Ist X eine Zufallsvariable, so gilt:

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) \cdot Pr[\omega]$$

Sei X eine Zufallsvariable mit $W_X \subseteq \mathbb{N}_0$. Dann gilt

$$\mathbb{E}[X] = \sum_{i=1}^{\infty} Pr[X \geq i]$$

Beobachtung: Für ein Ereignis $A \subseteq \Omega$ ist die zugehörige Indikatorvariable X_A definiert durch:

$$X_A(\omega) = \begin{cases} 1 & \text{falls } \omega \in A \\ 0 & \text{sonst.} \end{cases}$$

Für den Erwartungswert von X_A gilt: $\mathbb{E}[X_A] = Pr[A]$

- Schnitt: $A \cap B \quad X_{A \cap B} = X_A \cdot X_B$
- Komplement: $\bar{A} := \Omega \setminus A \quad X_{\bar{A}} = 1 - X_A$
- Vereinigung: $A \cup B \quad X_{A \cup B}$

Beispiel:

Beispiel: Wir werfen eine Münze 100 Mal $X := \text{Anzahl Kopf}$

Setze für alle $i = 1, \dots, 100$:

$X_i := \text{Indikatorvariable für „Kopf“ im } i\text{ten Wurf}$

Dann $X = X_1 + \dots + X_{100}$

und $\mathbb{E}[X_i] = 1/2$ und wegen der Linearität des Erwartungswertes

daher $\mathbb{E}[X] = \mathbb{E}[X_1] + \dots + \mathbb{E}[X_{100}] = 50$

Figure 2.3

Linearität des Erwartungswerts: Für Zufallsvariablen X_1, \dots, X_n und $X = a_1 X_1 + \dots + a_n X_n + b$ mit $a_1, \dots, a_n, b \in \mathbb{R}$ gilt:

$$\mathbb{E}[X] = a_1 \mathbb{E}[X_1] + \dots + a_n \mathbb{E}[X_n] + b$$

Stabiler Menge: Knoten, die nicht durch Kanten verbunden sind.

Satz Für jeden Graphen $G=(V,E)$ mit $|V| = n$ und $|E| = m$ bestimmt der Algorithmus (gehe durch die Knotenmenge und entferne den Knoten und inzidente Kanten mit wahrscheinlichkeit $1-p$, bei den übrig gebliebene kanten lösche ein Knoten) eine stabile Menge S mit

$$\mathbb{E}[S] \geq np - mp^2$$

Beweis:

$X :=$ Anzahl Knoten, die erste Runde „überleben“

\Rightarrow jeder einzelne Knoten überlebt mit Wahrscheinlichkeit p ,

wir haben n Knoten

\Rightarrow (Linearität des Erwartungswertes) $E[X] = np$

$Y :=$ Anzahl Kanten, die erste Runde „überleben“

\Rightarrow jede einzelne Kante überlebt mit Wahrscheinlichkeit p^2 ,

wir haben m Kanten

\Rightarrow (Linearität des Erwartungswertes) $E[Y] = mp^2$

$S \geq X - Y$ da wir höchstens einen Knoten pro Kante löschen

\Rightarrow (Linearität des Erwartungswertes) $E[S] \geq E[X] - E[Y]$

Figure 2.4

Coupon Collector:

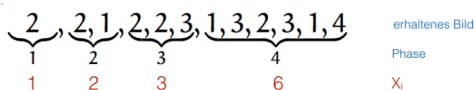
Szenario: Es gibt n verschiedene Bilder in jeder Runde erhalten wir (gleichwahrscheinlich) eines der Bilder
 $X :=$ Anzahl Runden bis wir alle n Bilder besitzen Ziel: Berechne $E[X]$

Lösungsansatz: betrachte n Phasen

Phase i : Runden während wir $i-1$ verschiedene Bilder besitzen

$X_i :=$ Anzahl Runden in Phase i

Beispiel $n=4$:



(a)

$X_i :=$ Anzahl Runden in Phase i , $X_i \sim \text{Geo}((n-(i-1))/n)$

$$E[X] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n \frac{n}{n-i+1} = n \cdot \sum_{i=1}^n \frac{1}{i} = n \cdot H_n,$$

$$H_n = \ln n + O(1)$$

(b)

\Rightarrow die Laufzeit der Coupon Collector ist $O(n \log n + n)$

Varianz: Für ein Zufallsvariable X mit $\mu = E[X]$ definieren wir die Varianz $Var[X]$ durch:

$$Var[X] := E[(X - \mu)^2] = \sum_{x \in W_X} (x - \mu)^2 \cdot Pr[X = x]$$

$$Var[X] = E[X^2] - E[X]^2$$

$$Var[a \cdot X + b] = a^2 \cdot Var[X] \quad X \text{ beliebig, } a, b \in \mathbb{R}$$

(b verschwindet, da verschieben der werte kein einfluss auf die Abweichung vom Durchschnitt hat)

Standardabweichung:

$$\sigma := \sqrt{Var[X]}$$

Dichten: X, Y Zufallsvariablen:

Gemeinsame Dichte:

$$f_{X,Y}(x, y) := Pr[X = x, Y = y]$$

Randdichte:

$$f_X(x) = \sum_{y \in W_Y} f_{X,Y}(x, y)$$

Unabhängigkeit: Zufallsvariablen X_1, \dots, X_n heißen unabhängig genau dann, wenn für alle $(x_1, \dots, x_n) \in W_{X_1} \times \dots \times W_{X_n}$ gilt:

$$Pr[X_1 = x_1, \dots, X_n = x_n] = Pr[X_1 = x_1] \cdot \dots \cdot Pr[X_n = x_n]$$

Alternativ:

$$f_{X_1, \dots, X_n}(x_1, \dots, x_n) = f_{X_1}(x_1) \cdot \dots \cdot f_{X_n}(x_n) \text{ für alle } (x_1, \dots, x_n) \in W_{X_1} \times \dots \times W_{X_n}$$

Beispiel:

$$\Omega = \{1, 2, 3, 6\} \text{ mit } Pr[\omega] = 1/4 \text{ für alle } \omega \in \Omega$$

$$\begin{aligned} X(\omega) &= \begin{cases} 1 & \text{wenn } \omega \text{ durch } 2 \text{ teilbar} \\ 0 & \text{sonst} \end{cases} & f_X(i) &= \begin{cases} 1/2 & \text{für } i=0,1 \\ 0 & \text{sonst} \end{cases} \\ Y(\omega) &= \begin{cases} 1 & \text{wenn } \omega \text{ durch } 3 \text{ teilbar} \\ 0 & \text{sonst} \end{cases} & f_Y(i) &= \begin{cases} 1/2 & \text{für } i=0,1 \\ 0 & \text{sonst} \end{cases} \\ f_{X,Y}(i,j) &= \begin{cases} 1/4 & \text{für alle } (i,j) \in \{0,1\} \times \{0,1\} \\ 0 & \text{sonst} \end{cases} & \Rightarrow & \boxed{\text{X und Y sind unabhängig}} \end{aligned}$$

Figure 2.6

Für zwei Indikatorvariablen X und Y gilt:

$$X \text{ und } Y \text{ sind unabhängig} \iff f_{X,Y}(1,1) = f_X(1) \cdot f_Y(1)$$

Lemma: Sind X_1, \dots, X_n unabhängige Zufallsvariablen und S_1, \dots, S_n beliebige Mengen mit $S_i \subseteq W_{X_i}$, dann gilt:

$$Pr[X_1 \in S_1, \dots, X_n \in S_n] = Pr[X_1 \in S_1] \cdot \dots \cdot Pr[X_n \in S_n]$$

Korollar: Sind X_1, \dots, X_n unabhängige Zufallsvariablen und ist $I = \{i_1, \dots, i_k\} \subseteq [n]$, dann sind X_{i_1}, \dots, X_{i_k} ebenfalls unabhängig.

Satz: f_1, \dots, f_n seien reellwertige Funktionen ($f_i: \mathbb{R} \rightarrow \mathbb{R}$ für $i = 1, \dots, n$). Wenn die Zufallsvariablen X_1, \dots, X_n unabhängig sind dann gilt dies auch für $f_1(X_1), \dots, f_n(X_n)$

Summe von Zufallsvariablen: Für zwei unabhängige Zufallsvariablen X und Y sei $Z := X + Y$. Es gilt:

$$f_Z(z) = \sum_{x \in W_X} f_X(x) \cdot f_Y(z - x)$$

Es folgt:

$$Poisson(\lambda_1) + Poisson(\lambda_2) = Poisson(\lambda_1 + \lambda_2) \quad Bin(n_1, p) + Bin(n_2, p) = Bin(n_1 + n_2, p)$$

falls die lambda's und n's unabhängig sind

Rechenregeln für Momente:

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y] \quad \forall X, Y$$

$$\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y] \quad \forall X, Y \text{ unabhängig}$$

$$\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y] \quad \forall X, Y \text{ unabhängig}$$

$$\text{Var}[X \cdot Y] \neq \text{Var}[X] \cdot \text{Var}[Y] \quad \text{muss ausgerechnet werden}$$

Multiplikativität des Erwartungswerts: Für unabhängige Zufallsvariablen X_1, \dots, X_n gilt:

$$\mathbb{E}[X_1 \cdot \dots \cdot X_n] = \mathbb{E}[X_1] \cdot \dots \cdot \mathbb{E}[X_n]$$

Satz: Für unabhängige Zufallsvariablen X_1, \dots, X_n und $X := X_1 + \dots + X_n$ gilt:

$$\text{Var}[X] = \text{Var}[X_1] + \dots + \text{Var}[X_n]$$

2.3 Abschätzen von Wahrscheinlichkeiten

Waldsche Identität: N und X seien zwei unabhängige Zufallsvariable, wobei für den Wertebereich von N gelte: $W_N \subseteq \mathbb{N}$. Weiter sei:

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

wobei X_1, X_2, \dots unabhängige Kopien von X seien. Dann gilt:

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

Ungleichung von Markov: Sei X eine Zufallsvariable, die nur nicht negative Werte annimmt. Dann gilt für alle $t \in \mathbb{R}$ mit $t > 0$, dass

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t} \quad \forall X \geq 0, \forall t > 0$$

Ungleichung von Chebyshev: Sei X eine Zufallsvariable und $t \in \mathbb{R}$ mit $t > 0$. Dann gilt:

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\text{Var}[X]}{t^2} \quad \forall X, \forall t > 0$$

insbesondere

$$\Pr[X \geq \mathbb{E}[X] + t] \leq \frac{\text{Var}[X]}{t^2}$$

Ungleichung von Chernoff: Die Oberschranke von Chernoff liefert ein viel kleineres Fehler als dass von Chebyshev

$$\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2\mathbb{E}[X]} \quad \forall X \sim \text{Bin}(n, p), \forall 0 < \delta < 1$$

Chernoff-Schranken: Seien X_1, \dots, X_n unabhängige Bernoulliverteilte Zufallsvariablen mit $\Pr[X_i = 1] = p_i$ und $\Pr[X_i = 0] = 1 - p_i$. Dann gilt für $X := \sum_{i=1}^n X_i$

- (i) $\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2\mathbb{E}[X]}$ für alle $0 < \delta \leq 1$
- (ii) $\Pr[X \leq (1 - \delta)\mathbb{E}[X]] \leq e^{-\frac{1}{2}\delta^2\mathbb{E}[X]}$ für alle $0 < \delta \leq 1$
- (iii) $\Pr[X \geq t] \leq 2^{-t}$ für $t \geq 2e\mathbb{E}[X]$

2.4 Randomisierte Algorithmen:

Target-Shooting: Gegeben zwei endliche Mengen $S \subseteq U$ bestimme $|S|/|U|$. Annahmen:

- Wir können ein Element aus U effizient zufällig gleichverteilt wählen
- es gibt eine effizient berechenbare Funktion

$$\mathbb{I}_S(u) := \begin{cases} 1 & \text{falls } u \in S \\ 0 & \text{sonst} \end{cases}$$

TARGET-SHOOTING
 1: Wähle $u_1, \dots, u_N \in U$ zufällig, gleichverteilt und unabhängig
 2: return $N^{-1} \cdot \sum_{i=1}^N \mathbb{I}_S(u_i)$

Notation: $Y_i := \mathbb{I}_S(u_i)$ für alle $i=1, \dots, N$

Y_1, \dots, Y_N unabhängige Bernoulli-Variablen mit $\Pr[Y_i = 1] = |S|/|U|$

$$Y := \frac{1}{N} \sum_{i=1}^N Y_i = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_S(u_i)$$

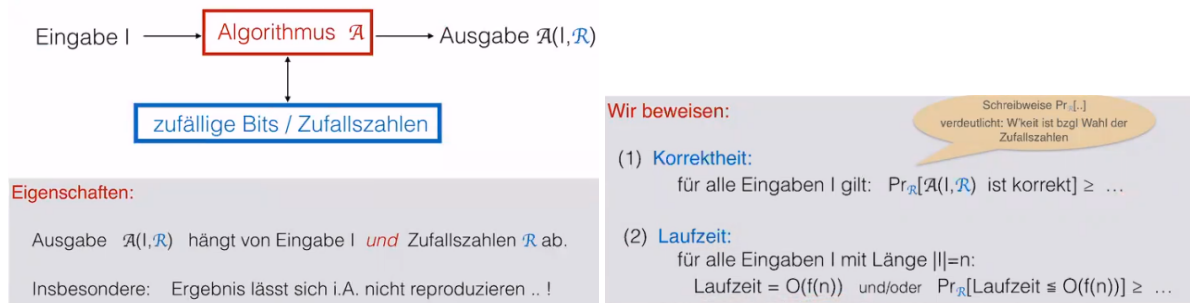
Dann gilt: $\mathbb{E}[Y] = |S|/|U|$... unabhängig von der Wahl von N

$$\text{Var}[Y] = \frac{1}{N} \left(\frac{|S|}{|U|} - \left(\frac{|S|}{|U|} \right)^2 \right)$$

Seien $\delta, \epsilon > 0$. Falls $N \geq 3 \frac{|U|}{|S|} \cdot \epsilon^{-2} \cdot \log(2/\delta)$ so ist die Ausgabe des Algorithmus TARGET-SHOOTING mit Wahrscheinlichkeit mindestens $1 - \delta$ im Intervall $[(1 - \epsilon) \frac{|S|}{|U|}, (1 + \epsilon) \frac{|S|}{|U|}]$. Oder äquivalent,

$$\Pr[|Y - \frac{|S|}{|U|}| > \epsilon \frac{|S|}{|U|}] = \Pr[|Y - \mathbb{E}[Y]| > \epsilon \mathbb{E}[Y]] = \Pr[|NY - \mathbb{E}[NY]| > \epsilon \mathbb{E}[NY]] \leq \delta$$

Da $NY = Y_1 + \dots + Y_N$ eine Summe von N unabhängigen Bernoulli-Variablen ist, kann dies nun leicht mit Hilfe der Chernoff Schranken hergeleitet werden.



Zufallsalgorithmen werden deterministisch indem wir den Zufallsgenerator einen Startwert geben.

Las-Vegas Algorithmen: Geben nie eine falsche Antwort, aber manchmal keine Antwort (Ausgabe = ???).

Ziel: $\Pr[\text{Antwort} = ???] = \text{winzig}$

Monte-Carlo Algorithmen: Geben immer eine Antwort aber manchmal auch eine falsche Antwort.

Ziel: $\Pr[\text{AntwortFalsch}] = \text{winzig}$

Entscheidungsproblem: Problem wo man entscheidet zwischen Ja oder Nein.

Fehlerkorrektur: Für Las Vegas und Monte Carlo Algorithmen kann man durch Wiederholungen den Fehler kleiner machen. Beispiel: Las Vegas

$$\Pr[\text{Antwort} = ???] \leq 1 - \epsilon$$

$$\Rightarrow N := \epsilon^{-1} \ln(\delta^{-1}) \text{ Wiederholungen reduzieren den Fehler auf } \Pr[\text{Antwort immer} ???] \leq (1 - \epsilon)^N \leq \delta$$

In der Praxis ruft man Las-Vegas-Algorithmen meist einfach so lange auf, bis sie eine Antwort geben.

Beispiel Monte Carlo Algorithmen für Entscheidungsprobleme: Einseitiger Fehler:

$$\Pr[\text{Antwortfalsch} | \text{Antwort} = \text{Ja}] = 0$$

$$\Pr[\text{Antwortfalsch} | \text{Antwort} = \text{Nein}] \leq 1 - \epsilon$$

$$\Rightarrow \epsilon^{-1} \ln(\delta^{-1}) \text{ Wiederholungen reduzieren Fehler auf } \Pr[\text{Antwort falsch}] \leq \delta$$

Der Algorithmus liefert Ja, wenn ein Aufruf Ja ausgibt und Nein, wenn alle Wiederholungen Nein ausgeben.
Zweiseitiger Fehler (Algorithmus kann falsch sein wenn es die Antwort Ja oder Nein ausgibt):

$$\Pr[\text{Antwortfalsch}] \leq \frac{1}{2} - \epsilon \Rightarrow 12(\frac{1}{2} - \epsilon) \ln(\delta^{-1}) \text{ Wiederholungen reduzieren Fehler auf } \Pr[\text{Antwortfalsch}] \leq \delta$$

Der Algorithmus liefert die Mehrheit der gesehenen Antworten (i.e Ja falls mehr Ausgaben Ja waren sonst Nein)

Satz 2.72: (Las Vegas Algorithmus) Sei A ein randomisierter Algorithmus, der nie eine falsche Antwort gibt, aber zuweilen '???' ausgibt, wobei

$$\Pr[A(I) \text{ korrekt}] \geq \epsilon \quad \forall I$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit A_δ den Algorithmus der A solange aufruft bis entweder ein Wert verschieden von '???' ausgegeben wird (und A_δ diesen Wert dann ebenfalls ausgibt) oder bis $N = \epsilon^{-1} \ln(\delta^{-1})$ mal '???' ausgegeben wurde (und A_δ dann ebenfalls '???' ausgibt), so gilt für den Algorithmus A_δ , dass

$$\Pr[A_\delta(I) \text{ korrekt}] \geq 1 - \delta$$

Satz 2.74: (Monte-Carlo Algorithmus Einseitiger Fehler) Sei A ein randomisierter Algorithmus, der immer eine der beiden Antworten Ja oder Nein ausgibt, wobei

$$\Pr[A(I) = \text{Ja}] = 1 \quad \text{falls } I \text{ eine Ja-Instanz ist}$$

$$\Pr[A(I) = \text{Nein}] \geq \epsilon \quad \text{falls } I \text{ eine Nein-Instanz ist}$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit A_δ den Algorithmus der A solange aufruft bis entweder der Wert Nein ausgegeben wird (und A_δ dann ebenfalls Nein ausgibt) oder bis $N = \epsilon^{-1} \ln(\delta)^{-1}$ mal Ja ausgegeben wurde (und A_δ dann ebenfalls Ja ausgibt), so gilt für alle Instanzen I

$$\Pr[A_\delta(I) \text{ korrekt}] \geq 1 - \delta$$

Satz 2.75: (Monte-Carlo Algorithmus Beidseitiger Fehler) Sei $\epsilon > 0$ und A ein randomisierter Algorithmus, der immer eine der beiden Antworten Ja oder Nein ausgibt, wobei

$$\Pr[A(I) \text{ korrekt}] \geq \frac{1}{2} + \epsilon \quad \forall I$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit A_δ den Algorithmus, der $N = 4\epsilon^{-2} \ln \delta^{-1}$ unabhängige Aufrufe von A macht und dann die Mehrheit der erhaltenen Antworten ausgibt, so gilt für den Algorithmus A_δ , dass

$$Pr[A_\delta(I) \text{ korrekt}] \geq 1 - \delta$$

Satz 2.76: Sei $\epsilon > 0$ und A ein randomisierter Algorithmus für ein Maximierungsproblem, wobei gelte:

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

Dann gilt für all $\delta > 0$: bezeichnet man mit A_δ den Algorithmus, der $N = \epsilon^{-1} \ln(\delta^{-1})$ unabhängige Aufrufe von A macht und die beste der erhaltenen Antworten ausgibt, so gilt für den Algorithmus A_δ , dass

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

(Für Minimierungsprobleme gilt eine analoge Aussage wenn wir $\geq f(I)$ durch $\leq f(I)$ ersetzen)

Abschätzungen für Binomialverteilung:

- **Markov:** $Pr[X \geq C \cdot \mathbb{E}[X]] \leq \frac{1}{C}$
- **Chebyshev:** $Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{Var[X]}{t^2}$
z.B $t = c \cdot \mathbb{E}[X]$
 $\Rightarrow Pr[X \leq (1 - c)\mathbb{E}[X] \text{ oder } X \geq (1 + c)\mathbb{E}[X]] \leq \frac{n \cdot p \cdot (1 - p)}{(c \cdot np)^2} = \frac{1}{cn} \cdot \frac{1 - p}{p}$
Dieses term = 1 für $p = \frac{1}{2}$ aber gross für $p \approx 0$ bzw $p \approx 1 \rightarrow$ Man kann Chebyshev anwenden wenn der Binomialverteilung ein vernünftig grosses n hat und wenn p von 0 und 1 weg beschränkt ist.
- **Chernoff:** $Pr[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]] \leq e^{-\frac{1}{2} \delta^2 \mathbb{E}[X]} + e^{-\frac{1}{3} \delta^2 \mathbb{E}[X]} \leq 2 \cdot e^{-\frac{1}{2} \delta^2 \mathbb{E}[X]}$
wir wählen $\delta = c \Rightarrow Pr[X \leq (1 - c)\mathbb{E}[X] \text{ oder } X \geq (1 + c)\mathbb{E}[X]] \leq 2 \cdot e^{-\frac{1}{2} c^2 np}$

2.5 Sortieren

```

QUICKSORT(A, ℓ, r)
1: if ℓ < r then
2:   p ← Uniform({ℓ, ℓ + 1, ..., r})    ▷ wähle Pivotelement zufällig
3:   t ← PARTITION(A, ℓ, r, p)
4:   QUICKSORT(A, ℓ, t - 1)
5:   QUICKSORT(A, t + 1, r)

```

Satz:

- QuickSort bestimmt immer das richtige Ergebnis
- $\mathbb{E}[\text{Laufzeit}] = O(n \ln n)$

Quicksort: Da Quicksort immer das richtige Ergebnis gibt ist es ein Las-Vegas Algorithmus. Wir wollen zeigen dass $\mathbb{E}[\text{Laufzeit}] = O(n \cdot \ln(n))$

$t_n := \mathbb{E}[\text{zeit beim Sortieren von n Elementen}]$

$T_n := \# \text{Vergleiche beim sortieren von n Elementen}$

$$T_n = n - 1 (\text{vergleiche mit Pivot element}) + \sum_{i=1}^n Pr[\text{pivot ist der i kleinste element}] \cdot (T_{i-1} + T_{n-i})$$

$$\Rightarrow T_n = n - 1 + \frac{1}{n} \sum_{i=1}^n (t_{i-1} + t_{n-i}) \text{ Durch linearität der Erwartungswert kann T durch t ersetzt werden}$$

$$\Rightarrow t_n = n - 1 + \frac{1}{n} \sum_{i=1}^n (t_{i-1} + t_{n-i}) \rightarrow \text{Löse Rekursion} \Rightarrow O(n \cdot \ln(n))$$

Selektieren: Aufgabe: finde das k-te kleinste Element aus einem unsortierten Array

Selektieren mit QuickSelect: . Wenn das Array A sortiert ist, gibt k den platz der k kleinste elements an. Wir

```

QUICKSELECT(A, ℓ, r, k)
1: p ← Uniform({ℓ, ℓ + 1, ..., r})    ▷ wähle Pivotelement zufällig
2: t ← PARTITION(A, ℓ, r, p)
3: if t = k - 1 then
4:   return A[t]                      ▷ gesuchtes Element ist gefunden
5: else if t > k - 1 then
6:   return QUICKSELECT(A, ℓ, t - 1, k)  ▷ gesuchtes Element ist links
7: else
8:   return QUICKSELECT(A, t + 1, r, k - t) ▷ gesuchtes Element ist rechts

```

Satz:

- QuickSelect bestimmt immer das richtige Ergebnis
- $\mathbb{E}[\text{Laufzeit}] = O(n)$

wählen ein Pivot p und wir partitionieren das array so dass alle elemente kleiner als p links von p sind und alle elemente die grösser sind rechts von p. Danach vergleichen wir den index von p und k. Falls k kleiner als p ist, wissen wir dass unser gesuchte element sich im linken teilarray befindet sonst im rechten. (falls index von p = index von k ist p unser gesuchtes element).

Beweis $\mathbb{E}[\text{Laufzeit}] = \mathcal{O}(n)$

Die Laufzeit ist proportional zur anzahl vergleiche. In Jeder schritt wird der funktion auf der linke seite oder rechte seite.

$$\Rightarrow T := \# \text{vergleiche} = \sum_{i=0}^N (r_i - l_i)$$

Wir definieren eine neue Zufallsvariable $N_i := \#i$ mit $(\frac{3}{4})^j r_i - l_i \leq (\frac{3}{4})^{j-1} n$

$$\Rightarrow T \leq \sum N_j \cdot (\frac{3}{4})^{j-1} \cdot n$$

$$\Rightarrow \mathbb{E}[T] \leq \sum \mathbb{E}[N_j] \cdot (\frac{3}{4})^{j-1} \cdot n$$

Beobachtung: Wir werden mit Wahrscheinlichkeit $\geq \frac{1}{2}$ mind $\frac{1}{4}$ der Elemente wegerfen. $(r_i - l_i \rightarrow r_{i+1} - l_{i+1})$, sodass $r_{i+1} - l_{i+1} \leq \frac{3}{4}(r_i - l_i)$

$$\Rightarrow \mathbb{E}[N_j] \leq \mathbb{E}[\text{Geo}(\frac{1}{2})] = 2$$

$$\Rightarrow \mathbb{E}[T] \leq 2n \sum (\frac{3}{4})^{j-1} \leq 2n \frac{1}{1-\frac{3}{4}} = 8n \Rightarrow \text{Laufzeit} = \mathcal{O}(n)$$

2.6 Primzahltest

Primzahlfunktion: Gibt an wie viele primzahlen bis zu einer zahl x es gibt.

$$\pi(x) := |\{n \in \mathbb{N} | n \leq x, n \text{ prim}\}| \sim \frac{x}{\ln x}$$

Grösster gemeinsamer Teiler: Für $m, n \in \mathbb{Z}$ sei $\text{ggT}(m, n)$ der grösste gemeinsame Teiler von m und n . Er kann mit Hilfe des Euklid'schen Algorithmus schnell berechnet werden ($\mathcal{O}((\log nm)^3)$). Natürlich gilt:

$$\text{ggT}(a, n) > 1 \text{ für } a \in [n-1] \Rightarrow n \text{ nicht prim}$$

Euklid-Primzahltest(n)
1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: if $\text{ggT}(a, n) > 1$ then return 'keine Primzahl'
3: else return 'Primzahl'

Bemerkungen:

- Ausgabe "keine Primzahl" ist immer richtig
- Falls n nicht prim: Falsche Ausgabe "Primzahl" mit W'keit $\frac{|\mathbb{Z}_n^*|}{n-1}$

Multiplikative Gruppe mod n :

$$\mathbb{Z}_n^* = \{a \in [n-1] | \text{ggT}(a, n) = 1\}$$

Eulersche Phi-Funktion:

$$\phi(n) := |\mathbb{Z}_n^*|$$

Falls n prim: $\mathbb{Z}_n^* = [n-1]$ und $\phi(n) = n-1$

Falls $n = p^2$ p prim: $\phi(n) = p(p-1) = n - \sqrt{n}$ ($\Rightarrow \frac{|\mathbb{Z}_n^*|}{n-1} \approx 1 - \frac{1}{\sqrt{n}}$ D.h die wahrscheinlichkeit dass wir in Irre geführt werden ist gross) Bemerkungen zu Fermats Primzahltest:

Erinnerung Gruppentheorie

- Satz von Lagrange \Rightarrow Ist $H \leq G$ (H Untergruppe von G), dann ist $|H|$ ein Teiler von $|G|$.
- $\text{ord}(a) := \min\{i \in \mathbb{N} | a^i = 1\}$, die Ordnung der von a erzeugten Untergruppe $\langle a \rangle := \{a^0 = 1, a^1, a^2, \dots\}$.
- $\text{ord}(a)$ ist Teiler von $|G|$. Folglich gilt

$$a^{|G|} = a^{\text{ord}(a) \cdot \frac{|G|}{\text{ord}(a)}} = 1^{\frac{|G|}{\text{ord}(a)}} = 1.$$

- In \mathbb{Z}_n^* : $a^{\phi(n)} = 1$ für alle $a \in \mathbb{Z}_n^*$ (weil $\phi(n) = |\mathbb{Z}_n^*|$)
- In \mathbb{Z}_n^* , n prim: $a^{n-1} = 1$ für alle $a \in \mathbb{Z}_n^*$ (weil $\phi(n) = n-1$).

Satz (Kleiner fermatscher Satz)

Ist $n \in \mathbb{N}$ prim, so gilt für alle Zahlen $a \in [n-1]$

$$a^{n-1} \equiv 1 \pmod{n} \quad (\text{bzw. } a^{n-1} = 1 \text{ in } \mathbb{Z}_n^*).$$

Fermat-Primzahltest(n)
1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: if $\text{ggT}(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ then
3: return 'keine Primzahl'
4: else
5: return 'Primzahl'

- $a^{n-1} \not\equiv 1 \pmod{n}$ kann schnell mit binärer Exponentiation berechnet werden
- Die Ausgabe keine Primzahl ist immer richtig
- Die Ausgabe Primzahl ist falsch mit W'keit $\frac{|PB_n|}{n-1} < \frac{1}{2}$ es sei denn n ist eine Carmichael-Zahl
- Wir machen einen Fehler, falls n nicht prim, $\text{ggT}(a, n) = 1$ und $a^{n-1} \equiv 1 \pmod{n}$. Ein solches a nennen wir **Pseudoprimzahlbasis** von n .

Pseudoprimzahlbasen und Carmichael-Zahlen:

$$PB_n := \{a \in [n-1] | \text{ggT}(a, n) = 1 \text{ und } a^{n-1} \equiv 1 \pmod{n}\} = \{a \in \mathbb{Z}_n^* | a^{n-1} \equiv 1 \pmod{n}\}$$

n heisst Carmichael-Zahl, falls n nicht prim ist und $PB_n = \mathbb{Z}_n^*$
 PB_n ist eine Untergruppe von \mathbb{Z}_n^* , weil

$$a^{n-1} \equiv 1 \pmod{n} \wedge b^{n-1} \equiv 1 \pmod{n} \Rightarrow (ab \pmod{n})^{n-1} \equiv 1 \pmod{n}$$

Folglich, wenn $PB_n \neq \mathbb{Z}_n^*$ (d.h. n ist nicht Carmichael), dann ist $|PB_n|$ ein echter Teiler von \mathbb{Z}_n^* und $|PB_n| \leq \frac{\phi(n)}{2} < \frac{n-1}{2}$

Zertifikate für nicht prim:

- triviales Zertifikat, falls $a > 2$ und $a|n$: Anteil $> \frac{1}{n}$
- Euklid Zertifikat, falls $\text{ggT}(a, n) > 1$: Anteil $> \frac{1}{\sqrt{n}}$
- Fermat Zertifikat, falls $a^{n-1} \pmod{n} \neq 1$: Anteil $> \frac{1}{2}$, ausser n ist Carmichael
- Miller-Rabin Zertifikat, falls

$$(a^d, a^{2d}, \dots, a^{2^k d}) \neq \begin{cases} (1, 1, 1, \dots, 1) & (\iff a^d \pmod{n} = 1) \\ (\cdot, \dots, \cdot, n-1, 1, \dots, 1) & (\iff \exists i : a^{2^i d} \pmod{n} = n-1) \end{cases}$$

Anteil $> \frac{3}{4}$ (n ungerade)

Miller-Rabin Test: See below

Miller-Rabin Zertifikat

$$\mathbb{Z}_n := \{0, 1, \dots, n-1\}$$

Für n prim, ist $(\mathbb{Z}_n, +, \cdot, ', ' \text{ und } '+ \pmod{n}$, ein Körper (mit 0 additives, und 1 multiplikatives neutrales Element).

In einem Körper hat die Gleichung $x^2 = 1$, zwei Lösungen:
 $x = 1$ und $x = -1 = n-1$

Ist $n > 2$ prim und $a \in [n-1]$, dann gilt

- ▶ $a^{n-1} = 1$, (kleiner fermatscher Satz)
- ▶ $a^{\frac{n-1}{2}} \in \{1, n-1\}$, (Körpereigenschaft oben)
- ▶ falls $a^{\frac{n-1}{2}} = 1$ und $\frac{n-1}{2}$ gerade, dann $a^{\frac{n-1}{4}} \in \{1, n-1\}$,
- ▶ ... bis $a^{\frac{n-1}{2^i}} = n-1$ oder $\frac{n-1}{2^i}$ ungerade.

Miller-Rabin-Primzahltest(n) ($n > 1$, ungerade!)

- 1: $d, k \in \mathbb{N}$, mit $n-1 = 2^k d$, d ungerade
- 2: Wähle $a \in [n-1]$, zufällig gleichverteilt
- 3: **if** $a^d \pmod{n} \neq 1$ und $\nexists i < k : a^{2^i d} \pmod{n} = n-1$ **then**
- 4: **return** 'keine Primzahl'
- 5: **else**
- 6: **return** 'Primzahl'

- ▶ Die Ausgabe 'keine Primzahl' ist immer richtig.
- ▶ Die Ausgabe 'Primzahl' ist falsch mit W'keit $\leq \frac{1}{4}$.

2.7 Duplikate Finden in grossen Datensätzen

Problemstellung: $S = (s_1, s_2, \dots, s_n)$, Folge von n Elementen (Datensatz). (i, j) , $1 \leq i < j \leq n$ heisst Duplikat in S , falls $s_i = s_j$. Beispiel: Gegeben ein Datensatz S , finde alle Duplikate.

$$S = (A \ C \ B \ Z \ C \ B \ C)$$

Duplikate $\text{Dupl}(S) = \{(2, 5), (6, 7), (5, 7), (3, 6)\}$ (4 Duplikate)

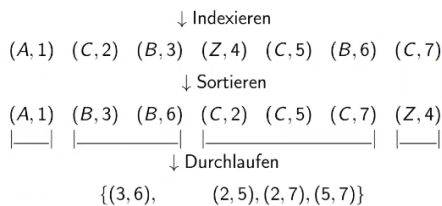
weitere Annahme:

Die Elemente in S sind gross d.h. Speicherzugriffe und Vergleiche sind teuer.

Lösung:

- Erste Lösung:

Sortiere die Folge $((s_1, 1), (s_2, 2), \dots, (s_n, n))$ nach erster Koordinate. Dann sind die Duplikate leicht durch einen Durchlauf der sortierten Folge zu finden.



- (i) Sortieren: $O(n \log n)$ Vergleiche und Speicherzugriffe
(ii) Durchlaufen: $O(n + |\text{Dupl}(S)|)$

- Zweite Lösung: Hash Funktion

Seien die Elemente in S aus einem Universum U . Wir verwenden eine Hashfunktion $h: U \rightarrow [m]$ und setzen voraus:

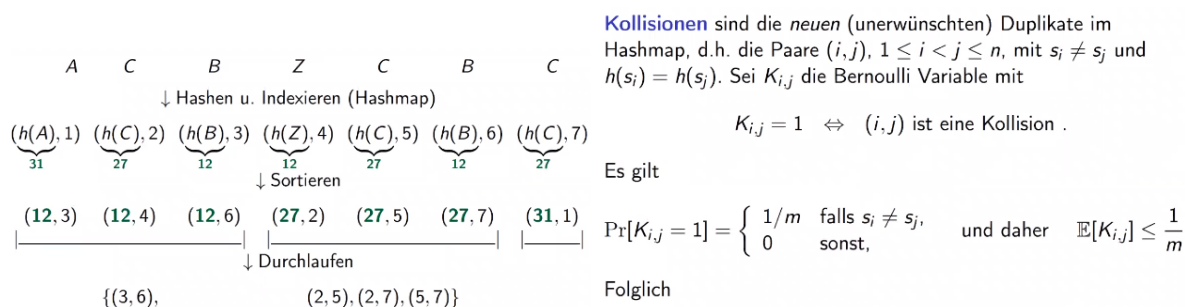
- h ist effizient berechenbar
- h verhält sich wie eine Zufallsfunktion d.h.

$$\forall u \in U \quad \forall i \in [m] : \Pr[h(u) = i] = \frac{1}{m}$$

Bei Hashfunktionen wird m viel kleiner als $|U|$ gewählt d.h es gibt eine Komprimierung. Jedes $h(s_i)$ ist zufällig gleichverteilt in $[m]$, aber

$$s_i = s_j \Rightarrow h(s_i) = h(s_j)$$

Es kann sein, dass zwei verschiedene Werte auf den gleiche Zahl abgebildet werden, dass nennt man **Kollision**



Beim Durchlaufen müssen eventuelle Kollisionen (hier z.B. wegen $h(B) = h(Z)$) im Hashmap noch geprüft werden.

$$\mathbb{E}[\#\text{Kollisionen}] = \sum_{1 \leq i < j \leq n} \mathbb{E}[K_{i,j}] \leq \binom{n}{2} \frac{1}{m}$$

$$\mathbb{E}[\#\text{Kollisionen}] \leq \binom{n}{2} \frac{1}{m} < 1 \quad \text{für } m = n^2$$

Mit $m = n^2$ ist der Mehraufwand durch Kollisionen konstant.

$$\text{Zeit: } \underbrace{O(n)}_{\text{Hashmap}} + \underbrace{O(n \log n)}_{\text{Sortieren von } \lceil 2 \log n \rceil\text{-Bit Zahlen}} + \underbrace{O(n + |\text{Dupl}(S)|)}_{\text{Durchlaufen}}$$

$$\text{zusätzlicher Speicher: } \underbrace{O(n \log n)}_{\text{Indices}} + \underbrace{n \log m}_{\text{Hashwerte}} \stackrel{m=n^2}{=} O(n \log n)$$

- Dritter Lösung: Bloom-Filter Wir wählen $m, k \in \mathbb{N}$ und k Hashfunktionen (zufällig)

$$h_i : U \rightarrow [m] \quad i = 1, \dots, k$$

d.h jede s_i in S wird ein Hash-Vektor zugeordnet. Wir benutzen auch ein boolsches Feld $M[1..m]$ welche am anfangs all Einträge auf 0 gesetzt werden. Wir gehen jetzt die elemente von S durch und setzen $M[i]$ auf 1 falls der indize in der Hashvektor enthalten ist und $M[i]$ vorher 0 war. Falls alle $M[i] = 1$ für diese hashvektor waren, dann nehmen wir an dass wir dieses element schon einmal gesehen haben und fügen den indizes in \mathcal{L} eine Liste der Wiederholungen ein. Wir interessieren uns für die Wahrscheinlichkeit einer falsche eintrag in \mathcal{L}

Betrachten wir s_i mit Hash-Vektor (x_1, \dots, x_k) .

$$X_i = 1 \Leftrightarrow \begin{cases} s_i \text{ tritt in } (s_1, \dots, s_{i-1}) \text{ nicht auf und} \\ M[x_1] = \dots = M[x_k] = 1 \text{ vor dem Abarbeiten von } s_i. \end{cases}$$

d.h. i wird ein falscher \mathcal{L} -Eintrag.

Nach den ersten $i-1$ Daten, $\ell \leq i-1$ verschieden, gilt für $x \in [m]$:

$$\Pr[M[x] = 0] = \left(1 - \frac{1}{m}\right)^{k\ell} \geq \left(1 - \frac{1}{m}\right)^{k(i-1)}$$

$$\Pr[X_i = 1] \leq \Pr[\forall j: M[x_j] = 1] \stackrel{(*)}{\leq} \left(1 - \left(1 - \frac{1}{m}\right)^{k(i-1)}\right)^k$$

(*) nutzt "negative Korrelation", hier nicht behandelt.

$$\mathbb{E}[\#\text{falsche Einträge in } \mathcal{L}] = \sum_{i=1}^n \mathbb{E}[X_i] \leq n \cdot \left(1 - \left(1 - \frac{1}{m}\right)^{k(n-1)}\right)^k$$

Dieser Wert ist $O(1)$ für $k = \ln n$ und $m = n \ln n$.

k und m gross \Rightarrow #Falsche Einträge klein.

Laufzeit: kn Hashfunktionsberechnungen. k gross \Rightarrow langsamer.

Zusätzlicher Speicher: m . m gross \Rightarrow mehr Speicher.

2.8 Lange Pfade

LONG-PATH Problem: Gegeben (G, B) , G eine Graph und $B \in \mathbb{N}_0$ stelle fest ob es einen Pfad der Länge B in G gibt. Ein Pfad der Länge l in einem Graph $G = (V, E)$ ist eine Folge von paarweise verschiedenen Knoten

$$\langle v_0, v_1, \dots, v_l \rangle \text{ mit } \{v_{i-1}, v_i\} \in E \text{ für } i = 1, \dots, l$$

d.h. es liegen $l+1$ Knoten auf einem Pfad der Länge l .

Wenn es einen Polynomialzeit-Algorithmus für LONG-PATH gibt, dann gibt es einen Polynomialzeit-Algorithmus für das Hamiltonkreis Problem was NP-Vollständig ist d.h. wir vermuten es gibt keinen Polynomialzeit algorithmus für den LONG-PATH. In der Regel sucht man kurze lange pfade i.e. $B = O(\log n)$. Wir werden folgende hilfsmittel brauchen:

- ▶ Für $n \in \mathbb{N}_0$ gilt $\sum_{i=0}^n \binom{n}{i} = 2^n$, eine Anwendung des Binomialsatzes: $\sum_{i=0}^n \binom{n}{i} x^i y^{n-i} = (x+y)^n$.
- ▶ Für $n \in \mathbb{N}_0$ gilt $\frac{n!}{n^n} \geq e^{-n}$. Potenzreihenentwicklung der Exponentialfunktion:

$$e^n = \sum_{i=0}^{\infty} \frac{n^i}{i!} \geq \frac{n^n}{n!} \quad (\text{der Term in der Summe für } i = n)$$
- ▶ $[n] := \{1, 2, \dots, n\}$; $[n]^k$ ist die Menge der Folgen über $[n]$ der Länge k , es gilt $|[n]^k| = n^k$; $\binom{[n]}{k}$ ist die Menge der k -elementigen Teilmengen von $[n]$ und es gilt $\left|\binom{[n]}{k}\right| = \binom{n}{k}$.
- ▶ Für jeden Graph $G = (V, E)$ gilt $\sum_{v \in V} \deg(v) = 2|E|$.
- ▶ k Knoten man mit $[k]$ auf genau k^k Arten färben, $k!$ dieser Färbungen nutzen jede Farbe genau einmal.
- ▶ Wiederholt man ein Experiment mit Erfolgswahrscheinlichkeit p solange bis man Erfolg hat, dann ist der Erwartungswert der Anzahl der Versuche $\frac{1}{p}$ (geometrische Verteilung $\text{Geo}(p)$).
- ▶ Für $c, n \in \mathbb{R}^+$, gilt $c^{\log n} = n^{\log c}$. Also, z.B. $2^{\log n} = n^{\log 2} = n$ und $2^{O(\log n)} = n^{O(1)}$ ist polynomiell in n .
- ▶ Dynamische Programmierung, ...

Bunte Pfade: Ein Pfad heisst bunt, falls alle seine Knoten verschiedene Farben haben.

COLORFUL-PATH Problem: Gegeben (G, γ) , $G = (V, E)$ ein Graph und $\gamma: V \rightarrow [k]$, stelle fest ob es einen bunten Pfad der Länge $k-1$ (d.h. mit k Knoten) in G gibt. (G muss nicht notwendigerweise eine gültige Färbung haben d.h. dass beliebig viele Knoten dürfen die gleiche Farbe besitzen) Ein bunter Pfad der Länge i zu v besteht aus einem $\gamma(v)$ -freien

Wir fixieren einen Knoten v und und $i \in \mathbb{N}_0$. Wir betrachten nur bunte Pfade der Länge i (mit $i+1$ Knoten), die in v enden, und sammeln alle Farbmengen, die auf solchen Pfaden auftreten können.

$$P_i(v) := \{S \in \binom{[k]}{i+1} \mid \exists \text{ in } v \text{ endender genau mit } S \text{ gefärbter bunter Pfad}\};$$

- ▶ $\forall S \in P_i(v) : \gamma(v) \in S$.
- ▶ $P_0(v) = \{\{\gamma(v)\}\}$.
- ▶ $P_1(v) = \{\{\gamma(x), \gamma(v)\} \mid x \in N(v), \gamma(x) \neq \gamma(v)\}$.
 $N(x) :=$ Menge der Nachbarn von x .
- ▶ \exists bunter Pfad mit k Knoten $\Leftrightarrow \bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$.

bunten Pfad der Länge $i-1$ zu einem Nachbarn x von v plus dem Schritt zu v .

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \gamma(v) \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}$$

Wir berechnen alle $P_i(v)$, $v \in V$ mit den $P_{i-1}(v)$, $v \in V$ gegeben:

	Regenbogen(G, γ)	G Graph, γ k -Färbung
	1: for all $v \in V$ do $P_0(v) \leftarrow \{\{\gamma(v)\}\}$	
	2: for $i = 1..k-1$ do Bunt(G, i)	
	3: return $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$	
Bunt(G, i)	1: for all $v \in V$ do	
	2: for all $x \in N(v)$ do $\deg(v)$	
	3: for all $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ do $P_{i-1}(x) \cdot i$	
	4: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$	

$$O\left(|V| + \sum_{i=1}^{k-1} \left(\binom{k}{i} \cdot i \cdot m\right) + |V|\right) = O(2^k km).$$

Mit $m := |E|$ und wegen $|P_{i-1}(x)| \leq \binom{k}{i}$ ist der Zeitaufwand

wegen $\sum_{i=1}^{k-1} \binom{k}{i} \leq \sum_{i=0}^k \binom{k}{i} = 2^k$

$$O\left(\sum_{v \in V} \overbrace{\deg(v)}^{2m} \cdot \binom{k}{i} \cdot i\right) = O\left(\binom{k}{i} \cdot i \cdot m\right).$$

D.h. für $k = \log n$: Laufzeit $O(mn \log n)$,
 $k = O(\log n)$: Laufzeit $O(\text{poly}(n))$.

Damit kann man den LONG-PATH Problem lösen:

Setze $k := B+1$, färbe G zufällig mit k Farben, und suche einen bunten Pfad mit k Knoten. Angenommen G hat einen bunten Pfad P mit k Knoten. Dann ist die W'keit p_{erfolg} , dass es in (G, γ) einen bunten Pfad gibt:

$$p_{\text{erfolg}} := Pr[\exists \text{ bunter Pfad mit } k \text{ Knoten}] \geq Pr[P \text{ ist bunt}] = \frac{k!}{k^k} \geq e^{-k}$$

Daraus folgt die erwartete Anzahl versuche, bis man einen bunten Pfad findet $\leq e^k (Geo(e^{-k}))$

Der Laufzeit für einen Versuch ist in: $O(2^k km)$ $p_{\text{erfolg}} \geq e^{-k}$

Macht man $\lceil \lambda e^k \rceil$ Versuche:

- Laufzeit: $O(\lambda(2e)^k km)$
- W'keit dass der Algorithmus den Pfad nicht findet ist höchstens

$$\leq (1 - e^{-k})^{\lceil \lambda e^k \rceil} \leq (e^{-e^{-k}})^{\lceil \lambda e^k \rceil} \leq e^{-\lambda} \text{ (wobei wir beim ersten schritt } 1 + x \leq e^x \text{ angewendet haben)}$$

Chapter 3

Flüsse in Netzwerken

3.1 Einführung & Modellierung

MaxFlow Problem: Gegeben ein Netzwerk, finde einen Fluss grössten Werts.

Netzwerk: Ein Netzwerk ist ein Tupel $N = (V, A, c, s, t)$ wobei gilt:

- (V, A) ist ein gerichteter Graph (ohne Schleifen)
- $s \in V$ die Quelle
- $t \in V \setminus \{s\}$, die Senke
- $c : A \rightarrow \mathbb{R}_0^+$ die Kapazitätsfunktion

Fluss: Sei $N = (V, A, c, s, t)$ ein Netzwerk. Ein Fluss in N ist eine Funktion $f : A \rightarrow \mathbb{R}$ mit den Bedingungen:

- **Zulässigkeit:** $0 \leq f(e) \leq c(e)$ für alle $e \in A$
- **Flusserhaltung:** Für alle $v \in V \setminus \{s, t\}$ gilt:

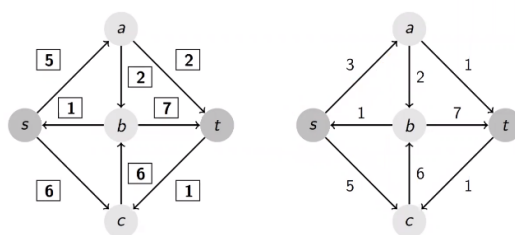
$$\sum_{u \in V : (u, v) \in A} f(u, v) = \sum_{u \in V : (v, u) \in A} f(v, u)$$

i.e die Summe der Flüsse in einen Knoten ist gleich die Flüsse aus dem Knoten.

Die Flüsse sind gerichtet. Der Wert eines Flusses f ist definiert als:

$$\text{val}(f) := \text{netoutflow}(s) := \sum_{u \in V : (s, u) \in A} f(s, u) - \sum_{u \in V : (u, s) \in A} f(u, s)$$

Beispiel: links ist der Netzwerk mit der Kapazitäten, rechts die Flüsse. Bei alle Knoten ausser der Quelle und Senke wollen



Fluss mit Wert $3 - 1 + 5 = 7$

wir dass gleich viel hineinfliesst wie herausfliesst. Der **Nettozufluss** der Senke t gleicht dem Wert des Flusses, d.h

$$\text{netinflow}(t) := \sum_{u \in V : (u, t) \in A} f(u, t) - \sum_{u \in V : (t, u) \in A} f(t, u) = \text{val}(f)$$

Schnitt: Ein s - t -Schnitt für ein Netzwerk ist eine Partition (S, T) von V mit $s \in S$ und $t \in T$. Die **Kapazität** eines s - t -Schnitts (S, T) ist durch

$$\text{cap}(S, T) := \sum_{(u, w) \in (S \times T) \cap A} c(u, w)$$

definiert. Die Kapazität eines Schnitts (S, T) ignoriert die Kanten von T nach S !

\Rightarrow Ist f ein Fluss und (S, T) ein s - t -Schnitt in einem Netzwerk (V, A, c, s, t) so gilt:

$$\text{val}(f) \leq \text{cap}(S, T)$$

d.h ein Fluss kann nie grösser sein als die Kapazität eines s-t-Schnitts. Finden wir zu einem Fluss f einen s-t-Schnitt (S,T) mit $\text{cap}(S,T) = \text{val}(f)$, so ist f ein maximaler Fluss.

Maxflow-Mincut Theorem: Jedes Netzwerk $N = (V,A,c,s,t)$ erfüllt

$$\max_{f \text{ Fluss}} \text{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \text{cap}(S,T)$$

Da es nur endlich viele s-t-Schnitte gibt:

s-t-MinCut ist ein endliches algorithmisches Problem
ein minimaler Schnitt existiert immer