Definition

Landau-Symbole

CoMa

Zahlentheorie

Modulo

CoMa

Definition

Maschinenzahlbereich $F(b, m, E_{\min}, E_{\max})$

CoMa

Python

Was sind `round(0.5)` und `round(1.5)`

CoMa

Definition

Inzidenzmatrix von $G = (V, E, \beta)$

CoMa

Definition

Baum

CoMa

Definition

Adjazenzliste- und Matrix und
Adjazenz/Inzidenz

CoMa

Definition

(schwacher, starker) Zusammenhang

CoMa

Definition

Branching, Aboreszenz, Eulertour

CoMa

Definition

Kantenzug, Weg und Kreis

CoMa

Zu $a \in \mathbb{Z}$, $m \in \mathbb{N}\backslash\{0\}$ existieren eindeutige $p \in \mathbb{Z}$ und $q \in \{0, \dots, m-1\}$ mit $a = pm + q$. Wir schreiben $q = a \mod m$. $(a \oplus b) \mod m = ((a \mod m) \oplus (b \mod m)) \mod m$ für $\oplus \in \{+, \cdot\}$.

Für $f\colon \mathbb{N} \to \mathbb{R}$ gilt

$O(f) := \{g\colon \mathbb{N} \to \mathbb{R} : \exists a > 0,\ N \in \mathbb{N} : 0 \leqslant g(n) \leqslant af(n)\ \forall n > N\}$

$\Omega(f) := \{g\colon \mathbb{N} \to \mathbb{R} : \exists a > 0,\ N \in \mathbb{N} : af(n) \leqslant g(n) \geqslant 0\ \forall n > N\}$

$\Theta(f) := O(f) \cap \Omega(f) \qquad (\ \beta f(n) \leqslant g(n) \leqslant af(n)\ )$

$f \in O(g) \iff f$ wächst höchstens wie g.
$f \in \Omega(g) \iff f$ wächst mindestens wie g.
Für Funktionen f, g gilt *nicht* stets $f \in O(g)$ oder $g \in O(f)$.
$f \in O(g) \implies \exists N \in \mathbb{N} : \forall n \geqslant N\ \exists a > 0 : f(n) \leqslant ag(n)$.

0 und 2...

Basis b, Mantissenlänge m, Exponent $E \in [E_{\min}, E_{\max}] \cap \mathbb{N}$:

$$\mathbb{Z} \ni x = \sigma \left(\sum_{k=0}^{m-1} z_k b^{E-k} \right) \text{ (m-stellige b-adische Darstellung)}$$

mit Vorzeichen $\sigma \in \{\pm 1\}$.

Ein einfacher ungerichteter Graph $G = (V, E)$ ist ein *Baum*, wenn er einer der folgenden äquivalenten Bedingungen erfüllt.

- G ist zusammenhängend und kreisfrei.
- G ist kreisfrei und $|E| = |V| - 1$.
- G ist zusammenhängend und $|E| = |V| - 1$.
- G ist minimal zusammenhängend.
- G ist maximal kreisfrei.
- zwischen je zwei Knoten existiert ein eindeutiger Weg.

gerichtet: $A = (a_{x,e})_{x \in V, e \in E} \in \mathbb{Z}^{|V| \times |E|}$ mit

$$a_{x,e} = \begin{cases} -1, & \text{wenn } e \in \delta^+(x) := \{e \in E : \exists w \in V : e = (x, w)\}, \\ 1, & \text{wenn } e \in \delta^-(x) := \{e \in E : \exists w \in V : e = (w, x)\}, \\ 0, & \text{sonst.} \end{cases}$$

ungerichtet: $A = (a_{x,e})_{x \in V, e \in E} \in \mathbb{Z}^{|V| \times |E|}$ mit

$$a_{x,e} = \begin{cases} 1, & \text{wenn } e \in \delta(x) := \{e \in E : x \in e\}, \\ 0, & \text{sonst.} \end{cases}$$

Ein ungerichteter Graph ist *zusammenhängend*, wenn es für jede zwei Knoten $v, w \in V$ einen v-w-Weg in G gibt.
Es gibt genau dann einen v-w-Weg in G, wenn es einen Kantenzug von v nach w gibt.
G ist genau dann zusammenhängend, wenn G einen Baum als aufspannenden Teilgraphen enthält.
G heißt (schwach) zusammenhängend, falls der zugrunde liegende ungerichtete Graph zusammenhängend ist.
G heißt stark zusammenhängend, falls es für alle $s, t \in V$ einen Weg von s nach t und einen Weg von t nach s in G gibt.

Existiert eine Kante $e = \{v, w\}$ bzw. (v, w), so sind v und w *adjazent*; sie sind *Endknoten* von e und mit dieser inzident.
Adjazenzliste: Jeder Knoten verwaltet eine Liste der zu ihm inzidenten Kanten (oder alternativ zu ihm adjazenten Knoten). In gerichteten Graphen verwaltet jeder Knoten entweder eine Liste der ausgehenden Kanten oder zwei Listen, getrennt nach ausgehenden und eingehenden Kanten.
Adjazenzmatrix von $G = (V, E, \beta)$, $A = (a_{x,y})_{x,y \in V} \in \mathbb{Z}^{|V| \times |V|}$ mit $a_{x,y} = |\{e \in E : \beta(e) = \{x, y\} \text{ bzw. } \beta(e) = (x, y)\}|$.

Ein *Kantenzug* in eine Graphen $G = (V, E, \Psi_G)$ ist eine Folge $v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}$ mit $v_i \in V$, $e_i \in E$ und $\Psi_G(e_i) = \{v_i, v_{i+1}\}$.
Ist $v_1 = v_{k+1}$ ist der Kantenzug *geschlossen*.
Ein v_1-v_{k+1}-*Weg* ist ein Kantenzug, sodass $v_i \neq v_j$ für $1 \leqslant i < j \leqslant k+1$.
Ein Kantenzug der Länge Null besteht aus einen Knoten und ist ein Weg.
Gilt $v_i \neq v_j$ für alle $1 \leqslant i < j \leqslant k$, so ist ein geschlossener Kantenzug ein *Kreis*.

Branching: gerichteter Graph, dessen zugrunde liegender ungerichteter Graph ein Wald ist und $\delta^-(v) \leqslant 1\ \forall v \in V$
Eine Aboreszenz ist ein zusammenhängendes Branching und hat $|V| - 1$ Kanten. Somit existiert genau ein Knoten r mit $\delta^-(r) = \varnothing$; er ist die *Wurzel*.
Eine *Eulertour* ist ein geschlossener Kantenzug, der jede Kante genau einmal enthält.

Definition

Eulertour

CoMa

Definition

(einfacher) (un)gerichteter Graph

CoMa

Definition

(Aufspannender) Teilgraph bzw. - Baum

CoMa

Definition

(starke) Zusammenhangskomponenten, Wald, Blatt

CoMa

Definition

Zugrunde liegende ungerichtete Graph

CoMa

Listen vs Matrizen

CoMa

Spezielle Listen: Stacks und Queues

CoMa

Sortieralgorithmus

Selectionsort

CoMa

Algorithmisches Schema

Graphendurchmusterung

CoMa

Definition + Beispiel

Partielle und totale Ordnungen

CoMa

Ungerichteter Graph: Ein Tripel (G, E, Ψ) mit $|V|, |E| < \infty$, $V \neq \varnothing$ und $\Psi \colon E \to \{X \subset V : |X| = 2\}$.
Einfach: keine parallele Kanten, i.e. $e, f \in E$,it $\Phi(e) = \Phi(f)$.

Eine *Eulertour* ist ein geschlossener Kantenzug, der jede Kante genau einmal enthält.
In einem ungerichten Graphen mit $|\delta(v)|$ gerade für alle $v \in V$ (oder mit $|\delta^-(x)| = |\delta^+(x)|$ wenn gerichtet), zerfällt die Kantenmenge in kantendisjunkte Kreise.

Die (starken) Zusammenhangskomponenten von G sind die maximalen (stark) zusammenhängenden Teilgraphen von G. G heißt Wald, falls alle seine Zusammenhangskomponenten Bäume sind.
Ist $G = (V, E)$ ein Wald und $v \in V$ vom Grad 1, so heißt v *Blatt*. Ein Baum mit mindestens zwei Knoten besitzt mindestens ein Blatt (sogar mindestens zwei).

Ein Graph $H = (V_H, E_H, \Psi_H)$ ist ein *Teilgraph* von $G = (V_G, E_G, \Psi_G)$, wenn $V_H \subset V_G$, $E_H \subset E_V$ und $\Psi_H = \Psi_G|_{E_H}$.
Ist $V_H = V_G$, s ist H ein *aufspannender Teilgraph*.
Ist $E_H = \{e \in E_G : \Psi_G(e) = \{v, w\},\ v, w \in V_H\}$ (bzw. mit (v, w)), so ist H *induzierte Teilgraph*.
Ein *Spannbaum* ist ein Teilgraph eines ungerichteten Graphen, der ein Baums ist und alle Knoten des Graphen enthält.

Vorteile von Listen

- Brauchen weniger Speicher, wenn $m \ll n$.
- Kann Algorithmen auf Graphen mit wenig (z.B. $O(n)$) Kanten beschleunigen.

Vorteile von Matrizen

- Schneller Test, ob eine Kante im Graph ist.
- Bei $\Omega(n^2)$ Kanten effizienter als Listen (weniger Overhead).

Sei $G = (V, E, \Psi)$ gerichtet. Der ungerichtete Graph $G' = (V, E, \Phi')$ mit $\Phi'(e) = \{u, v\}$ falls $\Psi(e) = (u, v)$ ist der G zugrunde liegende ungerichtete Graph.

Data: Array A der Länge n, partielle Ordnung $\preceq$
Result: A topologisch sortiert
for $i \leftarrow 0$ *to* $n - 2$ **do**
  for $j \leftarrow i + 1$ *to* $n - 1$ **do**
    if $A[j] \prec A[i]$ **then** `swap`$(A[i], A[j])$;

Im *best* und *worst case* $\binom{n}{2}$ Vergleiche, Laufzeit: $\Theta(n^2)$.

Stacks (Stapel) sind Listen, die Elemente hinten einfugen und entnehmen. Das letzte Element eines Stacks wird zuerst entnommen (LIFO – Last-in-First-out).
Queues (Warteschlange) sind Listen, die Elemente hinten einfugen und vorne entnehmen. Das erste Element einer Queue wird zuerst entnommen (FIFO – First-in-First-out).

$R \subset S \times S$ heißt *partielle Ordnung* auf einer Menge S, wenn für alle $a, b, c \in S$ gilt (wir schreiben $aRb := (a, b) \in R$) Reflexivität: aRa, Antisymmetrie: aRb und $bRa \implies a = b$ und Transitivität: aRb, $bRc \implies aRc$.
R ist *total* (oder *linear*), wen für alle $a, b \in S$ gilt: aRb oder bRa.
z.B. $S = \mathbb{N}$ und $R =$ teilt.
Sei $G = (V, E)$ gerichteter Graph ohne gerichtete Kreise. Dann ist die Relation „es existiert ein (gerichteter) Weg von v nach w“ eine partielle Ordnung auf V.

Data: Graph G, Knoten $r \in V$
Result: $R \subset V$: von r aus erreichbaren Knoten, $F \subset E$, sodass (R, F) Baum bzw. Aboreszenz mit Wurzel r ist.
$R, Q \leftarrow \{r\},\ \ F \leftarrow \varnothing$;
while $Q \neq \varnothing$ **do**
  sei v das nächstes Element in Q;
  if $\exists e = \{v, w\} \in \delta(v)$ *(bzw.* $(v, w) \in \delta^+(v)$*) mit* $w \in R$ **then**
    $R \leftarrow R \cup \{w\}$; $Q \leftarrow Q \cup \{w\}$; $F \leftarrow F \cup \{e\}$;
  else
    $Q \leftarrow Q \backslash \{v\}$;
return (R, F)

Realisierung von Q als Queue oder Stack. Speicherbedarf (in $|V| + |E|$) und Laufzeit linear (wenn Q geeignet realisiert).

DEFINITION

(Topologische) Sortierung

COMA

SORTIERALGORITHMUS

Insertionsort

COMA

Operation Merge

COMA

SORTIERALGORITHMUS

Mergesort

COMA

SATZ

Aufteilungs-Beschleunigungs-Satz

COMA

SATZ

Untere Schranke für Sortieralgorithmen

COMA

SORTIERALGORITHMUS

QuickSort

COMA

DEFINITION & ÜBERSICHT

stabil, in-place

COMA

DEFINITION

Alphabet, Wort, Zeichenkette

COMA

FRAGE

Gibt es zu jedem Berechnungsproblem einen Algorithmus, der es löst?

COMA

Data: Array A der Länge n, partielle Ordnung $\preceq$
Result: A sortiert
for $i \leftarrow 1$ *to* $n-1$ **do**
  $s \leftarrow A[i], \;\; k \leftarrow i$;
  while $k > 0$ *and* $s \prec A[k-1]$ **do**
    $A[k] \leftarrow A[k-1], \;\; k \leftarrow k-1$
  $A[k] \leftarrow s$

Insertionsort in *worst* und *average case* $\binom{n}{2}$, *best case* $n-1$) Vergleiche, Laufzeit: $\Theta(n^2)$. Ist $\preceq$ nur eine partielle Ordnung, so liefert Insertionsort keine topologische Sortierung, es gilt lediglich $A[0] \not\succ A[1] \not\succ \ldots \not\succ A[n-1]$.

Sei S eine n-elementige Menge und $\preceq$ eine partielle Ordnung auf S. Eine Bijektion $p\colon \{1,\ldots,n\} \to S$ ist ein *topologische Sortierung von S bzgl. $\preceq$*, wenn $p(j) \npreceq p(i)$ für $i < j$ gilt.
z.B. sind die topologischen Sortierung von $S := \{2,3,4\}$ bzgl. der Teilbarkeitsrelation $(2,3,4)$, $(2,4,3)$ und $(3,2,4)$.
Ist $\preceq$ eine totale Ordnung, so gilt $p(1) \preceq p(2) \preceq \ldots \preceq p(n)$ und p heißt *Sortierung* von S bzgl. $\preceq$.

Data: Array C der Länge n, totale Ordnung $\preceq$.
Result: Sortierung p der Elemente aus C.
Function `MergeSort`(C)**:**
  if $n = 1$ **then return** $1 \mapsto C[0]$;
  $A \leftarrow C[0], \ldots, C\left[\left\lfloor \frac{n}{2} \right\rfloor - 1\right]$;
  $B \leftarrow C\left[\left\lfloor \frac{n}{2} \right\rfloor\right], \ldots, C[n-1]$;
  $p_A \leftarrow$ `MergeSort`(A), $p_B \leftarrow$ `MergeSort`(B);
  return `Merge`(p_A, p_B)

best, *average* und *worst case*: $O(n\log(n))$.

Data: Sortierungen $p_A\colon \{1,\ldots,|A|\} \to A$ und für B analog.
Result: Sortierung $p\colon \{1,\ldots,|A|+|B|\} \to A \cup B$.
$a, b, c \leftarrow 1$;
while $a \leqslant |A|$ *and* $b \leqslant |B|$ **do**
  if $p_A(a) \succeq p_B(b)$ **then**
    $p(c) \to p_A(a), \;\; a \leftarrow a+1$
  else
    $p(c) \leftarrow p_A(b), \;\; b \leftarrow b+1$
  $c \leftarrow c+1$.
while $a \leqslant |A|$ **do**
  $p(c) \leftarrow p_A(a), \;\; a \leftarrow a+1, \;\; c \leftarrow c+1,$
while $b \leqslant |B|$ **do**
  $p(c) \leftarrow p_B(a), \;\; b \leftarrow b+1, \;\; c \leftarrow c+1,$

Merge benötigt $\leqslant |A|+|B|-1$ Vergleiche, Laufzeit: $O(|A|+|B|)$.

Jeder auf paarweisen Vergleichen basierende Algorithmus benötigt zum Sortieren einer n-elementigen Menge im worst case (sogar average) $O(n\log(n))$ Vergleiche.
Beweis. Entscheidungsbaum hat $\geqslant n!$ Blätter. Im worst case braucht man $\log_2(n!)$ Vergleiche (Höhe des Baums) $n\log_2(n) \geqslant \log_2(n!) \geqslant \left\lfloor \frac{n}{2} \right\rfloor \log_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$.

Sei $f\colon \mathbb{N} \to \mathbb{N}$ monoton wachsend und $a, b, c \in \mathbb{R}$ mit $a > 1$, $b, c > 0$ sodass $f(1) \leqslant \frac{c}{a}$ und die folgende rekursive Ungleichung gilt

$$f(an) \leqslant bf(n) + cn \quad \forall n \geqslant 1.$$

Dann gilt

$$f(n) \in \begin{cases} O(n), & \text{wenn } a > b, \\ O(n\log(n)), & \text{wenn } a = b, \\ O(n^{\log_a(b)}), & \text{wenn } a < b. \end{cases}$$

stabil: Elemente mit demselben Wert im Output in derselben Reihenfolge wie im Input auftauchen.
in-place: zusätzlich zum Eingabearray nur konstant viel Speicher benötigt.

	stabil	in-place		stabil	in-place
Select	ja	ja	**Radix**	ja	ja
Insert	ja / nein	ja	**Bucket**	ja	nein
Merge	ja	nein,	**Heap**	nein	ja
Quick	nein	(ja)	**Counting**	ja	nein

Im Gegensatz zu Merge-Sort ist Divide aufwendiger, Conquer einfacher.
Data: Nichtleere, total geordnete Menge $(S, \preceq)$
Result: Sortierung von S
if $|S| \leqslant 1$ **then return** $[s : s \in S]$. ;
else
  wähle $x \in S$ beliebig ;
  $S_1 \leftarrow \{s \in S\backslash\{x\} : s \preceq x\}, \;\; S_2 \leftarrow \{s \in S : s \npreceq x\}$;
  return `Quicksort`$(S_1) + [x] +$`Quicksort`(S_2)

best und *average case*: $n\log(n)$, *worst case* n^2 wenn pivot beliebig, mit Median $n\log(n)$.

Die Anzahl der python Programme ist abzählbar (da Σ endlich $\implies \Sigma^*$ abzählbar), aber die Menge der Probleme (als Obermenge von $\{f : \mathbb{N} \to \{0,1\}\} \cong \{0,1\}^{\mathbb{N}}$) ist überabzählbar; es gibt also mehr Probleme als Algorithmen!
Das Halteproblem (terminiert Algorithmus A bei Input x nach endlich vielen Schritten?) wird von keinem python-Programm gelöst.

Sei $\Sigma \neq \varnothing$ endlich. Es ist $\Sigma^k := \{f\colon \{1,\ldots,k\} \to \Sigma\} \cong (f(n))_{n=1}^k$ die Menge der *Wörter* der Länge k über dem *Alphabet* Σ. Eine Teilmenge $L \subset \Sigma^* := \bigcup_{k\in\mathbb{N}} \Sigma^k$ heißt *Sprache*.

Definition

Berechnungsproblem / Entscheidungsproblem

CoMa

Sortieralgorithmus

CountingSort

CoMa

Sortieralgorithmus

RadixSort

CoMa

Sortieralgorithmus

BucketSort

CoMa

Definition

Unabhängigkeitssystem & Matroid

CoMa

Definition

Unabhängige Menge, Basis, Kreis

CoMa

Satz

Basisaustauschsatz

CoMa

Definition und Satz

Graphische Matroide

CoMa

Algorithmische Charakterisierung von Matroiden

Optimierungsproblem über Unabhängigkeitssystemen

CoMa

Algorithmus

Kruskal

CoMa

Data: Array A der Länge n, $A[i] \in \{1, \ldots, k\}$.
Result: Sortiertes Array B.
C = array of length k; B = array of length n;
for $i \leftarrow 1$ *to* k **do** $C[i] \leftarrow 0$;
for $j \leftarrow 1$ *to* n **do** $C[A[j]] \leftarrow C[A[j]] + 1$;
for $i \leftarrow 1$ *to* $k-1$ **do** $C[i+1] \leftarrow C[i+1] + C[i]$;
for $j \leftarrow n$ *down to* 1 **do**
  $B[C[A[j]]] \leftarrow A[j]$, $C[A[j]] \leftarrow C[A[j]] - 1$.

Laufzeit $O(n+k)$, d.h. ist $k \in O(n)$, dann ist CountingSort linear. CountingSort ist stabil.

Ein Berechnungsproblem $P \subset D \times E$ ist eine *Relation*, sodass zu jedem $d \in D$ ein $e \in E$ mit dPe existiert.
Ist P eine Funktion, so ist das *Problem* P eindeutig. Ist zusätzlich $|E| = 2$, so ist P ein *Entscheidungsproblem*.

Data: Array A der Länge n mit $A[i] \in (0, 1]$.
Result: Sortiertes Array.
B = array of length n;
for $i = 1$ *to* n **do** $B[i] \leftarrow$ empty list.;
for $i = 1$ *to* n **do** insert $A[i]$ into list $B[\lceil n \cdot A[i] \rceil]$.;
for $i = 1$ *to* n **do** sort list $B[i]$ with InsertionSort.;
Concatenate lists $B[1], \ldots, B[n]$;

$A[i]$ unabh. + gleichverteilt, so ist BucketSort linear. Beliebige Zahlenbereiche anstatt $(0, 1]$ möglich, indem Bucketzuweisungen angepasst werden.

Data: Array A der Länge n, jede Eintrag ist d-stellige Zahl, 1. Stelle die niedrigste, d-te die höchste.
Result: Sortiertes Array.
for $i \leftarrow 1$ *to* d **do**
  sortiere A nach i-ter Stelle mit stabilem Sortierverfahren.

Für n Zahlen mit je d Stellen, bei denen jede Stelle bis zu k mögliche Werte annehmen kann, sortiert RadixSort diese Zahlen in $\Theta(d \cdot (n+k))$, falls das stabile Sortierverfahren $\Theta(n+k)$ besitzt.
Ist d konstant und $k \in O(n)$, so ist RadixSort linear.

Ist $(E, \mathcal{I})$ ein U-System, so heißen die Elemente in $\mathcal{I}$ *unabhängige Mengen* und diejenigen in $P(E) \backslash I$ *abhängig*.
Eine *Basis* ist eine (bzgl. Inklusion) max. unabhängige Menge.
Ein *Kreis* ist eine (bzgl. Inklusion) min. abhängige Menge.
Für $M \subset E$ nennt man die maximal unabhängigen Teilmengen von M *Basen von* M.
Der *Rang* von $M \subset E$ ist die maximale Kardinalität einer Basis von M.
Der *Rang von* E ist der Rang von $(E, \mathcal{I})$.

Für E endlich, $\mathcal{I} \subset P(E)$ ist $(E, \mathcal{I})$ ein Unabhängigkeitssystem mit Grundmenge E, wenn $\varnothing \in \mathcal{I}$ und aus $J \in \mathcal{I}$ und $I \subset J$ auch $I \in \mathcal{I}$ folgt.
Gilt zusätzlich: falls $I, J \in \mathcal{I}$ und $|I| < |J|$, so existiert ein $e \in J \backslash I$, sodass $I \cup \{e\} \in \mathcal{I}$, so ist $(E, \mathcal{I})$ ein *Matroid*.
Beispiele für Matroide: E = Zeilen (oder Spalten) einer Matrix, $\mathcal{I} = \{$ linear unabhängige Teilmengen $\}$. E endlich, $k \in \mathbb{N}_{>0}$, $\mathcal{I} := \{I \subset E : |I| < k\}$.

Sei $G = (V, E)$ ein ungerichteter Graph (möglicherweise mit Schleifen o. Mehrfachkanten)
Eine Teilmenge $I \subset E$ ist *unabhängig*, wenn der Teilgraph (V, I) von G ein Wald (d.h. kreisfrei) ist.
Das Paar $(E, \mathcal{I})$ ist ein Matroid.

Ein U-System $(E, \mathcal{I})$ ist genau dann ein Matroid wenn für jede Teilmenge $M \subset E$ *alle Basen von* M *dieselbe Kardinalität* haben.
Korollar (Basisaustauschsatz). Seien B_1 und B_2 Basen des Matroids $(E, \mathcal{I})$ und $e_1 \in B_1$. Dann existiert ein $e_2 \in B_2$, sodass $(B_1 \backslash \{e_1\}) \cup \{e_2\}$ eine Basis ist.
ACHTUNG: Kreise können unterschiedliche Kardinalität haben.

Data: $G = (\{v_i\}_{i=1}^n, \{e_i\}_{i=1}^m)$ zsmh., $w \colon E \to \mathbb{R}_{\geqslant 0}$.
Result: MST in G.
sortierte Kanten: $w(e_1) \leqslant \ldots \leqslant w(e_n)$;
$F := \{\{v_1\}, \ldots, \{v_n\}\}$, $B \leftarrow \varnothing$;
for $i \leftarrow 1$ *to* m **do**
  if e_i *verbindet* $C, D \in F$ $(C \neq D)$ **then**
    $F \leftarrow (F \backslash \{C, D\}) \cup \{C \cup D\}$, $B \leftarrow B \cup \{e_i\}$
return B

G darf Schleifen und Mehrfachkanten enthalten. Naiv: $O(m \times n)$. Mit UnionFind: $O(m \log(n))$.

Data: Unabhängigkeitssystem $(E, \mathcal{I})$, $w \colon E \to \mathbb{R}_{\geqslant 0}$.
Result: [Basis] $X \in \mathcal{I} : w(X) := \sum_{e \in X} w(e)$ max [min].
sortiere $E = \{e_1, \ldots, e_n\} : w(e_1) \geqslant \ldots \geqslant w(e_n)$ [$\leqslant$];
$X \leftarrow \varnothing$;
for $i \leftarrow 1$ *to* n **do**
  if $(X \cup \{e_i\}) \in \mathcal{I}$ **then** $X \leftarrow X \cup \{e_i\}$;
return X

Unabhängigkeitssystem $(E, \mathcal{I})$ ist ein Matroid $\iff$ Greedy-Algorithmus liefert optimale Lösung für alle $w \colon E \to \mathbb{R}_{\geqslant 0}$.

ALGORITHMUS

PRIM

COMA

DEFINITION

Minimal aufspannender Baum (MST)

COMA

ALGORITHMUS

Euklidischer Algorithmus

COMA

ALGORITHMUS

Tiefensuche (rekursiv)

COMA

DEFINITION

HASSE-Diagramm

COMA

SORTIERALGORITHMUS

Heapsort

COMA

ALGORITHMUS

Build-Max-Heap

COMA

COMA

COMA

COMA

Sei $G = (V, E)$ ein endlicher zusammenhängender Graph (möglicherweise mit Schleifen / Mehrfachkanten) und $w \colon E \to \mathbb{R}_{\geqslant 0}$. Ein aufspannender Baum (d.h. bzgl. Inklusion maximal kreisfreier oder minimal zusammenhängender Teilgraph) (V, B) heißt minimal bzgl. w, wenn $w(B) := \sum_{e \in B} w(e)$ minimal ist.
Ist w strikt positiv, so ist ein MST auch ein aufspannender zusammenhängender Teilgraph minimalen Gewichts.

Data: $G = (\{v_i\}_{i=1}^n, \{e_i\}_{i=1}^m)$ zsmh, $w \colon E \to \mathbb{R}_{\geqslant 0}$.
Result: MST in G.
wähle $v_0 \in V$, $U \leftarrow \{v_0\}$, $B \leftarrow \varnothing$;
while $|U| \leqslant |V|$ **do**
| finde $e = \{u, v\} \in E$ minimalen Gewichts, sodass $u \in U$, $v \in V \backslash U$;
| $U \leftarrow U \cup \{v\}$, $B \leftarrow B \cup \{e\}$;
return B

G darf Schleifen und Mehrfachkanten enthalten. Naiv: $O(m \times n)$. Mit MinHeaps: $O(m \log(n))$ bzw. sogar $O(m + n \log(n))$.

Data: Startknoten *start* und gesuchter Knoten *goal*
Result: True, wenn *goal* und *start* in derselben Zusammenhangskomponente sind.
Function `tiefensuche`(*start, goal*)**:**
| markiere(*start*);
| **for** v *in Adjazenzliste von start* **do**
| | **if** v *nicht markiert* **then** markierte v;
| | **if** v = *goal* **then return** True;
| | **if** *tiefensuche(v, goal)* **then return** True;

Data: $a, b \in \mathbb{N}_{>0}$
Result: $\mathrm{ggT}(a, b)$
while $b \neq 0$ **do**
| **if** $a > b$ **then**
| | $a \leftarrow a - b$
| **else**
| | $b \leftarrow b - a$
return a

Worst / average case $\Theta(n \log(n))$, Best-Case: $\mathcal{O}(n)$, in-place.
`Build-Max-Heap`(A)
while $n > 1$ **do**
| exchange $A[1]$ with $A[n]$
| $n \leftarrow n - 1$
| `Max-Heapify`$(A, 1)$

Sei $(S, \preceq)$ ein Partialordnung und $a \prec b \iff a \preceq b \wedge a \neq b$. Ist G endlich, so nennt man den gerichteten Graphen

$$G = (S, \{(x, y) : x \prec y \ \wedge \ \nexists z : x \prec z \prec y\})$$

das Hasse-*Diagramm* von $(S, \preceq)$.
Hasse-Diagramme sind azyklisch und somit besitzen endliche Mengen S ein (nicht eindeutiges) Min und Max (sonst hätte alle Knoten $|\delta^+(v)| \geqslant 1$ und dann gäbe es einen Kreis).
Hasse-Diagramm auf einem gerichteten Graph ohne gerichtete Kreise bzgl. der Partialordnung „es gibt eine Weg von v nach w" ist ein Spannbaum (??).

Laufzeit: linear.
for $i = \left\lfloor \frac{n}{2} \right\rfloor$ *down to 1* **do**
| `Max-Heapify`(A, i)

Definition

Eindeutig dekodierbar

CoMa

Definition

Präfixcode

CoMa

Definition

Blockcode

CoMa

Definition

Optimaler Präfixcode

CoMa

Algorithmus

Huffman

CoMa

Laufzeiten im Heap

CoMa

Algorithmus

RotateRight

CoMa

Algorithmus

RotateLeft

CoMa

RotateLR und RotateRL

CoMa

Definition

Balance, (extremaler) AVL-Baum

CoMa

Ein Code heißt **Präfixcode**, wenn kein Codewort als Präfix eines anderen Codeworts auftaucht.

a	b	c	d	e
00	01	100	11	101

ist Präfixcode, jedoch ist

a	b	c	d	e	f	g	␣
0	1	10	11	100	101	110	111

keiner.

Präfixcodes sind eindeutig dekodierbar und lässt sich eindeutig mit einem binären Baum identifizieren: Codewort eines Zeichens entspricht Weg von der Wurzel zum Blatt (0 = links, 1 = rechts).

Ein Code heißt **eindeutig dekodierbar**, falls verschiedene Originaldateien zu verschiedenen kodierten Dateien führen (injektive Kodierung).
Blockcodes sind eindeutig decodierbar.

a	b	c
1	10	00

ist eindeutig dekodierbar, jedoch ist

a	b	c	d	e	f	g	␣
0	1	10	11	100	101	110	111

es nicht.

Sei `c.key` die Häufigkeit eines Zeichens $c \in C$ in der zu kodierenden Datei.
Für einen Präfixcode (mit zugehörigem Binärbaum) T ist die Größe (# bits) der kodierten Datei $B(T) = \sum_{c \in C} \texttt{c.key} \cdot d_T(c)$, wobei $d_T(c)$ die Tiefes des Blatts c im Baum T (= Länge des Codeworts) ist.
Ein Präfixcode T mit $B(T) \leqslant B(T')$ für alle Präfixcodes T' heißt **optimal**.

Ein Code heißt **Blockcode**, wenn alle Zeichen als 0−1-Strings fester Länge kodiert werden. Vorteil: einfache (De)Kodierung. Blockcode ist eindeutig dekodierbar.

Beispiel:

a	b	c	d	e	f	g	␣
000	001	010	011	100	101	110	111

Für einen Präfixcode T für C sei $M(T)$ die maximale Länge eines Codeworts in T. Es gibt einen Blockcode T_B für C mit $M(T_B) \leqslant M(T)$ für alle Präfixcodes T für C. **(TODO: PROOF!!)**

ExtractMin / ExtractMax, Insert: $\log(n)$.

Q ist ein **min-Heap**.

Data: Zeichensatz C mit Häufigkeiten `c.key`, $c \in C$
Result: Optimaler Präfixcode T
for $c \in C$ **do** `Insert`(Q, c)
while Q *enthält mehr als einen Baum* **do**
  $T_1 \leftarrow$ `ExtractMin`(Q); $T_2 \leftarrow$ `ExtractMin`(Q)
  sei T Baum mit linkem TB T_1 und rechtem TB T_2
  T`.key` $= T_1$`.key` $+T_2$`.key`
  `Insert`(Q, T)
return *ExtractMin*(Q)

$\mathcal{O}(2(2n-1)\log(n)) = \mathcal{O}(n\log(n))$.

```
y = x.rightChild
if x.parent == None then  self.root = y
else if x.parent.leftChild == x then
 | x.parent.leftChild = y
else x.parent.rightChild = y
y.parent = x.parent;  x.rightChild =
 y.leftChild
if x.rightChild then  x.rightChild.parent = x
y.leftChild = x;  x.parent = y
```

Algorithm 1: rotate_left(self,x)

```
y = x.leftChild
if x.parent == None then self.root = y
else if x.parent.leftChild == x then
 | x.parent.leftChild = y
else x.parent.rightChild = y
y.parent = x.parent;  x.leftChild =
 y.rightChild
if x.leftChild then x.leftChild.parent = x
y.rightChild = x;  x.parent = y
```

Sei T ein Binärbaum, v ein Knoten von T und T_r, T_ℓ die rechten bzw. linken Teilbäume unterhalb von v. $\beta(v) := h(T_r) - h(T_\ell)$, $h(\varnothing) = -1$ ist die **Balance** von v.
Ein binärer Suchbaum T heißt **AVL-Baum**, falls $|\beta(v)| \leqslant 1$ („Knoten v ist balanciert“) für jeden Knoten $v \in T$. In jedem Knoten v eines AVL-Baums ist die Balance $\beta(v)$ gespeichert
Ein AVL-Baum der Höhe h heißt **extremal**, wenn er $\min(\{n :$ es gibt einen AVL-Baum der Höhe h mit n Knoten $\})$ Knoten enthält. Es ist $n(0) = 1$, $n(1) = 2$ und $n(h+2) = n(h+1)+n(h)$ („Fibonacci-Bäume“). Es gilt $n(h) \geqslant 2^{\frac{h}{2}}$, d.h. log. Höhe.

```
self.rotate_left(x.leftChild)
self.rotate_right(x)
```

Algorithm 2: rotate_LR(self,x)

```
self.rotate_right(x.rightChild)
self.rotate_right(x)
```

Algorithm 3: rotate_RL(self,x)

Dijkstra

CoMa

Splay Bäume

CoMa

Bellman-Ford

CoMa

CoMa

CoMa

CoMa

CoMa

CoMa
