Theorie

Einheitskostenmass $\leftrightarrow$ Log. Mass (ab Ch. 5), #Bits um n zu speichern $= \lceil \log_2(n+1) \rceil$

$f(n) = \Omega(g(n)) \Leftrightarrow \liminf \left|\frac{f(n)}{g(n)}\right| > 0$, $f(n) = \omega(g(n)) \Leftrightarrow \frac{|f(n)|}{|g(n)|} \to \infty$, $f(n) = o(g(n)) \Leftrightarrow \lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$.

$f(n) = O(g(n)) \Leftrightarrow \limsup \left|\frac{f(n)}{g(n)}\right| < \infty$, $f(n) = \Theta(g(n)) \Leftrightarrow f = O(g)$ & $f = \Omega(g)$

1.4 Graphenalgorithmen

- $\sum \deg(v) = 2|E|$
- $\forall G(V,E)$: Komponenten $\geq |V| - |E|$ (Ind. über n)
- $\forall G(V,E)$ zsmh.: $|E| \geq |V| - 1$
- $\forall$ Bäume $T = (V,E)$ mit $|V| \geq 2$: $\exists$ min 2 Blätter
- $T = (V,E)$ Baum $|V| \geq 2$, $u \in V$ Blatt $\Rightarrow T[V \setminus \{u\}]$ Baum
- $G = (V,E)$ zsmh. C Kreis $\forall e \in C \Rightarrow G_e = (V, E \setminus \{e\})$ zsmh.
- $G = (V,E)$ zsmh, $E = |V| - 1 \Rightarrow G$ Baum

$\to$ Adjezanzmatrix $O(n^2)$, Adjezanzliste $O(n+m)$

BREITENSUCHE

$\Theta(n^2)$ (Admatrix)
$\Theta(n+m)$ (Adjezenzliste)

Eingabe: Graph $G = (V,E)$, Startknoten $s \in V$
Ausgabe: Felder $d[v]$, $pred[v]$ mit $v \in V$

$\forall v \in V$ dobegin
  if $v = s$ then $d[v] \leftarrow 0$ else $d[v] \leftarrow \infty$;
  $pred[v] \leftarrow nil$;
end
$Q \leftarrow$ new Queue;
Q.Insert(s)
whilenot Q.isEmpty() do begin
  $v \leftarrow Q$.Dequeue();
  for all $u \in \Gamma(v)$ do
    if $d[u] = \infty$ then begin
      $d[u] \leftarrow d[v] + 1$;
      $pred[u] \leftarrow v$;
      Q.Insert(u)
    end
end

$d[v]$: shortest s-v path

G zsmh. $\{v, pred[v]\}$ bilden Spannbaum

Anzahl kürz. s-v pfade.
(Initialize $a[v] = 0, \forall v$, $a[s] = 1$.

if $d[u] = d[v] + 1$ then
  $a[u] = a[u] + a[v]$
end if

BERECHNUNG ZSMH-KOMPONENTE

Eingabe: $G = (V,E)$
Ausgabe: Feld $comp[v]$ (Nummern der Zsmhg. Komp)

$\forall v \in V$: $comp[v] \leftarrow 0$
$Q \leftarrow$ new Queue;
$i \leftarrow 0$;
repeat
  wähle bel. Knoten $s \in V$ mit $comp[s] = 0$;
  $i \leftarrow i + 1$
  $comp[s] \leftarrow i$;
  Q.insert(s);
  whilenot Q.isEmpty() dobegin
    $v \leftarrow Q$.dequeue();
    $\forall u \in \Gamma(v)$ do
      if $comp[u] = 0$ then begin
        $comp[u] \leftarrow comp[v]$;
        Q.Insert(u)
      end
  end
until $comp[v] > 0$, $\forall v \in V$

TIEFENSUCHE (DFS)

$\Theta(n^2)$ (AdMatrix)
$\Theta(n+m)$ (Adj.liste)

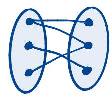
Eingabe: Graph $G = (V,E)$, Startknoten $s \in V$
Ausgabe: Feld $pred[v]$ mit $v \in V$.

for all $v \in V(G)$ do $pred[v] = nil$;
$S \leftarrow$ new STACK
$v \leftarrow s$;
repeat
  if $\exists u \in \Gamma(v) \setminus \{s\}$ mit $pred[u] = nil$ then
    S.PUSH(v);
    $pred[u] \leftarrow v$;
    $v \leftarrow u$;
  elseifnot S.ISEMPTY() then
    $v \leftarrow$ S.POP();
  else
    $v \leftarrow nil$;
until $v = nil$;

$\{v, pred[v]\}$ $\forall v \in V \setminus \{s\}$ bildet Spannbaum (falls G zsmhg)

BIPART-CHECK

(modded BFS) $\Theta(|V| + |E|)$



$\forall v \in V$ do
  $c[v] = nil$
end for
Q = newQueue
repeat
  wähle bel. $s \in V$ mit $c[s] = nil$
  $c[s] = red$
  Q.Insert(s)
  whilenot Q.IsEmpty() do
    $v = Q$.Dequeue()
    for all $u \in \Gamma(v)$ do
      if $c[u] = c[v]$ then
        return "G ist nicht bipartit"
      end if
      if $c[v] = red$ then
        $c[u] = blue$
      else
        $c[u] = red$
      end if
      Q.Insert(u)
    end for
  end while
until $c[v] \neq nil$, $\forall v \in V$
return "G ist bipartit"

	Mat	Liste	erw. Liste
find $\deg(v)$	$\Theta(n)$	$\Theta(\deg(v))$	$\Theta(1)$
find bel. $u \in \Gamma(v)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
$\{u,v\} \in E$?	$\Theta(1)$	$\Theta(\min\{\deg(u), \deg(v)\})$ *	=
delete $\{u,v\} \in E$	$\Theta(1)$	$\Theta(\deg(u) + \deg(v))$	"
add $\{u,v\}$ to E	$\Theta(1)$	$\Theta(1)$ if we know $\notin E$ else *	=
delete v und inz. Kanten	$\Theta(n^2)$	$\Theta(\sum_{u \in \Gamma(v)} \deg(u))$ worst case $\Theta(m)$	$\Theta(\deg(v))$

FLOYD-WARSHALL (All-to-All shortest path) $\mathcal{O}(n^3)$ (mod)

Eingabe: zsmhg. Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und $V = \{1, \dots, n\}$

Ausgabe: Längen aller kürzester i-j Pfade in N, $\forall i, j \in V$

{Initialisierung}

$$\forall i, j \in V : F_0[i,j] := \begin{cases} \ell(\{i,j\}) & \text{falls } \{i,j\} \in E \\ 0 & \text{falls } i = j \\ \infty & \text{sonst} \end{cases}$$

{Rekursion}

for $k = 1$ to n do

$\quad \forall i, j \in V : F_k[i,j] = \min\{F_{k-1}[i,j], F_{k-1}[i,k] + F_{k-1}[k,j]\}$

{Ausgabe}

falls neg. Dann gerichtet

if $(\exists i \in V$ mit $F_n[i,i] < 0)$ then

$\quad$ return "Graph enthält neg. Kreis"

else

$\quad F_n[i,j]$ gibt die Länge eines kürzesten i-j Pfades an

2.6 Minimal spannende Bäume

PRIM $\mathcal{O}(n^2)$, $\mathcal{O}(n \log(n) + m)$ mit F-Heaps

Eingabe: zsmh. Netzwerk $N = (V, E, \ell)$

Ausgabe: min. Spannbaum $T = (V, F) \subset N$

Wähle $s \in V$ beliebig;

$W := \{s\}$; $F := \emptyset$;

for all $v \in V \setminus (\{s\} \cup \Gamma(s))$ do $\sigma[v] := \infty$

for all $v \in \Gamma(s)$ do $\sigma[v] := \ell(s, v)$; pred$[v] := s$;

while $W \neq V$ do

$\quad$ Finde $x_0 \in V \setminus W$ s.d $\sigma[x_0] = \min\{\sigma[v] \mid v \in V \setminus W\}$;

$\quad W := W \cup \{x_0\}$; $F := F \cup \{\{x_0, \text{pred}[x_0]\}\}$;

$\quad$ for all $v \in \Gamma(x_0) \cap (V \setminus W)$ s.d $\sigma[v] > \ell(v_0, v)$ do

$\quad\quad \sigma[v] := \ell(x_0, v)$; pred$[v] := x_0$;

KRUSKAL $\mathcal{O}(n \cdot m)$, $\mathcal{O}(m \log(n))$ Union-Find

Eingabe: zsmh. Netzwerk $N = (V, E, \ell)$

Ausgabe: min. Spannbaum $T = (V, F) \subset N$

Sortiere Kanten s.d: $\ell(e_1) \leq \dots \leq \ell(e_m)$

Setze $F := \emptyset$

for $i := 1$ to m do

$\quad$ if $(V, F \cup \{e_i\})$ kreisfrei then $F := F \cup \{e_i\}$

KRUSKAL

Eingabe: ein zsmh. Netzwerk $N = (V, E, \ell)$

Ausgabe: ein min Spannbaum $T = (V, F) \subset N$

Sortiere Kanten s.d: $\ell(e_1) \leq \dots \leq \ell(e_m)$

Setze $F := \emptyset$

for $i := 1$ to m do

$\quad$ Sei $e_i = \{x, y\}$;

$\quad$ if Find$(x) \neq$ Find(y) then

$\quad\quad F := F \cup \{e_i\}$;

$\quad\quad$ Union(Find(x), Find(y));

$\mathcal{O}(m \log(n) + n[\langle$Laufz. MakeNewSet$\rangle + \langle$Laufz Union$\rangle] + m\langle$Laufz Find$\rangle)$

3. Algorithmische Grundprinzipien

3.1 Divide & Conquer Algos.

MergeSort
QuickSort
Binäre Suche

$c =$

Master-Theorem

Seien $\alpha \geq 1$, $\beta > 1$ und $C \geq 0$ Konstanten, $f(n) > 0$

$c_1(n), \dots, c_\alpha(n)$ mit $|c_i(n)| \leq C$, $\forall 1 \leq i \leq \alpha$, $n \in \mathbb{N}$.

Ist $T(n)$ eine Funktion mit $T(0) = 1$, s.d $\forall n$:

$T(n) = T(n/\beta + c_1(n)) + \dots + T(n/\beta + c_\alpha(n)) + f(n)$ erfüllt.

$$\Rightarrow T(n) = \begin{cases} \Theta(n^{\log_\beta \alpha}) & , f(n) = \mathcal{O}(n^{\log_\beta \alpha - \varepsilon}), \varepsilon > 0 \\ \Theta(f(n) \log(n)) & , f(n) = \Theta(n^{\log_\beta \alpha} (\log n)^\delta), \delta > 0 \\ \Theta(f(n)) & , f(n) = \Omega(n^{\log_\beta \alpha + \varepsilon}), \varepsilon > 0 \end{cases}$$

3.2 Dynamisches Programming

KNAPSACK-PACKING $\mathcal{O}(n^2 \cdot \max_{i \in [n]} p_i)$

Eingabe: $n, w_1, \dots, w_n, p_1, \dots, p_n, B$

Ausgabe: $\max\{\sum_{i \in I} p_i \mid I \subset [n], \sum_{i \in I} w_i \leq B\}$

$p \leftarrow \sum_{i=1}^{n} p_i$

for t from 1 to p_1 do $f[1,t] \leftarrow w_1$;

for t from $p_1 + 1$ to p do $f[1,t] \leftarrow \infty$;

for i from 2 to n dobegin

$\quad$ for t from 1 to p dobegin

$\quad\quad$ if $t \leq p_i$ then

$\quad\quad\quad f[i,t] \leftarrow \min\{f[i-1,t], w_i\}$;

$\quad\quad$ else

$\quad\quad\quad f[i,t] \leftarrow \min\{f[i-1,t], w_i + f[i-1, t-p_i]\}$;

$\quad$ end

end

return $\max\{t \mid f[n,b] \leq B\}$;

3.3 Greedy Algos

Def Matroid $M = (S, \mathcal{U})$, S endliche Menge

$\mathcal{U}$ unabh. Mengen von Teilmengen aus S

1) $\emptyset \in \mathcal{U}$

2) $A \in \mathcal{U}, B \subseteq A \Rightarrow B \in \mathcal{U}$

3) $A, B \in \mathcal{U}, |B| = |A| + 1 \Rightarrow \exists x \in B \setminus A$ s.d $A \cup \{x\} \in \mathcal{U}$.

A heisst Basis falls Inklusionsmaximal.

GREEDY-ALGO FÜR MATROIDE

Eingabe: Matroid $M = (S, \mathcal{U})$, Gewfkt. $w : S \to \mathbb{Z}$

Ausgabe: Basis A mit min. Gewicht.

$A := \emptyset$

while A ist keine Basis von M dobegin

$\quad X = \{x \in S \setminus A \mid A \cup \{x\} \in \mathcal{U}\}$

$\quad$ wähle $x_0 \in X$, s.d $w(x_0) = \min_{x \in X} w(x)$

$\quad A = A \cup \{x_0\}$

end

4.1 Suchbäume

info nur an Blättern

(a,b) Baum ist ein externer Suchbaum, bei dem alle Blätter dieselbe Tiefe haben und:

- $a \le \#\text{Knoten} \le b$, $\forall$Knoten ausser Wurzel
- $2 \le \#\text{Knoten} \le b$ bei Wurzel
- $b \ge 2a-1$, $a \ge 2$
- $\forall$ Knoten v bezeichne $\rho(v)$ die Anzahl Kinder von v
- An jedem inneren Knoten v gibt es $\rho(v)-1$ Schlüssel $K_1, .., K_{\rho(v)-1}$ s.d. $K_{i-1} <$ (Schlüssel im i-ten Unterbaum von v) $\le K_i$, $K_0 = -\infty$, $K_{\rho(v)} = +\infty$
- unterstütz: Insert, Delete, various Find.

Sei T ein (a,b)-Baum mit n Blättern und Höhe h
$\Rightarrow$ i) $2 \cdot a^{h-1} \le n \le b^h$, ii) $\log_b(n) \le h \le 1 + \log_a(\frac{n}{2})$

Find(k,T): $\mathcal{O}(\log(n))$

v = Wurzel von T
while (v nicht Blatt) do
  $i = \min\{j \mid 1 \le j \le \rho(v)$ und $K_j \ge k\}$
  v = i-tes Kind von v.
if key(v) = k
  then return u
else return nil

Insert(x,T) $\mathcal{O}(\log(n))$

1. Find(x) findet kleinsten Schlüssel w, der grösser als x ist.
2. Füge links von w x ein, $\rho(v)$ erhöht sich
3. Falls $\rho(v) > b$:
   while $\rho(v) > b$ do
     if ($v \ne$ Wurzel) then u = Vater von v
     else u := neue Wurzel mit einzigem Kind v;
     Zerteile v in v_1, v_2 und mache v_1 zu Vater der $\lceil \frac{b+1}{2} \rceil$ kleinsten Kindern und v_2 vom Rest;
     v = u; (Falls u nun Grad $\rho(u) > b$ hat)

Delete(x,T) $\mathcal{O}(\log(n))$

1. Find
2. Lösche, falls vorhanden
3. Wird $\rho(v) < a$: Falls $\exists$Nachbar mit Grad $> a$ übernehme eins. Sonst verschmelze mit Nachbar
   $\Rightarrow \rho(\text{Vater}) < a$ maybe (check).

4.2 Mengen / Union-Find Strukturen

disjunkt. Mengen, Bäume, Wurzel, Repräsentant

MakeNewSet(x)	$\mathcal{O}(1)$
Find(x)	$\mathcal{O}(\log(n))$
Union(r,s)	$\mathcal{O}(1)$

- Union-Find-Struktur Baum T: size $T \ge 2^{\text{height}(T)}$
- leere Datenstrukt. k Operation, davon n MakeNewSet
  $\Rightarrow \mathcal{O}(k \underbrace{\log^*(n)}_{\le 6})$ (Mit Path compression)

4.3 Wörterbücher

Hash Funktion h: Data$_{,N} \to$ Speicher$_{,m}$
Ziel $|h^{-1}(i)| = \frac{N}{m}$, $\forall i \in$ Speicher
Menge von Hash $\mathcal{H}$ heisst universell falls
$Pr_{h \in \mathcal{H}}[h(x) = h(y)] = \frac{|\{h \in \mathcal{H} \mid h(x) = h(y)\}|}{|\mathcal{H}|} \le \frac{1}{m}$, $\forall x, y \in$ Data, $x \ne y$

Satz $\mathcal{H}$ universell, $\forall x \in$ Data: $h \in \mathcal{H}$ zufällig, so ist die erwartete Erwartete Anzahl Kollisionen < 1

4.4 Vorrangwarteschlange

Insert, Extract-min, Decrease-Key

(List or Feld)	sorted	unsorted	Suchbaum
Insert	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Extract-Min	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
Decrease-Key	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$

Fibonacci-Heaps ← effiziente Struktur

Sammlung von geordn. Bäumen
(key(v) ≤ key[Kinder(v)]), Wurzelliste, min[H]
Info bei Knoten: • Zeiger auf Kindern & Eltern
  • key(x), rank(x), marked(x) — lost kid or no

Insert(H,x)
parent[x] = nil; child[x] = nil;
marked(x) = false; rank(x) = 0;
add x to Wurzellist, update min[H]
n[H] = n[H]+1;

Extract-min(H)
z = min[H];
if z ≠ nil then
  for jedes Kind x von z do
    parent(x) = nil; marked(x) = false;
    add x to Wurzelliste;
  entferne z aus Wurzelliste;
  n[H] = n[H] − 1
  if Wurzelliste = ∅
    then min[H] = nil
  else Consolidate(H);
return z

Consolidate(H)
for i = 0 to $\lfloor \log_{3/2} n[H] \rfloor$ do
  A[i] = nil;
repeat
  Entferne bel. Knoten x aus Wrzlliste
  while A[rank(x)] ≠ nill do
    y = A[rank(x)]; A[rank(x)] = nil;
    if key[x] > key[y] then vertausche x↔y;
    mache y zum Kind von x, erhöhe rank[x];
  A[rank(x)] = x;
until Wurzelliste von H leer ist;
min[H] = nil;
for i = 0 to $\lfloor \log_{3/2} n[H] \rfloor$ do
  if A[i] ≠ nil then füge A[i] to Wrzllist, update min[H];

Decrease-Key(H,x,k)
if k ≥ key[x] then error
key[x] = k;
if parent(x) ≠ nil und key(x) < key[parent(x)] then
  Cut(H,x)
elseif parent(x) = nil then update min[H];

Cut(H,x)

```
y = parent(x)
parent(x) = nil, marked(x) = false
entferne x aus Kinderliste von y, reduce rank(y);
Add x to Wrzellist and update min(H);
if parent(y) ≠ nil then
    if marked(y) = true then
        Cut(H,y);
    else
        marked(y) = true;
```

- x mit Kinder $y_1, \dots, y_k$ (y_1 frühstens)
  $\Rightarrow \text{rank}(y_i) \geq i-2$
- x bel. , $k = \text{rank}(x) \Rightarrow$ Teilbaum mit Wurzel x mind F_{k+2} viele Elemente.
- $n(H) \leq n \Rightarrow \forall x \in H : \text{rank}(x) \leq \log_\varphi n$, $\varphi = \frac{1+\sqrt{5}}{2}$

Satz leerer Heap. k Operationen, davon ℓ Extract-min, n max. Anzahl Elemente die irgendwann im Heap waren.

5. Berechenbarkeit

Sprache $L \subset \{0,1\}^*$ heisst berechenbar falls $\exists$ Programm A s.d $A(x) = L(x)$, $\forall x \in \{0,1\}^*$

$$H(A,x) = \begin{cases} 1 & , A \text{ ein Prog. und } A(x) \text{ endl. Laufzeit} \\ 0 & \text{sonst.} \end{cases}$$

- Sei V widerspruchfrei und s.d $\forall A, x$ für die $A(x)$ hält, die Aussage $s = [A(x) \text{ hält}]$ V-bew.bar ist. $\Rightarrow \exists A, x$ s.d $A(x)$ nicht hält und ($A(x)$ hält nicht) nicht V-beweisbar ist.

proof: Prog. H^* berechnet $s = [A(x) \text{ hält}]$ und $(\neg s)$. Es führt dann, für jedes $p \in \{0,1\}^*$, $V(s,p)$ und dann $V(\neg s, p)$ aus. If $V(s,p) = 1 \Rightarrow H^* \to 1$, falls $V(\neg s, p) = 1 \Rightarrow H^* \to 0$
Falls $A(x)$ hält $\Rightarrow \exists p$ s.d $V(s,p) = 1 \Rightarrow H^* \to 1$.
Falls nicht $\Rightarrow \exists p$ s.d $V(\neg s, p) = 1 \Rightarrow H^* \to 0 \quad \Rightarrow H^* = H$ ✓

- Ein Axiomatisches System die Widerspr. freiheit desselben Systems nicht beweisen kann (ausser wenn das System widersprüchlich ist)

5.3 Die Klasse P

Def Menge aller Sprachen $L \subseteq \{0,1\}^*$, für welche es ein $c \in \mathbb{N}$ und Prog. A gibt s.d $\forall x \in \{0,1\}^*$ $A(x) = L(x)$ gilt und $A(x)$ höchstens Laufzeit $|x|^c + c$ (im log. mass)

Bsp:
- Zsuhs Graphen : BFS, DFS
- Durchmesser: Menge aller Paare (G,b), G Netzwerk in dem alle Knotenpaare (s,t) mit einem Pfad länge $\leq b$ verbunden sind
- Menge der Primzahlen
- Menge $G = (V,E)$ s.d $\exists E' \subseteq E$ s.d jeder Knoten genau in einer Kante $\in E'$ enthalten ist.

Ziel: $f : \{0,1\}^* \to \{0,1\}$ speichern

Satz: $\forall f \; \exists$ Formel F mit höchstens $3 \cdot 2^n$ Operatoren $(\neg, \wedge, \vee)$ welche f berechnet.

DNF: $F = D_1 \vee \dots \vee D_m$, $D_i = y_{i_1} \wedge \dots \wedge y_{i_k}$
KNF: $F = C_1 \wedge \dots \wedge C_m$, $C_i = y_{i_1} \vee \dots \vee y_{i_k}$
F ist k-KNF falls C_i höchstens k Literale enthält.

5.4 Die Klasse NP

Def NP enthält alle Sprachen $L \subseteq \{0,1\}^*$, für welche es ein $c \in \mathbb{N}$ und $L' \in P$ gibt s.d
$x \in L \Leftrightarrow \exists w \in \{0,1\}^* : (x,w) \in L'$ & $|w| \leq |x|^c + c$
(w: Zeugen)

5.6 Reduktionen

L ist polynomiell auf L' reduzierbar ($L \leq_p L'$) falls $\exists f : \{0,1\}^* \to \{0,1\}^*$ mit $x \in L \Leftrightarrow f(x) \in L'$, f polytime

3-SAT $\leq_p$ 3-COL

5.7 NP-Vollständigkeit

$L \subseteq \{0,1\}^*$ NP schwer, falls $\forall L' \in NP$ gilt: $L' \leq_p L$
Falls $L \in NP \Rightarrow L$ NP vollständig.

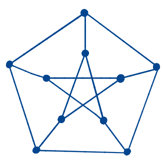
Bsp 3-SAT, Nullstelle mod — Literal $L \to P_L = \begin{cases} x_i & , L = x_i \\ (1-x_i) & , L = \bar{x}_i \end{cases}$ for D_i : $Q_i := 1 - \prod_{L \in D_i} (1 - P_L)$

Tripartites Matching, KnapSack

Andere

$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$, $n \to \infty$.

Petersen graph



Dijkstra Beweis

Nach jeder Iteration gilt:

i) $\forall v \in W$ ist die Länge des kürzesten s-v Pfades gleich $p[v]$ und $\exists$ kürzester s-v Pfad, der mit $\{pred[v], v\}$ endet

ii) $\forall v \in V \setminus W$ mit $p[v] < \infty$ ist die Länge des kürzesten s-v Pfades in dem Netzwerk $N[W \cup \{v\}]$ gleich $p[v]$ und es gibt einen kürzesten s-v Pfad, der mit $\{pred[v], v\}$ endet

Gilt beides nach Initialisierung. Betrachte Knoten x_0 der zu W hinzugefügt wird. Nach Annahme ii) $\exists$ s-x_0 Pfad P in $N[W \cup \{x_0\}]$ der mit $\{pred(x_0), x_0\}$ endet und Länge $\ell(P) = p(x_0)$ hat. Sei P' ein bel. Pfad mit Knoten $y \notin W \cup \{x_0\}$. Es gilt $\ell(P) = p(x_0) \leq p(y) \leq \ell(P')$
$\Rightarrow$ i) für $W \cup \{x_0\}$. ii) gilt wegen Markierung des Algos.

TURNIER

$v_1, \ldots, v_n$ Path, füge v ein $O(\log(n))$

if $a_{v,v_1} = 1$ then return $v, v_1, \ldots, v_n$
else if $a_{v_n,v} = 1$ then return $v_1, \ldots, v_n, v$
else
  $\ell = 1$
  $r = n$
while $r - 1 > \ell$
  $m = \lceil \ell + \frac{r-\ell}{2} \rceil$
  if $a_{v,v_m} = 1$ then
    $r = m$
  else
    $\ell = m$
  end if
end while
return $v_1, \ldots, v_\ell, v, v_r, \ldots, v_n$
end if

GERICHTETER KREIS in G $O(n+m)$

Total $= |V|$
S = new Stack
$\forall v \in V$: $c[v]$ = white
repeat
  wähle $v \in V$ bel mit $c[v]$ = white
  repeat
    if $\exists u \in \Gamma(v)$ mit $c[u]$ = grey then return $\exists$ circle
    (gericht. Ums.)
    if $\exists u \in \Gamma(v)$ mit $c[u]$ = white then
      $c[v]$ = grey
      S.push(v);
      $v = u$;
    end if
    if $\forall u \in \Gamma(v)$: $c[u]$ = black then
      $c(v)$ = black
      Total = Total $- 1$
      $v = S$.pop()
    end if
  until v = nil
until Total $= 0$
return $\nexists$ circle

MEDIAN

(2 sortierte Arrays $a(1,\ldots,n), b(1,\ldots,n)$ $O(\log(n))$

if $n \leq 2$ then
  return MedianOfFour$(a(1,\ldots,n), b(1,\ldots,n))$
endif
if $a(\lfloor \frac{n+1}{2} \rfloor) \leq b(\lceil \frac{n+1}{2} \rceil)$ then
  Median $(a(\lfloor \frac{n+1}{2} \rfloor, \ldots, n), b(1, \ldots, \lceil \frac{n+1}{2} \rceil))$
else
  Median $(a(1, \ldots, \lfloor \frac{n+1}{2} \rfloor), b(\lceil \frac{n+1}{2} \rceil, \ldots, n))$
endif

TOPSORT

(Digraph) $(u,v) \in E \Rightarrow f(u) < f(v)$ $O(n+m)$

Total $= |V|$
S = new Stack
$\forall v \in V$: $c[v]$ = white
for all $w \in V$ do
  if $c(w)$ = white then
    $v = w$
    repeat
      Sei u der nächste Knoten aus der Adj.liste der ausgehend Kanten von v (am Ende u = nil)
      if $c(u)$ = white then
        $c(v)$ = grey
        S.push(v)
        $v = u$
      else if $c(u)$ = grey then
        return "no top. order"
      else if u = nil then
        $c(v)$ = Black
        $f(v)$ = Total
        Total = Total $- 1$
        $v = S$.pop()
      end if
    until v = nil
  end if
end for
Gib $(v, f(v))$ $\forall v \in V$ aus.

Serien

Erweitere Datenstruktur (a,b)-Bäume s.d SELECT (k,T) in $\mathcal{O}(\log n)$ ausgeführt werden kann.

speichere für jeden inneren Knoten v,
$\ell(v)$ = Wieviele keys sich im Teilbaum mit Wzel v befinden

Find: lassen es sein

Insert: Bei der Rebalancierung wenn v zu v_1, v_2, dann bestimmen wir $\ell(v_1), \ell(v_2)$ indem wir die ℓ-Werte der Kinder aufsummieren.
Sobald keine Rebalancierung nötig ist, so laufen wir von v bis zur Wurzel und erhöhen den ℓ-Wert jedes besuchten Knoten um 1. Für alle andere Knoten ändert the #keys don't change. For every rebalance $O(1)$ um ℓ zu berechnen, da Baum höhe $O(\log n) \to$ Laufzeit $O(\log(n))$

delete: Falls keine rebal. nötig → reduziere die ℓ-Werte auf dem Weg von v zur Wurzel um 1.
Falls v ein Kind von u adoptiert → reduce ℓ-Wert auf dem Weg von u bis Wurzel um 1.
Falls Verschmelzung, $\ell(v)$ = Addition ℓ von Kinder → $O(\log n)$

Anzahl Schlüssel ändert sich nicht.

Select(k, v): Falls v no kids → key(v)
Falls $v_1, \ldots, v_n$ kinder, search v_i : $v_1 + \ldots + v_{i-1} \leq k \leq v_1 + \ldots + v_i$
and call Select$(k - (v_1 + \ldots + v_{i-1}), v_i)$.

Select (k,T) = Select(k, w), w wurzel.

two Stacks → Queue

ENQUEUE(x)
S_1.push(x)

```
DEQUEUE()
if S2.isEmpty then
   while S2.isEmpty do
      S2.push(S1.pop)
   end
end
return S2.pop()
```

Amortisierte Laufzeit

$\phi(i) := 2 \cdot$ (Anzhl. Obj. in S_1 nach den ersten i Operationen)
$t(i)$:= Zeit bis i.

$\phi(0) = 0$, $t(0) = 0$ und $\phi(i) > 0$

induktiv: $t(i) + \phi(i) \leq 3i$. $(\Rightarrow t(i) \leq 3i)$
└ falls enqueue: $t(i) + \phi(i) = (t(i-1)+1) + (\phi(i-1)+2) \leq 3i$
falls dequeue: $S_2 \neq \phi \Rightarrow t(i) + \phi(i) = (t(i-1)+1) + \phi(i-1)$
falls $S_1 = \phi \Rightarrow \forall$ Objekt in S_1 genau 2 Operationen:
(pop von S_1, push auf S_2) am Ende noch ein pop auf S_2
$\Rightarrow t(i) + \phi(i) = (t(i-1) + \phi(i-1) + 1) + 0 \leq 3i$

n^n braucht $n\log(n)$ bits d.h $\Omega(n)$ Speicherplatz

KNAPSACK $(\omega[1,..,n], p[1,..,n], B)$. $\sum \omega < B$, p unbeschr.

F[0...n, 0...B] $\qquad \mathcal{O}(nB)$

```
for b=0 to B do
  if ω[1] ≤ b then
    F[1,b] = p[1]
  else
    F[1,b] = 0
  end if
end for
for i=2 to n do
  for b=0 to B
    if b ≥ ω[i] AND F[i-1,b-ω[i]]+p[i] > F[i-1,b] then
      F[i,b] := F[i-1,b]        ← L[i,b] = 1. sonst 0.
    endif
  endfor
endfor
return F[n,B]
```

$H_0(A) = \begin{cases} 1, & \text{falls A ein Programm und A(0) endl. Laufzeit} \\ 0, & \text{sonst} \end{cases}$

Nehme an P_0 berechnet H_0. B_x := ignoriert Eingabe und führt A(x) aus. $P(A,x) := 0$, falls A kein Programm. Sonst modifiziert A, s.d B_x entsteht, danach wird P_0 ausgeführt auf B_x und gibt das Resultat aus.
$P(A,x) = 1 \Rightarrow$ A programm und $P_0(B_x) = 1$, d.h B_x hält für 0 d.h A hält auf x. Analog für 0. $\Rightarrow P(A,x) = H(A,x)$

Reductions

3-SAT $\leq_p$ Clique:
$C_i = \ell_{i1} \vee \ell_{i2} \vee \ell_{i3} \mapsto v_{i1}, v_{i2}, v_{i3}$, $(v_{ip}, v_{jq}) \in E \iff i \neq j$ ∧ $\ell_{ip} = \overline{\ell_{jq}}$

3-SAT $\leq_p$ Double SAT
$F \in$ 3-SAT, $F' = F \wedge (x_{n+1} \vee \neg x_{n+1})$

HamC $\leq_p$ Dir.HC
$T: G = (V,E) \mapsto G'(V,E')$, E' replaces $\{u,v\}$ with (u,v) ∧ (v,u)

SAT $\leq_p$ NAE-SAT
$=0 \to w_j = 1, z=0$
$\lambda_1 \vee \lambda_2 = 1 \to w_j = 0 = z$
$F = C_1 \wedge \ldots \wedge C_k \in$ 3-SAT, $C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3 \mapsto D_j = (\lambda_1 \vee \lambda_2 \vee w_j) \wedge (\neg w_j \vee \lambda_3 \vee z)$

Vertex Cover $\leq_p$ Set cover
$G = (V,E)$, j $\mapsto$ Split E into n Subsets, $S_i = \{v \in V \mid \{v, v_i\} \in E\}$, k=j

HamiltonCycle $\leq_p$ TSP
$G = (V,E) \mapsto (V', E')$, $E' = E \cup E^c$, $\omega(v,u) = \begin{cases} 1, & \{v,u\} \in E \\ 2, & \{v,u\} \in E^c \end{cases}$

SAT $\leq_p$ exact covers
k-th Literal in C_j
$C_j = (L_{j,1} \vee \ldots \vee L_{j,m_j}) \mapsto U = \{x_i \mid 1 \leq i \leq n\} \cup \{C_j \mid 1 \leq j \leq \ell\} \cup \{p_{jk} \mid 1 \leq j \leq \ell, 1 \leq k \leq m\}$

3-SAT $\leq_p$ Independent Set
$C_i = (L_{i1} \wedge L_{i2} \wedge L_{i3}) \mapsto \{c_{i1}, c_{i2}, c_{i3}\}$ (triangle) and $\{c_{ik}, c_{j\ell}\} \iff \overline{L_{ik}} = L_{j\ell}$

Independent Set $\leq_p$ Clique
G has independ. Set $\iff G^c$ has a clique.

Ind.Set $\leq_p$ Node-Cover (Vertex-cover)
$G, K \mapsto G, (|V| - K)$

Exact cover $\leq_p$ Knapsack
$(U = \{u_1, \ldots, u_n\}, F = \{S_1, \ldots, S_m\}) \mapsto a_1, \ldots, a_m$, $a_{ji} = 1 \iff u_i \in S_j$ else 0
$K = 1 + m^2 + \ldots + m^{n-1}$

Inversion

max : $\frac{n(n-1)}{2}$, $(n, n-1, \ldots, 1)$

split into $L_\ell = (a_1, \ldots, a_{\lceil \frac{n}{2} \rceil})$ and $L_r = (b_1, \ldots, b_{\lfloor \frac{n}{2} \rfloor})$ (aufst. sortiert). Mod. the Mergestep in MergeSort to count the Anzahl Inversionen $O(|L|)$

$a_i \le b_i \rightarrow c = c$. $a_i > b_j \rightarrow c = c + (\lceil \frac{n}{2} \rceil - i + 1)$

Korrektheit: Argue every Inversion only gets counted once.

Kruskal mit UnionFind

Zu begin ist jeder Knoten ein ZHK für sich

jede zusätzliche Kante → (1) 2 Knoten im selben ZHK
(2) 2 Knoten versch ZHK verbinden

iff (1) wird ein Kreis geschlossen

Im Unionfind wird eine ZHK durch ihre Knoten repräsentiert. Falls $\{u,v\}$ hinzugefügt wird, wird Find(u) = Find(v) getestet Falls true → (1). Sonst (2) und wir fügen $\{u,v\}$ zu T hinzu und rufen Union(Find(u), Find(v))

Laufzeit: ... $O(|E| \cdot \log(|V|)$, da der Graph in jedem Schritt höchstens einen Kreis hat

mit naive BFS: $O((|V|+|E|) = O(|V|)$

zusammen mit sort $= O(|E| \cdot |V|)$

Längste aufst. Teilsequenz (=LATS)

L_i = Länge der LATS der in a_i endet

$$L_i = \begin{cases} 1 & \text{, falls } a_j > a_i, \forall a_j \in \Gamma(a_i) \\ \max_{a_j \in \Gamma(a_i): a_i > a_j} \{1 + L_j\} & \text{, sonst} \end{cases}$$

Nullstellen-mod in NP

Sei ℓ = # Monome von P, $P = P_1 + \ldots + P_\ell$, $P_j = a_j x_1^{b_{j,1}} \ldots x_k^{b_{j,k}}$, sei $b = \max(b_{j,i})$. So können wir jeden Exponent in $\log b$ bits kodieren

$\Rightarrow |P| = \ell(\log n + k \cdot \log b) + \log n + \log k$

Falls $(P, n, k) \in L$ so $\exists$ Nullstelle in P mod n, welches wir als Zertifikat verwenden. Sei $x = (x_1, \ldots, x_k)$ dieses Zert. $x_i \in \{0, \ldots, n-1\}, \forall i \rightarrow$ x in $k \cdot \log n = O(|P|^2)$ kodierbar

Addition und Subtr. in $\mathbb{Z}/n\mathbb{Z}$ können in $O(\log n)$ berechnet werden. ZB $a + b \bmod n = \begin{cases} a+b & , a+b < n \\ a+b-n & , a+b \ge n \end{cases}$

Multiplikation $O(\log^2 n)$, so können wir (durch iteriertes quadrieren) $x_i^{b_{ji}}$ in $2\lceil \log(b_{ji}) \rceil$ Mult. berechnen

$\Rightarrow P(x)$ in $O(\ell \cdot k \log b \cdot \log^2 n + \ell \cdot k \log^2 n + \ell \cdot \log(n)) = O(|P|^3)$ berechenbar

(2,5)-Baum Amort-Cost

Potential $\Phi(i)$ = # innere Knoten mit 5 Kindern nach i Oper.

Ang bei i-ten Schritt werden k rebal. vorgenommen

$\Rightarrow \Phi(i) \le \Phi(i-1) - k + 1$ und $t(i) \le t(i-1) + k$

$\Rightarrow a(i) = \Phi(i) + t(i) - (\Phi(i-1) + t(i-1)) \le 1 \Rightarrow \sum_{i=1}^{n} a(i) \le n$

Rucksack $\le_p$ Zero-One Linear Inequal.

$(B, K, \omega_1, \ldots, \omega_n, p_1, \ldots, p_n) \mapsto b_2 = -B, b_1 = K, p_j = a_{1j}, -\omega_j = a_{2j}$

Grad beschränkt-Spannbaum NP schwer

Falls (G,k) in der Sprache GBS liegt, dann besitzt G einen k-beschränkten Spannbaum T. Sei T das Zertifikat co. Da $T \subseteq G$ ist $|T|$ polynomiell beschränkt in $|G|$.

Gegeben T können wir in polyzeit überprüfen ob G wirklich einen k-beschränkten Spannbaum hat. Indem wir mit Breitensuche testen ob T ein Spannbaum ist und durch alle Knoten durchgehen um zu testen ob alle Knoten Grad $\le k$ haben.

Clique $\le_p$ Half-Clique

i) $2k > V$: add $2k - |V|$ isol. nodes

ii) $2k < V$: add $|V| - 2k$ nodes, connect all

VertexCover $\le_p$ Football CardCollector

$e \in E \rightarrow Y_e$ Player

E3SAT $\le_p$ E4SAT

$(x \vee y \vee z) = (x \vee y \vee z \vee a) \wedge (x \vee y \vee z \vee \bar{a})$

E4SAT $\le_p$ E3SAT

$(x_1 \vee x_2 \vee x_3 \vee x_4) = (x_1 \vee x_2 \vee a) \wedge (x_3 \vee x_4 \vee \bar{a})$

E5SAT $\le_p$ E3SAT

$(y_1 \vee y_2 \vee y_3 \vee y_4 \vee y_5) = (y_1 \vee y_2 \vee x) \wedge (y_3 \vee \bar{x} \vee z) \wedge (y_4 \vee y_5 \vee \bar{z})$

Annika study-plan (Rückwärts dyn. Prog.)

$z_i = \max\{a_i + z_{i+1}, b_i + z_{i+2}\}$

Monotones Matching

$$a_{ij} = \begin{cases} \max\{a_{i-1,j}, a_{i,j-1}\} & , x_i \neq y_j \text{ or } j = i \\ 1 + a_{i-1,j-1} & , \text{sonst} \end{cases}$$