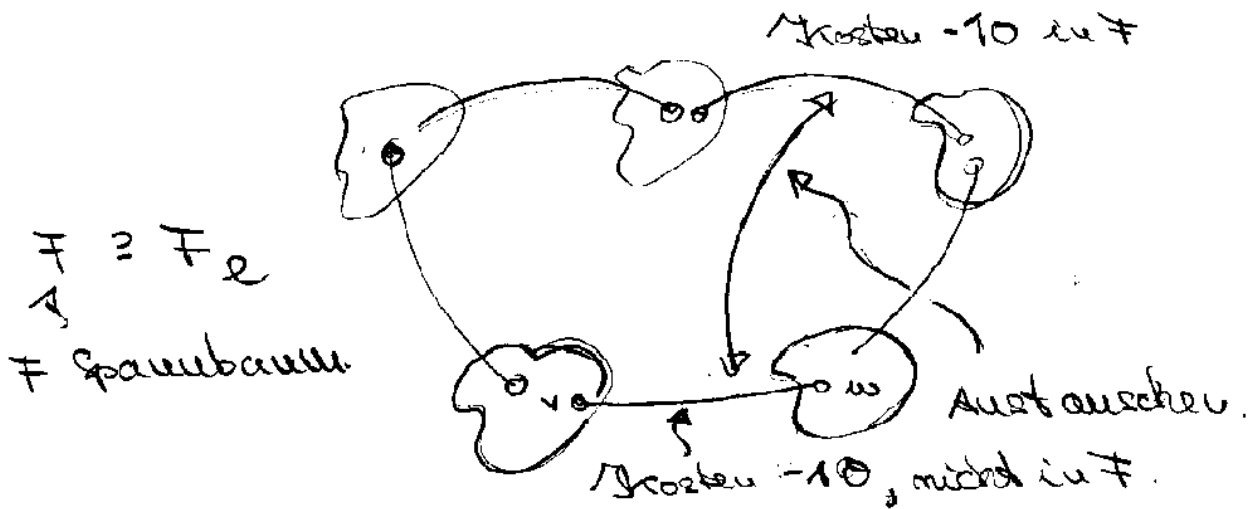


Nachtrag zu Korrektheit von Kruskal

1. Durchlauf: Die Kante mit minimalen Kosten wird genommen. Gibt es mehrere, so ist es egal welche.

2+1'tes Lauf: Die Kante mit minimalen Kosten, die zwei Komponenten verbindet.



Alle Kanten, die 2 Komponenten verbinden, haben Kosten ≥ -10 !

Beachte: Negative Kosten bei Kruskal kein Problem!

Nur bei der branch-and-bound. (Vorher vergrößern!) $\rightarrow b$

Binomialkoeffizient $\binom{m}{k}$ für $m \geq k \geq 0$.

$$\binom{m}{k} = \frac{m!}{(m-k)! \cdot k!} = \frac{\overbrace{m(m-1) \dots (m-k+1)}^{\text{Oberste } k \text{ Faktoren von } m!}}{k!}$$

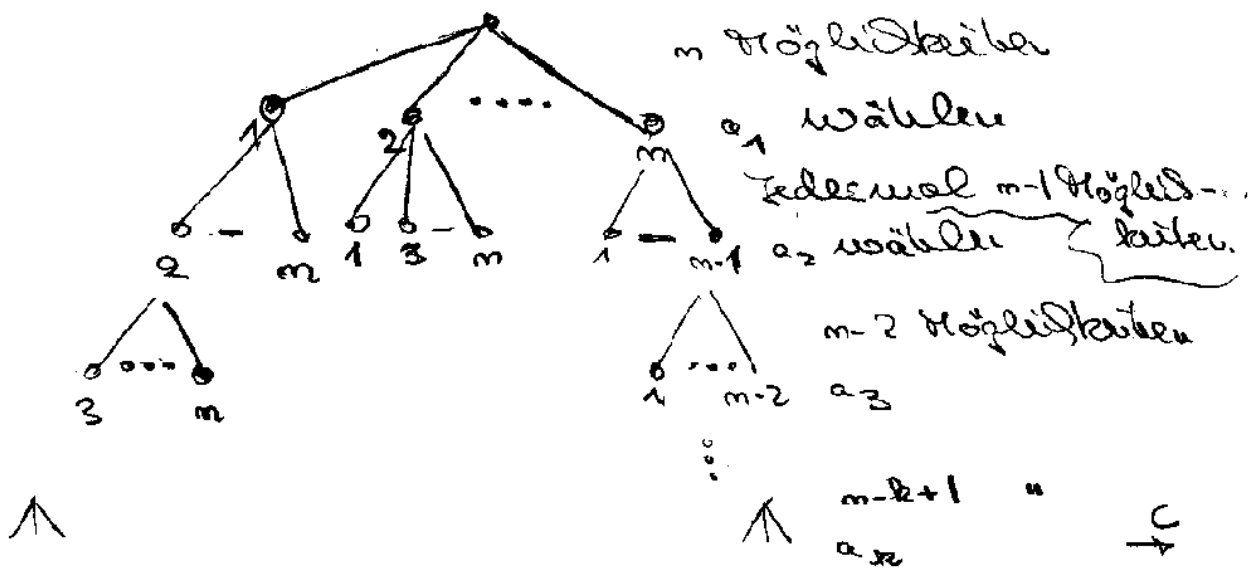
$0! = 1! = 1$.

$\binom{m}{k}$ = # Teilfolgen von $\{1, \dots, m\}$ mit genau k Elementen

$\binom{m}{1} = m, \binom{m}{2} = \frac{m(m-1)}{2}, \binom{m}{k} \leq m^k$

Beweis: Permutationen für

alle Folgen (a_1, \dots, a_k) mit $a_i \in \{1, \dots, m\}$,
 alle a_i verschieden.



Die Zahl hat

$$m(m-1) \dots \overbrace{(m-k+1)}^{m-k+1} = \frac{m!}{(m-k)!}$$

k Faktoren (mit k-1, da 0, 1, ..., k-1 genau k Zahlen)

Blätter.

Jedes Blatt \Leftrightarrow genau eine Folge (a_1, \dots, a_k)

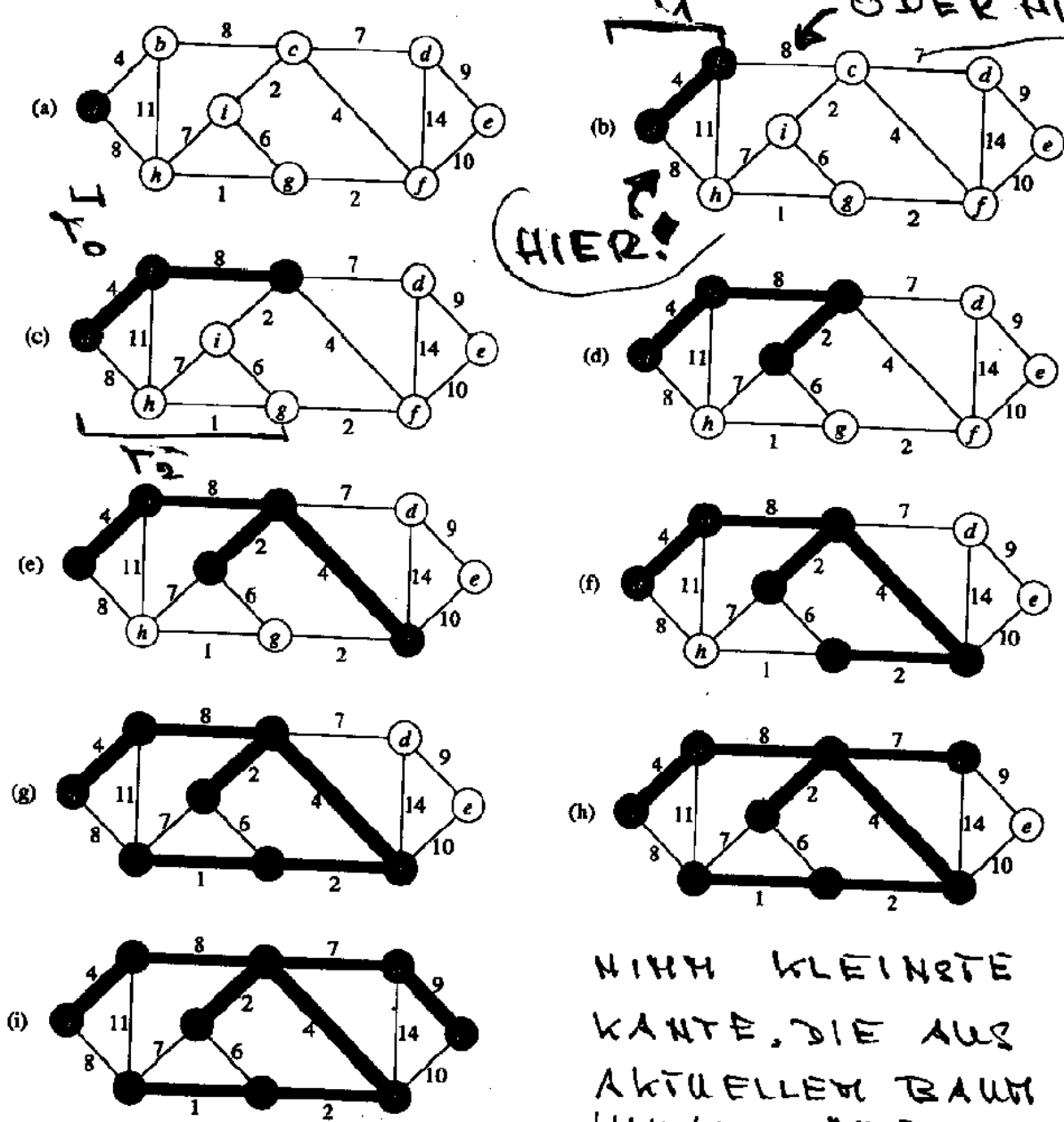
Jede Menge aus k Elementen kommt k!-mal vor: $\{a_1, \dots, a_k\}$ ungeordnet, geordnet als $(a_1, \dots, a_k), (a_2, a_1, \dots, a_k), \dots, (a_k, a_{k-1}, \dots, a_2, a_1)$ k! Permutationen.

Also $\frac{m!}{(m-k)! \cdot k!} = \binom{m}{k}$ Mengen

mit k verschiedenen Elementen.

241

BEISPIEL (PRIMS ALGORITHMUS)



NIMM KLEINSTE KANTE, DIE AUS AKTUELLEM BAUM HINAUSFÜHRT.

Figure 24.5 The execution of Prim's algorithm on the graph from Figure 24.1. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.

Algorithmus (Minimaler Spannbau
nach Prim 1963)

Eingabe: Wie bei Kruskal

Ausgabe: Menge von Kanten eines
minimalen Spannbau.

1. Wähle irgendeinen Startknoten v

$Q := V \setminus \{v\}$ // Q enthält immer

$\{v\}$ // die Knoten, die noch

behandelt werden müssen.

2. while $Q \neq \emptyset$

2.1 $v \in Q$

3. $M := \{e, w\} \mid v \in V \setminus Q, w \in Q$

3.1 $M := \{e, w\} \mid e = \{v, w\}$ // $M =$ Menge der Kanten mit

genau einem Knoten in Q

4. $\{e, w\} :=$ eine Kante von M mit

minimalem w - Knoten in Q

4.1 $F := F \cup \{e, w\}$

$Q := Q$ ohne die Knoten aus

$\{v, w\}$, die auch zu Q gehört.



Korrektheit mit der Invarianten:

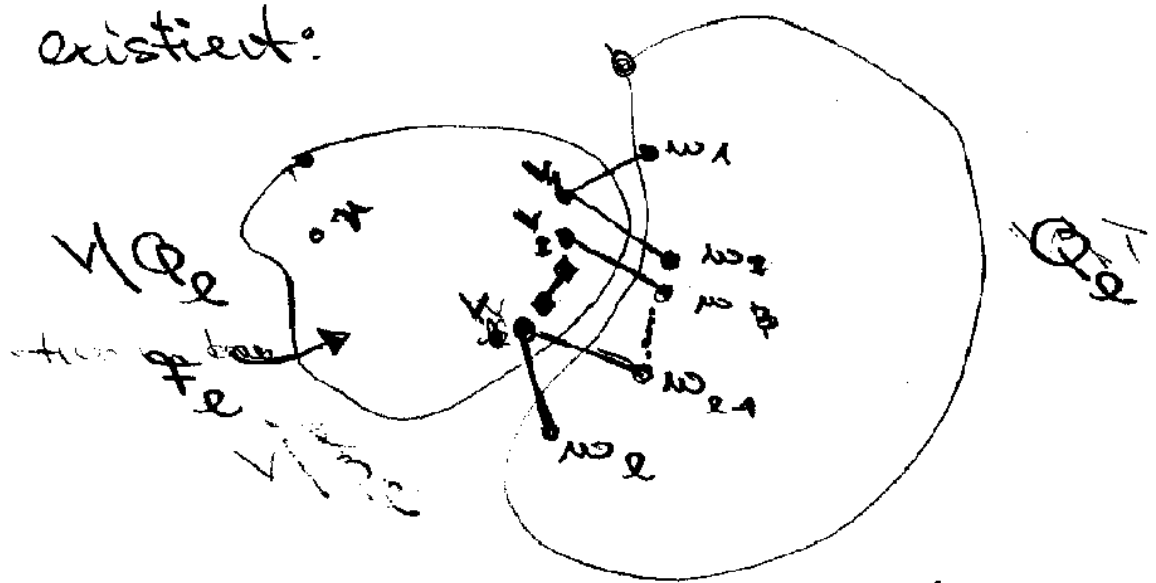
• Es gibt min. pb. $F \geq F_e$

• $Q_e = \bigvee_{F \geq F_e} \dots$

+ die In-
(4.11.15)
aus 1.,
wg. $F_0 = \text{pb.}$

$l = 0 \vee$

Gehe die Invarianten für l und
finde eine $l+1$ -tes Lauf der
Schleife statt. Sei $F \geq F_e$
eine min. pb., die nach Ind-Vor.
existiert:



Werde $\{v, w\}$ im $l+1$ ten
Lauf genommen. Falls $\{v, w\} \in F$, dann

(7.44)

gilt die Invariante auch für

$l+1$. Somit gilt $F \cup \{v, w\}$

enthält eine Kante, die S_v, w

enthält, Diese enthält

mindestens eine weitere Kante

$\{v, w'\}$ mit $v' \in V \setminus Q_l, w' \in Q_l$



$$\exists \text{ ist } k(S_v, w) \leq k(S_{v'}, w')$$

gemäß Prior. Also, da F minimal

$$k(S_v, w) = k(S_{v'}, w').$$

Können in F die Kante $S_{v'}, w'$

durch $\{v, w\}$ ersetzen und haben

immer noch -minimales Spannbauwerk

Dann Invariante bei $l+1$. $l+1$

Laufzeit von Prim

- $m-1$ Läufe durch 2.
- 3. + 4. etw. $O(|E|)$ nicht erhöht, da $|E| \geq |V|-1$.

Also $O(|V| \cdot |E|)$, bzw. $O(m^3)$ bei $|V|=m$.

Bessere Laufzeit durch bessere Vorgehensweise von Q.

- Maßgebend dafür, ob $w \in Q$ im den Baum aufgenommen wird, sind die minimalen Kosten einer Kante(!) $\{v, w\}$ für $v \notin Q$.

- Also array $key[u-v]$ of $\mathbb{R} \cup \{\infty\}$ mit der Interpretation: Für alle $w \in Q$ ist

$key[w] = \infty$
 ghes. keine solche Kante existiert.

$key[w] = \text{min. Kosten einer Kante } \{v, w\} \text{ wobei } v \notin Q.$

• Aufbauen array $ko[1, \dots, m]$ of $\{1, \dots, m\}$
 mit: Für alle $w \in Q$

$ko[w] = v \Leftrightarrow \{v, w\}$ ist keine Kante
 minimales Kosten mit $v \notin Q$

w werden Q mit einem Datenstruktur
 für die priority queue (Vorrang-
 warteschlange) implementieren:

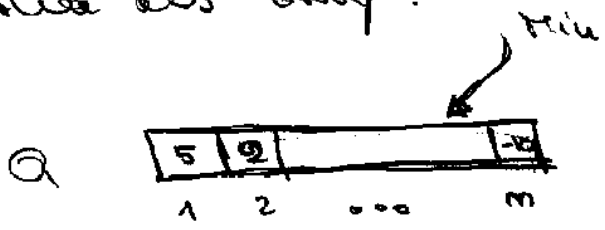
- Speichert Menge von Elementen,
 denen jedes einen Schlüsselwert
 hat. (Keine Eindeigkeitsforderung)
- Operation Min gibt uns (ein)
 Element mit minimalem Schlüssel-
 wert.
- Deletion Min löscht ein Element
 mit minimalem Schlüsselwert.

o $\text{Insert}(v, s)$ = Element v mit Schlüsselwert s einfügen.

Wie kann man eine solche DS duplizieren?

1. Möglichkeit

Erkenne als array.



Indices = Elemente, i

Einträge m_i Q = Schlüsselwerte,

Sonderwert (etwas $-\infty$) bei "nicht vorhanden"

Zeichen: $\text{Insert}(v, s)$ O/W

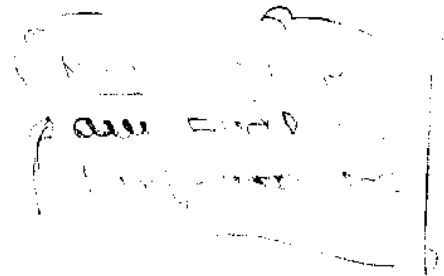
(sofern keine gleiche Elemente mehr da)

Min $\in O(n)$ $O(n)$

^{100%} Delta Min $O(n)$ ~~best~~ finden

Also das zu Löschen, also dann

$O(n)$ um das neue Minimum zu finden.



2. Möglichkeit

^{100%} Darstellung als Heap.

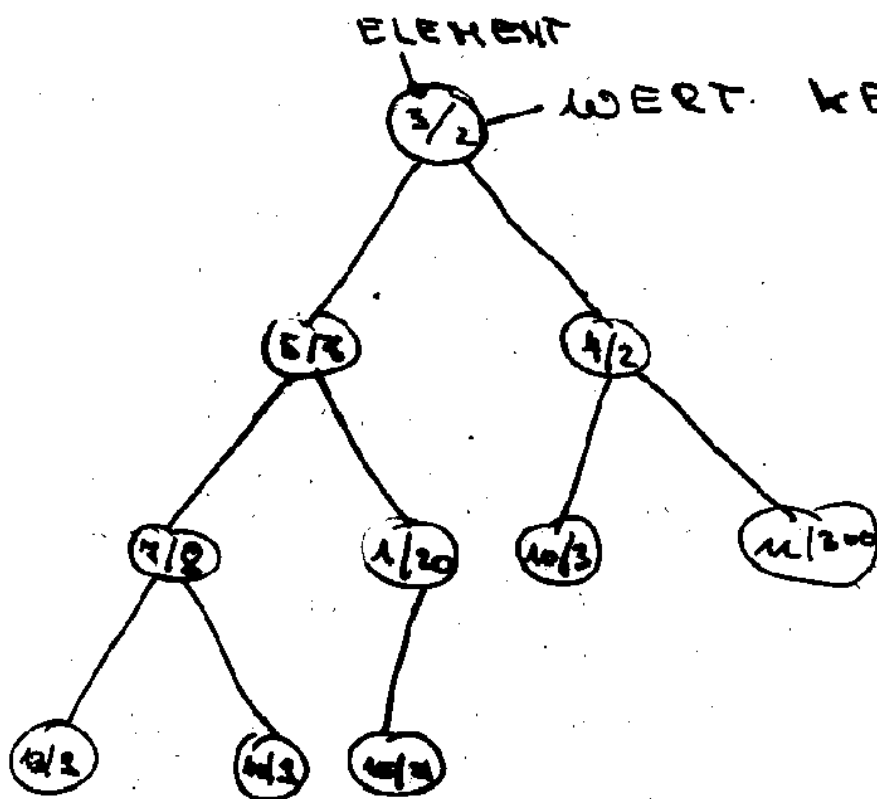
HEAP = "FAST VOLLSTÄNDIGER"

BINÄRER BAUM DESSEN

ELEMENTE (= KNOTEN) BEGL.

$key[i]$
FUNKTIONSWERTEN NACH OBEN HIN

KLEINER WERDEN.



TIEFE n ,

DANN:

INNERE:

$$2^n - 1$$

BLÄTTER:

$$\frac{2^n}{2} - 1$$

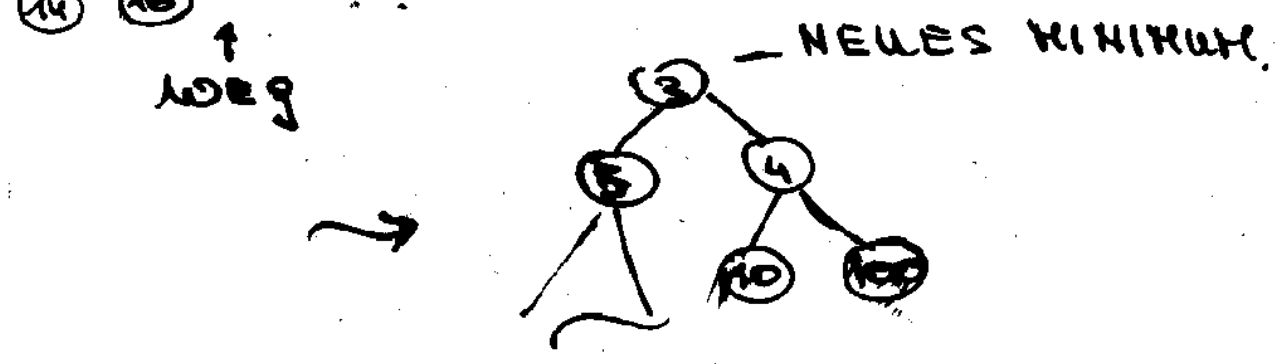
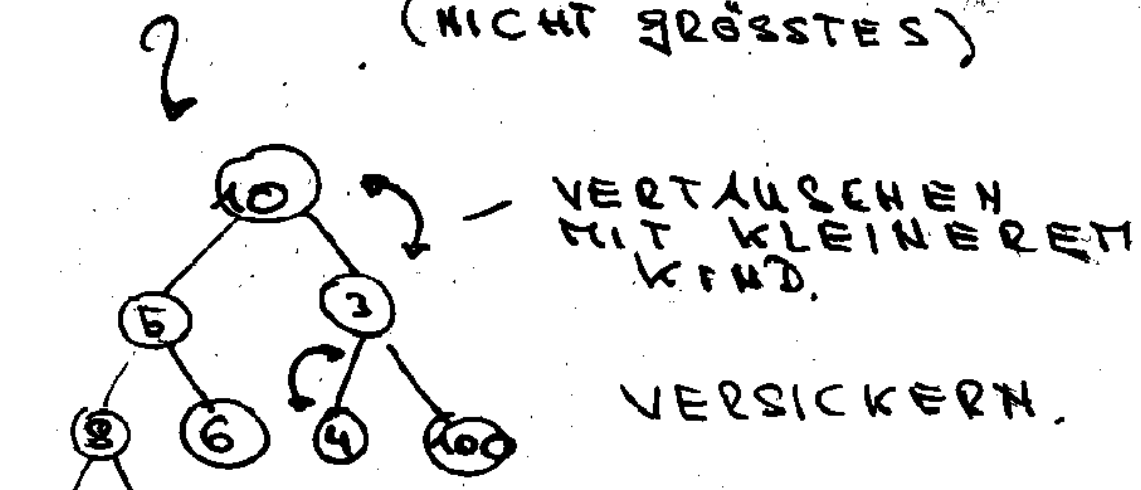
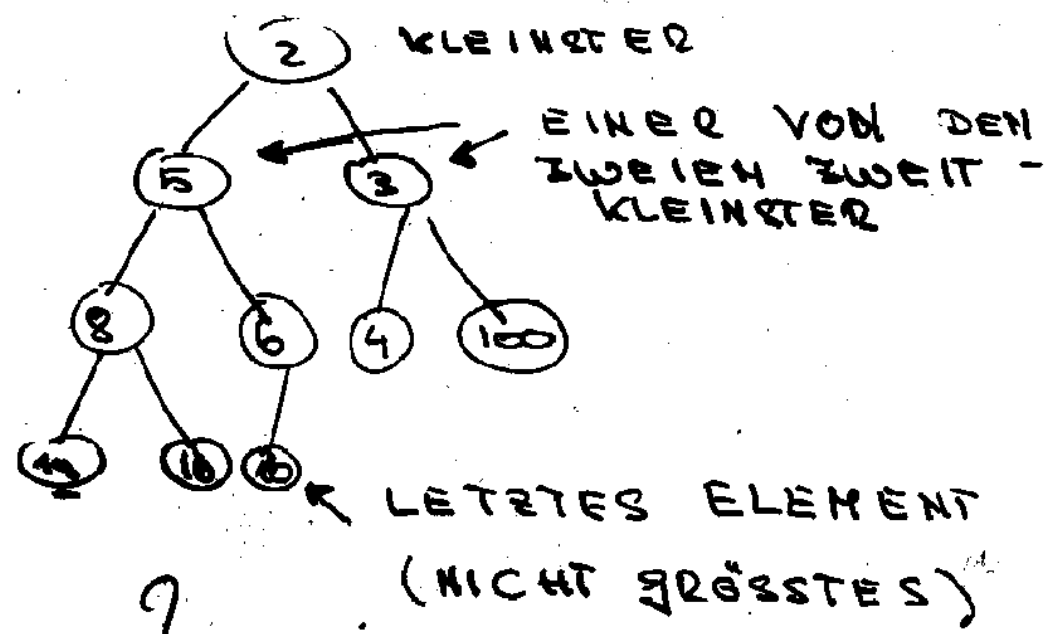
FAST VOLLSTÄNDIG.

HEAPEIGENSCHAFT: FÜR ALLE u, v

u VORFAHR VON $v \rightarrow key[u] \geq key[v]$

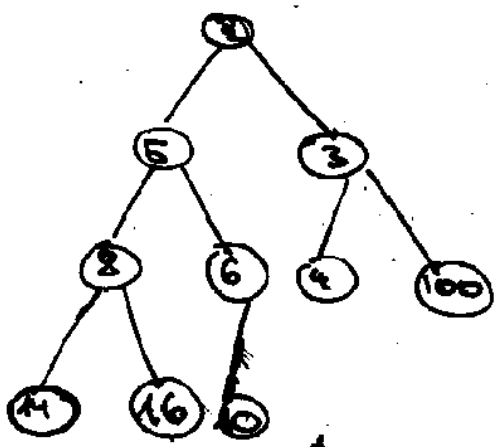
BEISPIEL MINIMUM LÖSCHEN

NUR WERTE: KEY[I]:

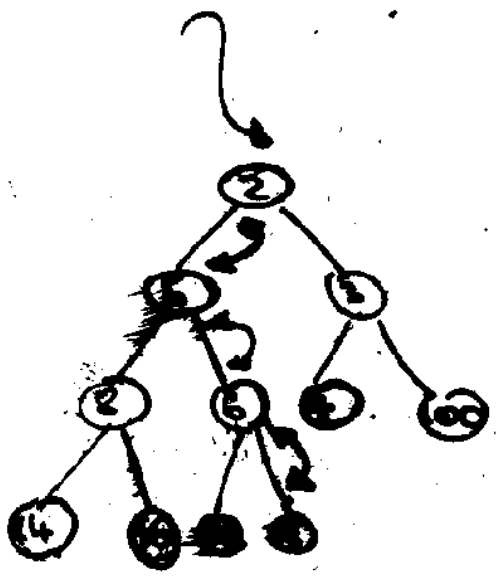


BEISPIEL EINFÜGEN

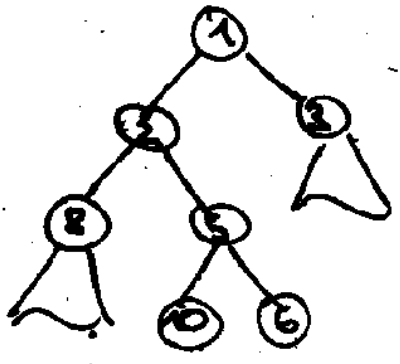
ELEMENT MIT WERT 1 EINFÜGEN



↑
LETZTER PLATZ



AUFSTEIGEN



PRIORITY QUEUE MIT HEAP, SAGEN WIR Q.

MIN {

1. GIB ELEMENT DER WURZEL
{
AUS $O(1)$

DELETEMIN {

1. NIMM „LETZTES“ ELEMENT
SETZE ES AUF WURZEL
x: = WURZEL(-ELEMENT)

2. WHILE KEY[x] > KEY[LINKER
SOHN VON x]
ODER ... RECHTER SOHN
{ TAUSCHE x MIT
KLEINEREM SOHN {
}

N ELEMENTE, $O(\log N)$, DA

MAXIMAL $O(\log N)$
DURCHLAUFE

INSERT (v, s) // KEY[v] - s

1. TUE V ALS LETZTES ELEMENT IN DEN HEAP.

2. WHILE KEY[VATER VON V] > KEY[V]

{ VERTAUSCHE V MIT SEINEM VATER }

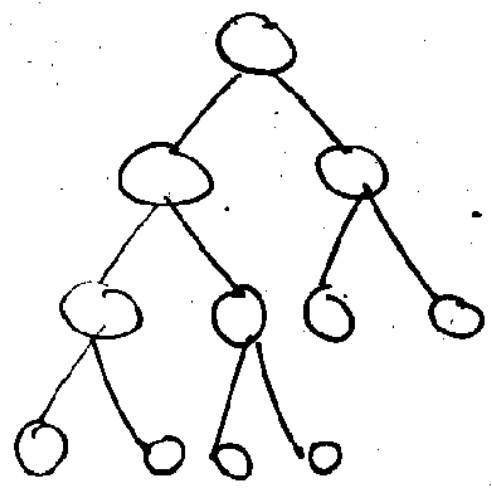
}

N ELEMENTE O(LOGN)

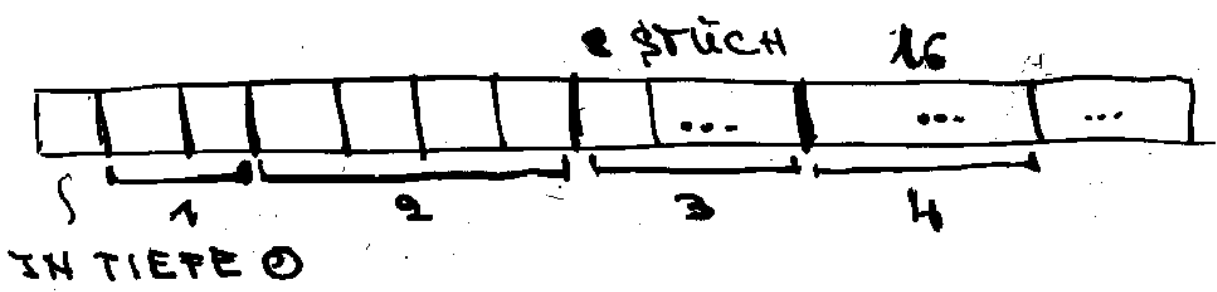
VERGLEICH DER MÖGLICHKEITEN DER PRIORITY QUEUE

	MIN	DELETEN	INSERT
HEAP	1	LOG N	LOG N
ARRAY			
ODER LISTE	1	N (5)	1
	ELEMENT FINDEN	HEAP N	LISTE N
		ARRAY 1	

HEAP ALS ARRAY $Q[1, \dots, N]$



- $Q[1]$
- $Q[2], Q[3]$
- $Q[4] \dots Q[7]$
- $Q[8] \dots Q[15]$



VATER i // i INDEX AUS $1, \dots, N$

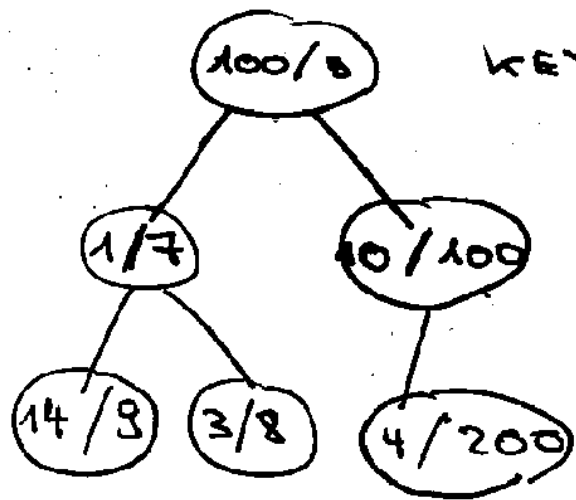
~~$i \neq i+1$~~

REBE $\lfloor \frac{i}{2} \rfloor$ AUS ?

LINKER SOHN i $2i$

RECHTER SOHN $2i+1$

ABSCHLIESSENDES BEISPIEL



KEY[100] = 3

ARRAY Q

Q[1] = 100

KEY[100] = 3

Q[2] = 1

KEY[2] = 7

Q[3] = 10

KEY[10] = 100

Q[4] = 14

⋮

Q[5] = 3

Q[6] = 4



WENN ELEMENTE
NUR EINMAL.

SUCHEN NACH ELEMENT, ODER

SCHLÜSSEL WIRD NICHT () UNTER-
STÜTZT.

Algorithmus (Prüfung mit Q in heap)

1. Wähle Startknoten r .

for each $v \in \text{Adj}[r]$ {

key[v] := $k(r, v)$;

kaute[v] := r ?

Für alle übrigen $v \in V$ {

key[v] := ∞ ; kaute[v] := ∞ ?

Früge $v \in V$ mit Schlüsselwert key[v]
in heap Q ein.

2. while $Q \neq \emptyset$ {

3. $w := \text{Min}_Q$; DeleteMin Q ;

$F = F \cup \{v \mid \text{kaute}[v] = w\}$;

4. for each $u \in \text{Adj}[w] \cap Q$ {

if $k(u, w) < \text{key}[u]$ {

key[u] := $k(u, w)$;

Q anpassen ?

}

}

Korrektheit mit zusätzliches Invarianten:

Für alle $is \in Q_e$

$key_e[is] =$ minimale Kosten einer Kante $\{v, w\}$ für $v \in Q_e$

$key_e[is] = \infty$, wenn eine solche Kante nicht existiert.

Laufzeit

1. $O(m) + O(m \cdot \log m)$ für das Füllen von Q . (Füllen von Q ist $O(m)$ - interessante Nebenlaufzeit.)
2. $m-1$ Läufe.
3. Einmal $O(\log m)$, insgesamt $O(m \cdot \log m)$
4. Insgesamt $O(|E|) + |E|$ -mal Auslesen von Q .

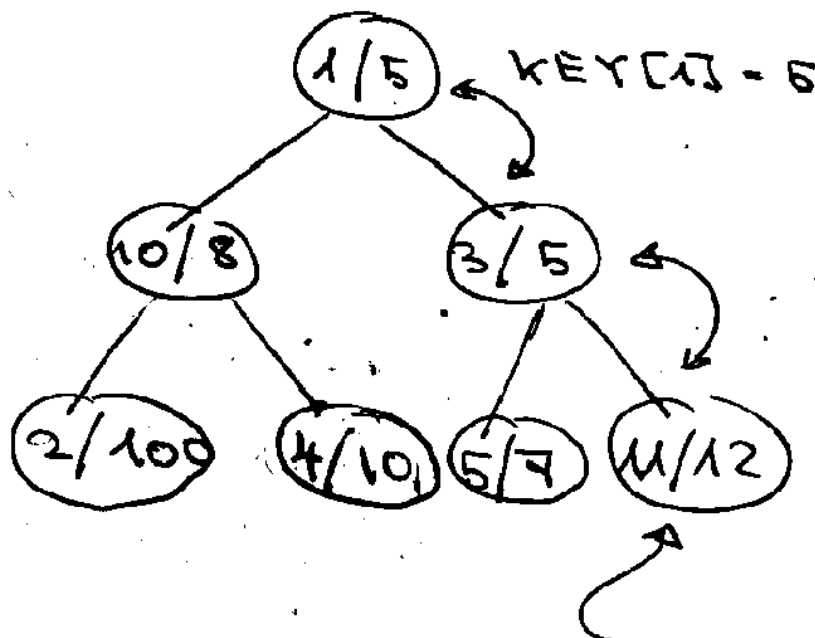
ANPASSEN VON HEAP Q

7.54

OPERATION

DECREASE KEY(V, S)

$S < \text{KEY}[V]$, NEUER SCHLÜSSEL



DECREASE KEY(U, 1)

DECREASE KEY(V, S)

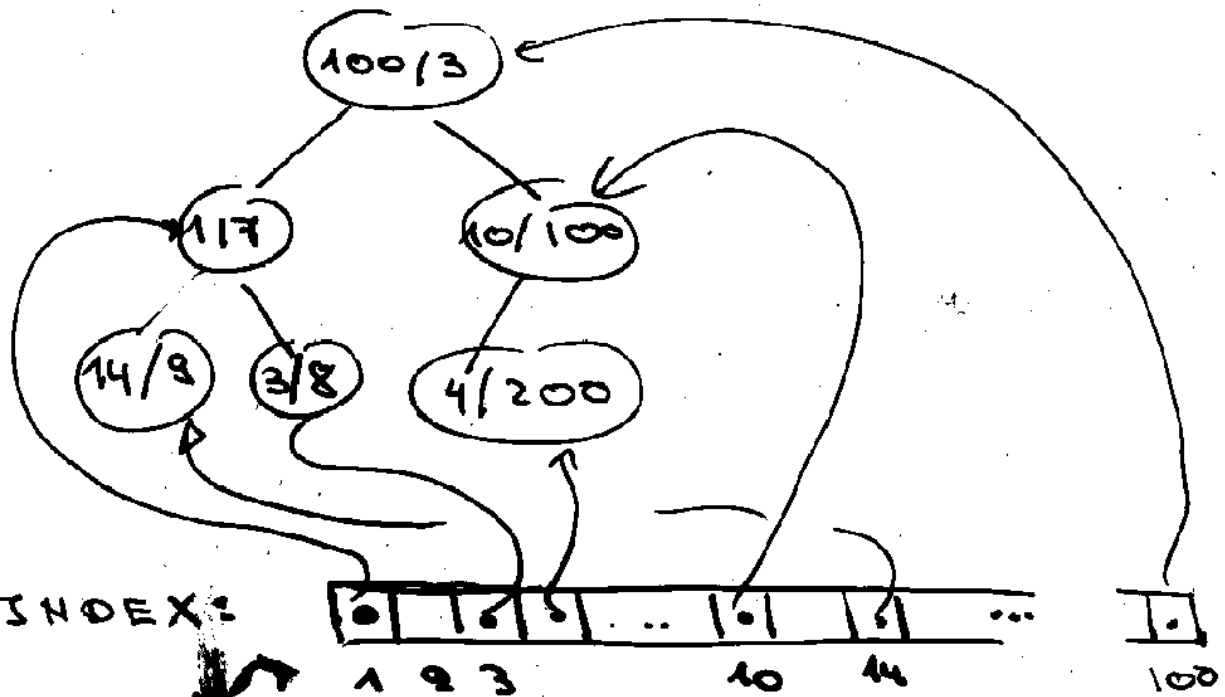
1. FINDE ELEMENT V IN Q
/ NICHT VON Q
/ UNTERSTÜTZT!

2. WHILE KEY[VATER VON V] > S
TAUSCHE V MIT VATER

LAUFZEIT DECKEY(V,S)

2. $O(\log N)$ 1. $O(N)$ (!!)

SUM FINDEN: INDEX DRÜBERLEGEN



DIREKTE ADRESSEN

INDEX=3
INDEX=5
INDEX=6

; INDEX=1-1

FINDEN $O(1)$.

FALLS GRUNDMENGE ZU GROS
DANN SUCHBAUM.

FINDEN $O(\log N)$

Falls direkte Adressen dabei:

Einmalige Lookup von G

(und Index) $O(\log N)$

$|E|$ -malige Lookup $O(|E| \log N)$

Dann Primus implement

$O(|E| \log V)$

Beachte vorher
 $O(|E| \cdot V)$

(wie Kruskal mit Union by size.)