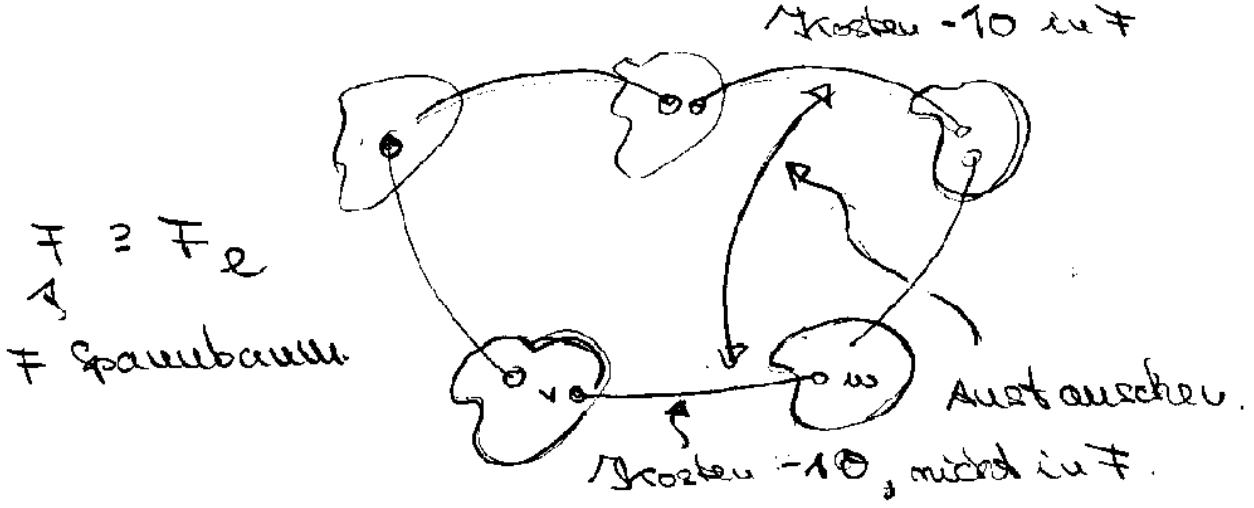
Nachtrag zu Korrektheit von Kruskal

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

l+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 beim branch-and-bound. (Vorher vergrößern!) $\to$ b

I-46b

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

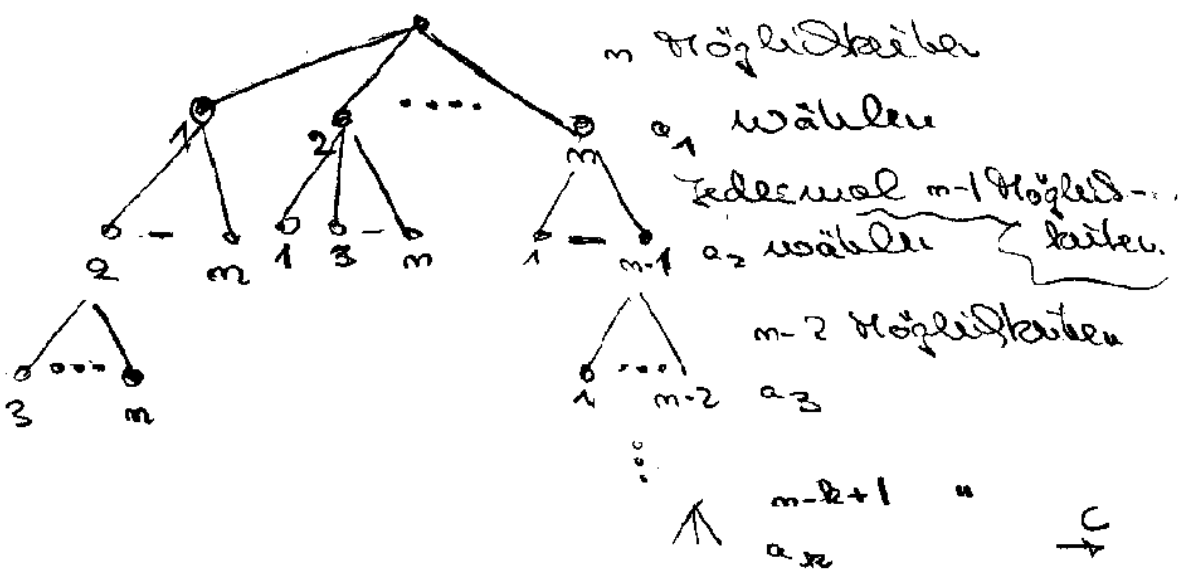
$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} = \frac{\overbrace{n(n-1)\cdots(n-k+1)}^{\text{Obersten } k \text{ Faktoren von } n!}}{k!}$$

$0! = 1! = 1$.

$\binom{n}{k}$ = # Teilmengen von $\{1, \ldots, n\}$ mit genau k Elementen

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

Beweis: Auszählbar für alle Folgen $(a_1, \ldots, a_k)$ mit $a_i \in \{1, \ldots, n\}$, alle a_i verschieden.



Da Zahlen hat

$$\underbrace{n(n-1)\cdots(n-k+1)}_{k \text{ Faktoren (weil } k-1, \text{ da } 0,1,-,\text{ 41 genau } k \text{ Zahlen})} = \frac{n!}{(n-k)!}$$

($n-(k-1)=n-k+1$)

Blätter.

Jedes Blatt $\Leftrightarrow$ genau eine Folge

$$(a_1, -, a_k)$$

Jede Wahl aus k Elementen kann k! - mal von $\{a_1, -, a_k\}$ unterschieden, geordnet als $(a_1, -, a_k)$, $(a_2, a_1, -, a_k)$, — $(a_k, a_{k-1}, -, a_2, a_1)$ k! Permutationen.

Also $\frac{n!}{(n-k)!\,k!} = \binom{n}{k}$ Wegen

mit k verschiedenen Elementen.

Diese Seite mußte aus rechtlichen Gründen entfernt werden!

Algorithmus (Minimales Spannbaum nach Prim 1963)

Eingabe: Wie bei Kruskal

Ausgabe: Menge von Kanten F eines minimalen Spannbaums.

1. Wähle irgendeinen Startknoten v
   $Q := V \setminus \{v\}$ // Q enthält immer die Knoten, die noch bearbeitet werden müssen.
   $F := \emptyset$ // F = Menge der Kanten im Spannbaum
2. while $Q \neq \emptyset$ {
3. $M := \{\{u,w\} \mid u \in V \setminus Q,\ w \in Q\}$ // M = Menge der Kanten mit genau einem Knoten in Q
4. $\{u,v\} :=$ eine Kante von minimalem Gewicht in M
   $F := F \cup \{\{u,v\}\}$;
   $Q := Q \setminus \{v\}$ // v = der Knoten aus $\{u,v\}$, der auch in Q liegt.

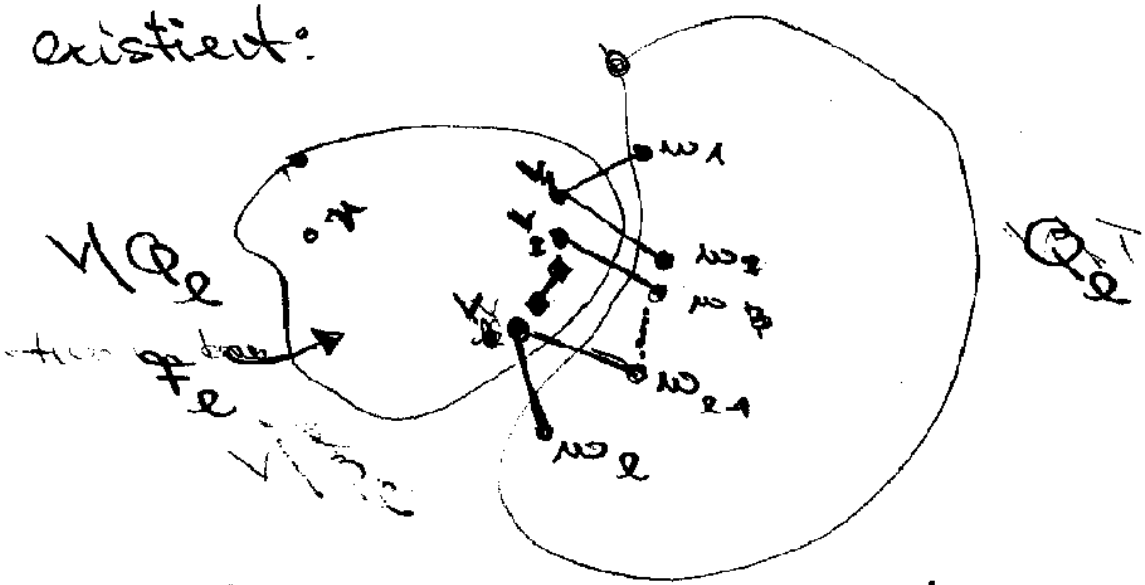
□

Korrektheit mit der Invariante:

- Es gibt max. $F_\ell \subseteq F$
- Q_ℓ = [illegible]

$\ell = 0$ ✓.

Gelte die Invariante für ℓ und finde ein $\ell+1$-ter Lauf der Schleife statt. Sei $F \supseteq F_\ell$ ein max. F_ℓ, das nach Induktionsvor. existiert:



Wenn $\{v, w\}$ im $\ell+1^{\text{ten}}$ Lauf gewählt. Falls $\{v, w\} \in F$, dann

gilt die Invariante auch für $l+1$. Sonst gilt $F \cup \{\{v,w\}\}$ enthält einen Kreis, der $\{v,w\}$ enthält. Dieser enthält mindestens eine weitere Kante $\{v',w'\}$ mit $v' \in V \setminus Q_l$, $w' \in Q_l$

Es ist $k(\{v,w\}) \leq k(\{v',w'\})$ gemäß Prim. Also, da F minimal

$$s(\{v,w\}) = s(\{v',w'\}).$$

Können in F die Kante $\{v',w'\}$ durch $\{v,w\}$ ersetzen und haben immer noch minimalen Spannbaum. Damit Invariante für $l+1$.

Laufzeit von Prim

- $n-1$ Läufe durch 2.
- 3. + 4. einmal $O(|E|)$ wegen schnell, da $|E| \geq |V|-1$.

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

Bessere Laufzeit durch bessere Verwaltung von Q.

- Abgefragt, ob $w \in Q$ ist, ob w in der Baum aufgenommen wird; sind die minimalen Kosten einer Kante(!) $\{v, w\}$ für $v \notin Q$.
- Also array $key[1-n]$ of Real mit der Intention: Für alle $w \in Q$ ist
  - $key[w] = \infty$ falls keine solche Kante existiert.
  - $key[w]$ = min. Kosten einer Kante $\{v, w\}$ wobei $v \notin Q$.

- Aufbau array key[1,...,m] of {1,...,m}
  mit: Für alle $v \in Q$

  key[v] = x $\Leftrightarrow$ {v,x} ist eine kante
  minimalen kosten mit $x \notin Q$

Wie wendet Q mit einer Datenstruktur für die priority queue (Vorrangwarteschlange) implementieren:

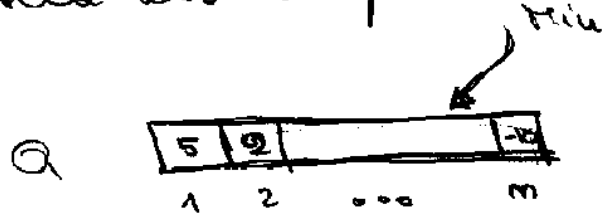
- Speichert Menge von Elementen, denen jedes einen Schlüsselwert hat. (kein Eindeutigkeitsforderung)
- Operation Min gibt ein (ein) Element mit minimalem Schlüsselwert.
- Delete Min löscht ein Element mit minimalem Schlüsselwert.

- Insert(x, s) = Element x mit Schlüsselwert s einfügen.

Wie kann man eine solche DS implementieren?

1. Möglichkeit

Etwa als array.



Indices = Elemente,

Einträge in Q = Schlüsselwerte,

Sonderwert (etwa -∞) bei „nicht vorhanden"

Zeiten: Insert(x, s) O(1)

(sofern keine gleichen Elemente mehrfach)

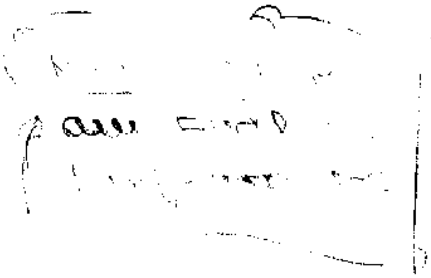
Min $O(1)$ $O(1)$

Delete Min $O(1)$ bis Finden

des zu Löschenden, das dann

$O(n)$ um das neue Minimum

zu finden.



2. Möglichkeit

Darstellung als Heap.

HEAP = „FAST VOLLSTÄNDIGER"

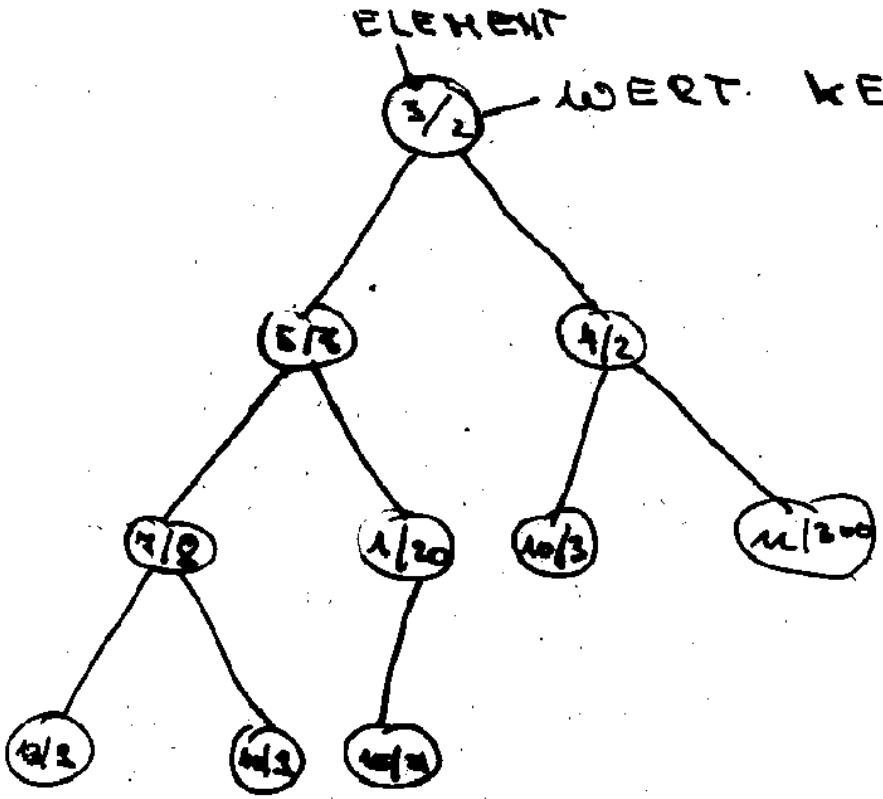
BINÄRER BAUM DESSEN

ELEMENTE (= KNOTEN) BZGL.

KEY[i]

FUNKTIONSWERTEN NACH OBEN HIN

KLEINER WERDEN.



TIEFE H,

DANN:

INNERE: $2^H - 1$

BLÄTTER: 2^H // 2^{H-1}

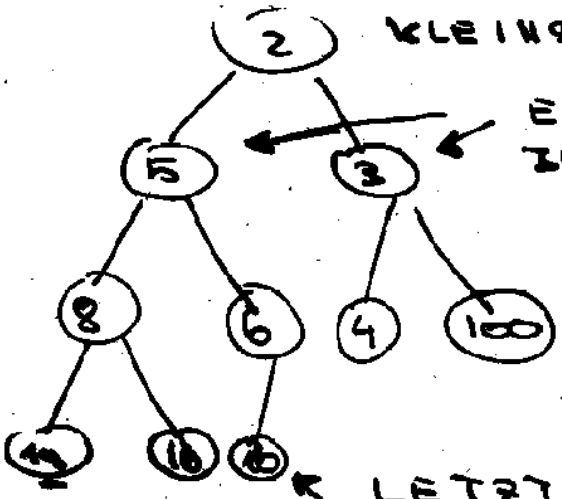
FAST VOLLSTÄNDIG.

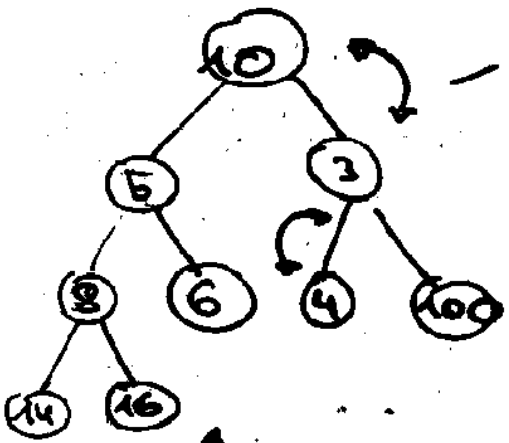
HEAPEIGENSCHAFT: FÜR ALLE U, V

U VORFAHR VON V → KEY[U] ≤ KEY[V].

BEISPIEL MINIMUM LÖSCHEN

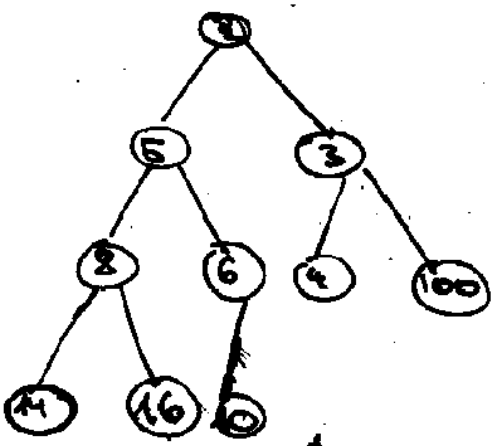
NUR WERTE: KEY[I]:

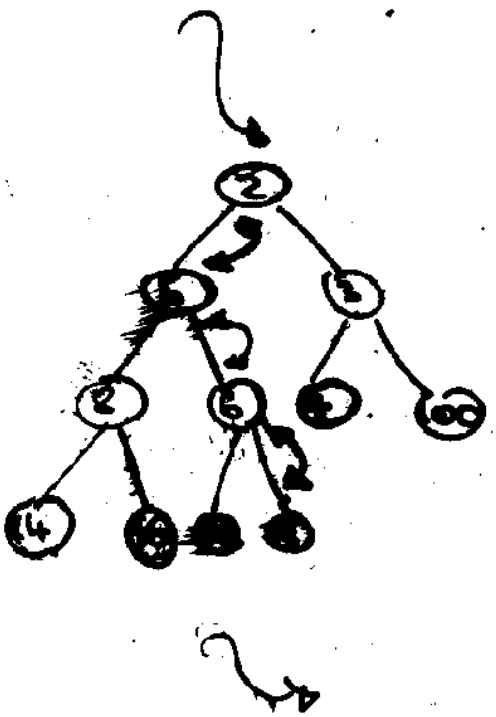




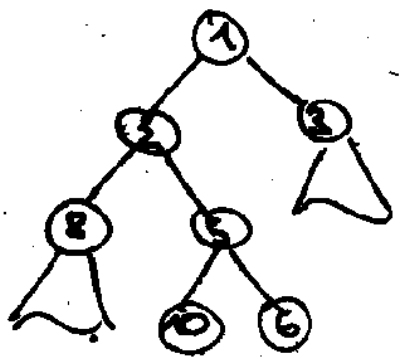
BEISPIEL EINFÜGEN

ELEMENT MIT WERT 1 EINFÜGEN





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) DURCHLÄUFE

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 }
}

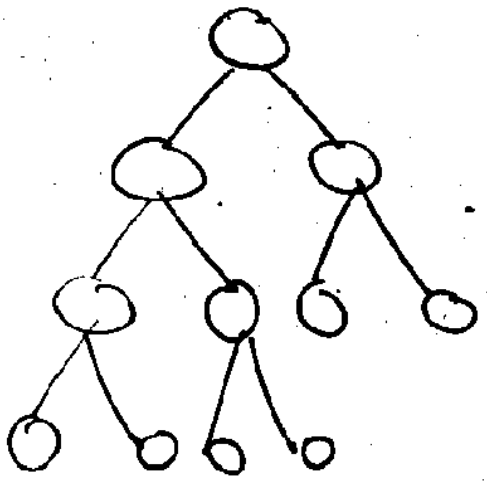
N ELEMENTE O(LOG N)

VERGLEICH DER MÖGLICHKEITEN
DER PRIORITY QUEUE

	MIN	DELETEMIN	INSERT
HEAP	1	LOG N	LOG N
ARRAY ODER LISTE	1	N (!)	1

· ELEMENT FINDEN: HEAP N, ARRAY 1, LISTE N.

HEAP ALS ARRAY Q[1,...,N]

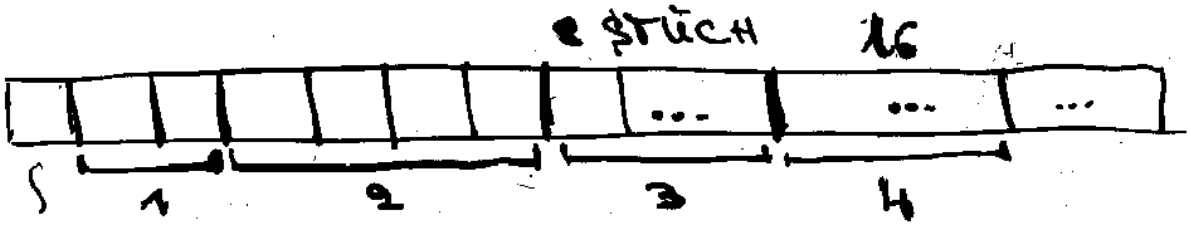


Q[1]

Q[2], Q[3]

Q[4] -- Q[7]

Q[8] ... Q[N]



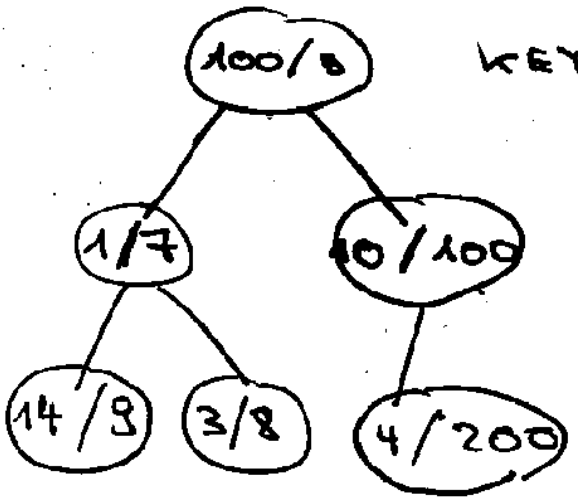
IN TIEFE ⊙

```
VATER := i // i INDEX aus 1,...,N
IF i > 1
   DANN i := ⌊i/2⌋ aus ℕ
```

LINKER SOHN := 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 (!) UNTERSTÜTZT.

Algorithmus (Prim mit Q in heap)

1. Wähle Startknoten r.

   for each $v \in Adj[r]$ {
      key[v] := $k(\{r,v\})$;
      konto[v] := r }
   Für alle übrigen $v \in V$ {
      key[v] := ∞; konto[v] := ∞ }
   Füge $V\{r\}$ mit Schlüsselwerten key[v]
   in heap Q ein.

2. while $Q \neq \emptyset$ {

3. w := Min$_Q$; DeleteMin$_Q$;
   T := T $\cup$ $\{$konto[w], w$\}$

4. for each $u \in Adj[w] \cap Q$ {
      if $k(\{u,w\}) <$ key[u] {
         key[u] := $k(\{u,w\})$;
         Q anpassen }
   }

}

Korrektheit mit zusätzlicher Invariante:

Für alle $w \in Q_\ell$

$key_\ell[w]$ = minimale Kosten eines Pfades $\langle v, w \rangle$ für $v \notin Q_\ell$

$key_\ell[w] = \infty$, wenn eine solche Kante nicht existiert.

Laufzeit

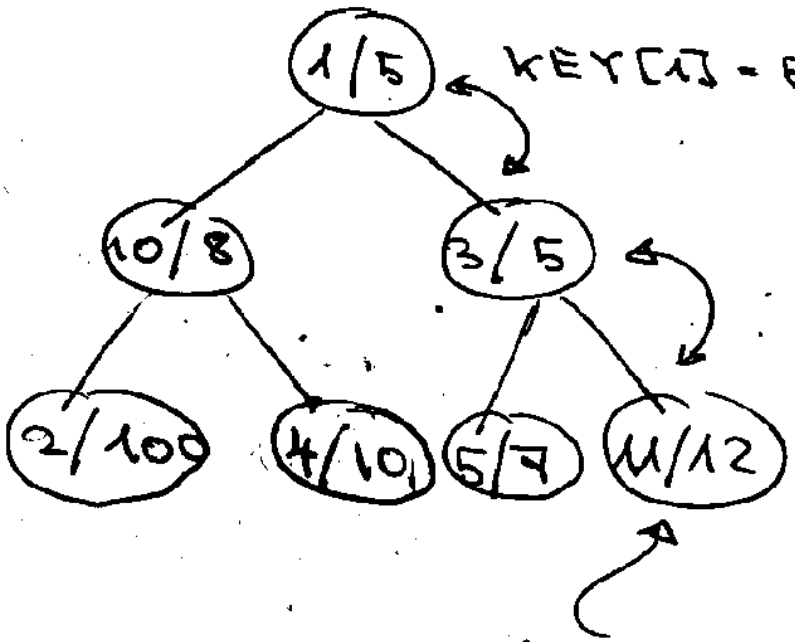
1. $O(n) + O(n \cdot \log n)$ für das Füllen von Q. (Füllen von Q in $O(n)$ – interessante Übungsaufgabe.)
2. $n-1$ Läufe
3. Einmal $O(\log n)$, insgesamt $O(n \cdot \log n)$
4. Insgesamt $O(|E|)$ + $|E|$-mal Anpassen von Q.

ANPASSEN VON HEAP Q

OPERATION

DECREASEKEY(V,S)

S < KEY[V] , NEUER SCHLÜSSEL



DECREASEKEY(11,1)

DECREASEKEY(V,S)

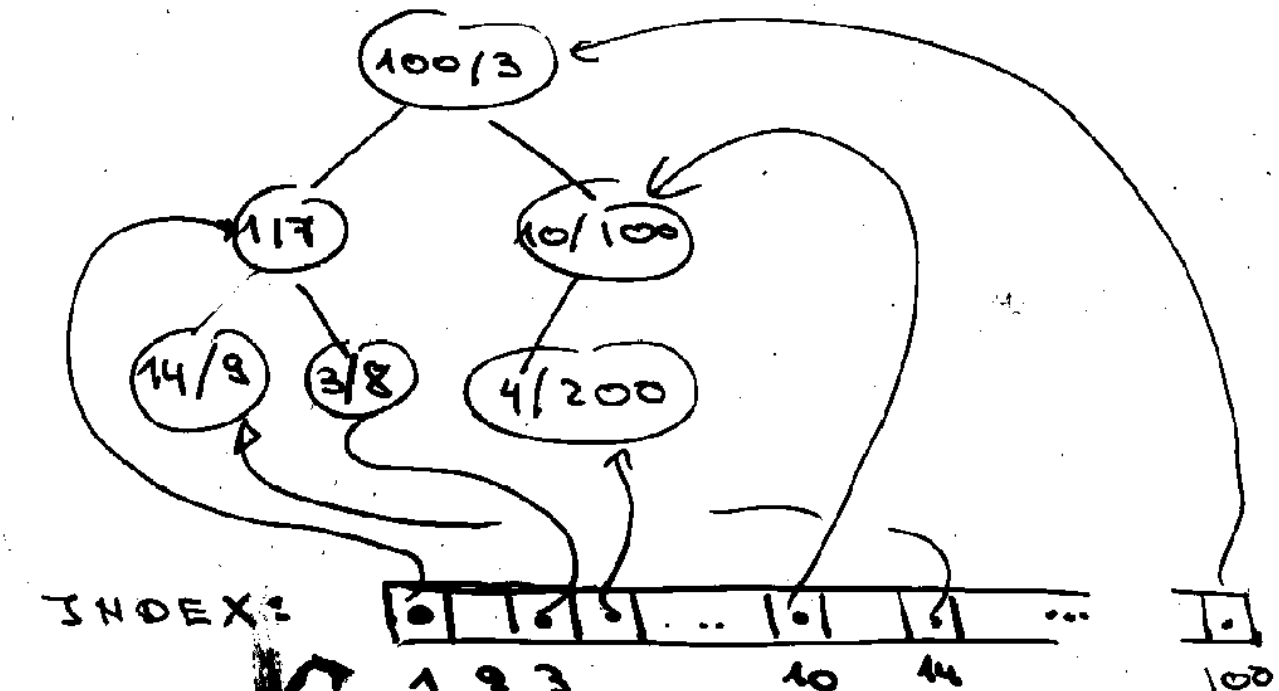
1. FINDE ELEMENT V IN Q
   // NICHT VON Q
   // UNTERSTÜTZT !

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

LAUFZEIT DECKER(V,S)

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

ZUM FINDEN: „INDEX DRÜBERLEGEN"



DIREKTE ADRESSEN

INDEX[1]=3
INDEX[3]=5
INDEX[14]=6

; INDEX[100]=1

FINDEN O(1).

FALLS GRUNDMENGE ZU GROß

DANN SUCHBAUM.

· FINDEN $O(\log N)$

Falls direkte Adressen dabei:

Einmaliges Kopieren von G

(und Index) $\Theta(\log|V|)$

$|E|$-maliges Kopieren $\Theta(|E|\log|V|)$

Dann Prim insgesamt

$\Theta(|E|\log|V|)$

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

(wie Kruskal mit Union by size.)