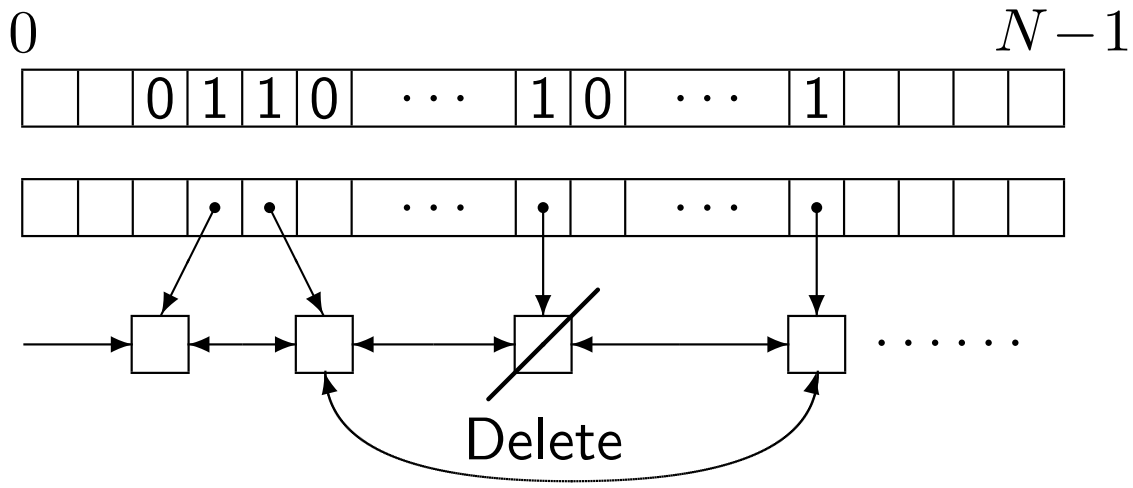
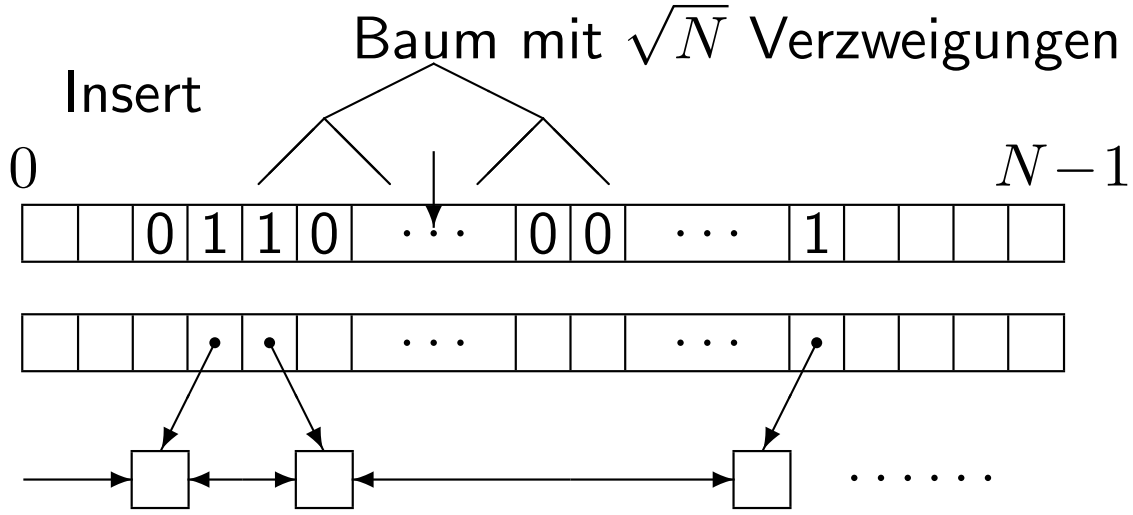
7.2 van Emde Boas-Priority Queues

Universum U, $|U| = N$, $U \subset \mathbb{N}$.
Hier: $U = \{0, 1, \ldots, N-1\}$.
Wir untersuchen hier eine bessere Priority Queue für den Fall $n \geq \log N$.


0
$N-1$
0 1 1 0 ··· 1 0 ··· 1
Delete


Baum mit $\sqrt{N}$ Verzweigungen
Insert
0
$N-1$
0 1 1 0 ··· 0 0 ··· 1

Notation

Sei

$$k \in \mathbb{N}, k \geq 2$$

$$k' = \left\lceil \frac{k}{2} \right\rceil$$

$$k'' = \left\lfloor \frac{k}{2} \right\rfloor$$

$$x \in [0..2^k - 1] \qquad (x \text{ hat } \leq k \text{ Bits})$$

$$x' = \left\lfloor \frac{x}{2^{k''}} \right\rfloor \qquad (x' \text{ vordere Hälfte von } x)$$

$$x'' = x \bmod 2^{k''} \qquad (x'' \text{ hintere Hälfte von } x)$$

Sei

$$S = \{x_1, \ldots, x_m\} \subseteq [0..2^k - 1]\,.$$

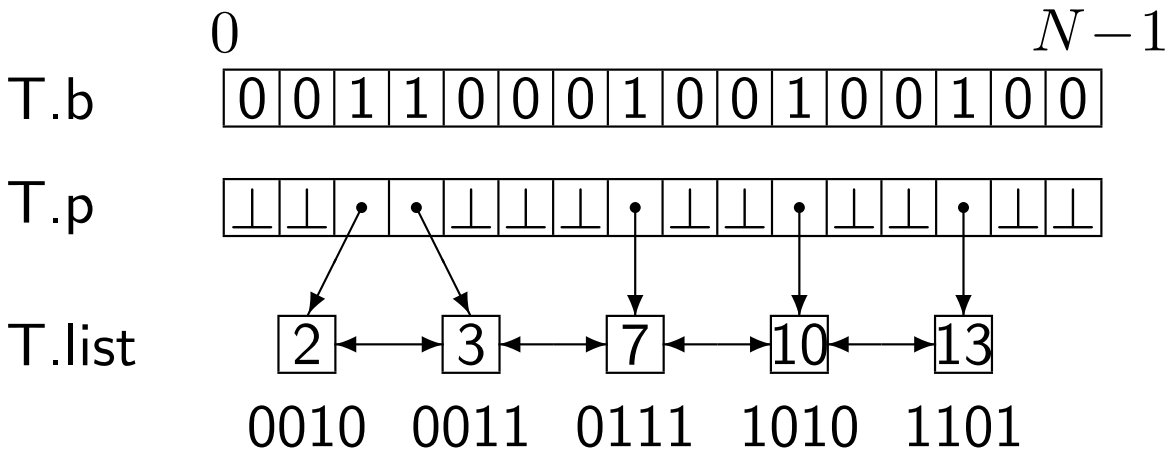
Definition 65

Eine k-Struktur T für S besteht aus:

1. der Zahl T.size $= |\{x_1, \ldots, x_m\}| = |S| = m$;
2. einer doppelt verketteten Liste T.list, die die Elemente von S in aufsteigender Reihenfolge enthält;
3. einem Bitvektor T.b$[0..2^k - 1]$ mit T.b$[i] = \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$, falls $\begin{smallmatrix} i \notin S \\ i \in S \end{smallmatrix}$ einem Zeiger (Pointer) Vektor T.p$[0 \ldots 2^k - 1]$. Falls T.b$[i] = 1$, dann zeigt T.p$[i]$ auf i in der Liste aus 2.
4. einer k'-Struktur T.top und einem Feld T.bottom$[0 \ldots 2^{k'} - 1]$ von k''-Strukturen. Falls $m = 1$, dann T.top, T.bottom und die zugehörigen k''-Strukturen leer, T.size $= 1$. T.list $= \{x\}$, der Bitvektor wird nicht benötigt. Falls $m > 1$, dann ist T.top eine k'-Struktur für die durch $\{x'_1, x'_2, \ldots, x'_m\}$ gegebene Menge, und für jedes $y \in [0 \ldots 2^{k'} - 1]$ ist T.bottom$[y]$ eine k''-Struktur für die Menge $\{x''_i;\ 1 \leq i \leq m;\ y = x'_i\}$

Beispiel 66

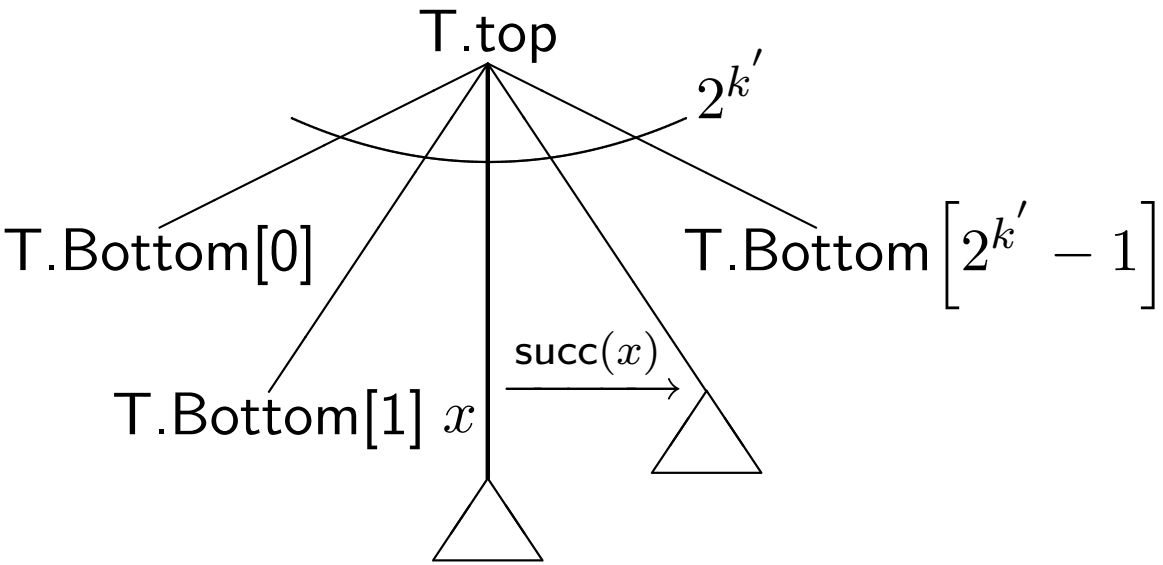
$k = 4$, $S = \{2, 3, 7, 10, 13\}$, T.size $= 5$:



T.top ist 2-Struktur für $\{0, 1, 2, 3\}$
T.bottom$[0]$ ist eine 2-Struktur für $\{2, 3\}$
T.bottom$[1]$ ist eine 2-Struktur für $\{3\}$
T.bottom$[2]$ ist eine 2-Struktur für $\{2\}$
T.bottom$[3]$ ist eine 2-Struktur für $\{1\}$

Operation *Succ*(x) findet $\min\{y \in S;\ y > x\}$ in der k-Struktur T.

if $k = 1$ **or** T.Size ≤ 2 **then**
  naive Suche
elif $x \geq \max$ in T.list **then**
    return *Succ*(x) gibt's nicht
else
  $x' := \left\lfloor \frac{x}{2^{k''}} \right\rfloor$;
  $x'' := x \bmod 2^{k''}$;
  if T.top.b$[x'] = 1$ **and** $x'' < \max\{$T.bottom$[x']\}$ **then**
    return $x' \cdot 2^{k''} +$ *Succ*$(x'',$ *T.bottom*$[x'])$
  else
    $z' :=$*Succ*$(x',$ *T.top*$)$; **return** $z' \cdot 2^{k''} + \min\{$T.bottom$[z']\}$
  fi
fi


T.top
$2^{k'}$
T.Bottom[0]
T.Bottom $\left[2^{k'}-1\right]$
succ(x)
T.Bottom[1] x

Kosten:

$$\mathsf{T}(k) \leq c + \mathsf{T}(\left\lceil \frac{k}{2} \right\rceil) = \mathcal{O}(\log k)\,.$$

Lemma 67

Die Succ-Operation hat Zeitbedarf $\mathcal{O}(\log k)$.

Beweis:

$\surd$ □

Insert Operation:

- falls x bereits in Priority Queue, dann fertig
- bestimme *Succ*(x, T), füge x davor ein
- bestimme x' und x''
- behandle die entsprechenden Unterstrukturen rekursiv: *Insert*(x',$T.top$), *Insert*(x'',$T.bottom[x']$) (nur ein nicht trivialer rekursiver Aufruf)

Zeitbedarf: naiv $\mathcal{O}(\log^2 k)$, Optimierung: das oberste *Succ* tut alles $\Rightarrow \mathcal{O}(\log k)$.

Delete Operation:
Komplexität von *Delete* in k-Struktur: $\mathcal{O}(\log k)$

Kosten der Initialisierung: $\sim$ Größe der Datenstruktur

Platzbedarf für k-Struktur: Sei $S(k)$ der Platzbedarf für eine k-Struktur. Dann gilt:

$$S(1) = c$$
$$S(k) = c2^k + S(k') + 2^{k'} S(k'') \text{ für } k \geq 2$$

Wir ersetzen zur Vereinfachung:

$$S(k) = c2^k + S(\frac{k}{2}) + 2^{\frac{k}{2}} S(\frac{k}{2})$$

Lemma 68

$$S(k) = \mathcal{O}(2^k \cdot \log k)$$

Beweis:

Zeige: $\mathsf{S}(k) := c'2^k \log k$ funktioniert.

Für $k = 1$ ist das klar.

Beweis (Forts.):

Rekursionsschritt:

$$\begin{aligned}
\text{Platz für } k\text{-Struktur} &\leq c2^k + c'2^{\frac{k}{2}}(\log k - 1) + 2^{\frac{k}{2}}c'2^{\frac{k}{2}}(\log k - 1) \\
&= c2^k + c'2^{\frac{k}{2}}(1 + 2^{\frac{k}{2}})(\log k - 1) \\
&= c2^k + c'2^{\frac{k}{2}}(2^{\frac{k}{2}}\log k + \log k - 2^{\frac{k}{2}} - 1) \\
&\leq c2^k + c'2^{\frac{k}{2}}(2^{\frac{k}{2}}\log k + \log k - 2^{\frac{k}{2}}) \\
&\leq c'2^k \log k\,, \text{ falls} \\
c2^k + c'2^k \log k + c'2^{\frac{k}{2}}\log k - c'2^k &\leq c'2^k \log k \\
\Leftrightarrow c'2^{\frac{k}{2}}\log k &\leq (c' - c)2^k \\
\Leftrightarrow \frac{c' - c}{c'} &\geq \frac{\log k}{2^k} \text{ (gilt für } c' \text{ groß genug.)}
\end{aligned}$$

□

Satz 69

Sei $N = 2^k$, Universum $U = \{0, \ldots, N-1\}$. Wird eine Teilmenge $S \subseteq U$ durch eine k-Struktur dargestellt, dann benötigen die Operationen Insert, Delete und FindMin jeweils Zeit $\mathcal{O}(\log\log N)$, die Initialisierung Zeit $\mathcal{O}(N \log\log N)$. Der Platzbedarf ist ebenfalls $\mathcal{O}(N \log\log N)$.

Beweis:

s.o. □

Literatur zu van Emde Boas-Priority Queue:

Kurt Mehlhorn:
Data structures and algorithms 1: Sorting and searching, pp. 290–296,
EATCS Monographs on Theoretical Computer Science,
Springer Verlag: Berlin-Heidelberg-New York-Tokyo, 1984

P. van Emde Boas, R. Kaas, E. Zijlstra:
Design and implementation of an efficient priority queue
Math. Systems Theory 10 (1976), pp. 99–127