Radix Heaps

Ziel: Reduziere amortisierte Laufzeit der langsamsten Zugriffsfunktion von $\mathcal{O}(\sqrt{C})$ bei 2–level–Buckets auf $\mathcal{O}(\log C)$.

Voraussetzungen an die Schlüssel:

- Die Schlüssel sind nicht-negative ganze Zahlen.
- Zu jedem Zeitpunkt gilt:

  maximaler Schlüssel - minimaler Schlüssel $\leq C$.
- Wenn durch `extractMin` ein Element mit Schlüssel k ausgegeben wurde, so werden danach nur noch Elemente mit Schlüssel $\geq k$ eingefügt. *(Monotonie des Heaps)*

Datenstrukturen:

Sei $B = \lceil \log(C+1) \rceil + 1$.

Buckets:	`b[1..B]`
Schranken der Buckets:	`u[0..B]`
Bucket-Nummern der Elemente:	`b_no[x]`, wobei `x` ein beliebiges Element des Heaps ist

Invarianten:

Während der Lebenszeit des Heaps müssen mehrere Invarianten erfüllt bleiben:

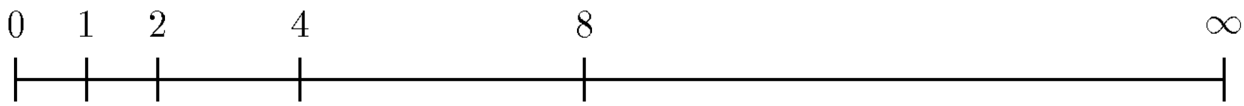
1. Einordnung der Elemente in die Buckets:
   $\forall 1 \leq i \leq B: \;$ `u[i-1]` $\leq$ key(x) $<$ `u[i]` für alle Elemente x in `b[i]`

2. Exponentielles Wachstum der Bucketgrößen:
   - `u[0]` = `u[1]`-1
   - $\forall\, 1 \leq i < B - 1: \quad 0 \leq$ `u[i+1]`-`u[i]` $\leq 2^{i-1}$

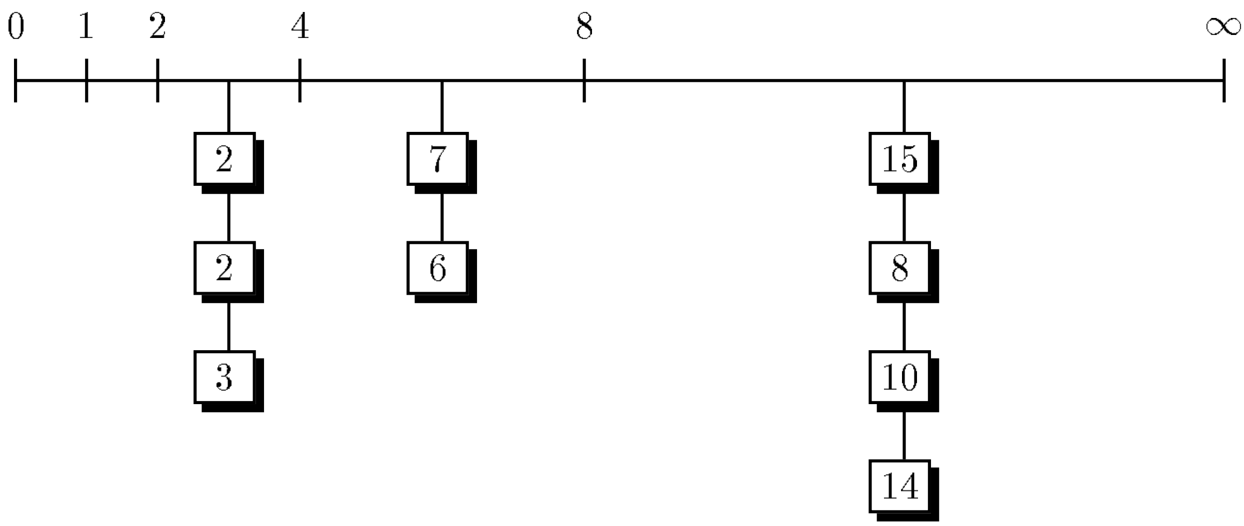
3. Monotonie der Schlüssel: Es werden immer nur Elemente x mit key(x) $\geq$ `u[0]` eingefügt

Initialize

```
// Alle Buckets sind leer
for (i = 1; i++; i ≤ B) {
        b[i] = empty;
}
// Bucketgrenzen setzen
u[0] = 0; u[1] = 1;
for (i = 2; i++; i < B) {
        u[i] = u[i-1] + 2^(i-2)
}
u[B] = ∞;
```



Ein leerer Radix–Heap



Beispiel eines gefüllten Heaps

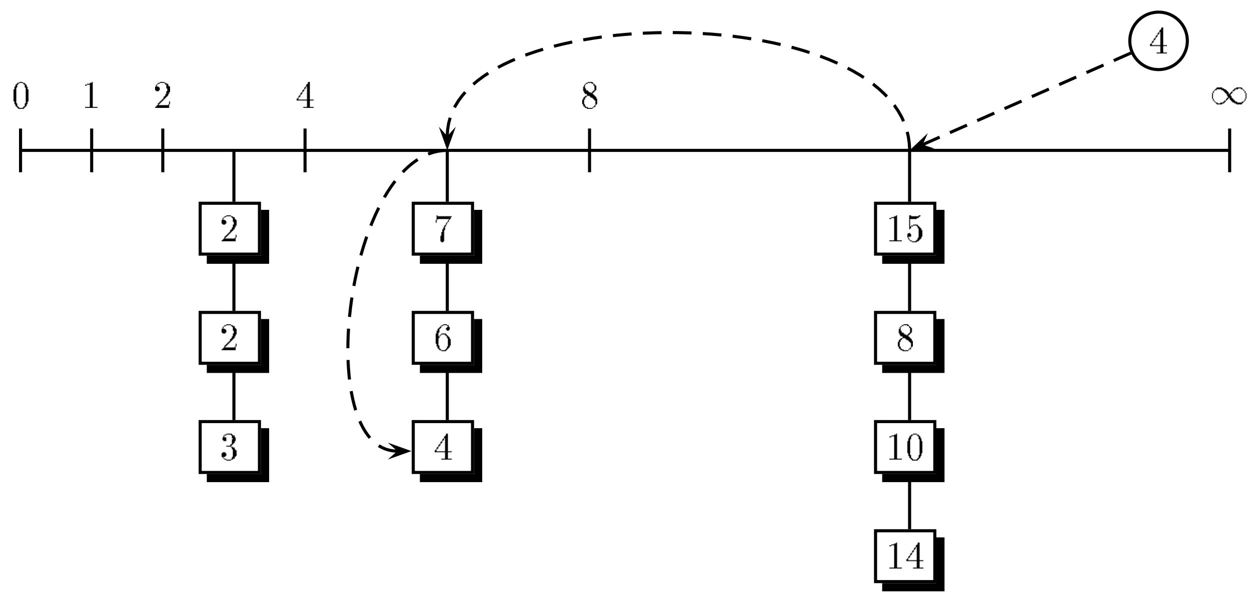
Einfügen eines neuen Elementes

Insert(x)

i = B;

while (u[i-1] $>$ key(x)) $i--$;

Füge x in b[i] ein;



Einfügen eines Elements mit Schlüssel „4“

Erniedrigen eines Schlüssels

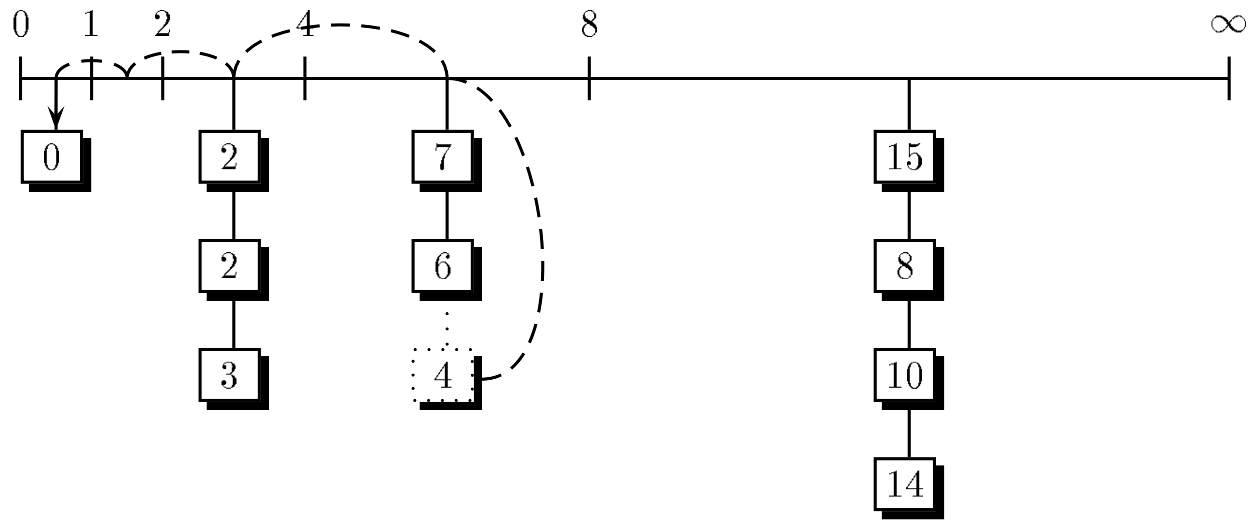
DecreaseKey(x)

i = b_no[x];

Entferne x aus b[i];

while (u[i-1] $>$ key(x)) i$--$;

Füge x in b[i] ein;



Erniedrigen des Elements „4“ auf „0“

Entfernen des kleinsten Elementes

<u>ExtractMin</u>

i = 0;

while (b[i] == empty) i++;

Bestimme kleinsten Schlüssel k in b[i];

// Neuer Heap beginnt bei $k \rightarrow$ Grenzen u[.] neu setzen

u[0] = k; u[1] = k+1;

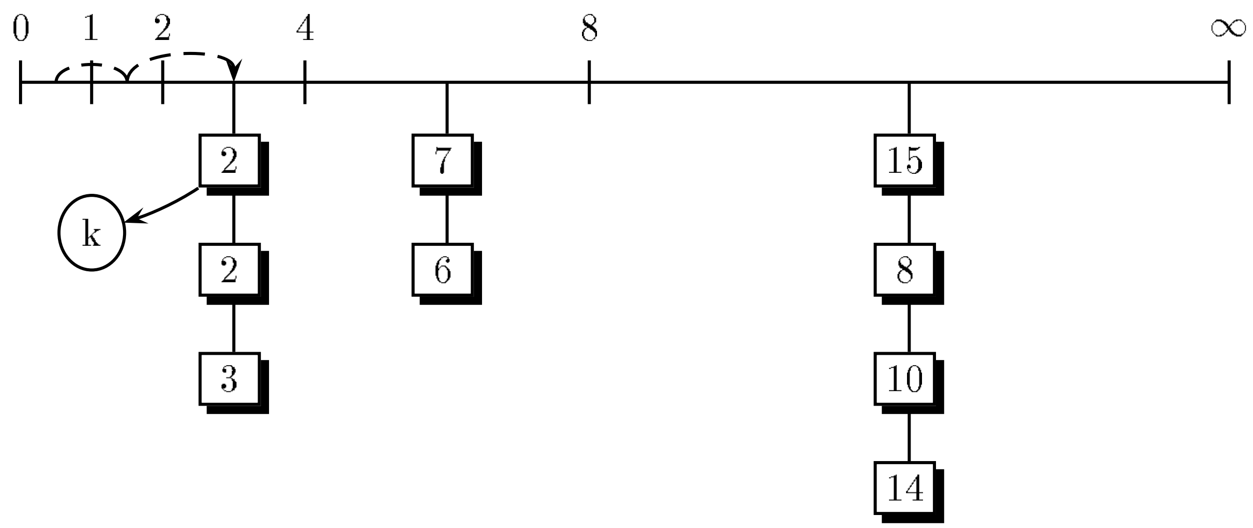
for (j=2; j++; j $\leq$ i) $\{$

  u[j] = min$\{$u[j-1]+2^{j-2}; u[i+1]$\}$

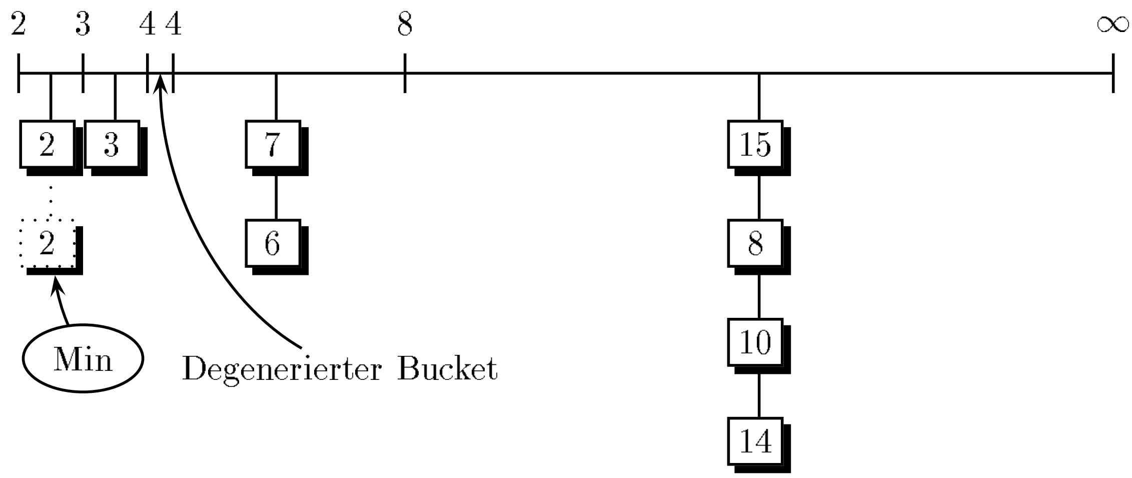
$\}$

Verteile alle Elemente von b[i] auf b[0],. . .,b[i-1];

Entferne ein beliebiges Element von b[0] und gib es aus;



`ExtractMin`: Suche nach dem kleinsten Schlüssel



`extractMin`: Reorganisation des Heaps

Satz: Ausgehend von einem leeren Heap beträgt die *worst–case*–Laufzeit von k `Insert`- und l `DecreaseKey`-/`ExtractMin`–Operationen in einem Radix–Heap $\mathcal{O}(k \log C + l)$.

Beweisidee: amortisierte Analyse

Wdh:

amortisierte Laufzeit $=$ tatsächliche Laufzeit + Potentialzunahme

Gesamtlaufzeit einer Folge von Operationen:

$$\sum_i \langle \text{amort. Laufz. } i\text{ter Oper.} \rangle = \sum_i \langle \text{tats. Laufz. } i\text{ter Oper.} \rangle + \Phi_{Ende} - \Phi_{Anfang}$$

Also:

$$\begin{aligned} \sum_i \langle \text{tats. Laufz. } i\text{ter Oper.} \rangle &= \sum_i \langle \text{amort. Laufz. } i\text{ter Oper.} \rangle + \underbrace{\Phi_{Anfang}}_{=0, leerer\ Heap} - \underbrace{\Phi_{Ende}}_{\geq 0} \\ &\leq \sum_i \langle \text{amort. Laufz. } i\text{ter Oper.} \rangle \end{aligned}$$

Hier: Potential eines Elementes x: $\phi(x) = c \cdot \texttt{b\_no}[x]$, für ein geeignetes c (s.u.)

Potential des Heaps: $$\Phi(H) = \sum_{x \in H} \phi(x)$$

Zu zeigen:

amortisierte Laufzeit von

`Insert`	ist $\mathcal{O}(\log C)$
`ExtractMin`	ist $\mathcal{O}(1)$
`DecreaseKey`	ist $\mathcal{O}(1)$