Karlsruher Institut für Technologie

Übung 2 – Algorithmen II

Hans-Peter Lehmann, Daniel Seemaier – *{hans-peter.lehmann, daniel.seemaier}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS21.php`

Institut für Theoretische Informatik - Algorithmik II

Inhalt

Themenübersicht für heute

- Ganzzahlige *Priority Queues*
  - *Bucket Queues*
  - *Radix Heaps*
- *Average case* Analyse am Beispiel MST

Spezielle *Priority Queues*

monoton, ganzzahlig

Warum das Ganze?

- spezialisierte Datenstruktur $\longrightarrow$ schneller
- Dijkstras Algorithmus

Idee

- speichere Schlüssel in *Buckets* statt Bäumen

Spezielle *Priority Queues*

Dijkstras Algorithmus

Laufzeit Dijkstras Algorithmus

- $T_{Dijkstra} = O(m \cdot T_{decreaseKey} + n \cdot (T_{deleteMin} + T_{insert}))$

amortisierte Laufzeiten

$O(\cdot)$	*Bin. Heap*	*Fib. Heap*	*Bkt. Queue*	*Radix Heap*
$T_{decreaseKey}$	$\log n$	1	1	1
T_{insert}	$\log n$	1	1	$\log C$
$T_{deleteMin}$	$\log n$	$\log n$	C	$\log C$
$T_{Djkstra}$	$(m+n)\log n$	$m + n \log n$	$m + nC$	$m + n \log C$

Spezielle *Priority Queues*

monoton, ganzzahlig

Bedingungen

- positive, ganzzahlige Schlüssel
- neue/geänderte Schlüssel $k \geq$ minimaler Schlüssel min
- maximales Schlüsselinkrement $C \longrightarrow min \leq k \leq min + C$

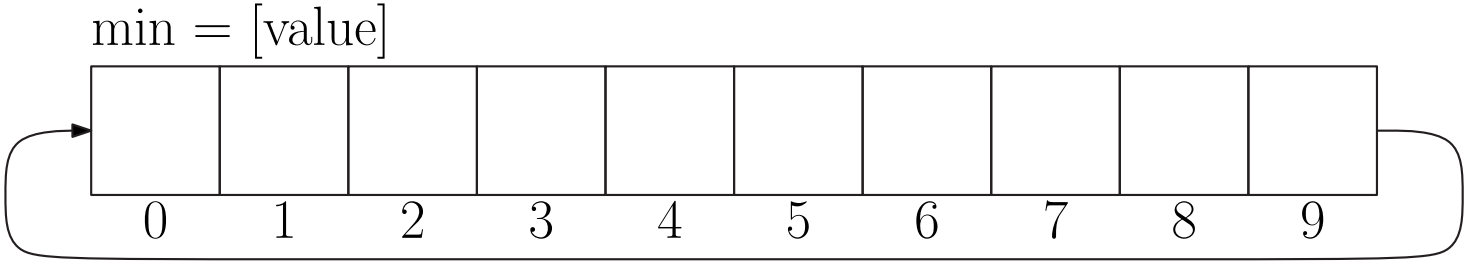
Bucket Queues

Aufbau:

- zirkuläre Liste aus Buckets $B[\cdot]$
- Variable mit minimalem Schlüssel *min*
  (der in die Datenstruktur aufgenommen werden kann)
- *min*: letzter `deleteMin`-Schlüssel, initialisiert mit 0

gegeben:

- max. Schlüsselinkrement $C \longrightarrow$ # Buckets $|B| = C + 1$



$$C = 9 \longrightarrow |B| = C + 1 = 10$$

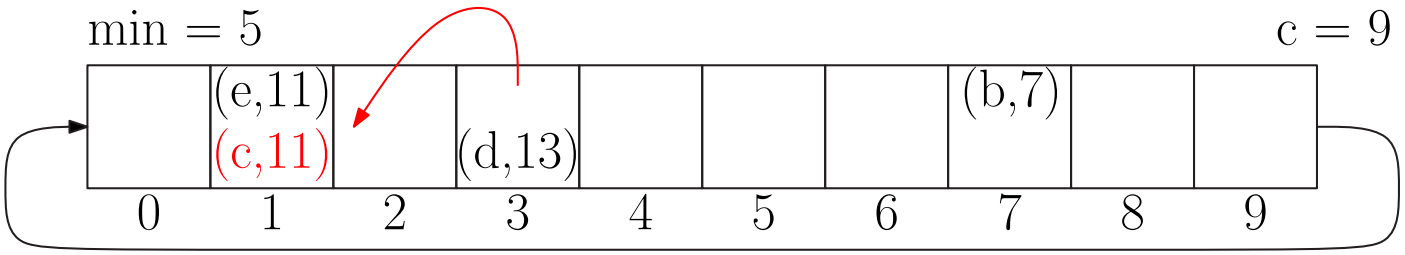
Bucket Queues

Ablauf:

`insert`:	Einfügen in Bucket $B[key \bmod (C+1)]$	$O(1)$
`deleteMin`:	Entfernen aus Bucket $B[min \bmod (C+1)]$	$O(C)$
	(bzw. aus erstem nicht-leeren Folgebucket)	
`decreaseKey`:	Verschieben von altem in neuen Bucket	$O(1)$

Beispiel:

- `insert(a,5)`
- `insert(b,7)`
- `deleteMin()`
- `insert(c,13)`
- `insert(d,13)`
- `insert(e,11)`
- `decreaseKey(c,11)`



Spezielle *Priority Queues*

Dijkstras Algorithmus

Laufzeit Dijkstras Algorithmus

- $T_{Dijkstra} = O(m \cdot T_{decreaseKey} + n \cdot (T_{deleteMin} + T_{insert}))$

amortisierte Laufzeiten

$O(\cdot)$	*Bin. Heap*	*Fib. Heap*	*Bkt. Queue*	*Radix Heap*
$T_{decreaseKey}$	$\log n$	1	1	1
T_{insert}	$\log n$	1	1	$\log C$
$T_{deleteMin}$	$\log n$	$\log n$	C	$\log C$
$T_{Djkstra}$	$(m+n)\log n$	$m + n \log n$	$m + nC$	$m + n \log C$

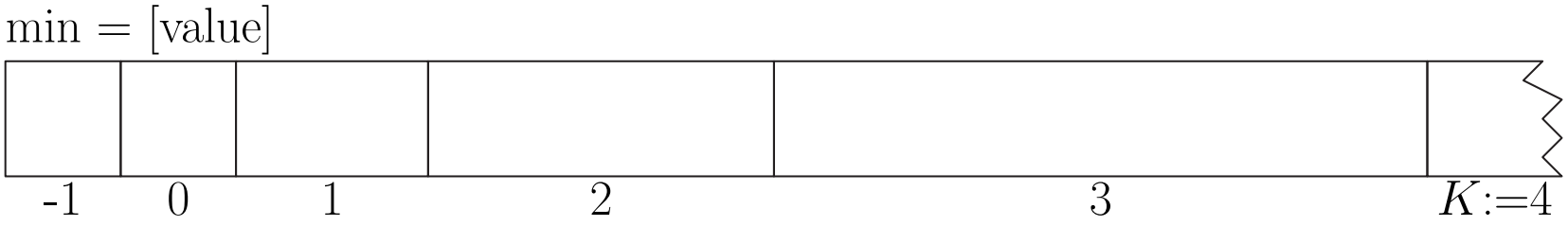
Radix Heaps

Aufbau:

- Liste aus logarithmischer Anzahl Buckets $B[\cdot]$
- Variable mit minimalem Schlüssel *min*
  (der in die Datenstruktur aufgenommen werden kann)

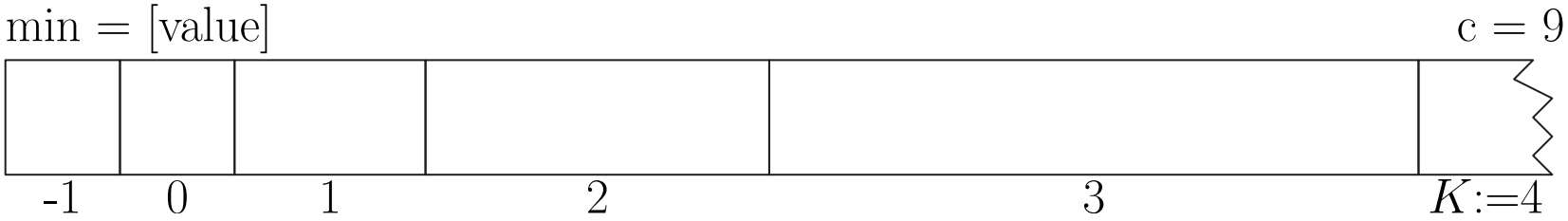
gegeben:

- max. Schlüsselinkrement $C \longrightarrow$ # Buckets $|B| = K + 2$
  $= (\lfloor \log C \rfloor + 1) + 2$



$$C = 9 \longrightarrow K + 2 = (\lfloor \log C \rfloor + 1) + 2 = (3 + 1) + 2 = 6$$

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

Bucket i: hält Elemente mit $i = \min(\mathrm{msd}(min, key), K)$

$\rightarrow$ Bucket K ist *overflow* Bucket

$\rightarrow$ höchstwertiges unterschiedliches Bit ist i

(bzw. -1 wenn $key = min$)

$\rightarrow$ $\max(1,\ 2^i)$ verschiedene Schlüssel

$\rightarrow$ leer, wenn Bit i von min gesetzt

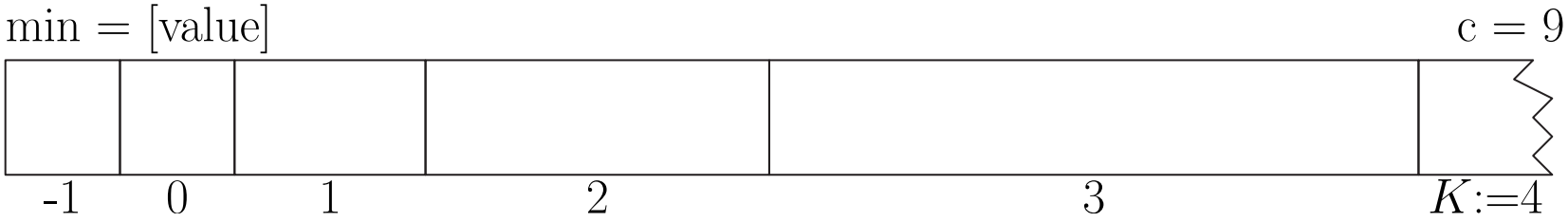
Beispiele

$min := \texttt{00001000}\ (08)$

$\mathrm{msd}(\texttt{00001000},\ \texttt{01001000}) = 6$

$\mathrm{msd}(\texttt{00001000},\ \texttt{00001010}) = 1$

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
1000 (08)	8	9	10..11	12..15	-	16..8+C
1010 (10)	10	11	-	12..15	-	16..10+C
1111 (15)	15	-	-	-	-	16..15+C
1000100 (68)	68	69	70..71	-	72..79 ?	80..68+C ?
1000100 (68)	68	69	70..71	-	72..77 !	- !

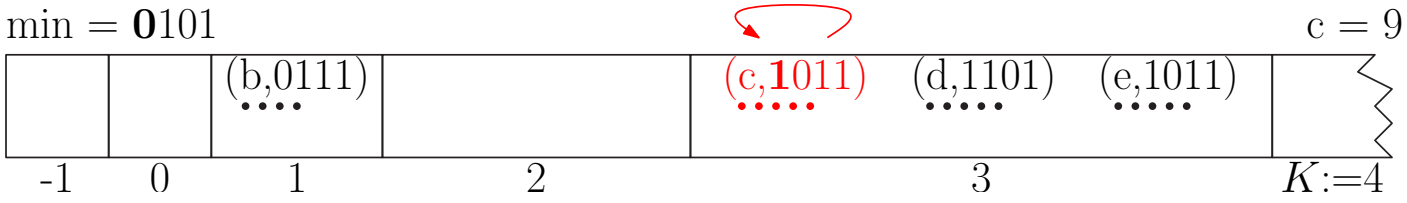
Radix Heaps

Ablauf und Laufzeiten (amortisiert)*:*

`insert`:	Einfügen in Bucket $B[\min(\mathrm{msd}(min, key), K)]$	$O(\log C)$
`deleteMin`:	$min =$ minimaler Schlüssel (aus Bucket i),	$O(\log C)$
	Elemente aus Bucket i verschieben,	
	Entfernen aus Bucket $B[-1]$	
`decreaseKey`:	Verschieben von altem in neues Bucket	$O(1)$

Beispiel:

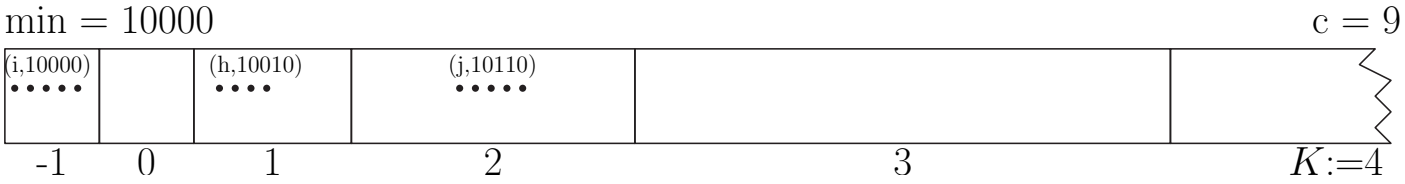
- `insert(a,5)`
- `insert(b,7)`
- `deleteMin()`
- `insert(c,13)`
- `insert(d,13)`
- `insert(e,11)`
- `decreaseKey(c,11)`



Radix Heaps

Beispiel (fortgesetzt):

- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Warum komplizierte Formel $\min(\mathrm{msd}(\mathit{min}, \mathit{key}), K)$ *?*

- viel anschaulicher wäre doch

$$i = \lceil \log(\mathit{key} - \mathit{min}) \rceil$$

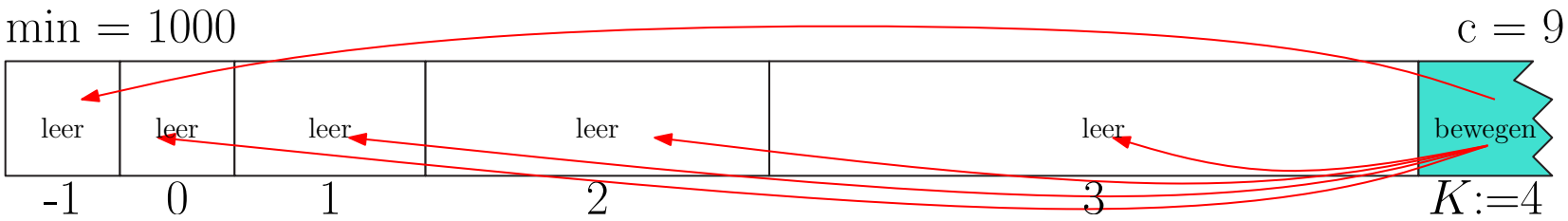
- okay, Anzahl Verschiebungen erhöht sich nicht ...
- ... aber, jede Änderung von *min* ändert potentiell alle Buckets
  $\rightarrow$ schlechtere Laufzeit!

Beispiel

$\mathit{min} := \texttt{01000}\ (08) \xrightarrow{\text{deleteMin}} \mathit{min} := \texttt{01010}\ (10)$

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
1000 (08)	8	9	10	11..12	13..16	17..8+C
1010 (10)	10	11	12	13..14	15..18	19..10+C

Radix Heaps



Änderung von min $\longrightarrow$ *Umverteilung eines Bucket genügt*

min := 01000 (08) $\xrightarrow{\text{deleteMin}}$ *min* :=
01001 (09)0101∗ (10..11)011 ∗ ∗ (12..15)−, *Bucket* 3 *ist leer* 1 ∗ ∗ ∗ ∗ (16..)

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..15	-	$\geq$ 16
01001 (09)	9	-	10..11	12..15	-	$\geq$ 16
01010 (10)	10	11	-	12..15	-	$\geq$ 16
01011 (11)	11	-	-	12..15	-	$\geq$ 16
01100 (12)	12	13	14..15	-	-	$\geq$ 16
01101 (13)	13	-	14..15	-	-	$\geq$ 16
01110 (14)	14	15	-	-	-	$\geq$ 16
01111 (15)	15	-	-	-	-	$\geq$ 16
10000 (16)	16	17	-	-	-	-
10001 (17)	17	-	-	-	-	-

Spezielle *Priority Queues*

Dijkstras Algorithmus

Laufzeit Dijkstras Algorithmus

- $T_{Dijkstra} = O(m \cdot T_{decreaseKey} + n \cdot (T_{deleteMin} + T_{insert}))$

amortisierte Laufzeiten

$O(\cdot)$	*Bin. Heap*	*Fib. Heap*	*Bkt. Queue*	*Radix Heap*
$T_{decreaseKey}$	$\log n$	1	1	1
T_{insert}	$\log n$	1	1	$\log C$
$T_{deleteMin}$	$\log n$	$\log n$	C	$\log C$
$T_{Djkstra}$	$(m+n)\log n$	$m + n \log n$	$m + nC$	$m + n \log C$

Average case Analyse für MST

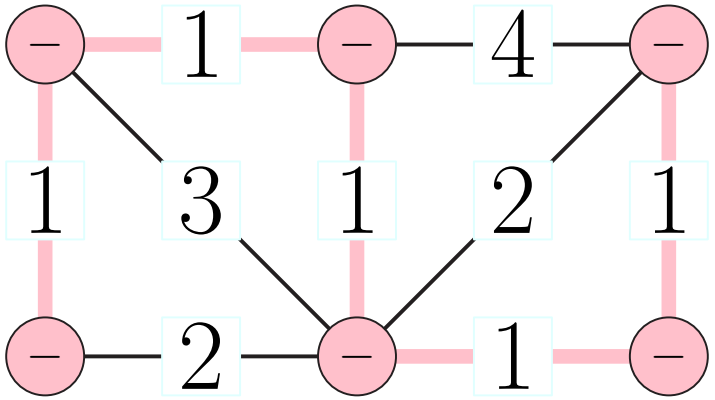
Motivation

- Algorithmus zur Berechnung minimaler Spannbäume (MSTs)
  (siehe Algorithmen I: Algorithmus von Jarnik-Prim)
- Beweis analog zu Dijkstras Algorithmus

Kurze Wiederholung von Jarnik-Prim

- Wähle beliebigen Startknoten s
- Füge iterativ Knotem v zu MST mit kleinstem Abstand $d[v]$

$\rightarrow$ Verwalte vorläufige Abstände $d[\cdot]$ in *Priority Queue*!



Average case Analyse für MST

Durchschnittliche Laufzeit Jarnik Prim Algorithmus

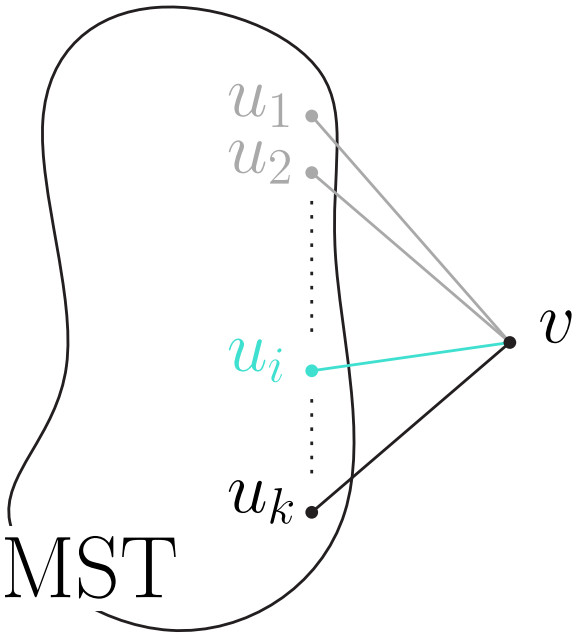
- Annahme: Gewichte $1, \ldots, m$ sind Kanten zufällig zugeordnet.
- $T_{Prim} = O(m + n \cdot (T_{insert} + T_{deleteMin} + T_{insertMST}) + x \cdot T_{decreaseKey})$

Average case Analyse für MST

Durchschnittliche Anzahl `decreaseKey` *Operationen*

- betrachte Knoten v:
  `decreaseKey(v)`, wenn Knoten u zu MST hinzugefügt wird mit $c(u, v) < d[v]$
  $\rightarrow$ Reihenfolge $u_1, \ldots, u_k$
  $\rightarrow$ $c(u_i, v) < \min_{j<i} c(u_j, v)$

- Wie oft tritt dies auf?
  $\rightarrow$ Erwartungswert M_k für Anzahl Präfixminima
  $\rightarrow$ $\mathbb{E}(M_k) = H_k$ (k'te harmonische Zahl)



Average case Analyse für MST

Präfixminima

Zufallsvariablen

- M_k: Anzahl Präfixminima einer Folge von k Zahlen
- I_i: $I_i = 1$ gdw. i-te Zahl Präfixminimum (Indikator) $\longrightarrow \mathbb{P}[I_i = 1] = \frac{1}{i}$

$$\begin{aligned}\mathbb{E}(M_k) &= \mathbb{E}(\sum_{i=1}^{k} I_i) = \sum_{i=1}^{k} \mathbb{E}(I_i) \\ &= \sum_{i=1}^{k} \frac{1}{i} = H_k\end{aligned}$$

Intuition

- Wahrscheinlichkeit, dass i-tes Element Präfixminimum ist, ist $\frac{1}{i}$, da nur ein Minimum in den ersten i Elementen existiert.

Average case Analyse für MST

Durchschnittliche Anzahl `decreaseKey` *Operationen (weiter)*

- erwartet H_k Präfixminima
  - Erstes Minimum $\rightarrow$ `insert(v)`
  - $H_k - 1$ Minima $\rightarrow$ `decreaseKey(v)`
    (Bemerkung: $H_k - 1 < \ln k = \ln \mathrm{grad}(v)$)

- Gesamt:

$$\begin{aligned} x &:= \sum_{v \in V} (H_k - 1) = \sum_{v \in V} (H_{\mathrm{grad}(v)} - 1) \\ &< \sum_{v \in V} \ln \mathrm{grad}(v) \overset{konkav}{\leq} n \ln \frac{2m}{n} = O(n \ln \frac{m}{n}) \end{aligned}$$

Ende!

Feierabend!