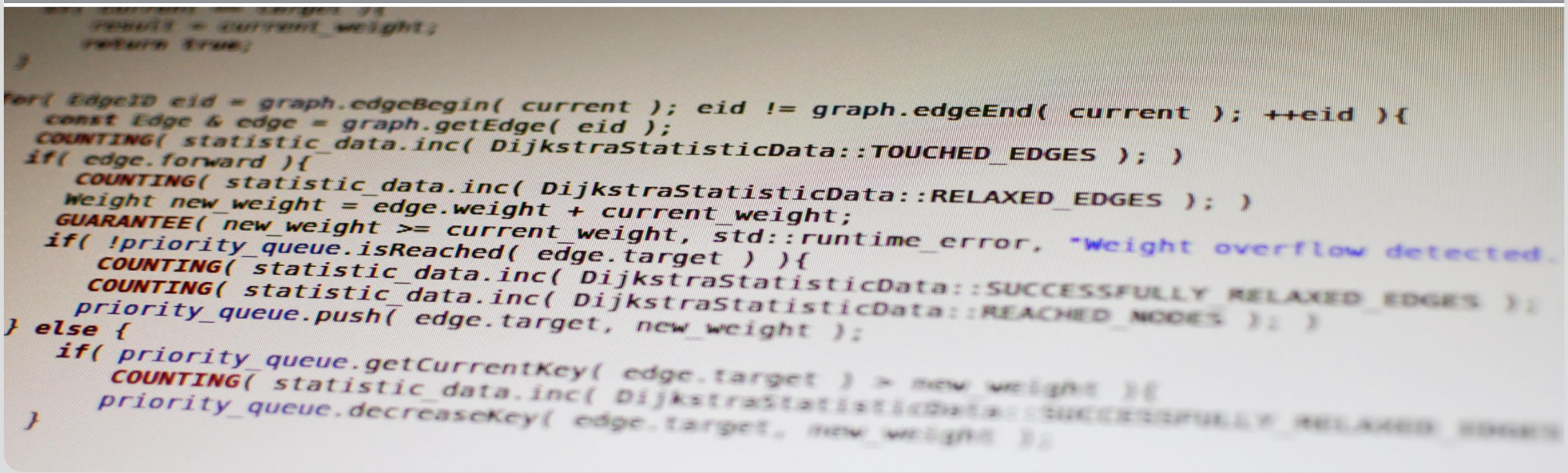
Karlsruher Institut für Technologie

Übung 2 – Algorithmen II

Moritz Laupichler, Nikolai Maas – *{moritz.laupichler, nikolai.maas}@kit.edu*

`https://algo2.iti.kit.edu/AlgorithmenII_WS23.php`

Institut für Theoretische Informatik - Algorithm Engineering



Inhalt

Themenübersicht für heute

- Ganzzahlige *Priority Queues*
  - *Bucket Queues*
  - *Radix Heaps*
- Bidirektionale Suche
- A*-Suche
- All-to-All Suche

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	$\log C$
$T_{Dijkstra}$	$(m+n)\log n$	$m + n\log n$	$m + nC$	$m + n\log C$

Spezielle *Priority Queues*

monoton, ganzzahlig

Bedingungen

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

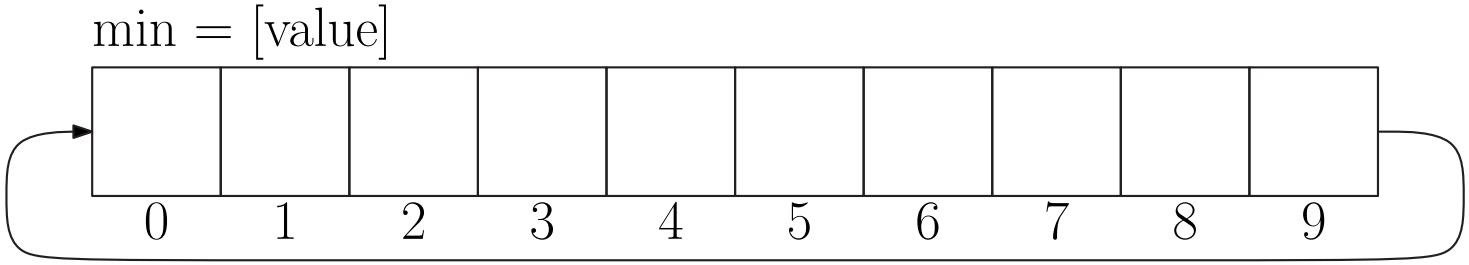
Bucket Queues

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 $|B| = C + 1$



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

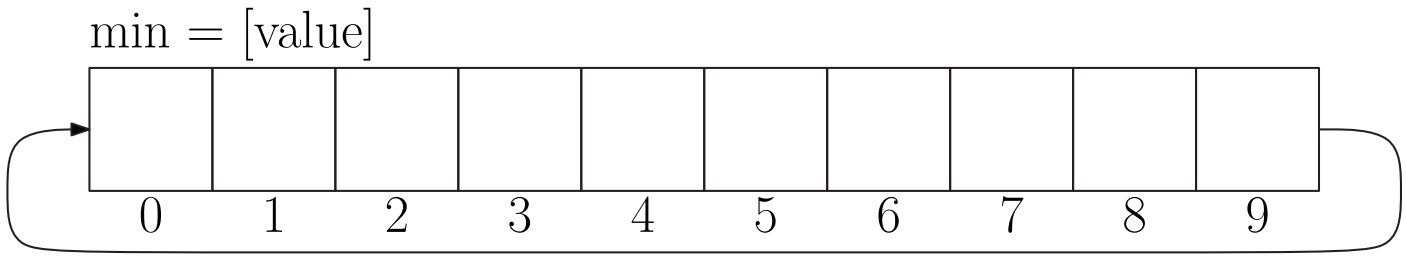
Bucket Queues

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)`



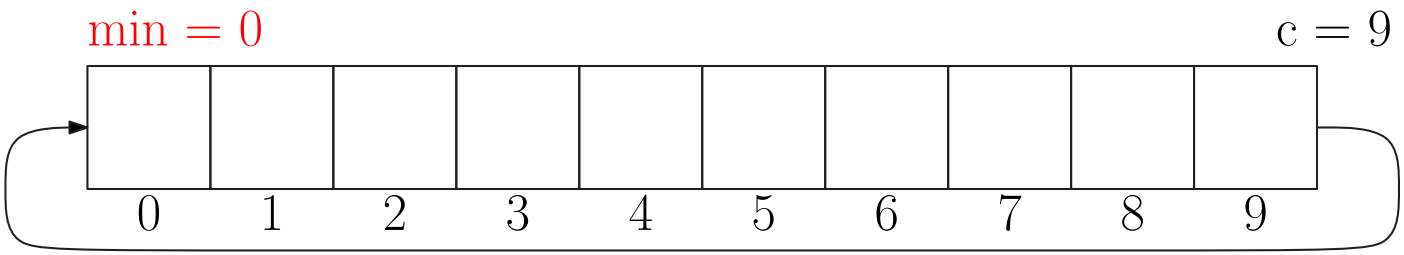
Bucket Queues

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)`



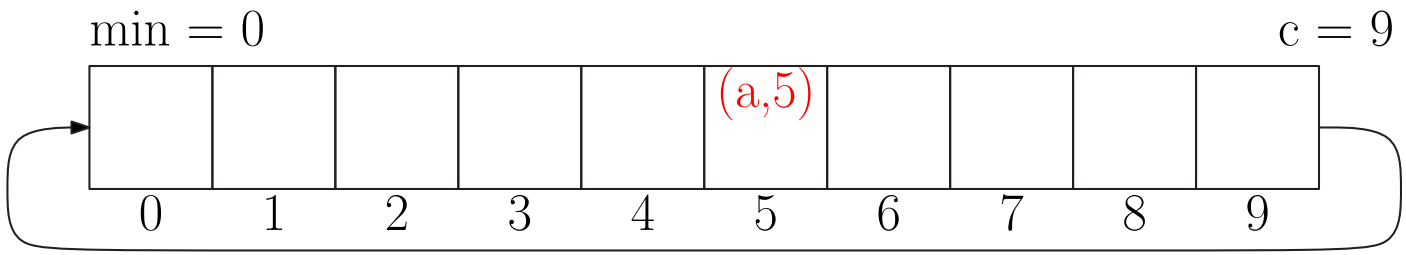
Bucket Queues

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)`



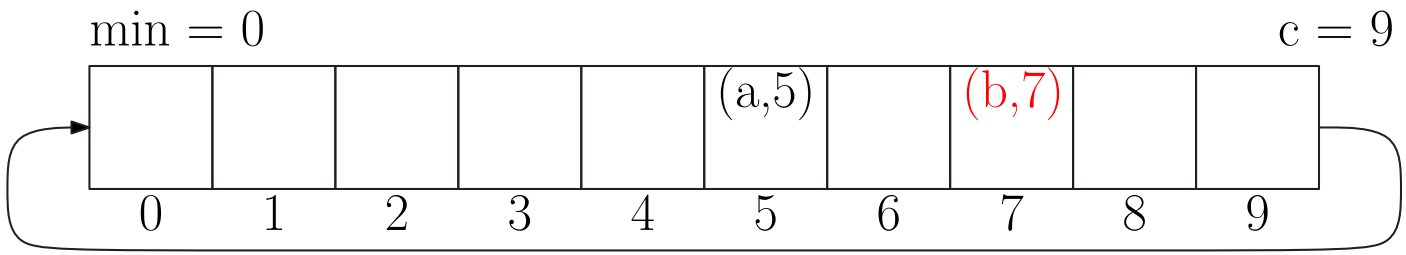
Bucket Queues

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)`



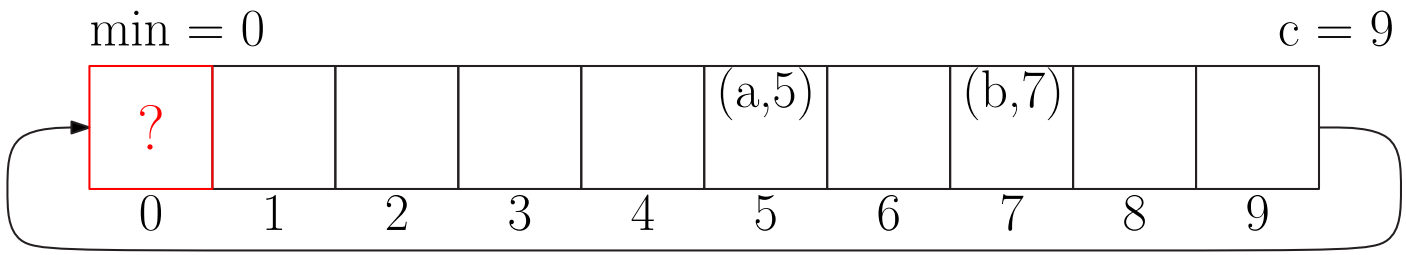
Bucket Queues

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)`



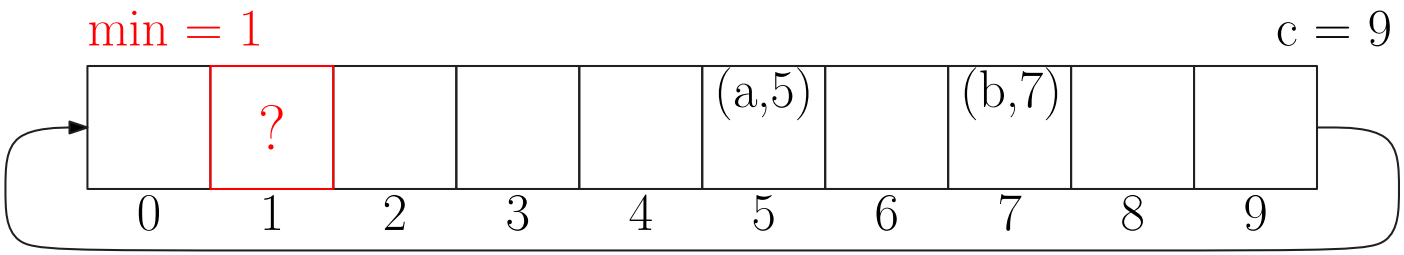
Bucket Queues

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)`



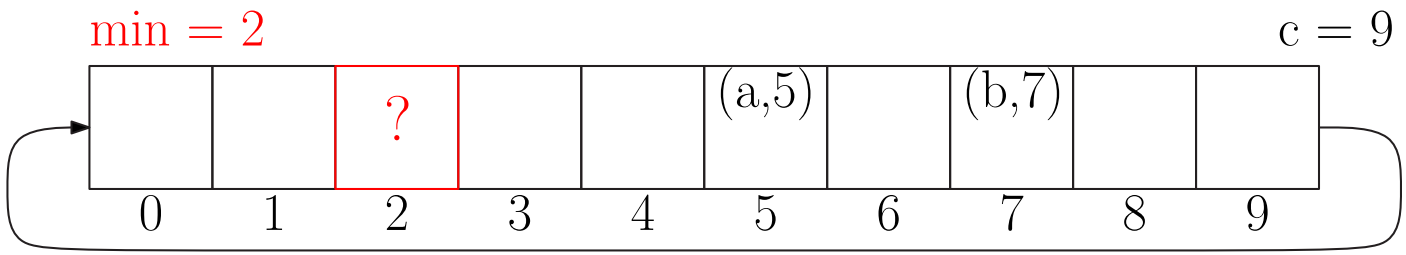
Bucket Queues

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)`



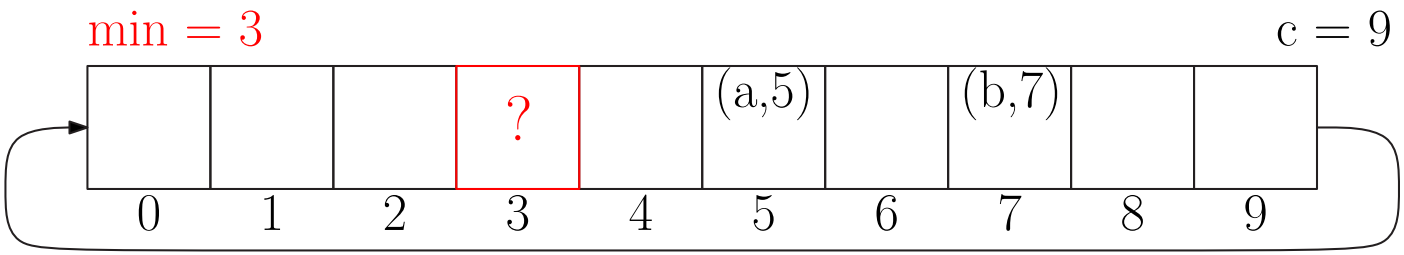
Bucket Queues

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)`



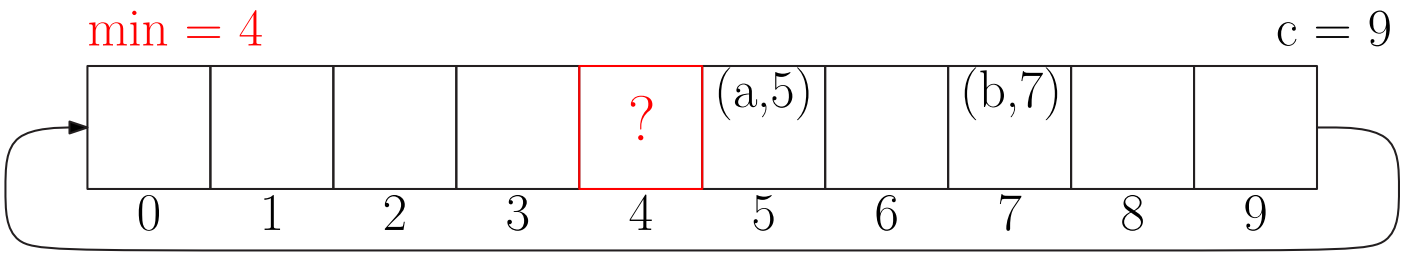
Bucket Queues

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)`



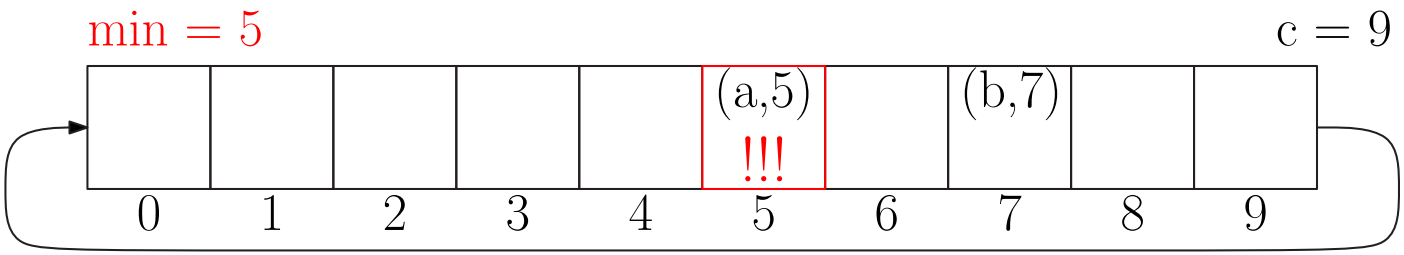
Bucket Queues

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)`



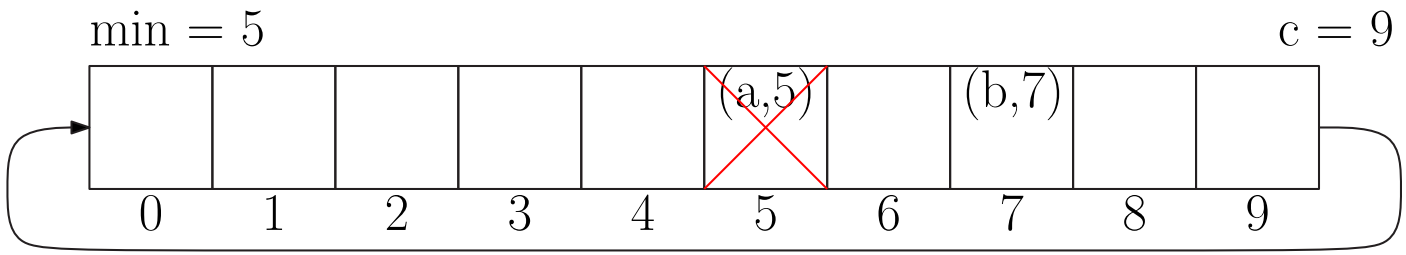
Bucket Queues

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)`



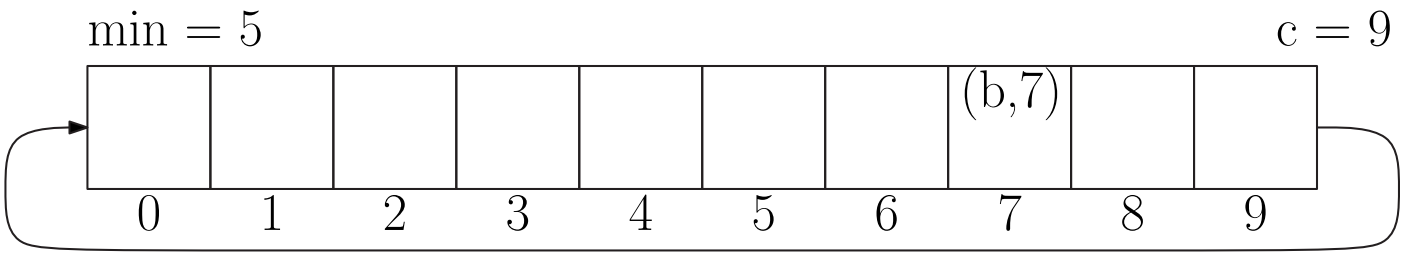
Bucket Queues

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)`



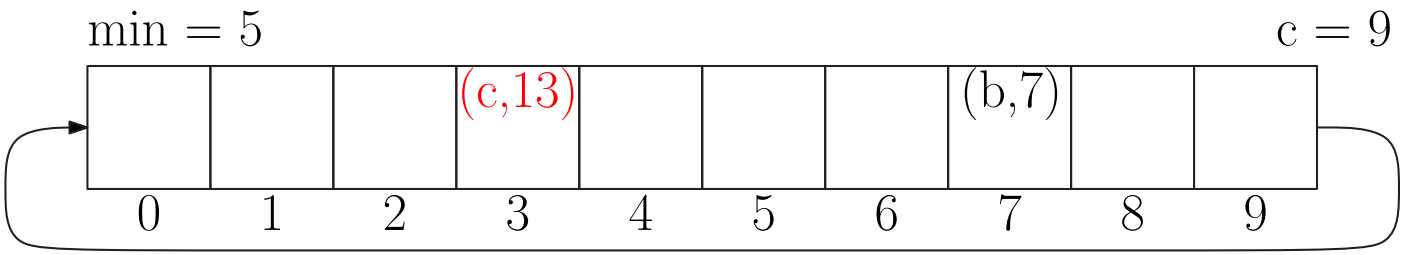
Bucket Queues

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)`



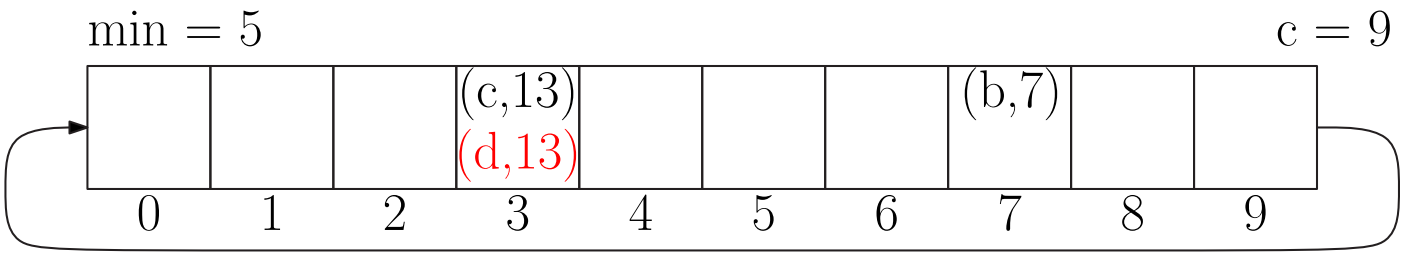
Bucket Queues

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)`



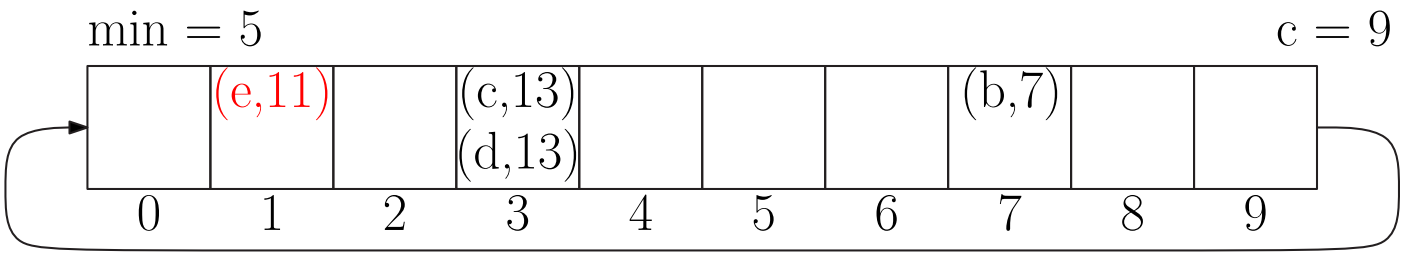
Bucket Queues

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)`



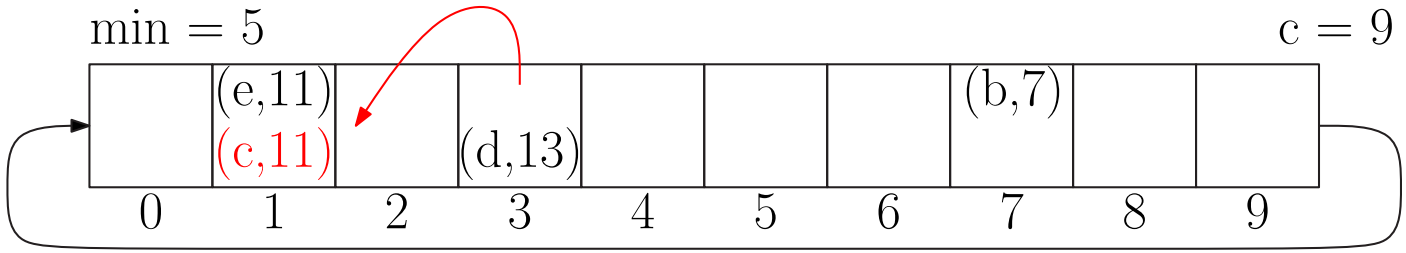
Bucket Queues

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	$\log C$
$T_{Djkstra}$	$(m+n)\log n$	$m + n\log n$	$m + nC$	$m + n\log C$

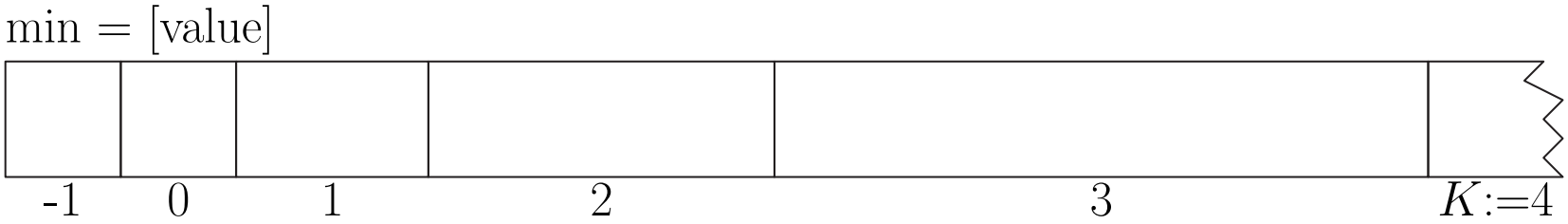
Radix Heaps

Aufbau:

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

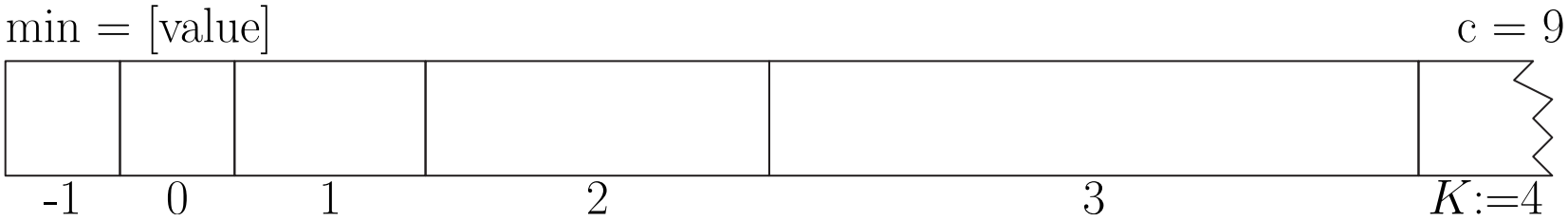
gegeben:

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



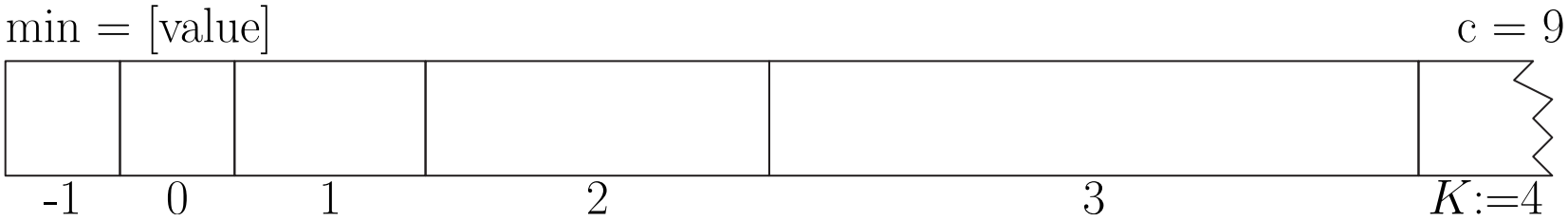
$$C = 9 \longrightarrow K + 2 = (\lfloor \log C \rfloor + 1) + 2 = (3 + 1) + 2 = 6$$

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

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

$\rightarrow$ Bucket K ist *overflow* Bucket

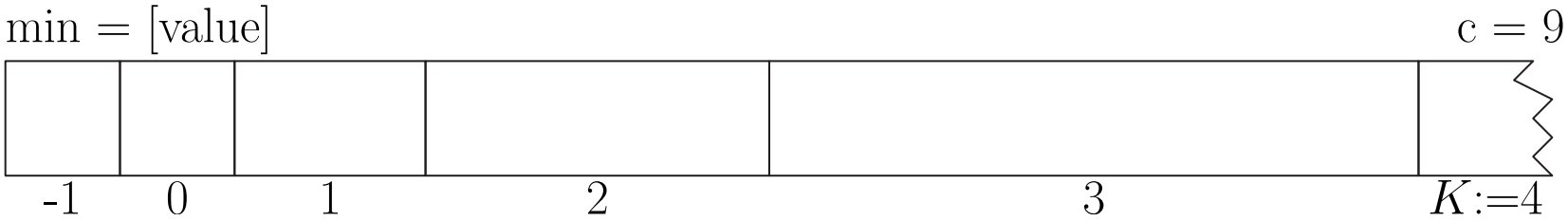
$\rightarrow$ höchstwertiges unterschiedliches Bit ist i

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

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

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

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

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

$\rightarrow$ Bucket K ist *overflow* Bucket

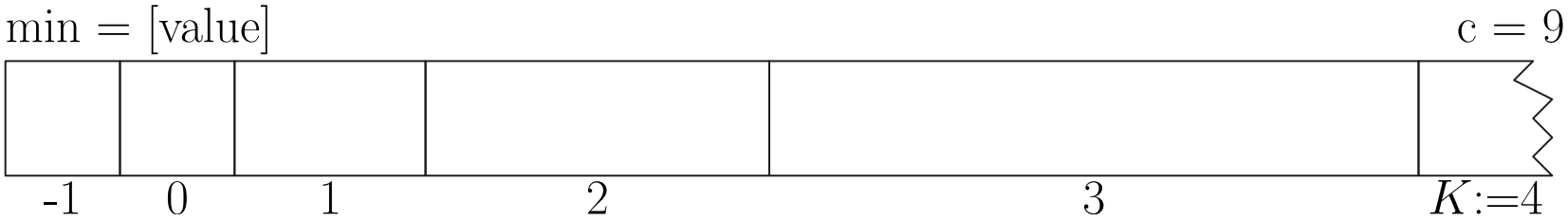
$\rightarrow$ höchstwertiges unterschiedliches Bit ist i

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

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

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

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

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

$\rightarrow$ Bucket K ist *overflow* Bucket

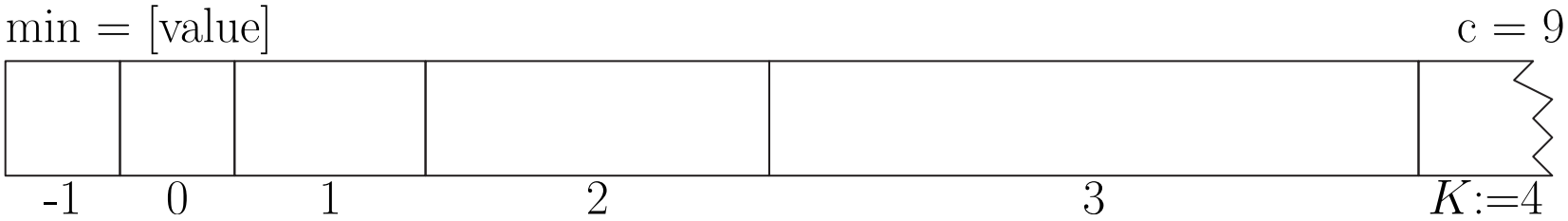
$\rightarrow$ höchstwertiges unterschiedliches Bit ist i

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

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

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

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

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

$\rightarrow$ 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

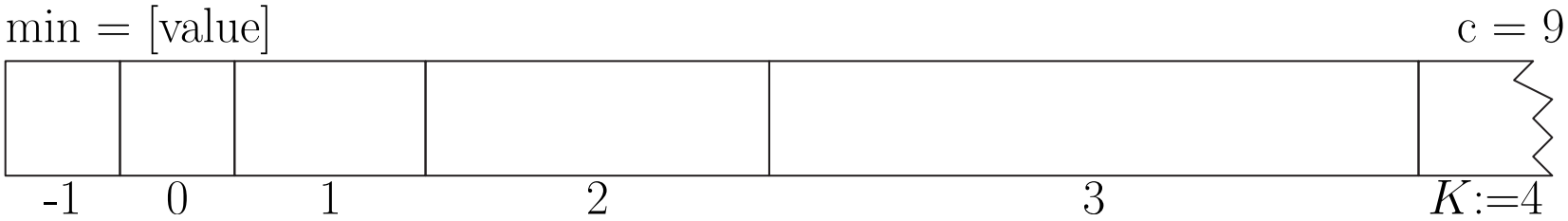
Beispiele

min := 00001000 (08)

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

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

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

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

$\rightarrow$ 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

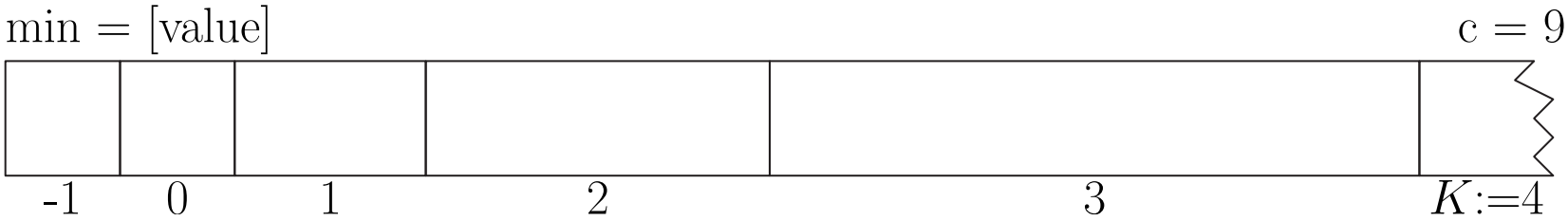
Beispiele

$min := 00001000\ (08)$

$\text{msd}(00001000, 01******) = 6 \longrightarrow 2^6 = 64$ Werte

$\text{msd}(00001000, 0000101*) = 1 \longrightarrow 2^1 = 2$ Werte

Radix Heaps



Welche Schlüssel kommen in welches Bucket?

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

$\rightarrow$ Bucket K ist *overflow* Bucket

$\rightarrow$ höchstwertiges unterschiedliches Bit ist i

(bzw. -1 wenn $\mathit{key} = \mathit{min}$)

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

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

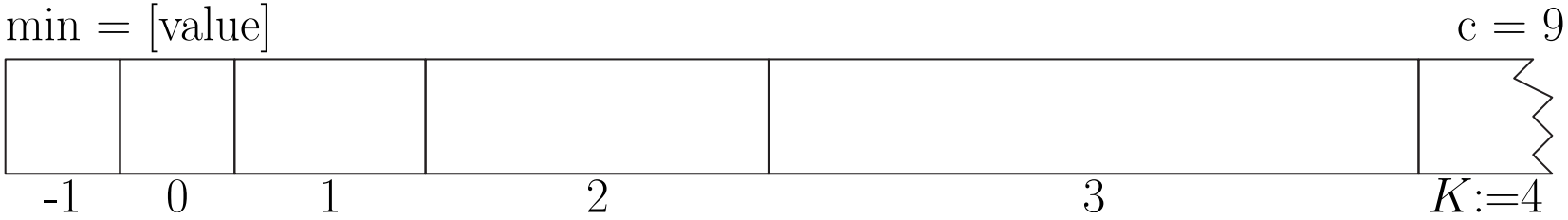
Beispiele

min := 00001000 (08)

$\text{msd}(00001000, 00000110) = 3 \longrightarrow$ kann nicht passieren, da $6 < 8$

$\text{msd}(00001000, 00000100) = 3 \longrightarrow$ kann nicht passieren, da $4 < 8$

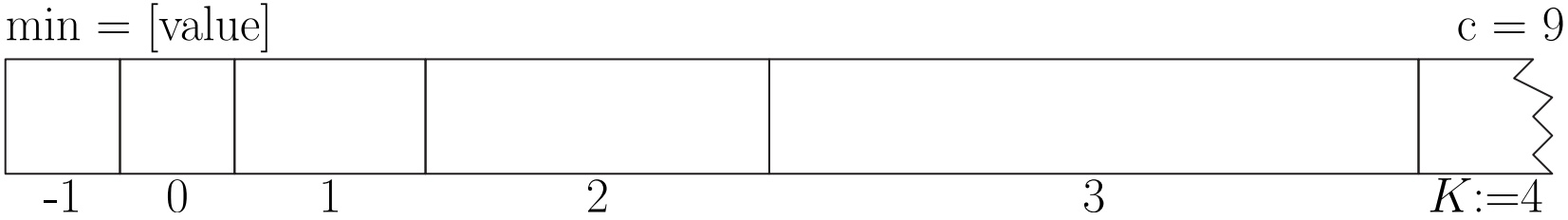
Radix Heaps



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 ?

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..77 !	- !

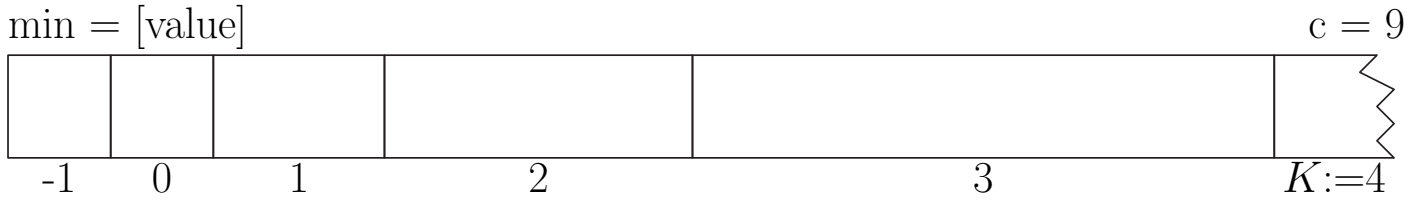
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



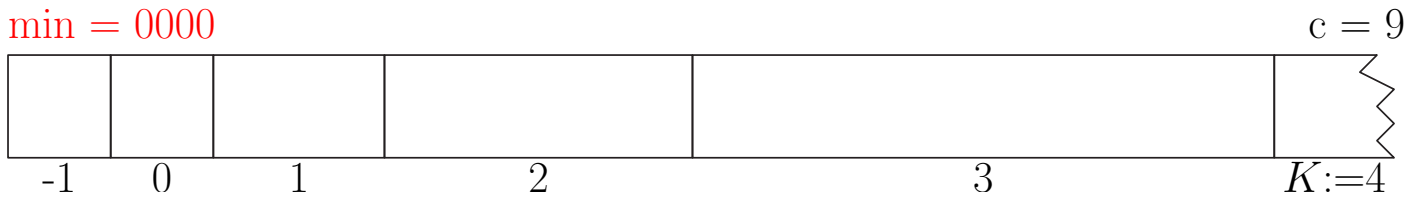
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



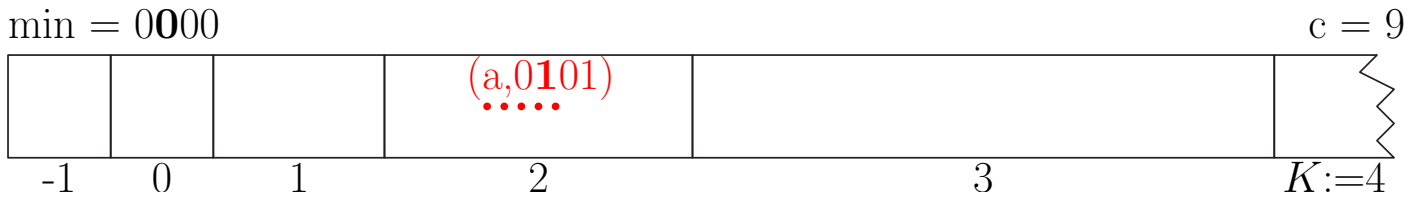
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



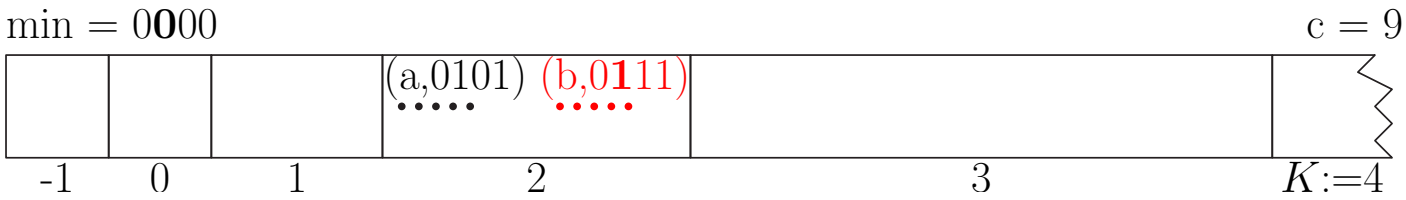
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



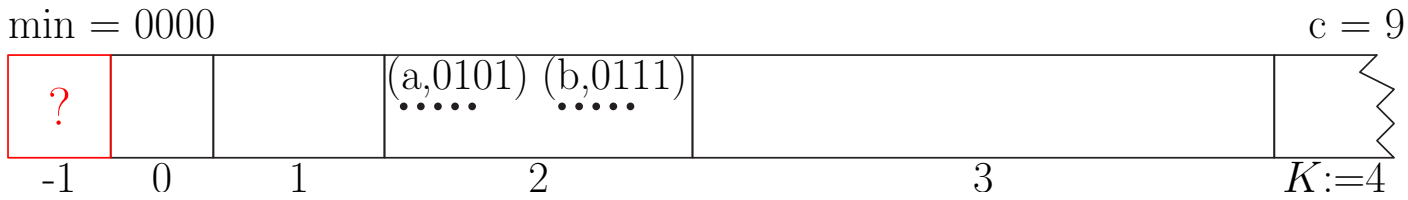
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



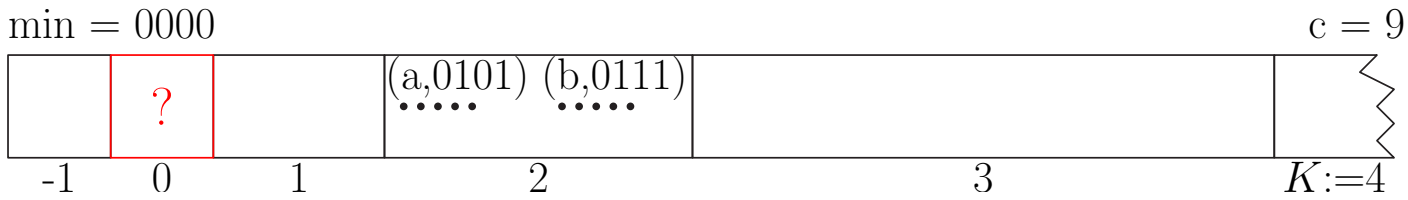
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



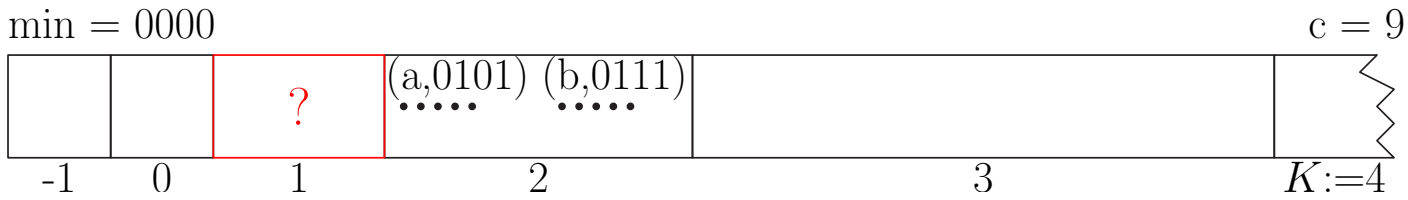
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



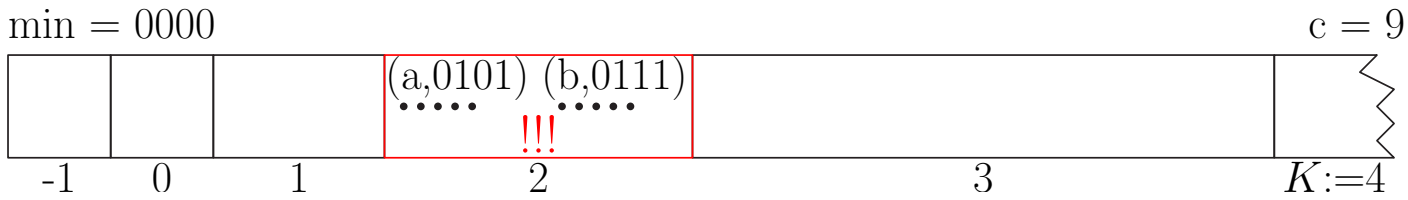
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



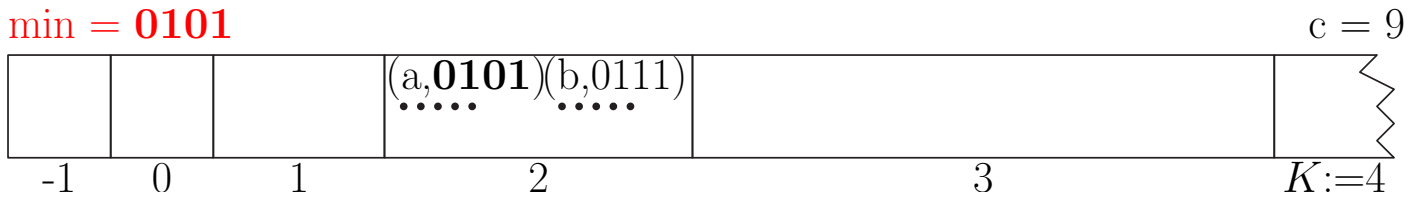
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



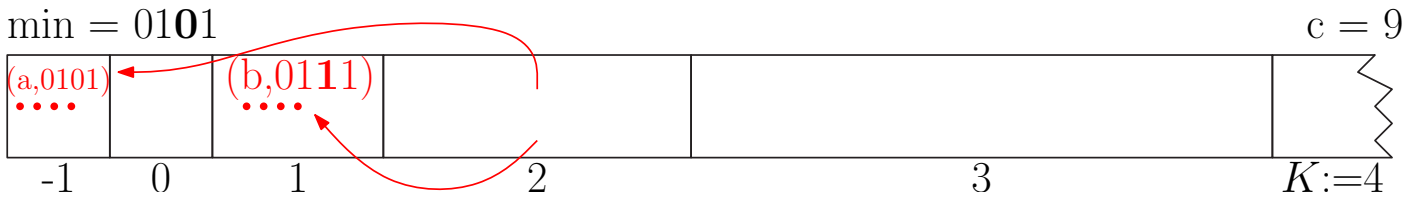
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



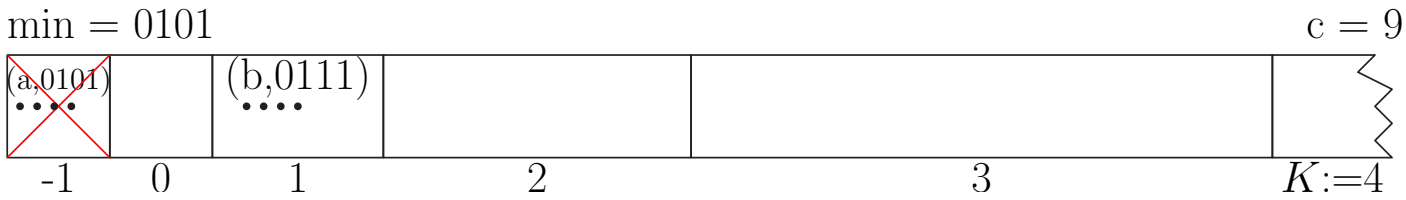
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



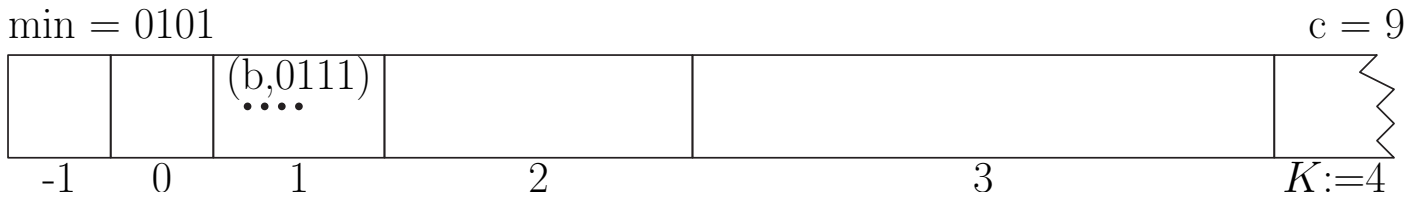
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



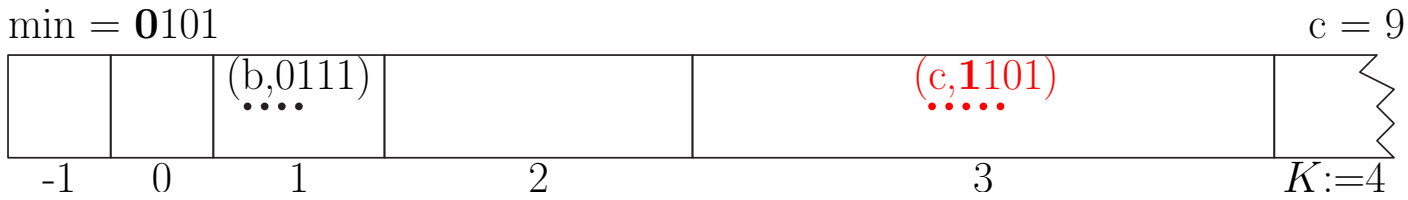
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



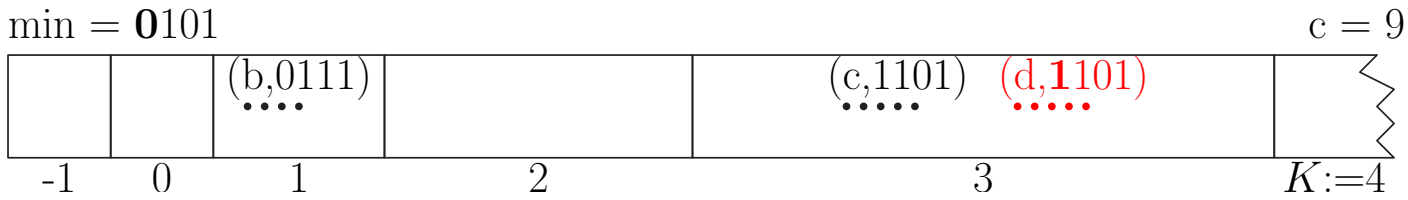
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



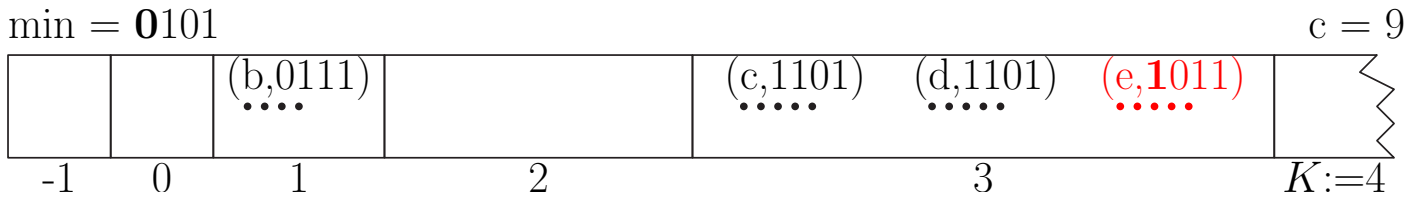
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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



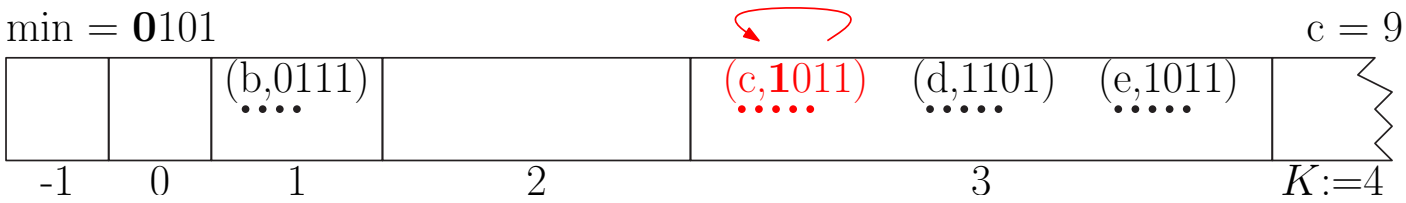
Radix Heaps

Ablauf und Laufzeiten (amortisiert):

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

Beispiel:

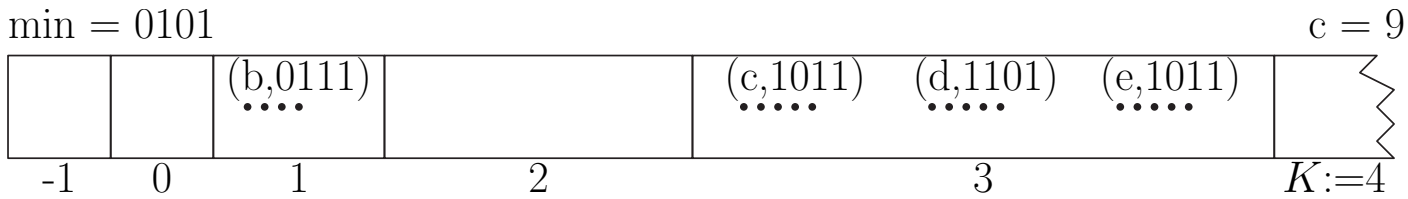
- `insert(a,5)`
- `insert(b,7)`
- `deleteMin()`
- `insert(c,13)`
- `insert(d,13)`
- `insert(e,11)`
- `decreaseKey(c,11)`



Radix Heaps

Beispiel (fortgesetzt):

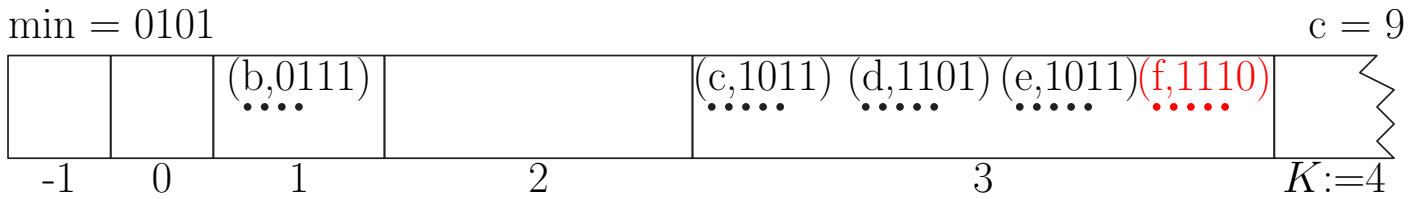
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

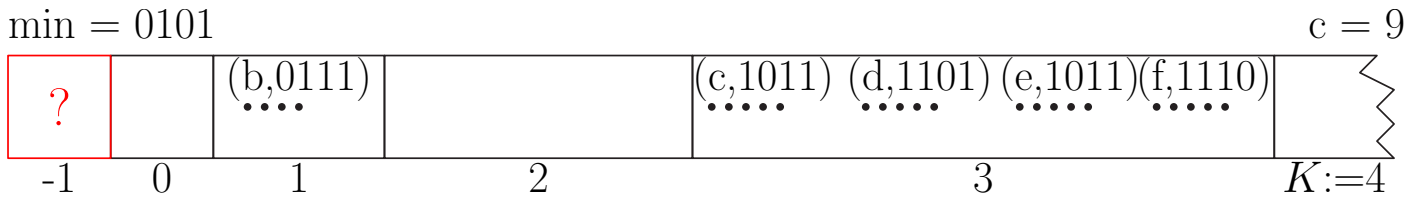
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

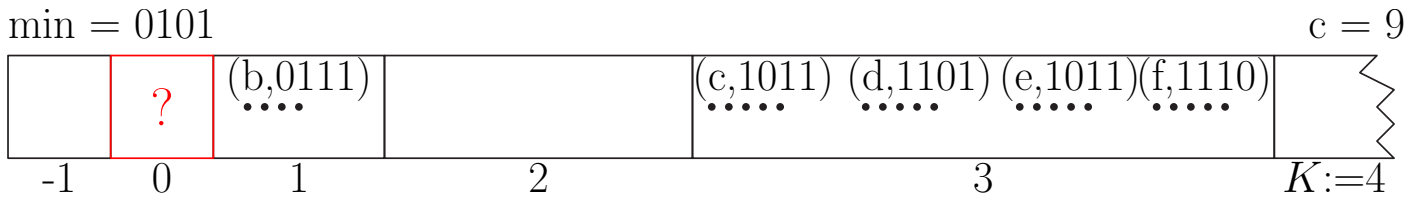
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

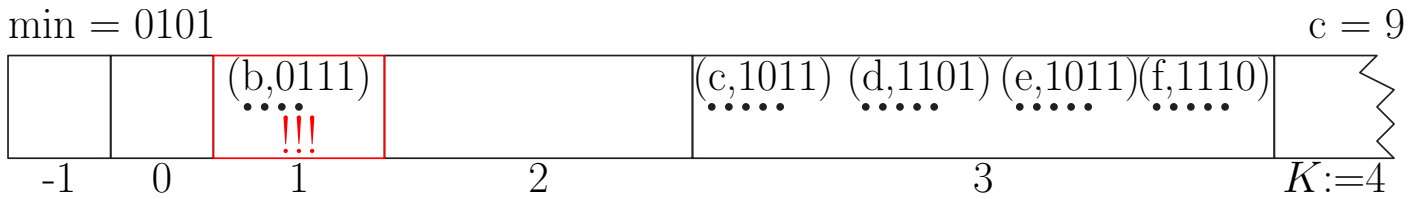
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

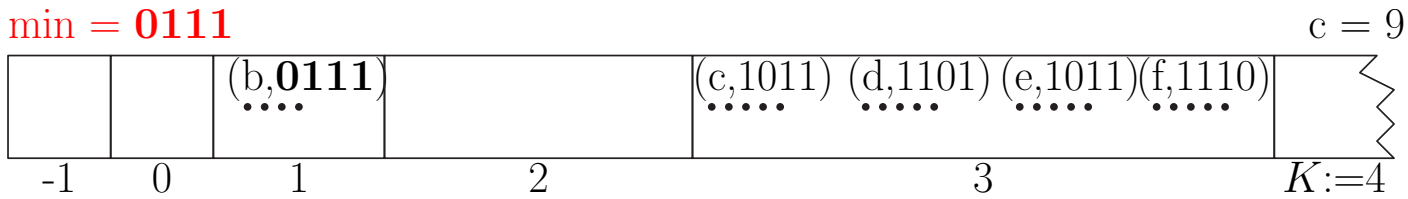
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

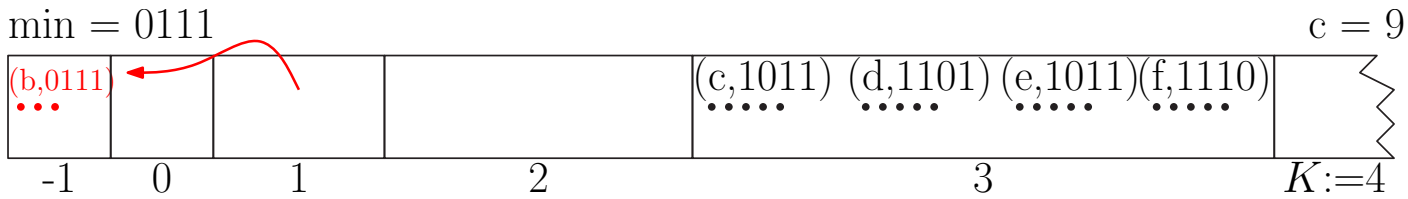
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

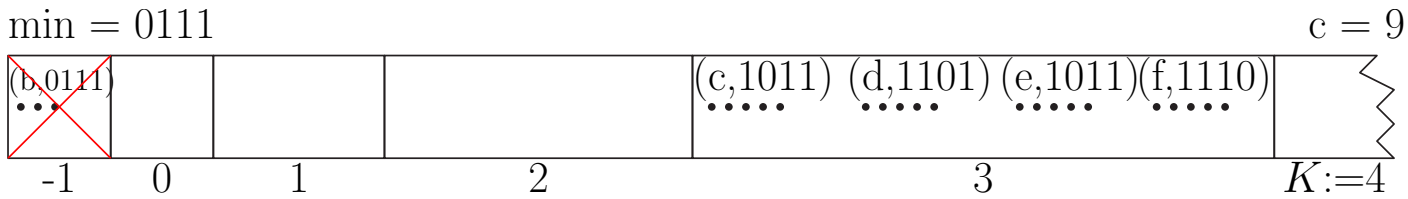
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

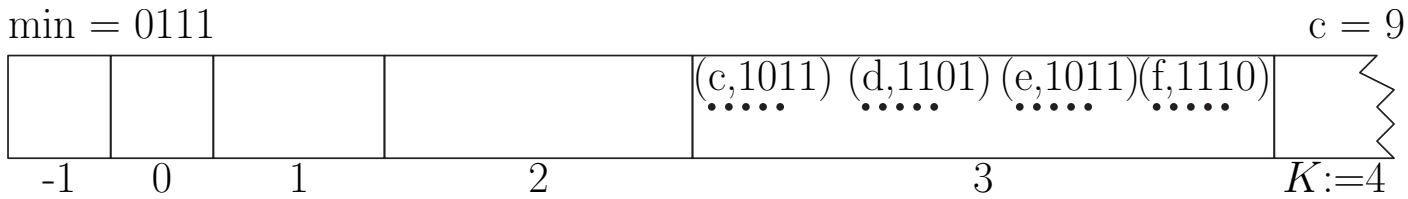
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

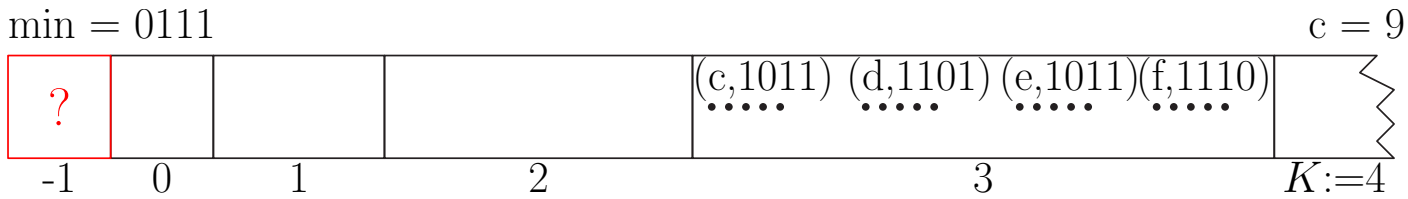
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

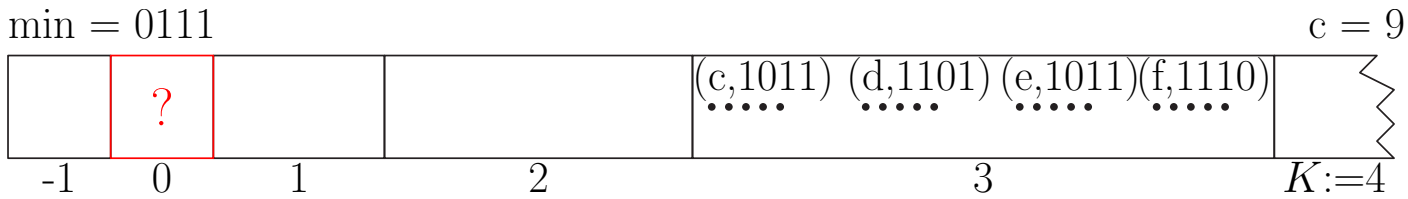
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

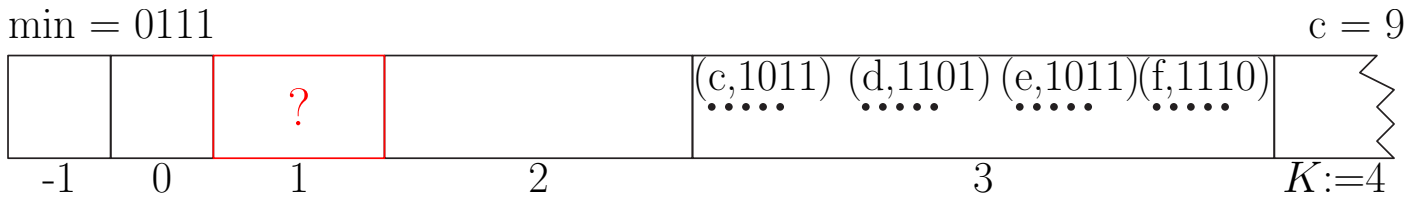
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

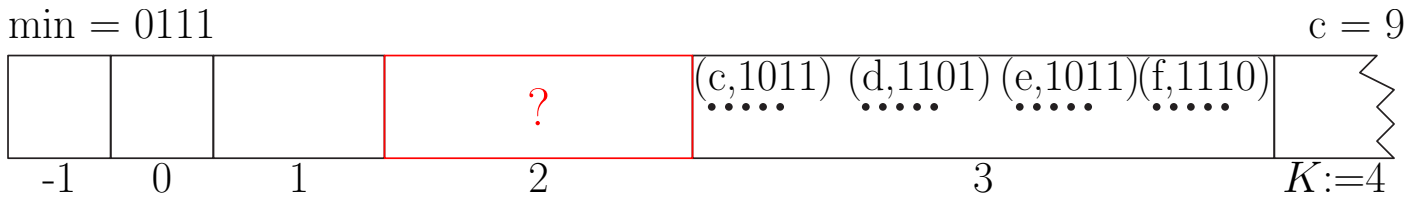
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

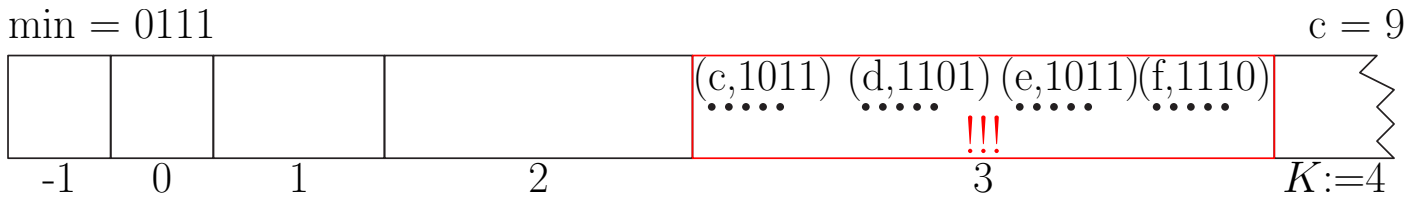
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

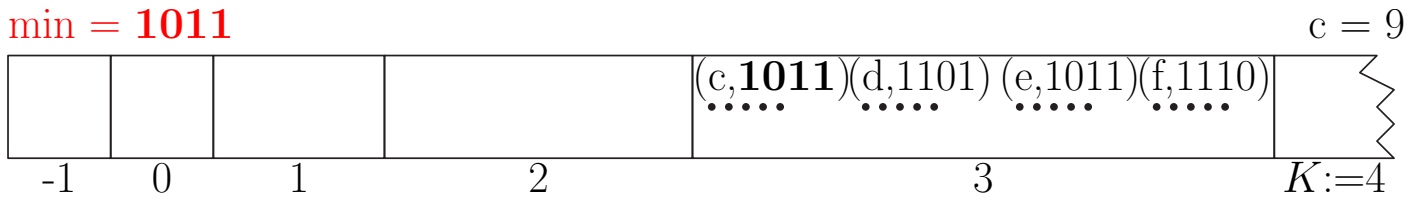
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

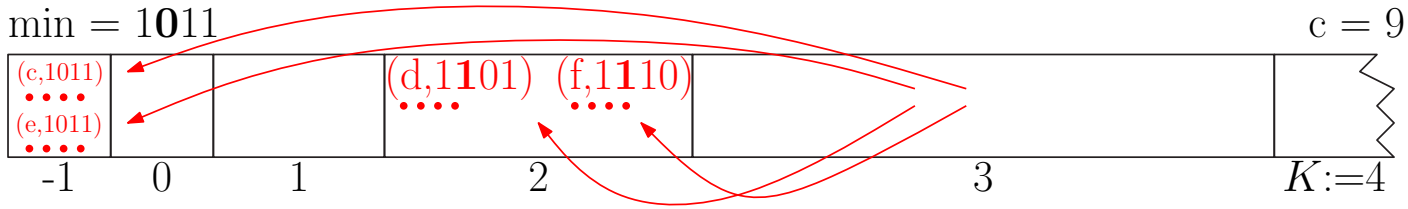
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

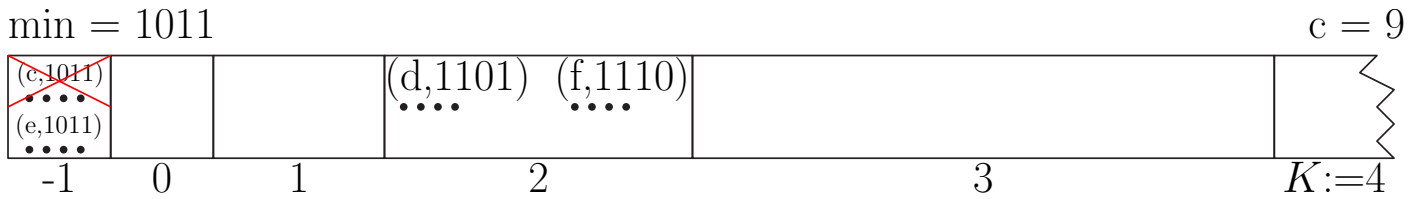
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

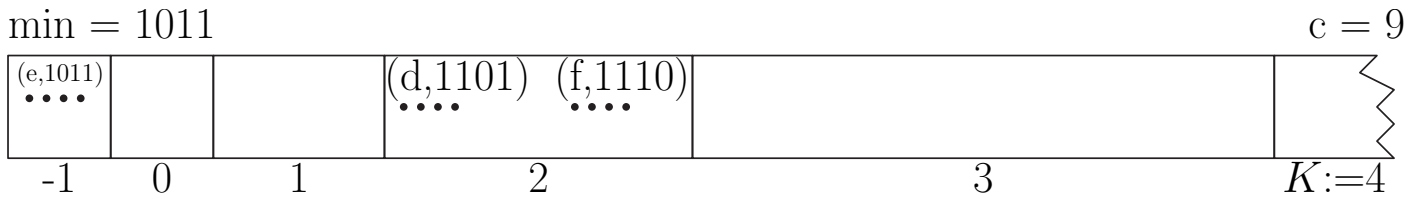
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

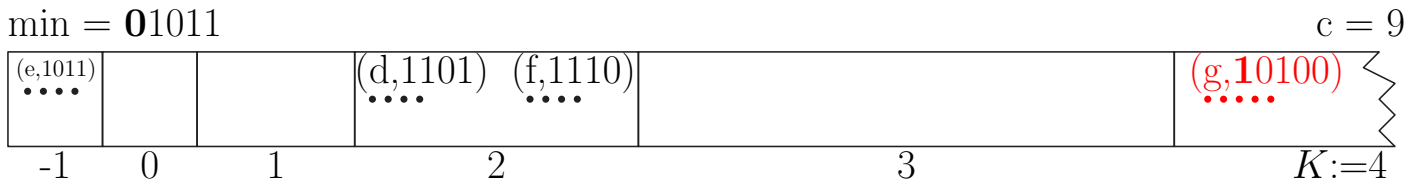
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

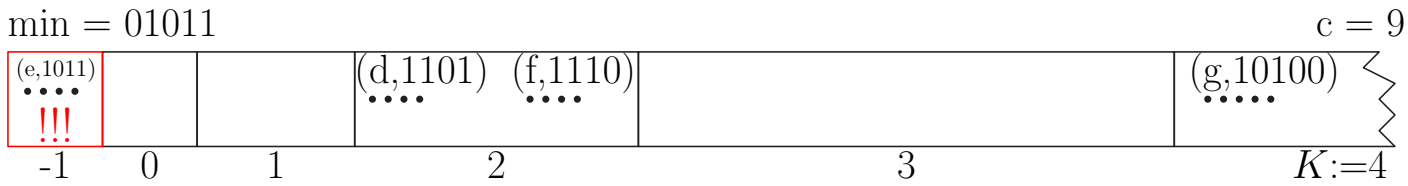
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

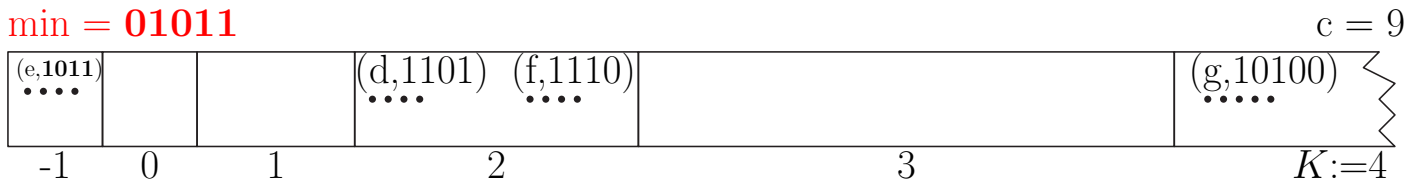
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

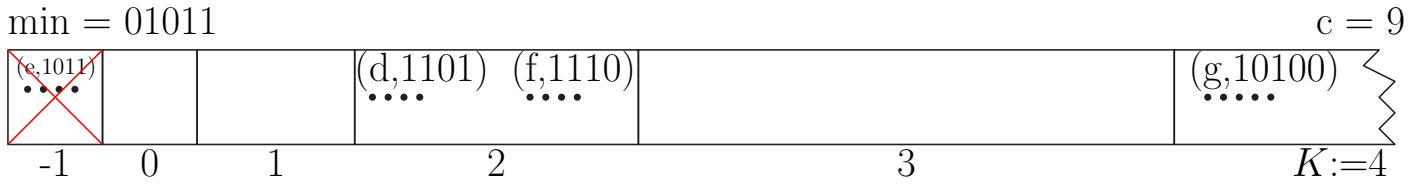
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

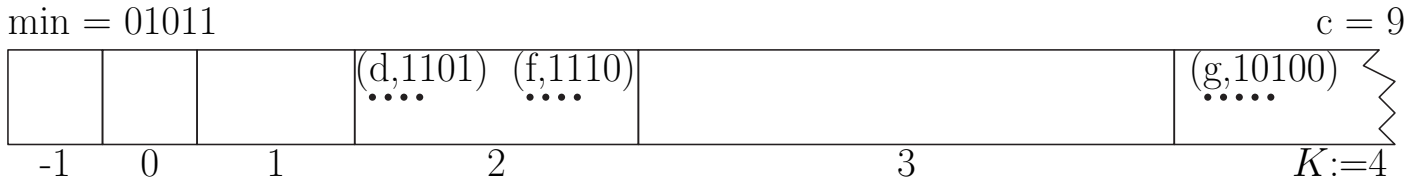
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

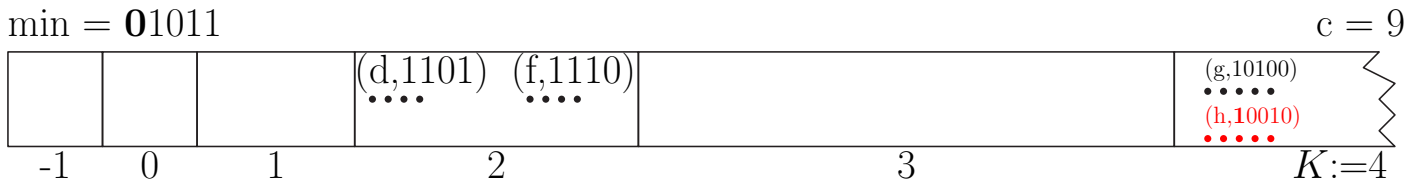
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

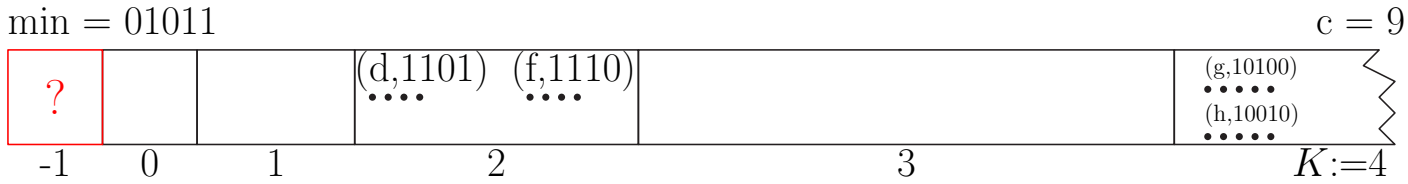
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

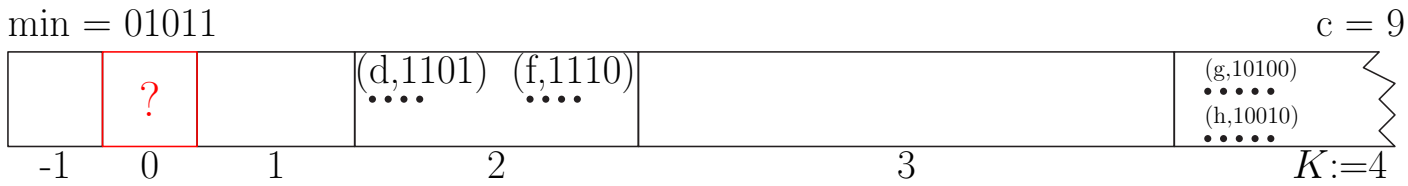
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

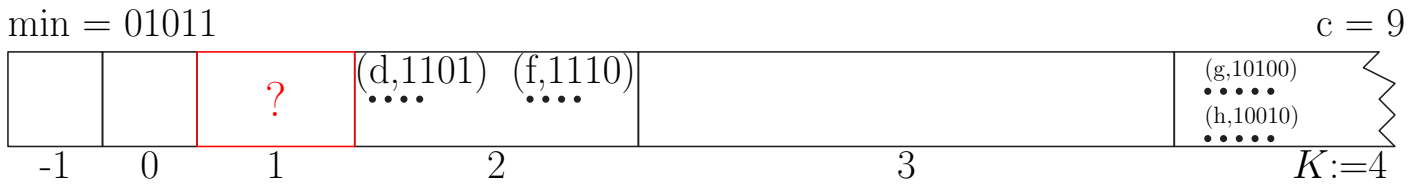
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

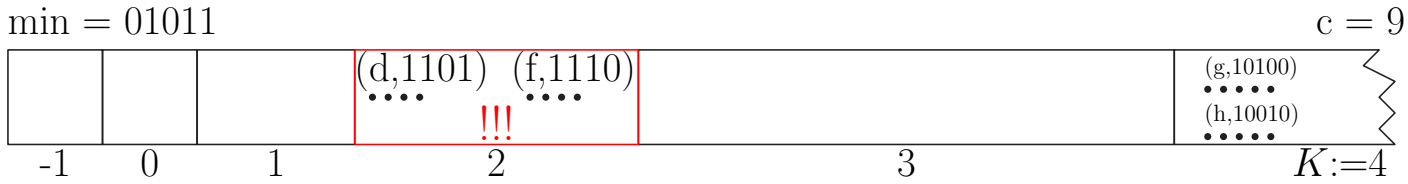
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

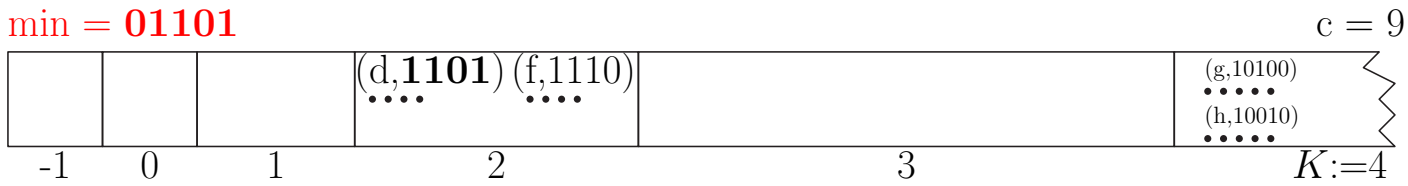
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

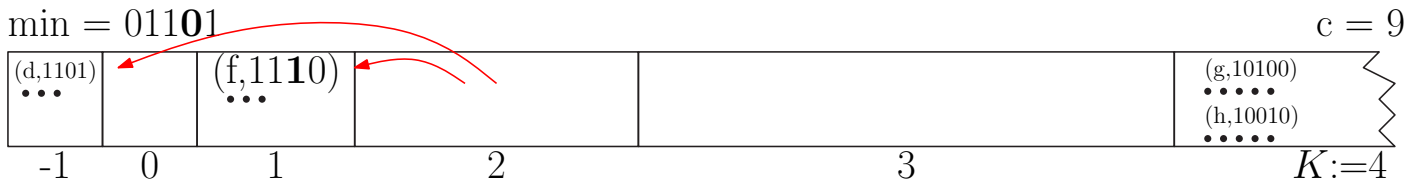
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

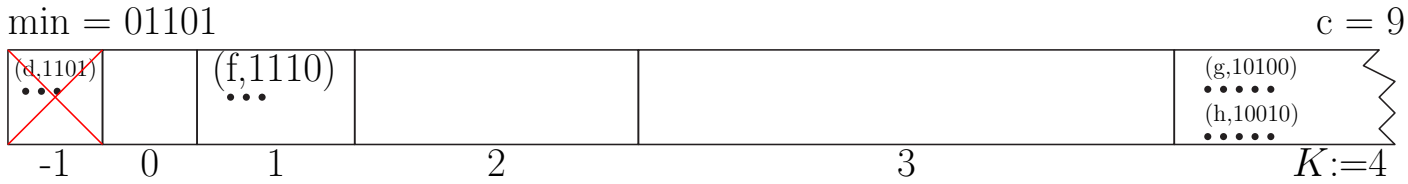
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

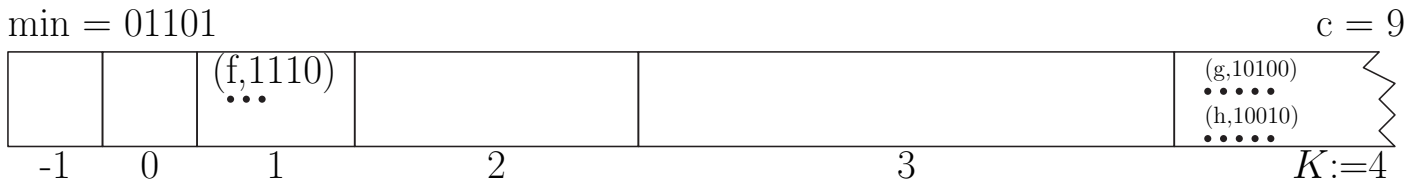
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

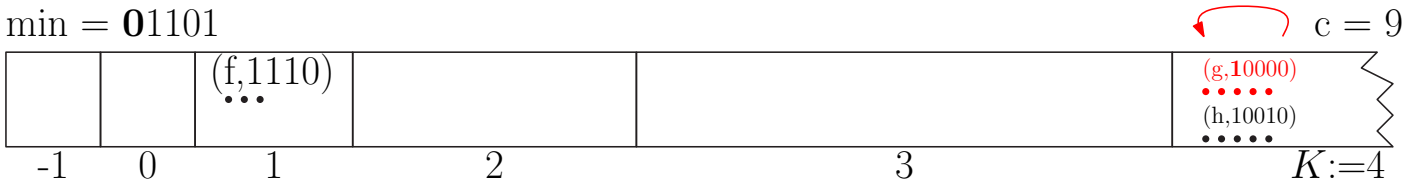
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

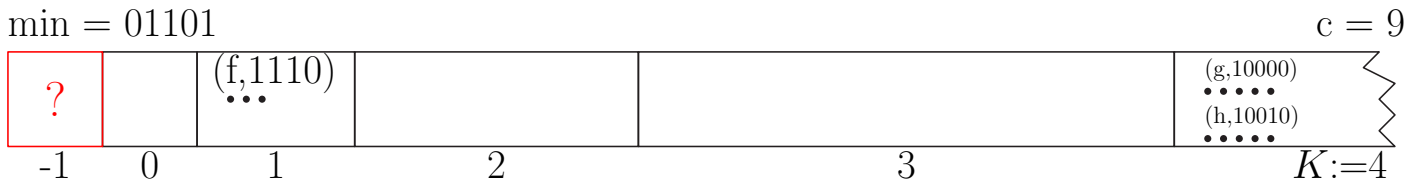
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

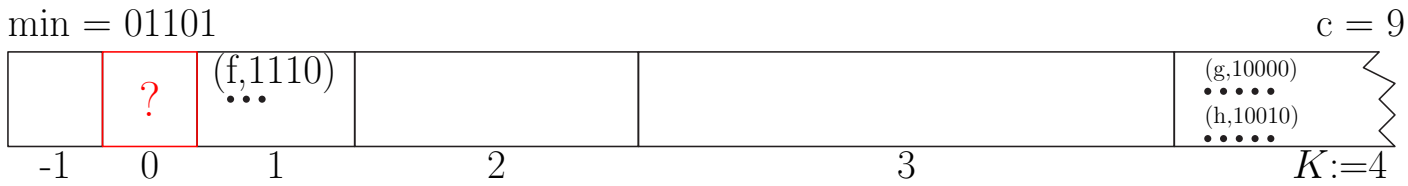
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

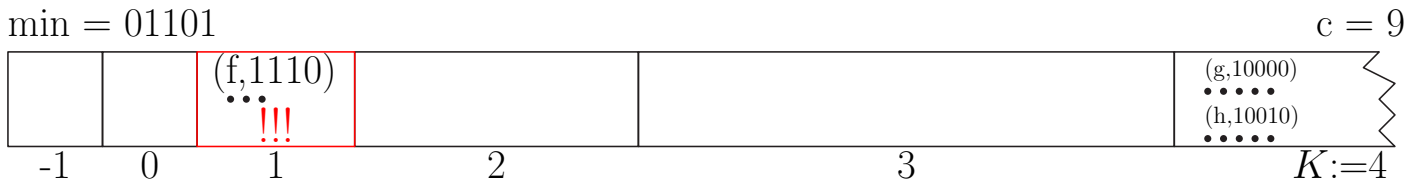
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

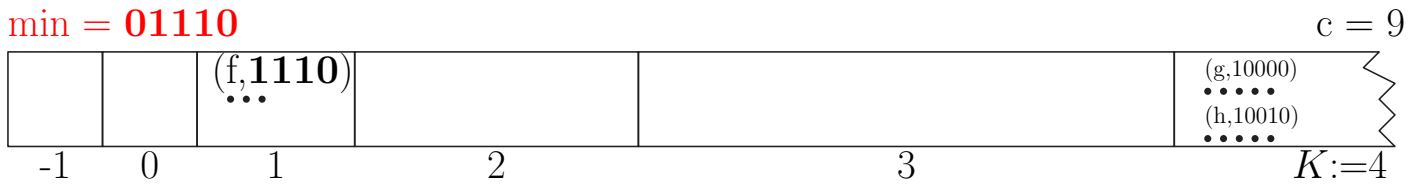
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

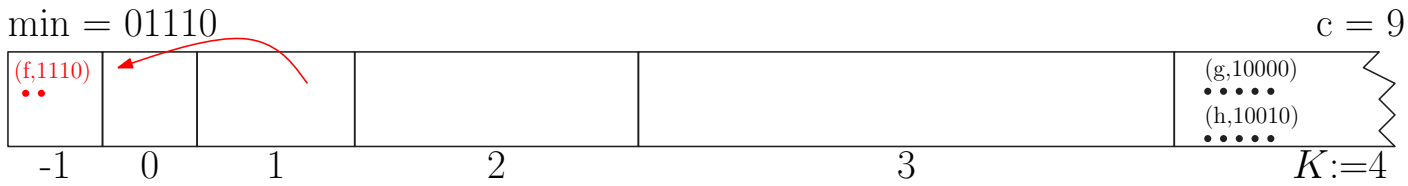
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

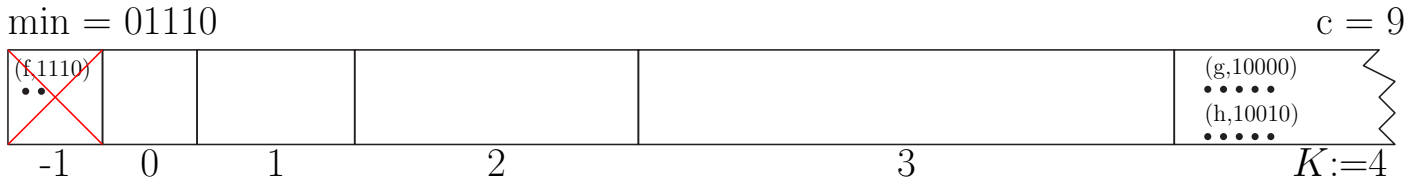
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

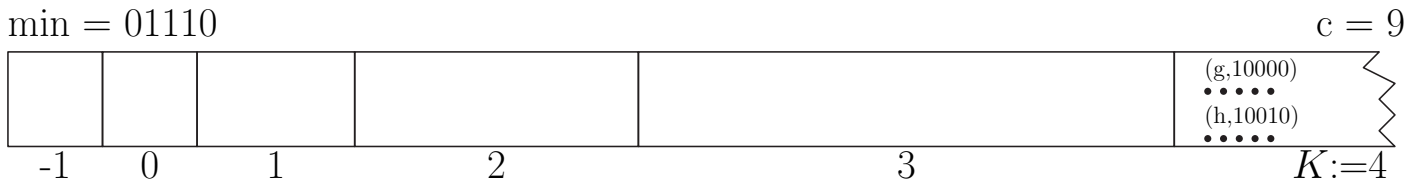
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

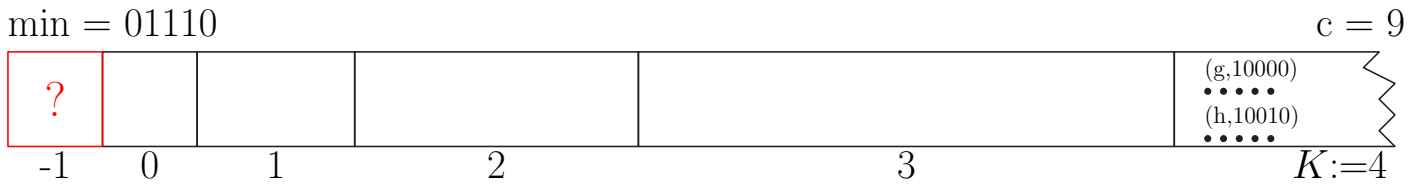
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

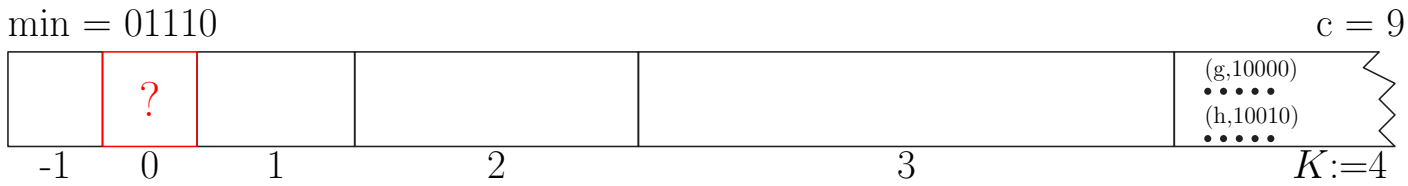
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

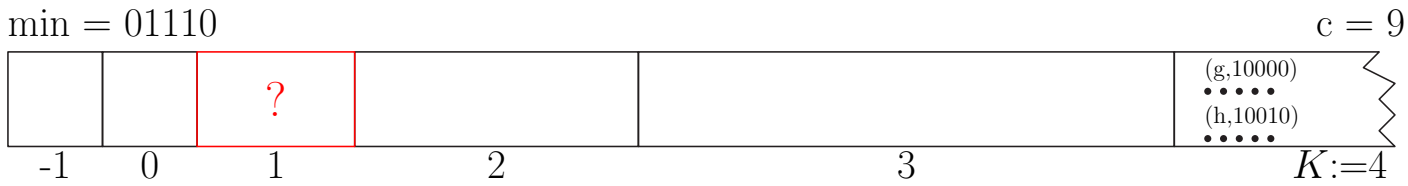
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

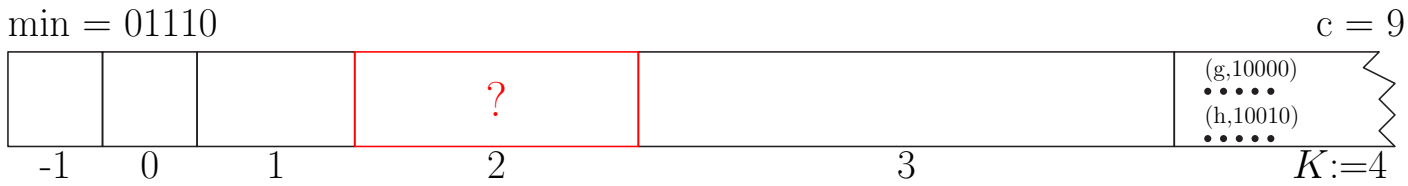
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

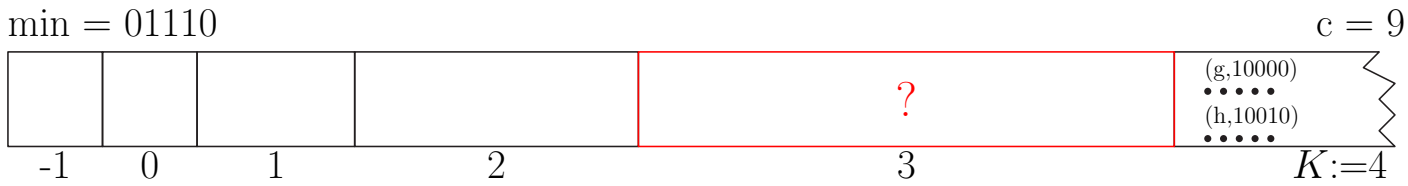
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

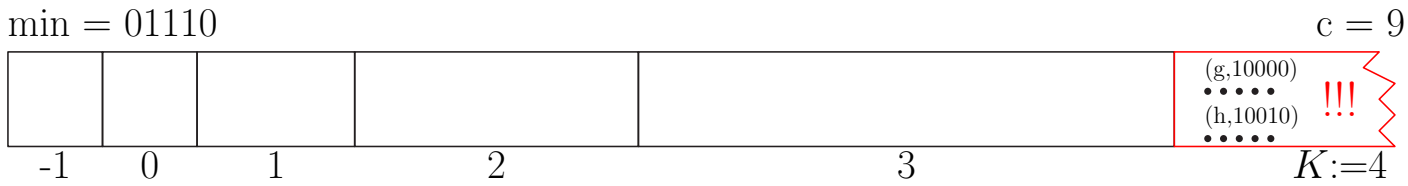
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

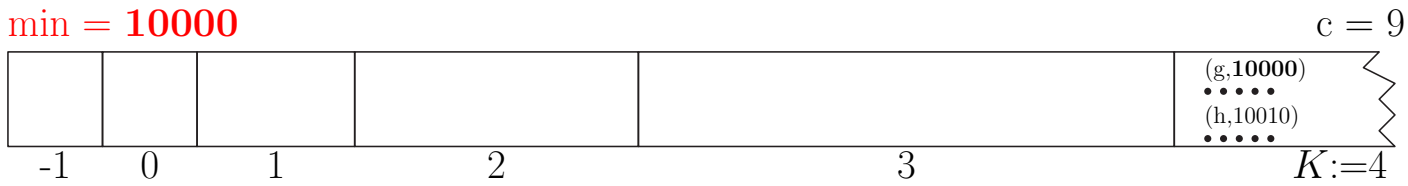
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

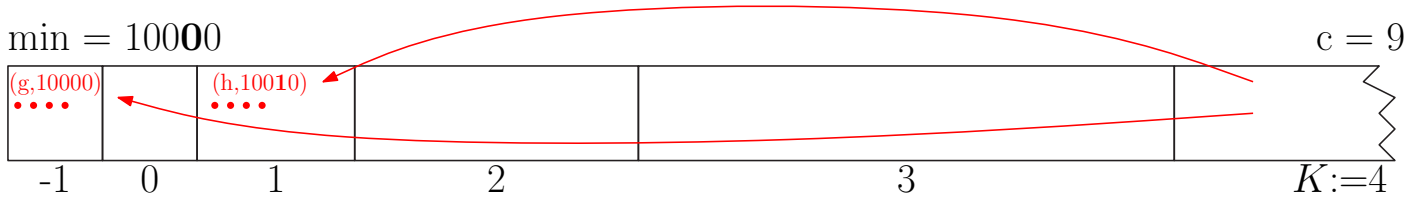
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

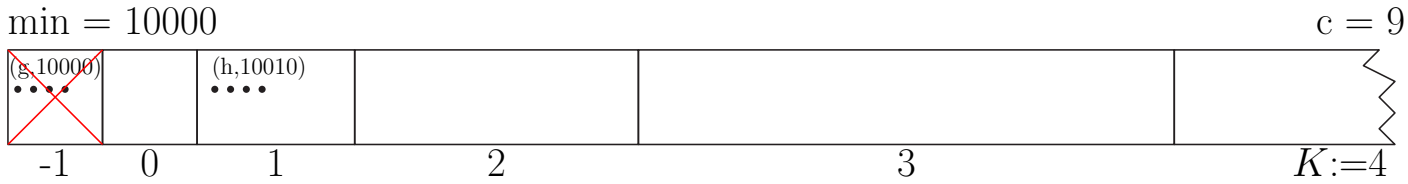
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

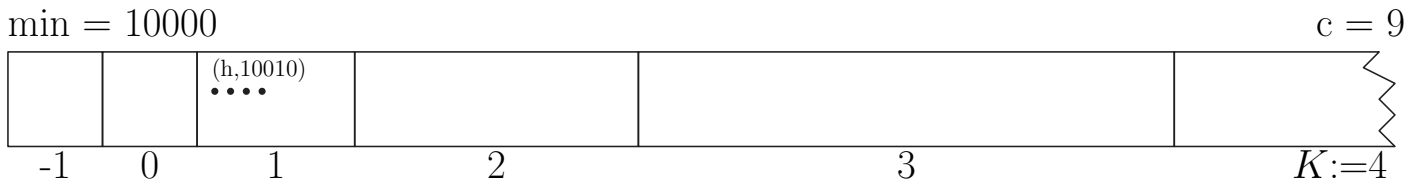
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

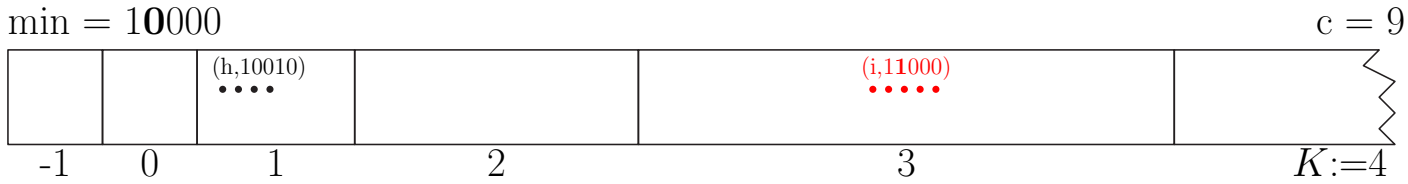
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

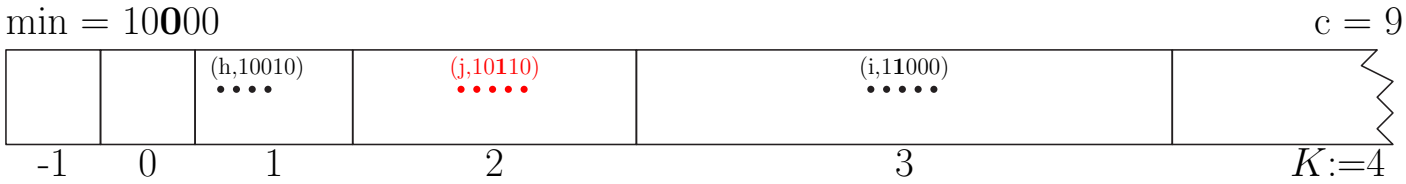
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

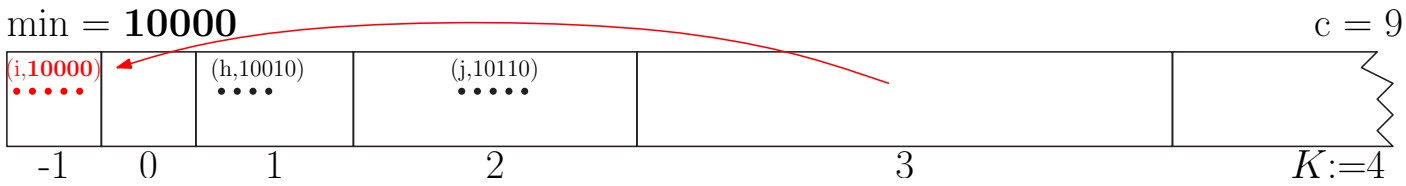
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

Beispiel (fortgesetzt):

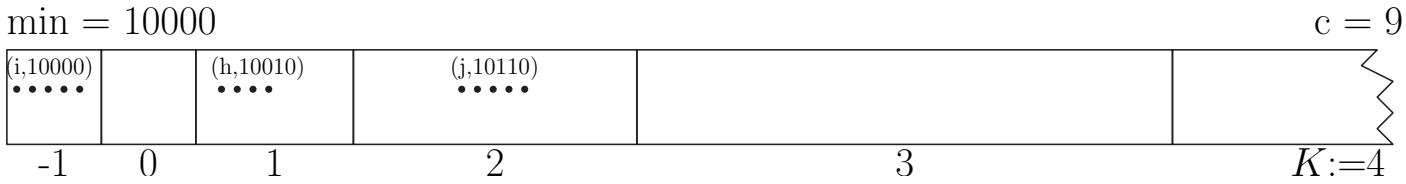
- `insert(f,14)`
- `deleteMin()`
- `deleteMin()`
- `insert(g,20)`
- `deleteMin()`
- `insert(h,18)`
- `deleteMin()`
- `decreaseKey(g,16)`
- `deleteMin()`
- `deleteMin()`
- `insert(i,24)`
- `insert(j,22)`
- `decreaseKey(i,16)`



Radix Heaps

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 $\min(\text{msd}(min, key), K)$ *?*

- viel anschaulicher wäre doch

$$i = \lfloor \log(key - min) \rfloor$$

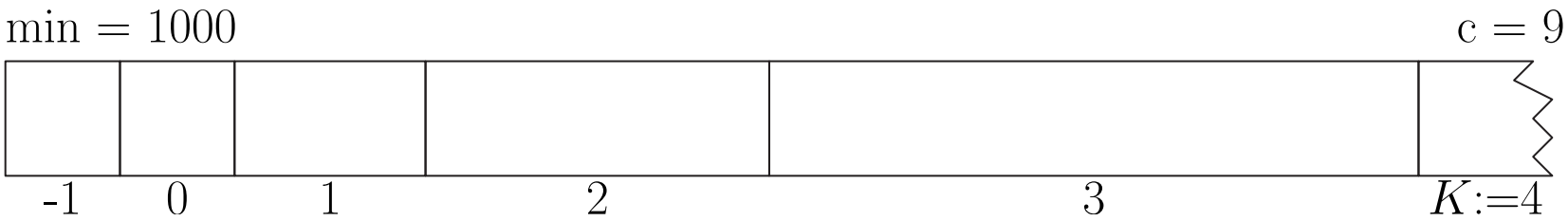
- okay, Anzahl Verschiebungen erhöht sich nicht ...
- ... aber, jede Änderung von *min* ändert potentiell alle Buckets
  $\rightarrow$ schlechtere Laufzeit!

Beispiel

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

	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..13	14..17	18..10+C	-

Radix Heaps

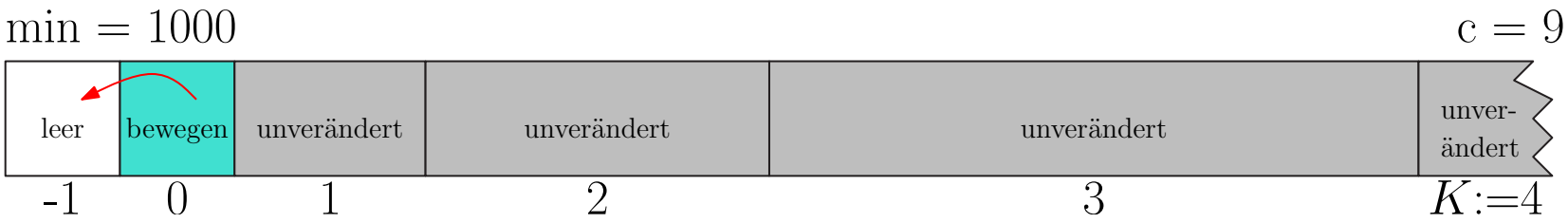


Änderung von min $\longrightarrow$ *Umverteilung eines Bucket genügt*

min $:=$ 01000 (08)

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..15	-	≥ 16

Radix Heaps

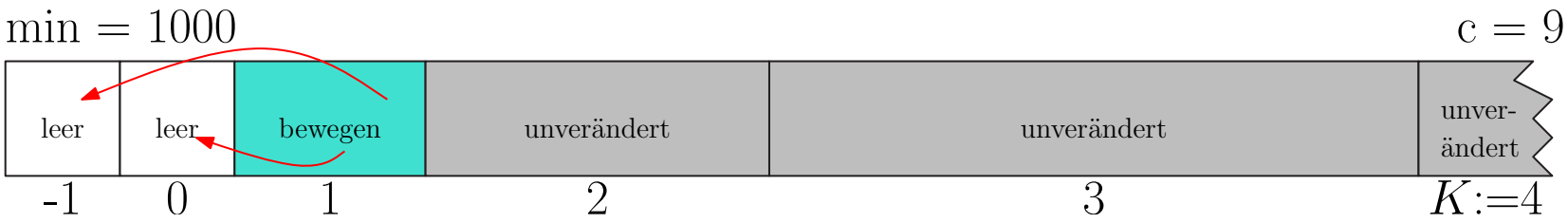


Änderung von min $\longrightarrow$ *Umverteilung eines Bucket genügt*

min := 01000 (08) $\xrightarrow{\text{deleteMin}}$ *min* := 01001 (09)

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..15	-	≥ 16
01001 (09)	9	-	10..11	12..15	-	≥ 16

Radix Heaps

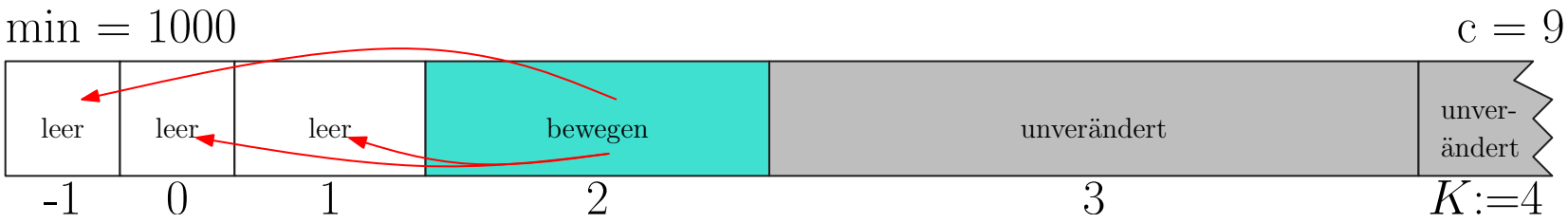


Änderung von min $\longrightarrow$ *Umverteilung eines Bucket genügt*

min := 01000 (08) $\overset{\text{deleteMin}}{\longrightarrow}$ *min* := 0101∗ (10..11)

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..15	-	≥ 16
01010 (10)	10	11	-	12..15	-	≥ 16
01011 (11)	11	-	-	12..15	-	≥ 16

Radix Heaps

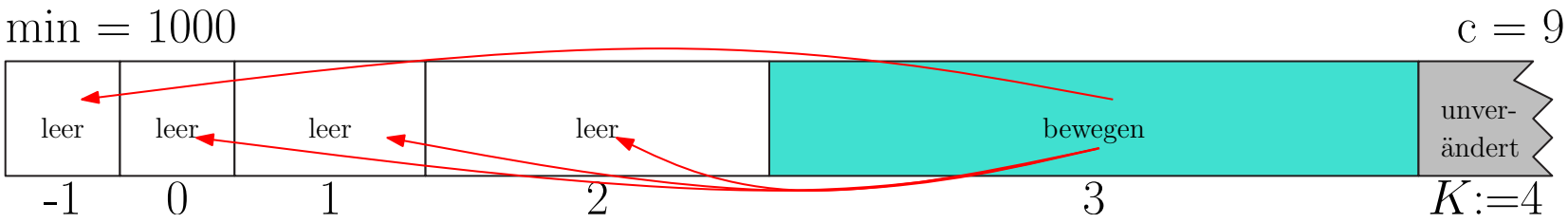


Änderung von min $\longrightarrow$ *Umverteilung eines Bucket genügt*

min := 01000 (08) $\overset{\text{deleteMin}}{\longrightarrow}$ *min* := 011 ∗ ∗ (12..15)

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..15	-	≥ 16
01100 (12)	12	13	14..15	-	-	≥ 16
01101 (13)	13	-	14..15	-	-	≥ 16
01110 (14)	14	15	-	-	-	≥ 16
01111 (15)	15	-	-	-	-	≥ 16

Radix Heaps

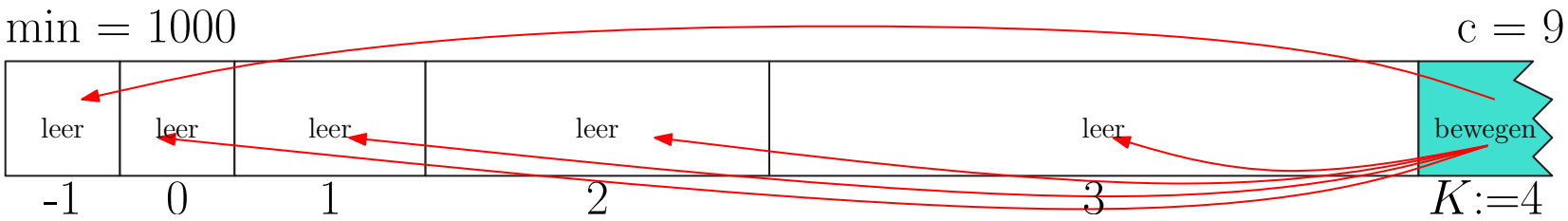


Änderung von min $\longrightarrow$ *Umverteilung eines Bucket genügt*

min $:= 01000\ (08) \overset{\text{deleteMin}}{\longrightarrow}$ *min* $:= -$, *Bucket* 3 *ist leer*

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..15	-	≥ 16

Radix Heaps



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

min := 01000 (08) $\overset{\text{deleteMin}}{\longrightarrow}$ *min* := 1 ∗ ∗ ∗ ∗ (16..)

	Bucket $B[\cdot]$					
min	-1	0	1	2	3	4
01000 (08)	8	9	10..11	12..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	$\log C$
$T_{Djkstra}$	$(m+n)\log n$	$m + n\log n$	$m + nC$	$m + n\log C$

Suche in Graphen

Kürzeste-Wege-Suche

Eigenschaften

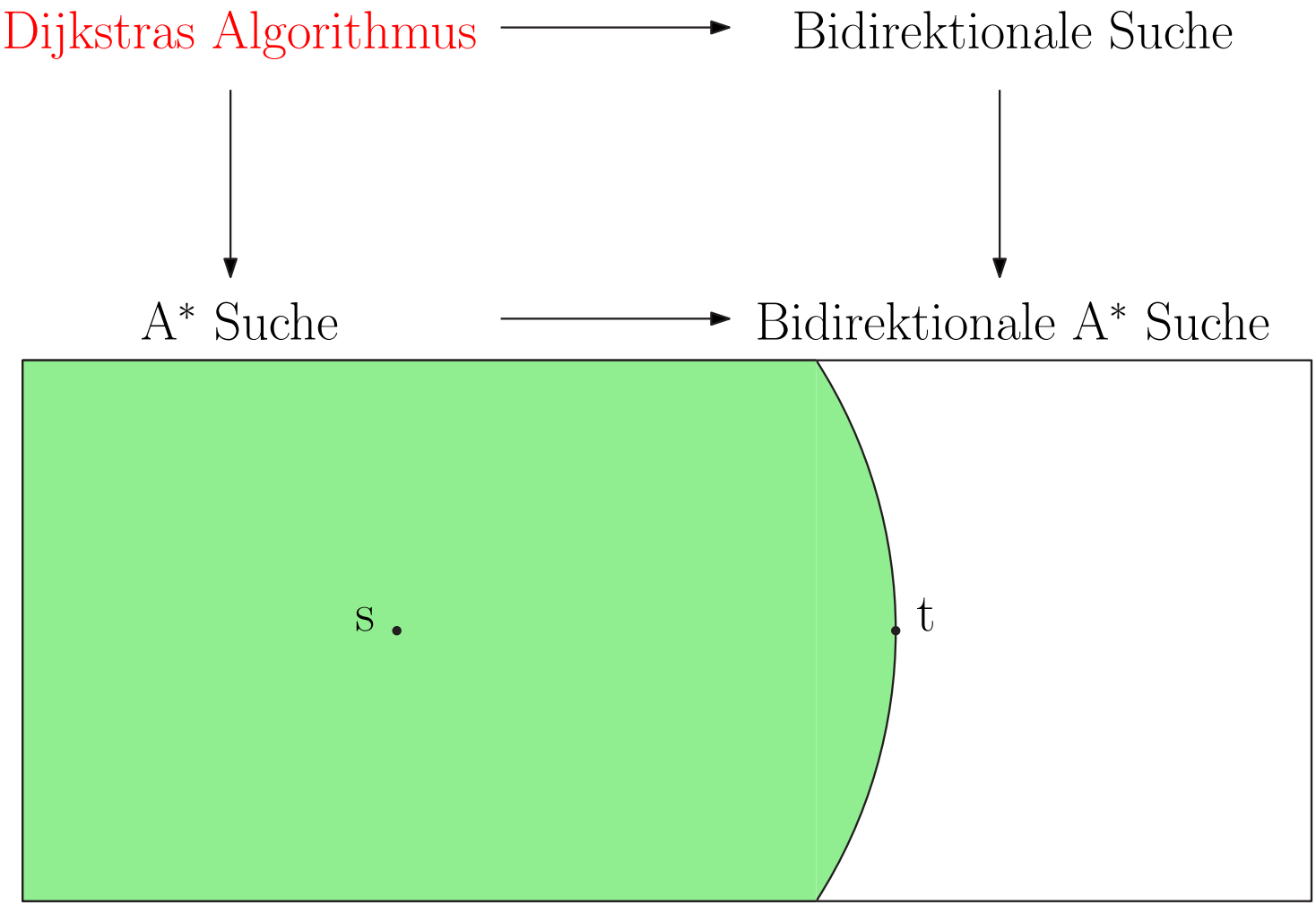
- gegeben
  - Graph $G = (V, E)$
  - Kantengewichte $c(u, v) : E \to \mathbb{R}_0^+$
- betrachte Suche von Start s nach Ziel t (One-to-One Query)
  $\to$ kürzeste Distanz $\mu(s, t)$ bestimmen

Übersicht über verschiedene Varianten

- Dijkstras Algorithmus (auch One-to-All Query)
- Bellmann Ford Algorithmus (auch negative Kantengewichte)
- Bidirektionale Suche (Beschleunigungstechnik)
- A*-Suche (Heuristik)
- ...

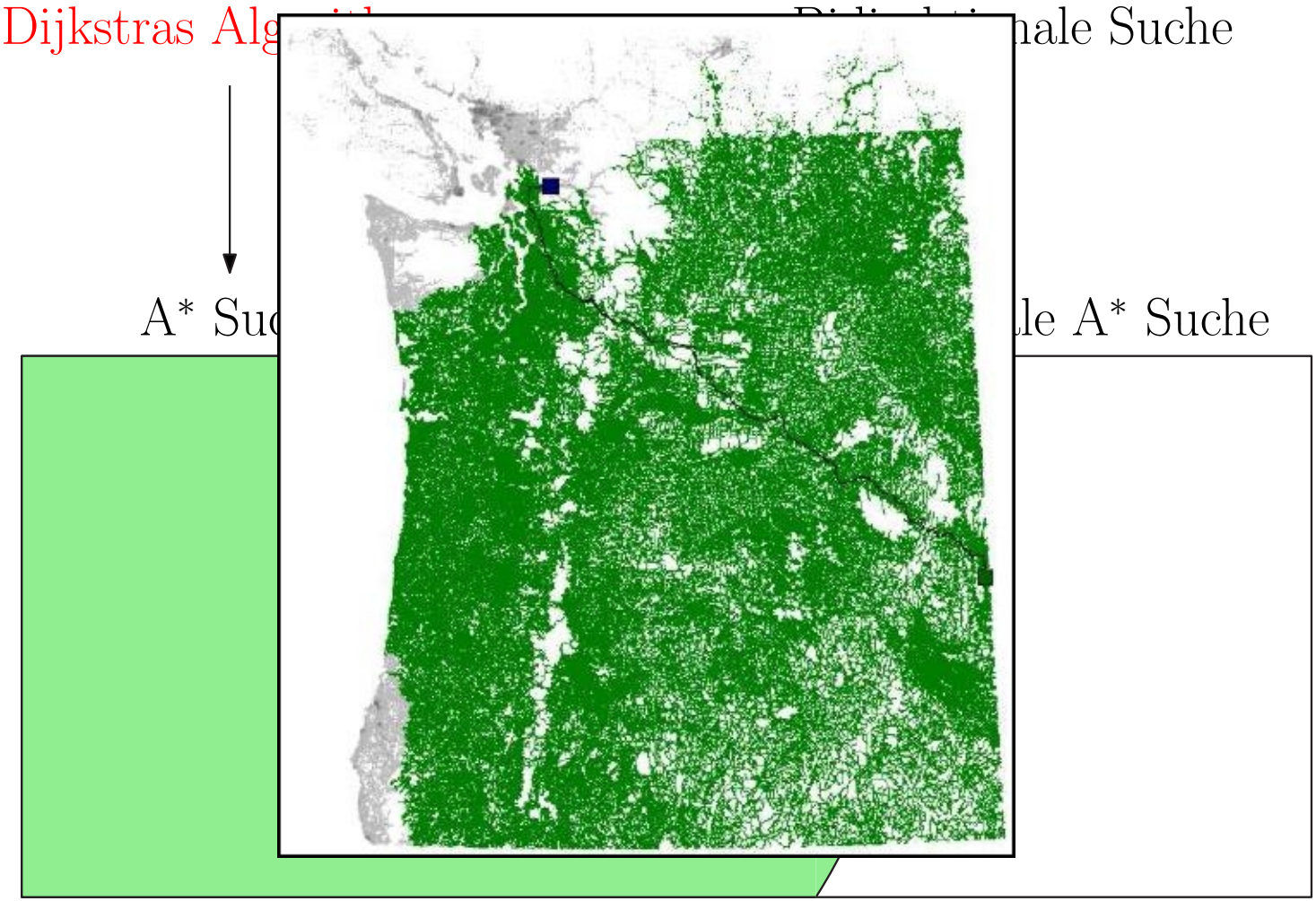
Suche in Graphen

Übersicht über verschiedene Varianten



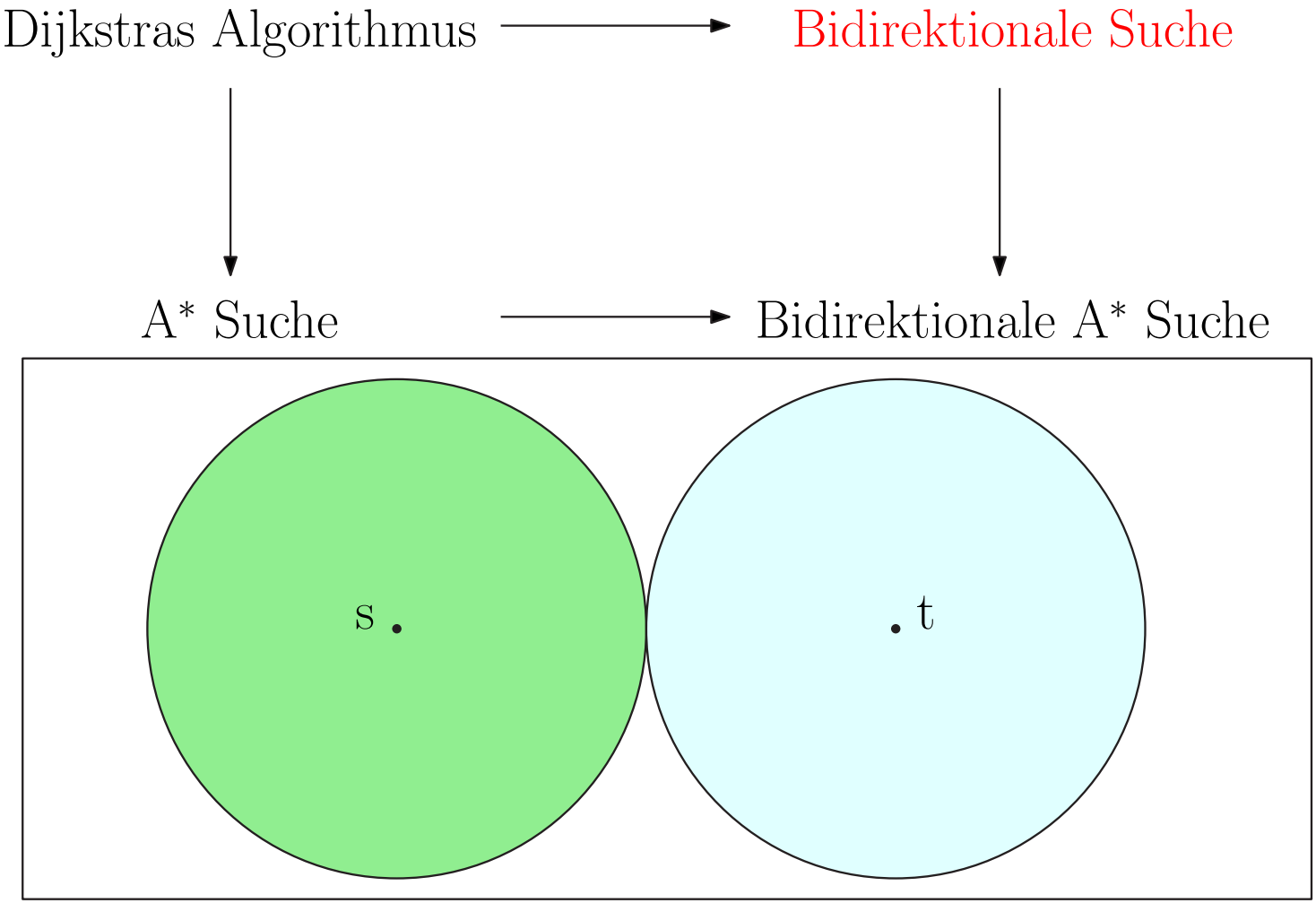
Suche in Graphen

Übersicht über verschiedene Varianten



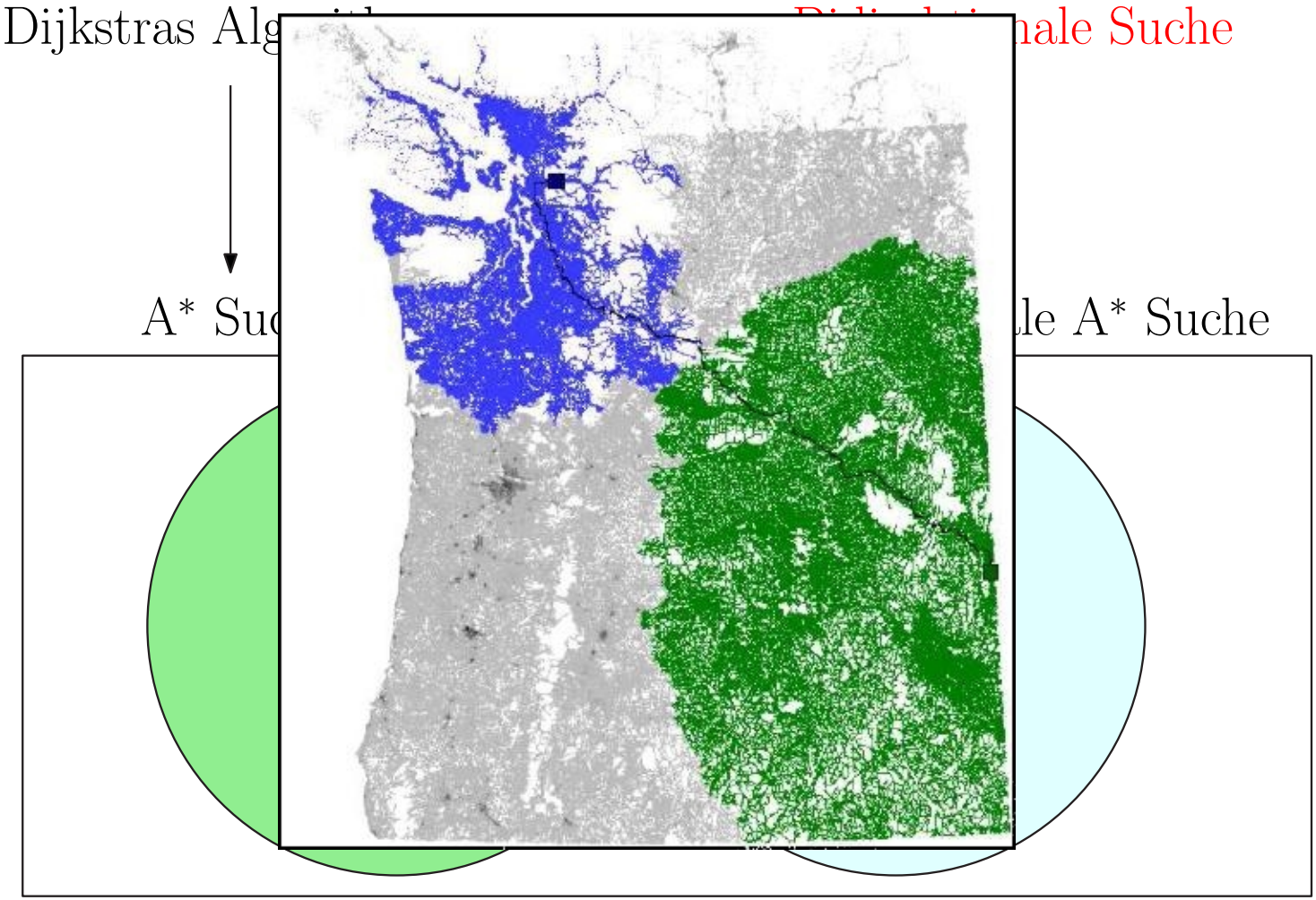
Suche in Graphen

Übersicht über verschiedene Varianten



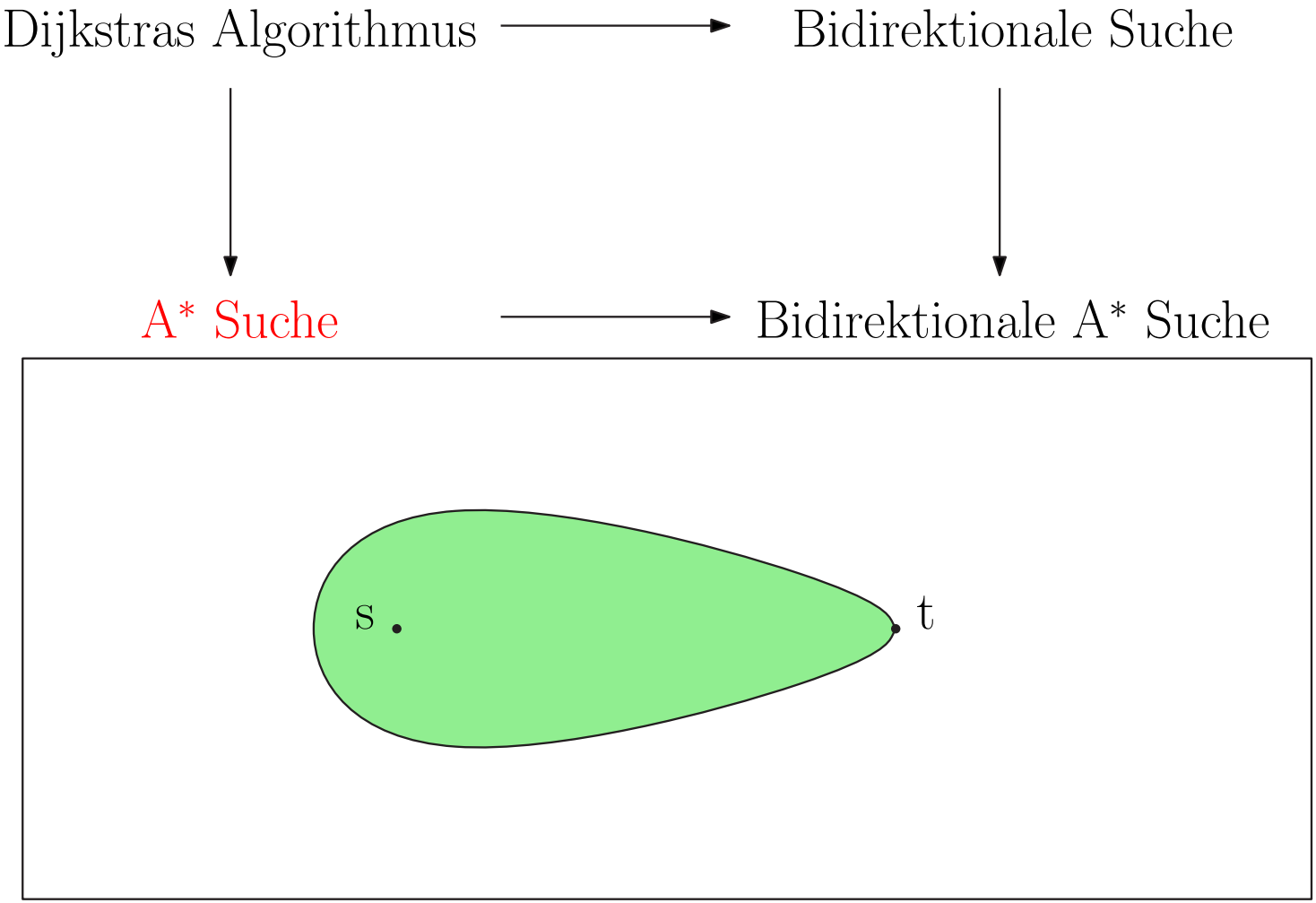
Suche in Graphen

Übersicht über verschiedene Varianten



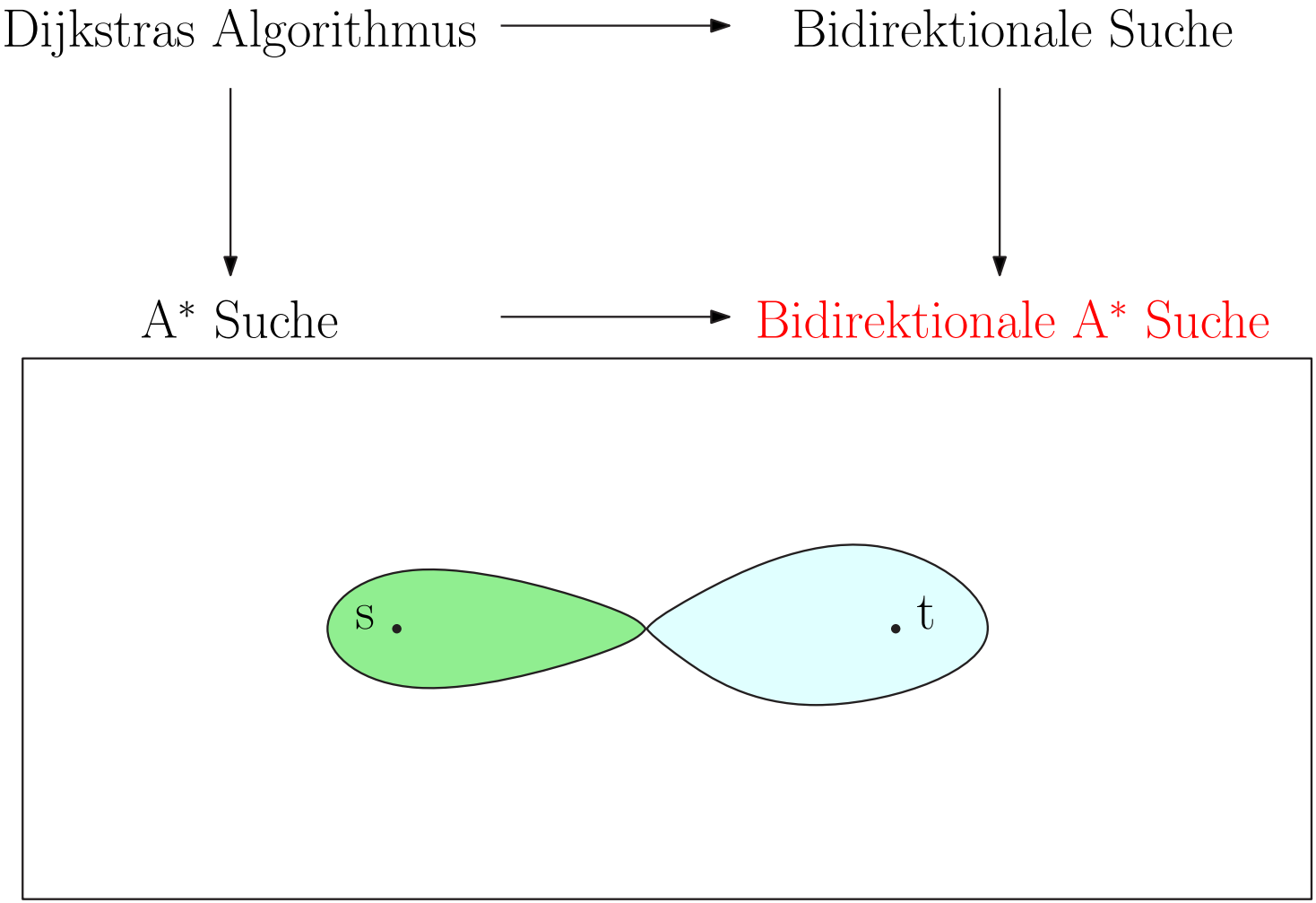
Suche in Graphen

Übersicht über verschiedene Varianten



Suche in Graphen

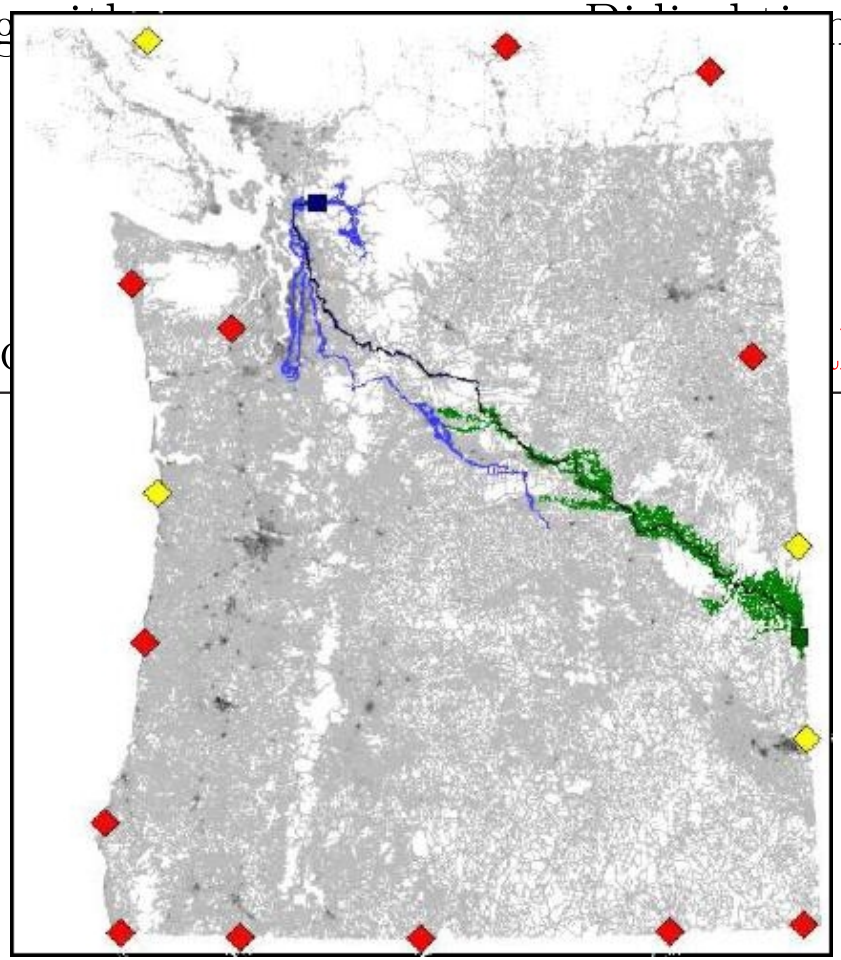
Übersicht über verschiedene Varianten



Suche in Graphen

Übersicht über verschiedene Varianten

Dijkstras Alg hale Suche



A* Suc le A* Suche

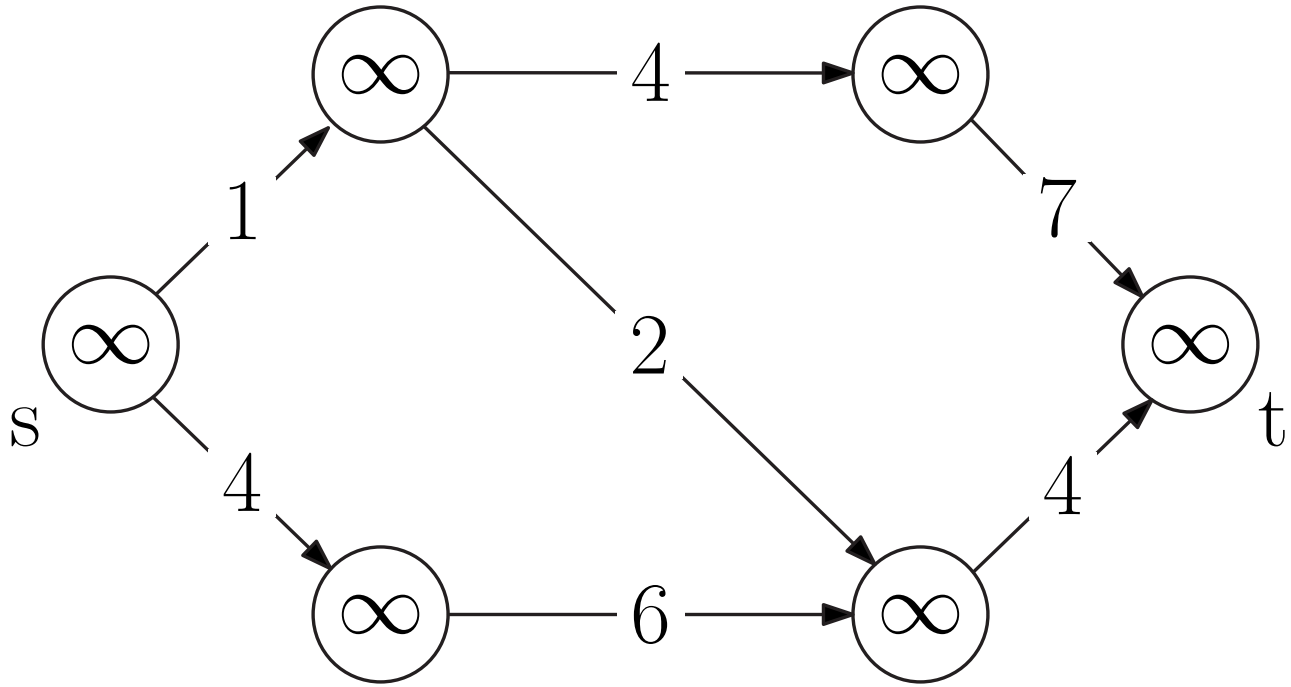
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



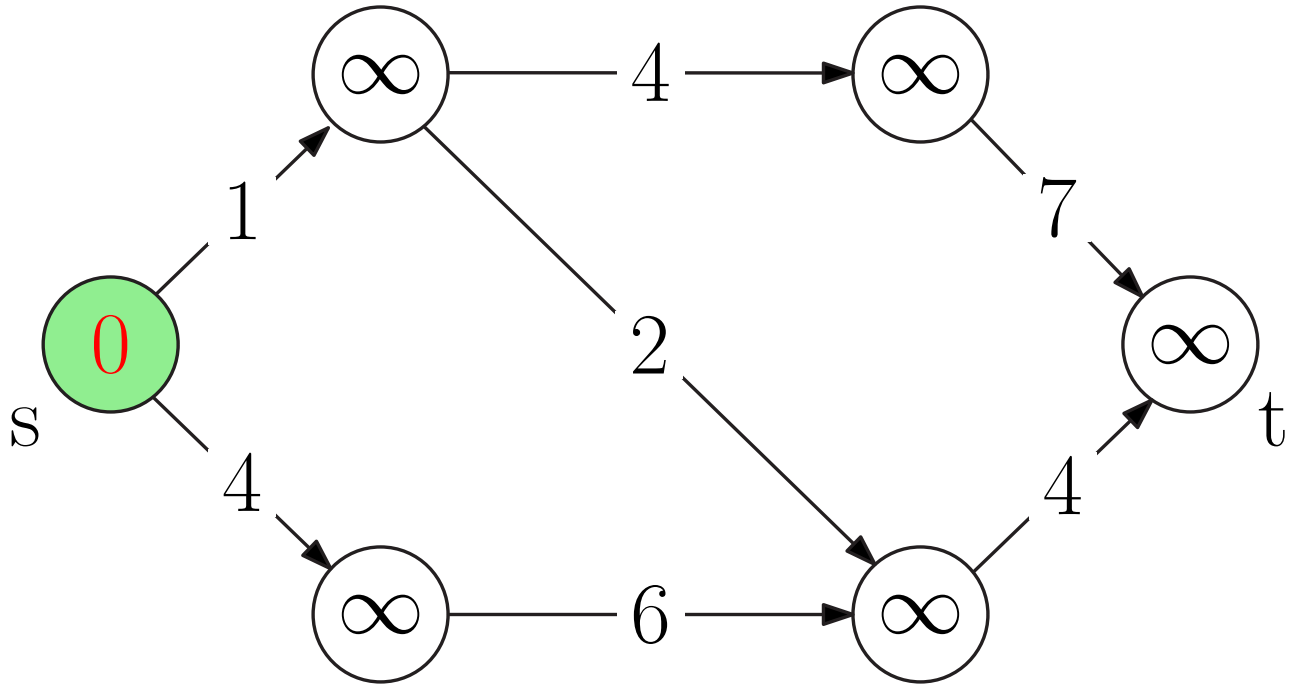
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



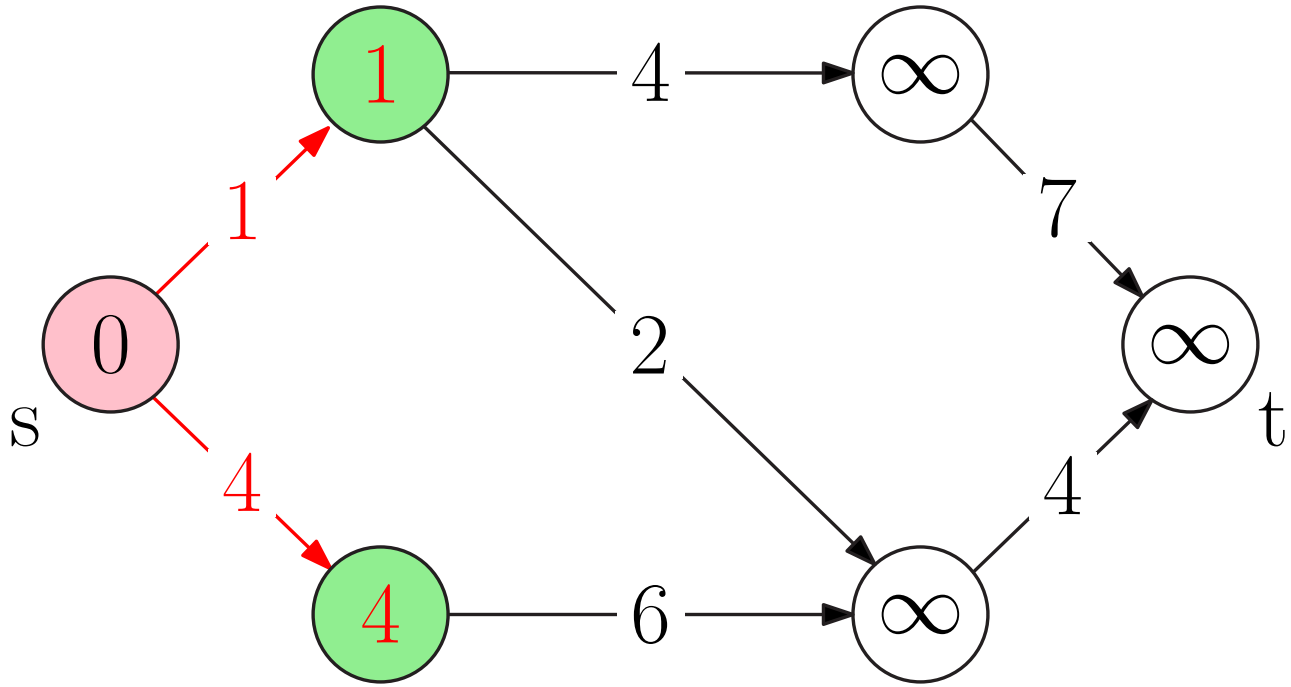
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



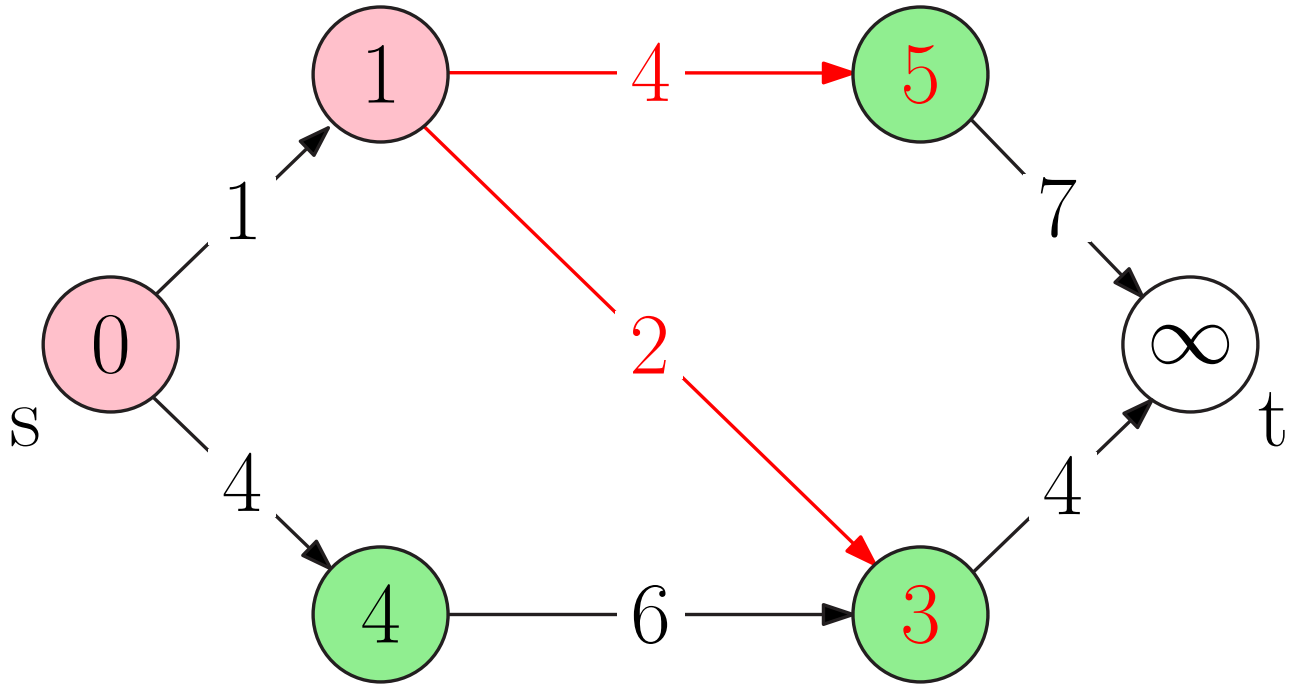
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



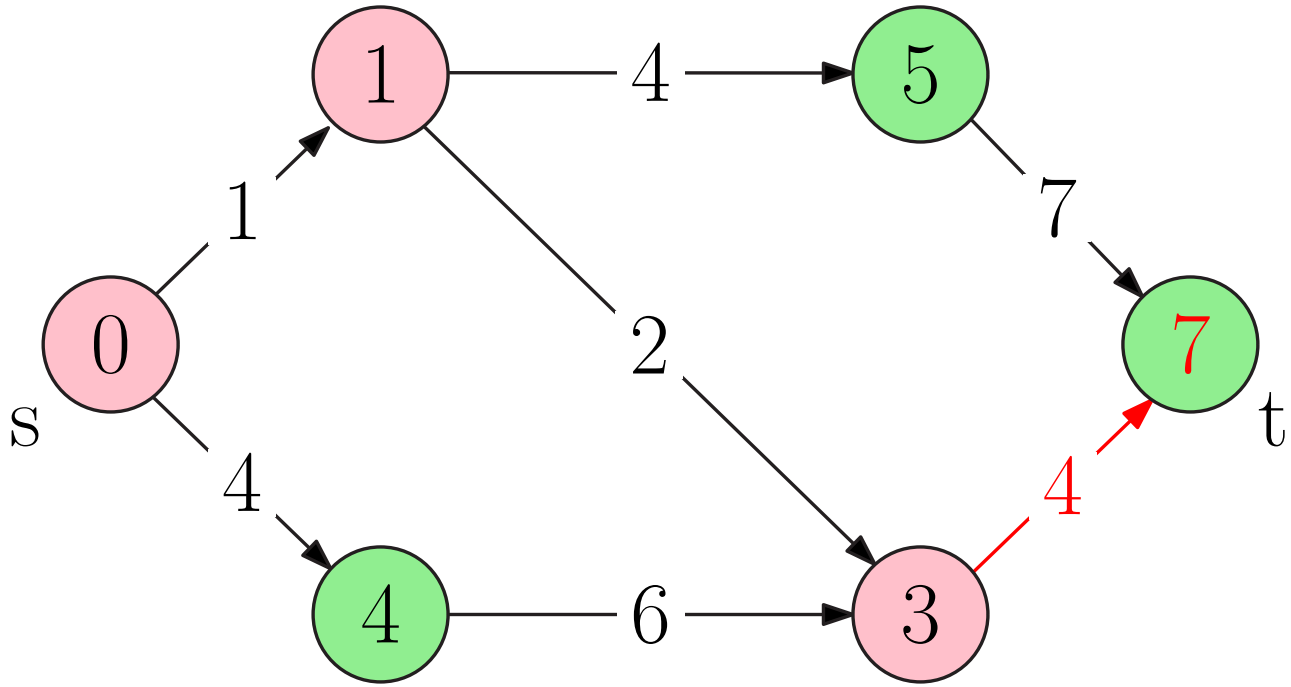
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



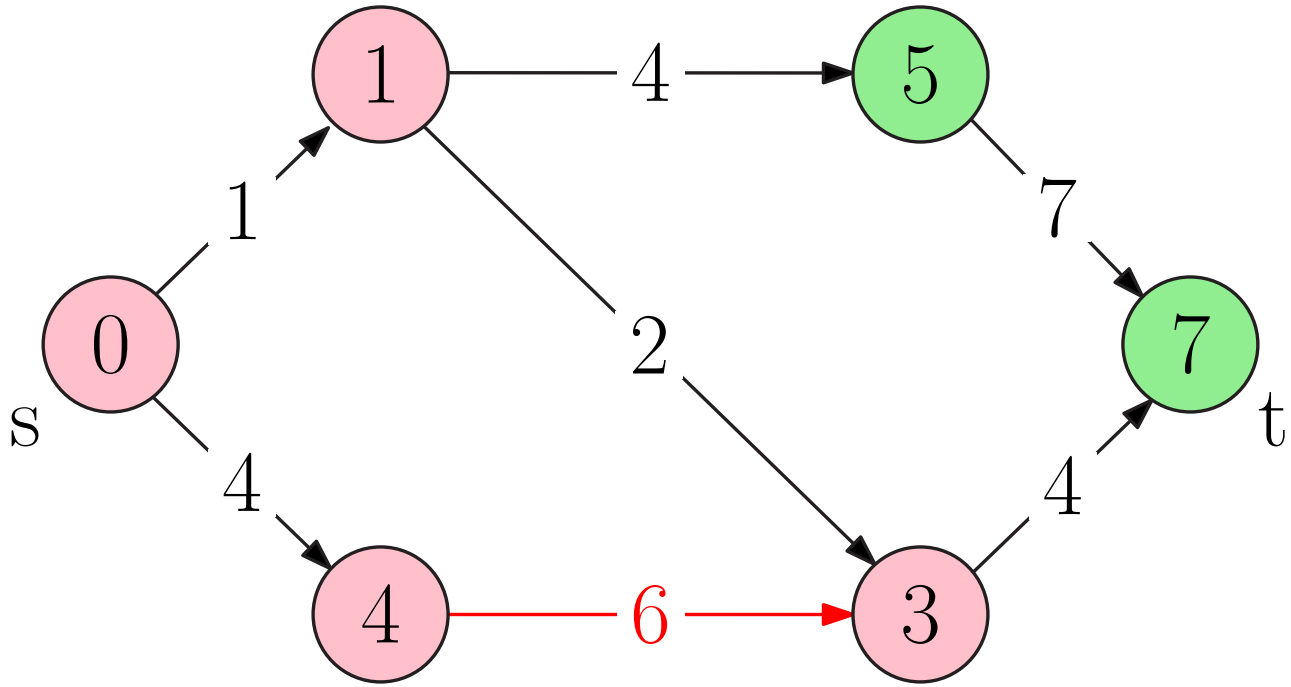
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



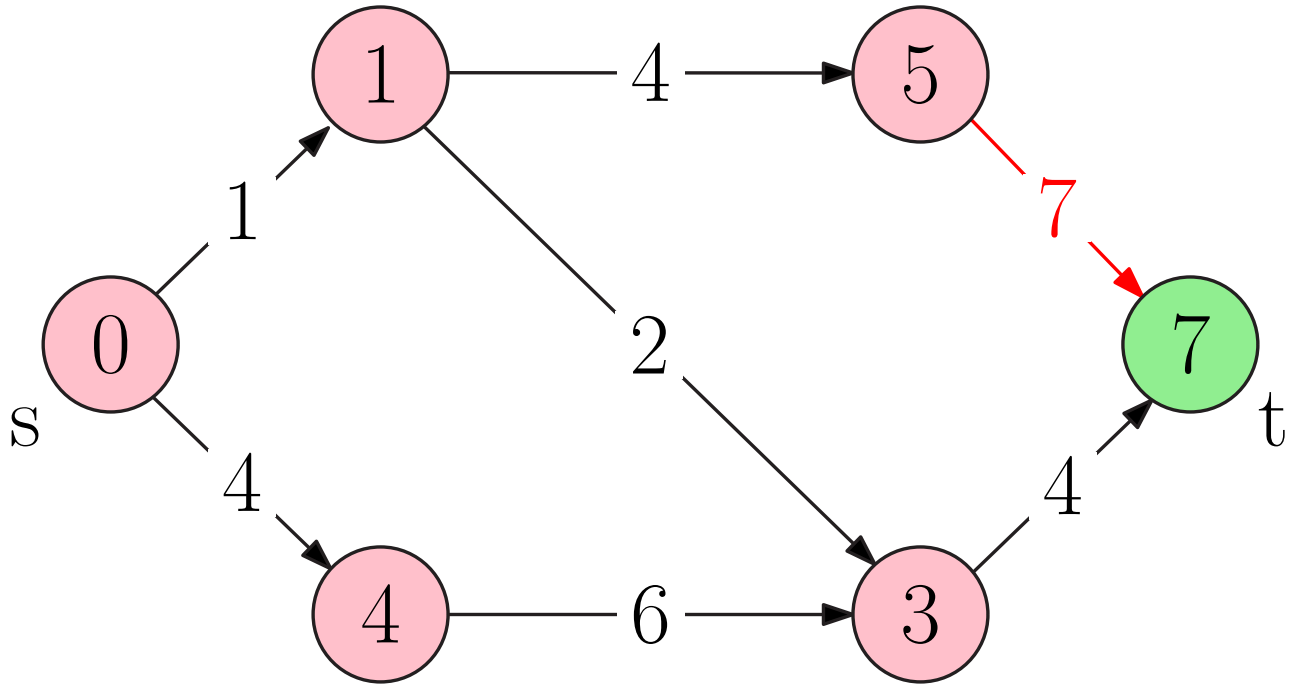
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



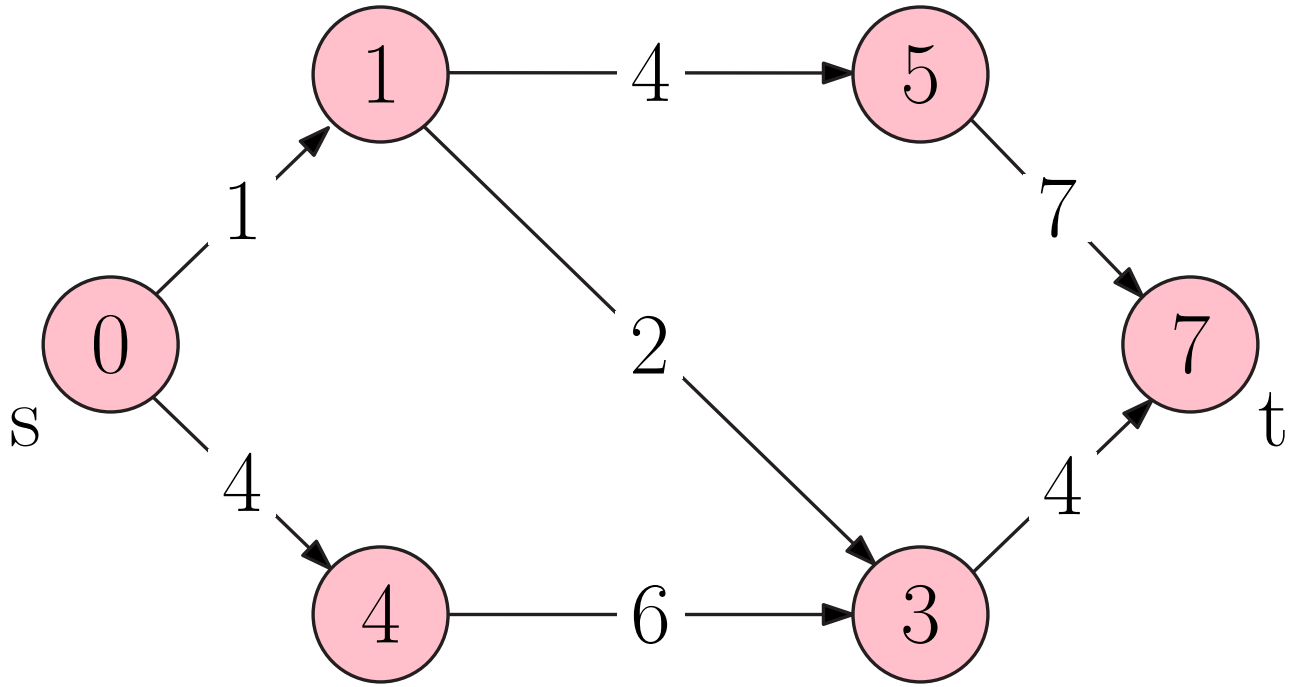
Suche in Graphen

Dijkstras Algorithmus

Einige Eigenschaften

- benötigt nicht-negative Kantengewichte
- verwaltet vorläufige Distanzen $d[\cdot]$ in *Priority Queue*
  $\rightarrow$ unterscheide *erreichte* Knoten und *gescannte* Knoten

Beispiel



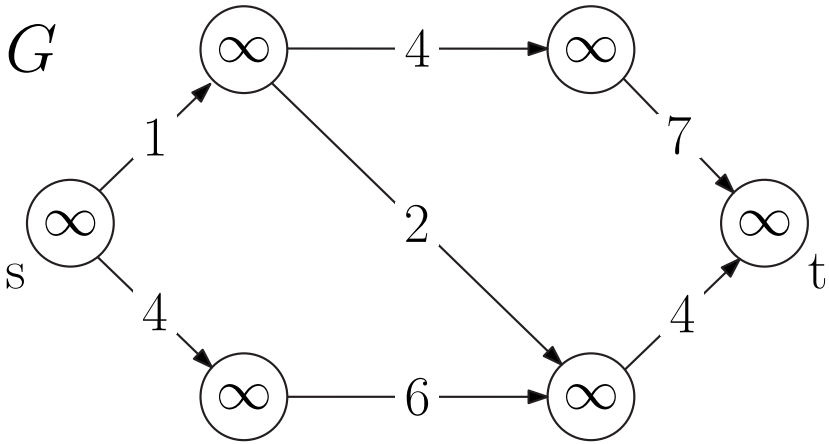
Suche in Graphen

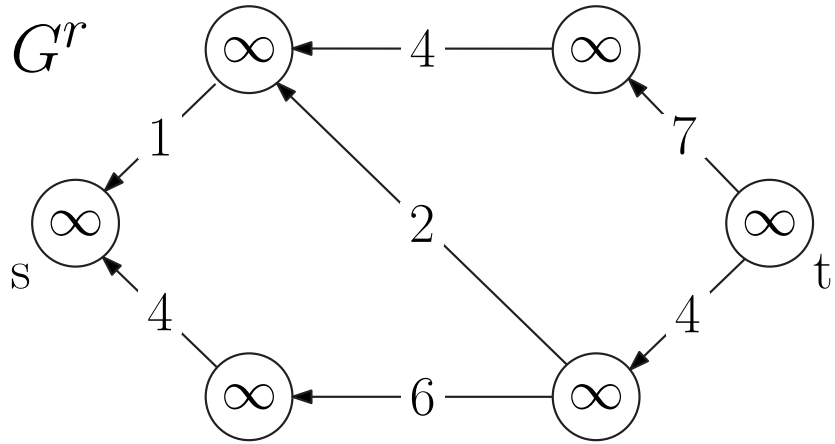
Bidirektionale Suche

Eigenschaften

- führt zweimal Dijkstras Algorithmus aus
  → Vorwärtssuche ($s \to t$) auf normalem Graph G,
  → Rückwärtssuche ($t \to s$) auf Rückwärtsgraph G^r

Beispiel





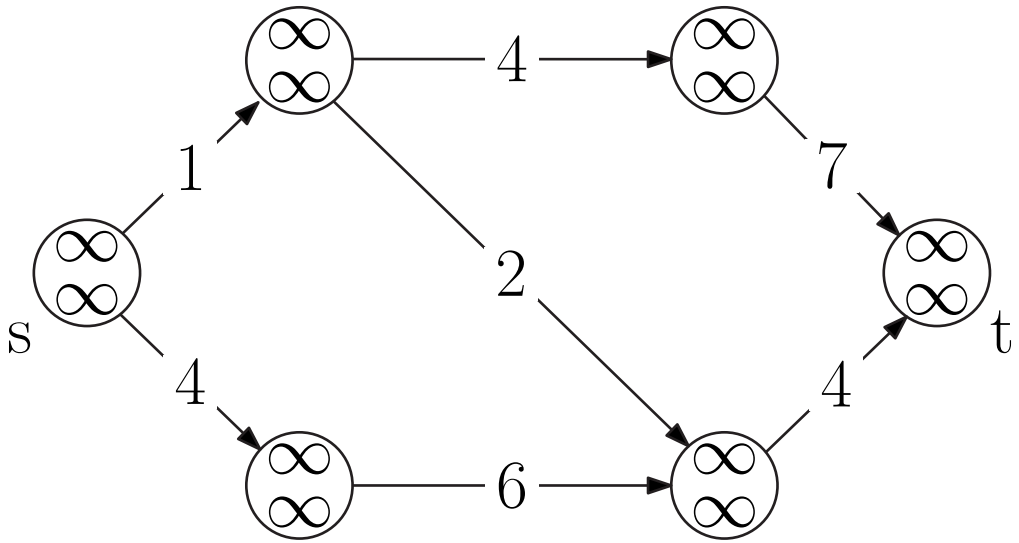
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



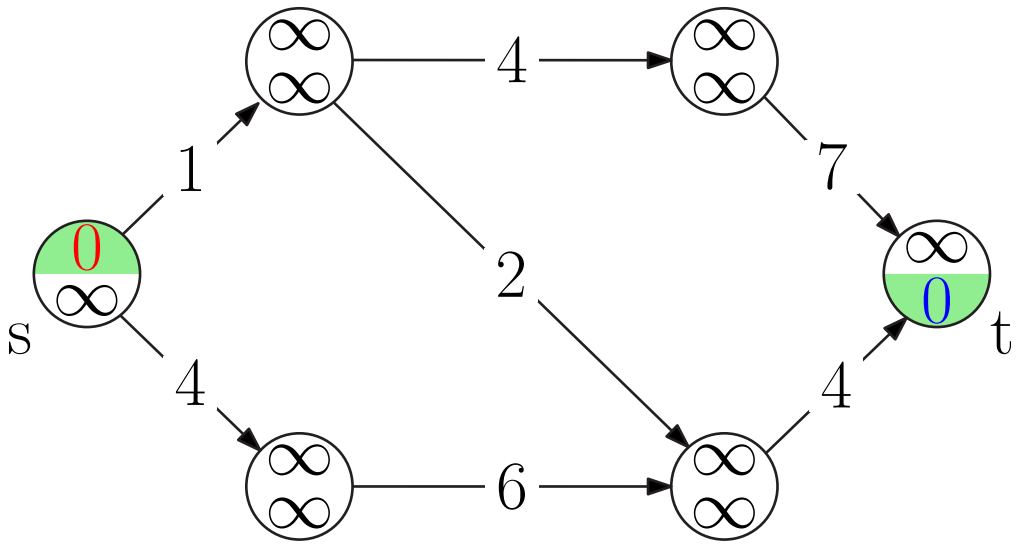
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



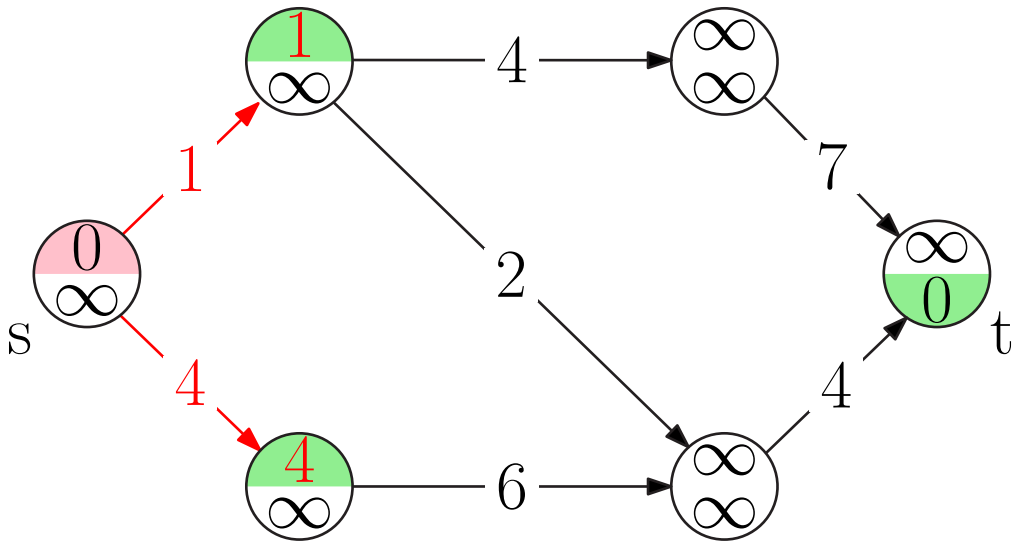
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



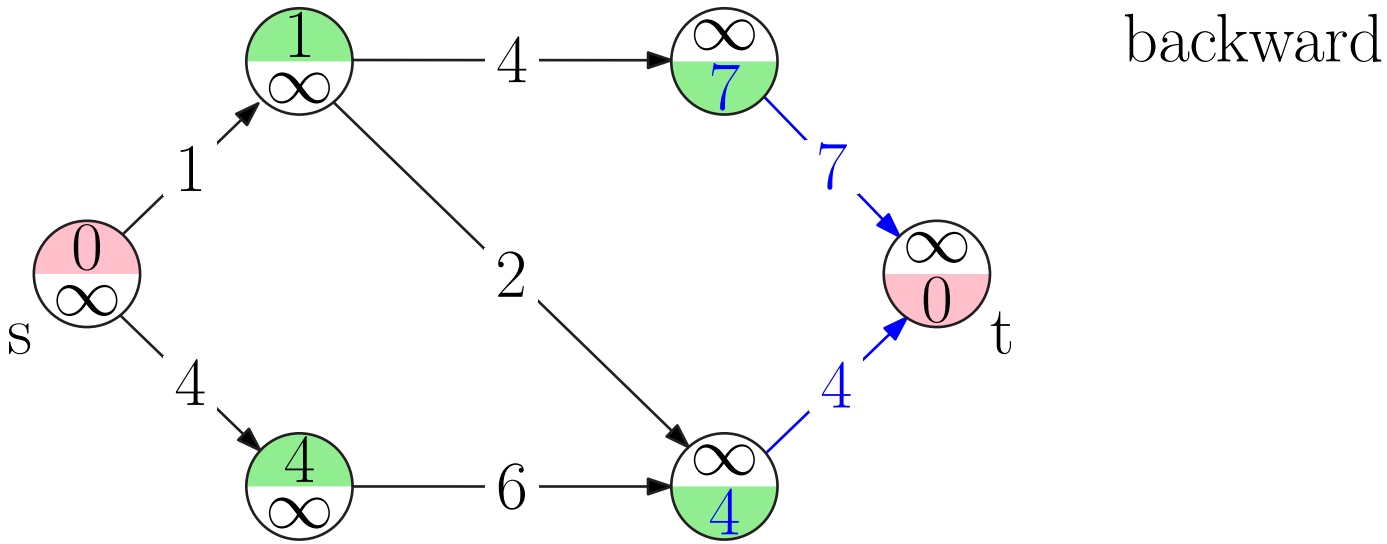
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



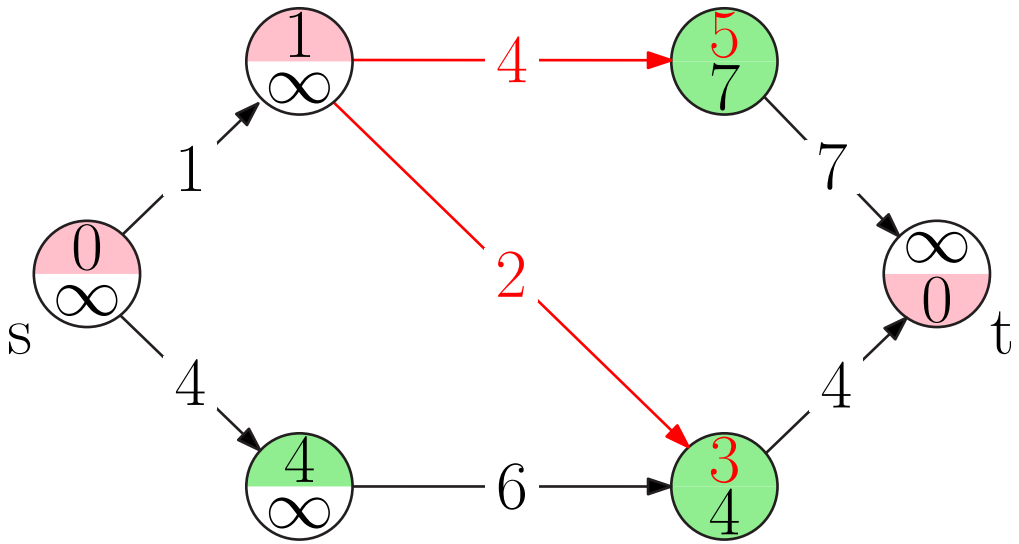
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



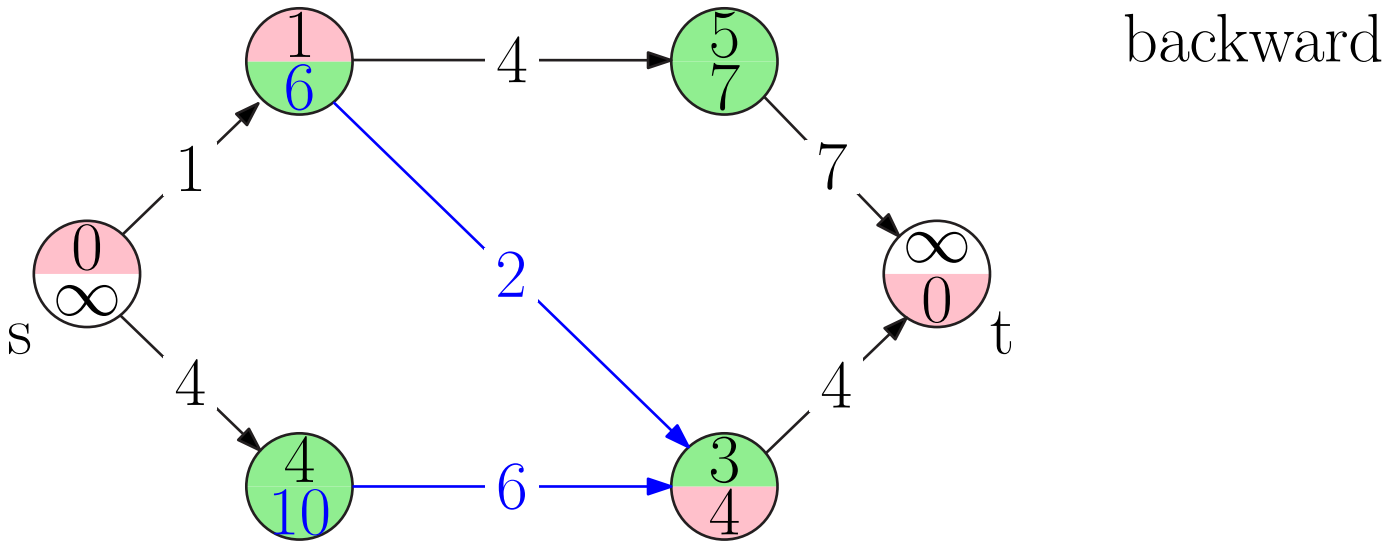
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



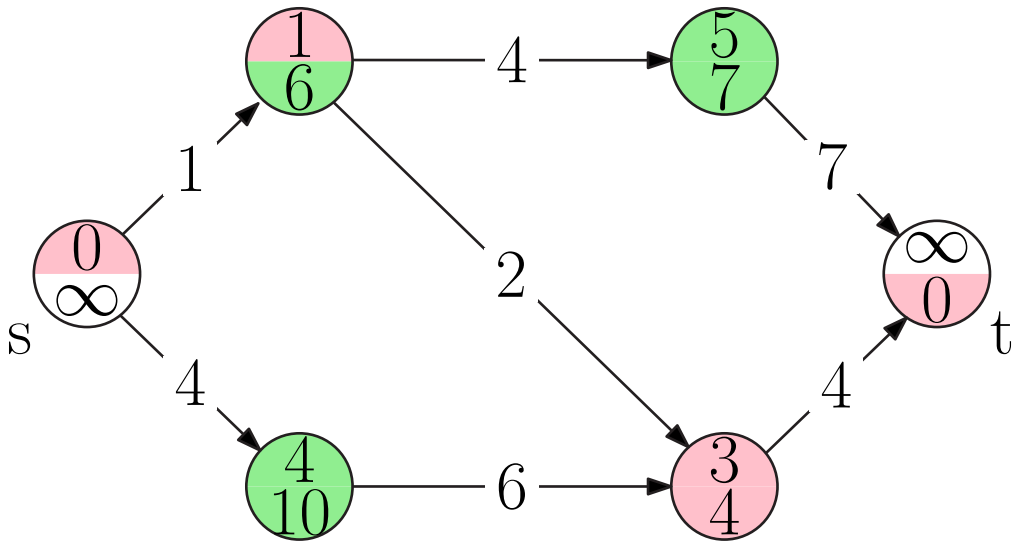
Suche in Graphen

Bidirektionale Suche

Eigenschaften (weiter)

- wechselt Suchrichtung in jedem Schritt
  (alternativ: wähle Richtung mit kleinerem `pq.min`)
- Abbruch, wenn ein Knoten in beiden Suchen gescannt wurde
  (aufpassen bei alternativer Wahl der Suchrichtung!)

Beispiel



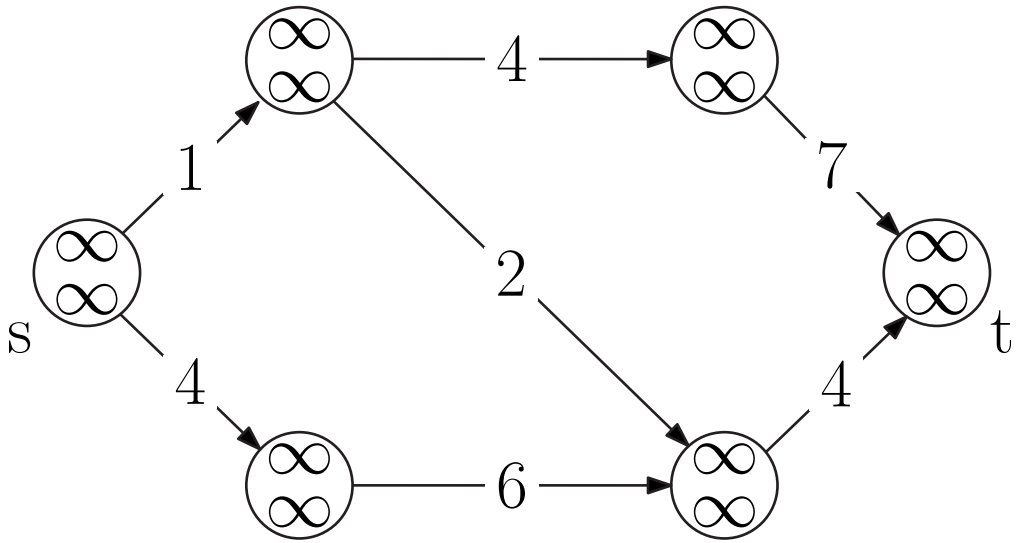
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



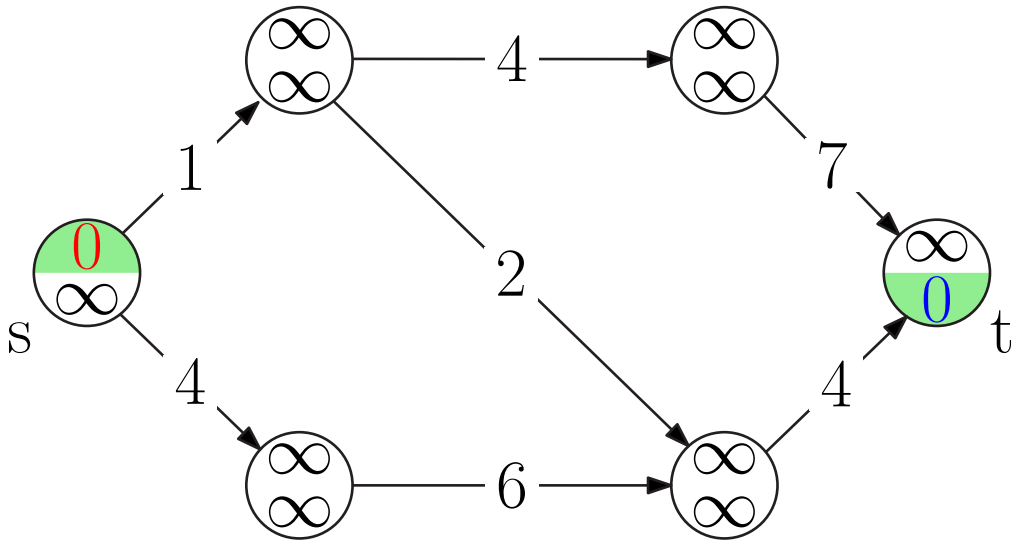
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



$d[s,t] = \infty$

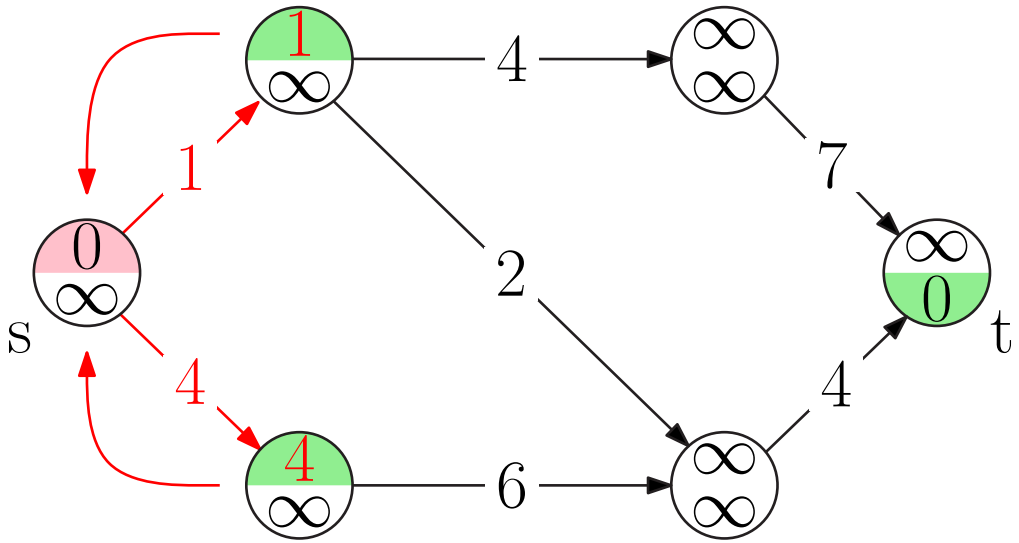
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



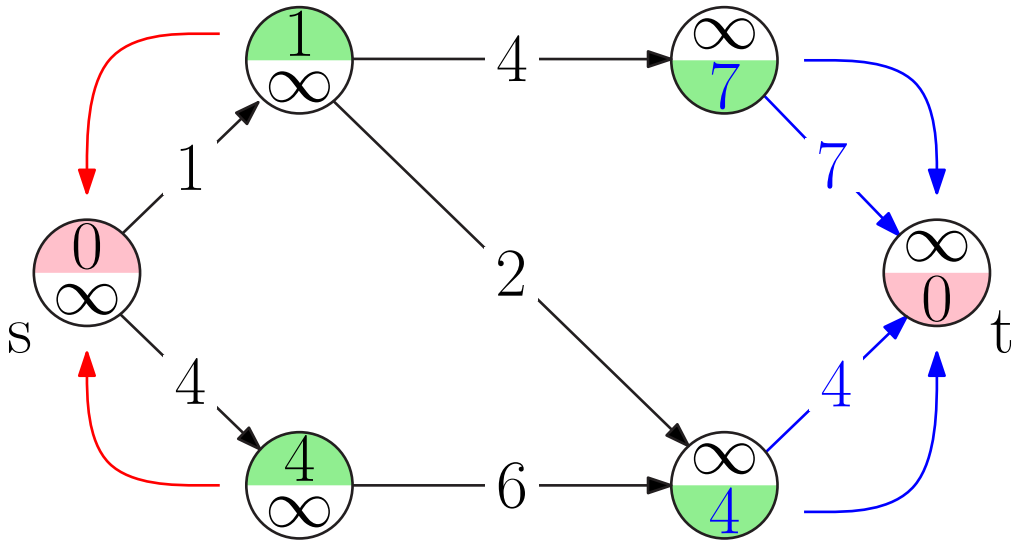
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



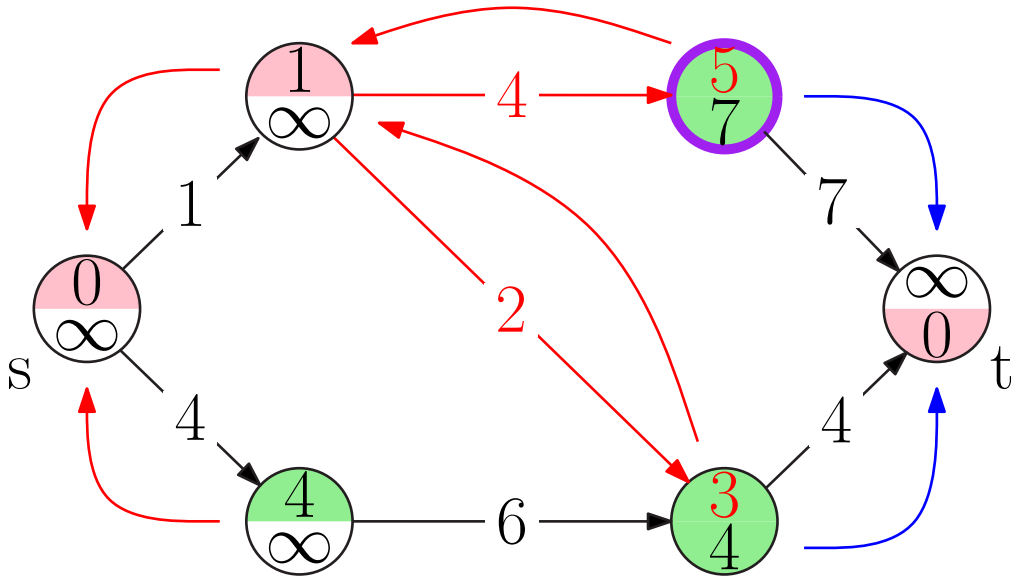
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



forward

$d[s,t] = 12$

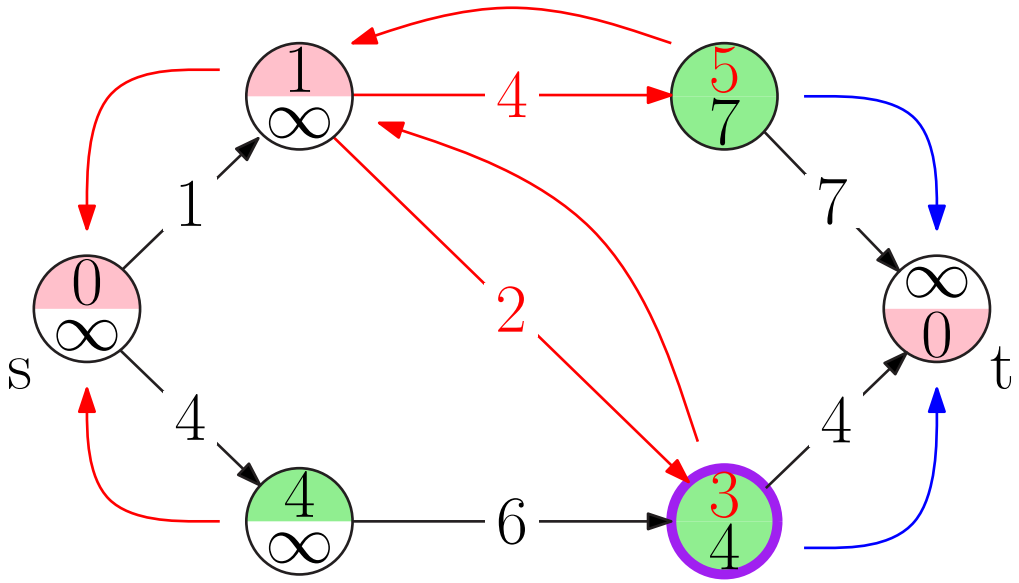
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



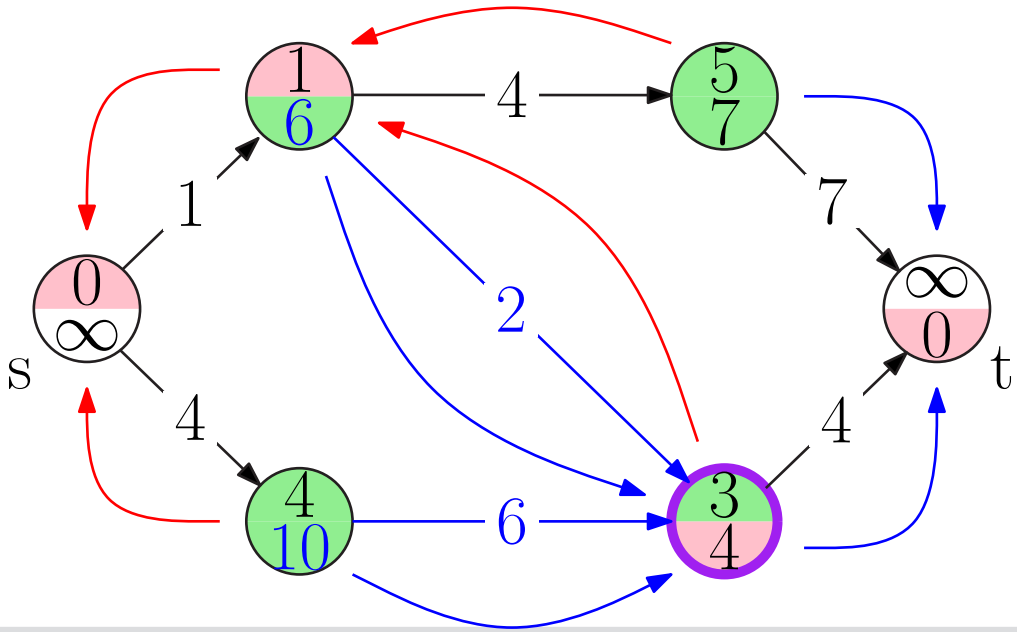
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



backward

$d[s,t] = 7$

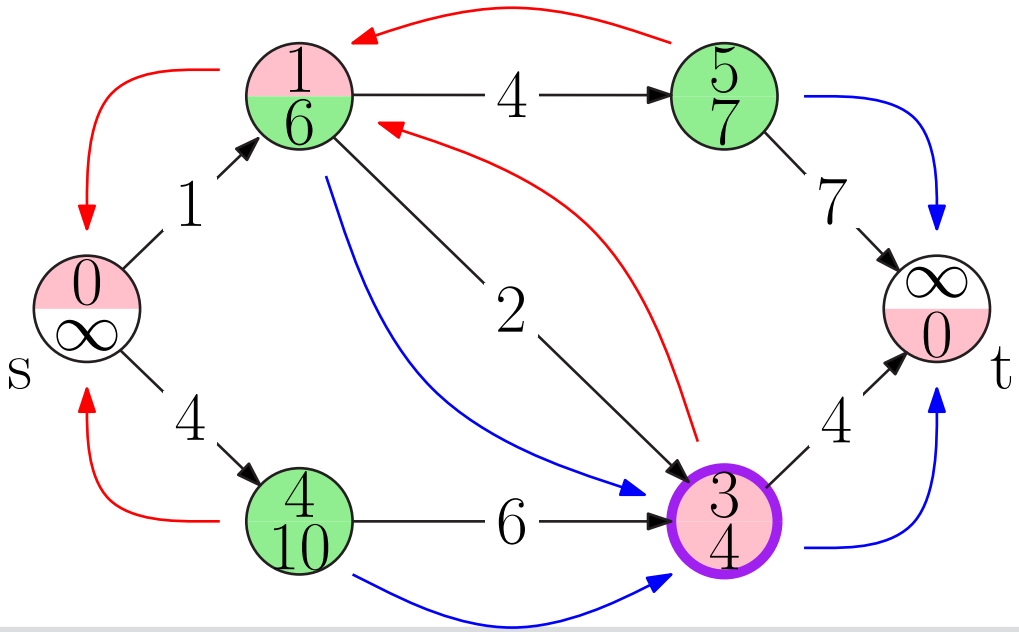
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



forward

$d[s,t] = 7$

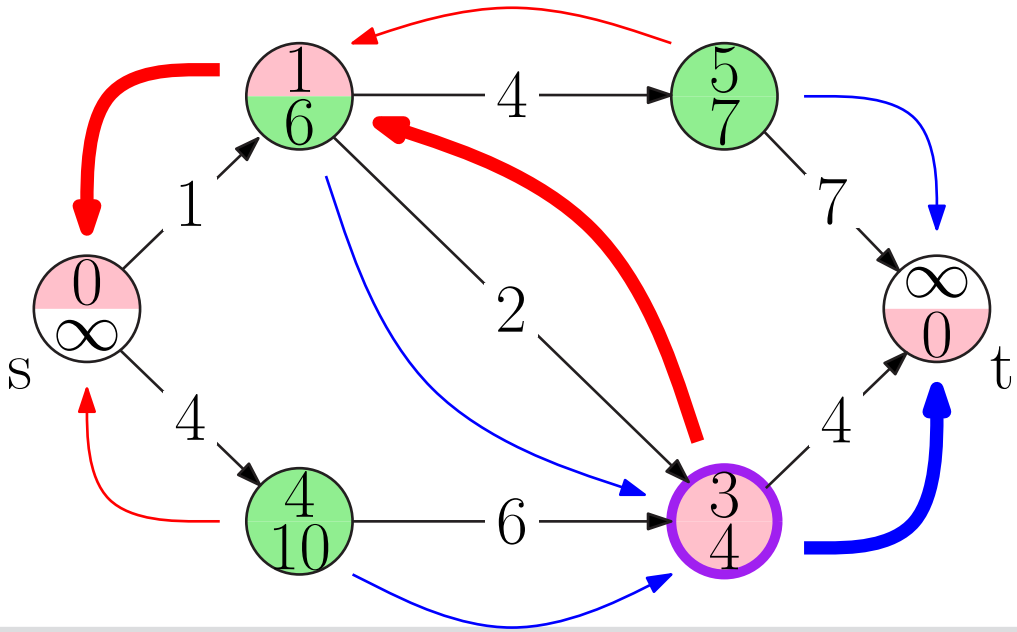
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



forward

$d[s,t] = 7$

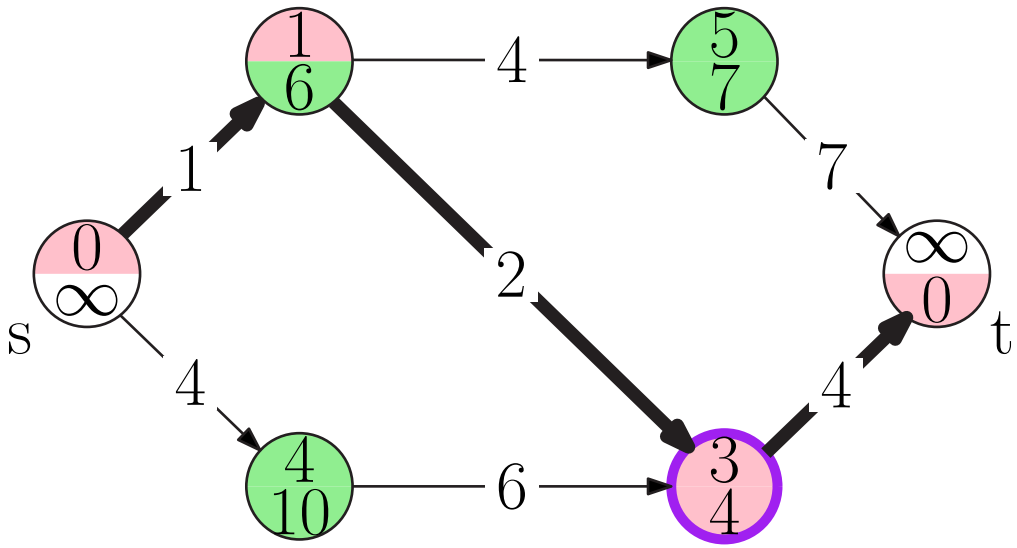
Suche in Graphen

Bidirektionale Suche

Wie wissen wir kürzeste Distanz und Weg nach Abbruch?

- wenn sich $d_{fwd}[v]$ oder $d_{bwd}[v]$ ändert,
  - aktualisiere Vorgänger u,
  - falls vorläufige kürzeste Distanz $d[s,t] > d_{fwd}[v] + d_{bwd}[v]$
    $\rightarrow$ aktualisiere $d[s,t]$,
    $\rightarrow$ aktualisiere Treffpunkt v

Beispiel



forward

$d[s,t] = 7$

Suche in Graphen

A*-Suche

Eigenschaften

- zielgerichtete Suche
- benötigt Potentialfunktion $\text{pot}(\cdot) : V \longrightarrow \mathbb{R}$
  - reduzierte Kantengewichte $\overline{c}(u, v) := c(u, v) + \text{pot}(v) - \text{pot}(u)$ bzw.
  - modifizierte Schlüssel $\overline{d}[v] := d[v] + \text{pot}(v)$

  $\rightarrow$ Abarbeitung der Knoten wird geändert (verbessert?)
  $\rightarrow$ kürzeste Pfade bleiben erhalten

Suche in Graphen

A*-Suche – Potentialfunktionen

Eigenschaften

- Potentialfunktion $\text{pot}(\cdot) : V \longrightarrow \mathbb{R}$
- heuristische Funktion
  - modelliert vorhandenes Wissen über Graph
  - schätzt Distanz zum Ziel $\rightarrow \overline{d}[v]$ schätzt $\mu(s,t)$

gültige Potentialfunktion $\text{pot}(\cdot)$

- untere Schranke für Distanz zum Ziel t
  $\rightarrow \text{pot}(u) \leq \mu(u,t) \quad \forall u \in V$
  (beendet Suche sobald t gescannt wurde)
- nicht-negative reduzierte Kantengewichte
  $\rightarrow \overline{c}(u,v) := c(u,v) + \text{pot}(v) - \text{pot}(u) \geq 0 \quad \forall (u,v) \in E$
  (für Dijkstras Algorithmus)

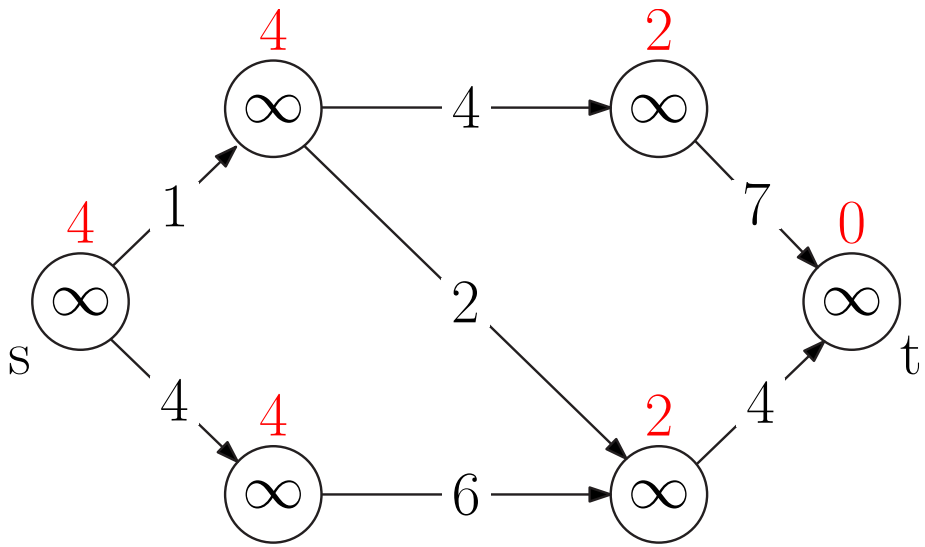
Suche in Graphen

A*-Suche – Potentialfunktionen

Woher nehmen?

- Manhattan-Distanz (nur Gittergraphen), euklidischer Abstand, . . .
  (benötigt geometrische Einbettung des Graphen)
- Distanzen zu Landmarken und Dreiecksungleichung
  (funktioniert ohne Einbettung)
- Erfahrungswerte
  (z.B. Bewertung von Spielsituationen)

Beispiel



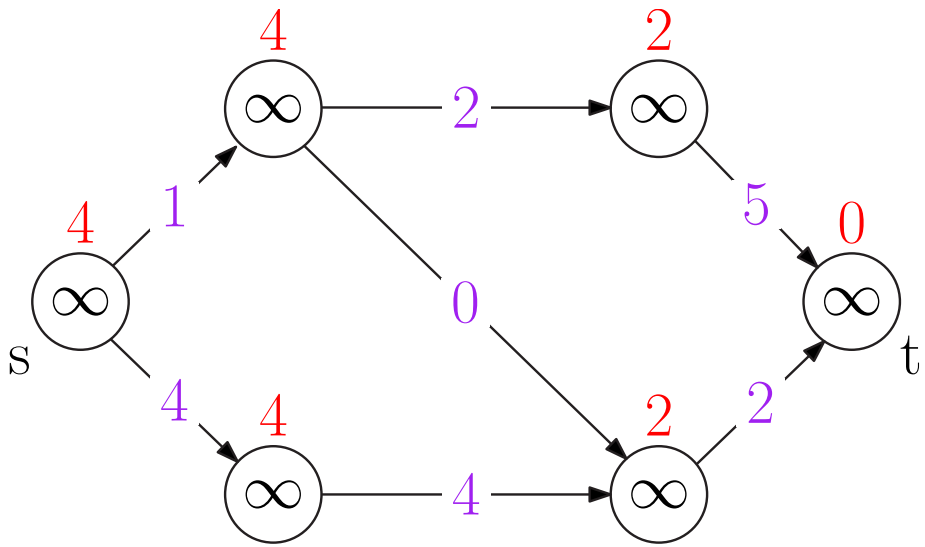
Suche in Graphen

A*-Suche – Potentialfunktionen

Woher nehmen?

- Manhattan-Distanz (nur Gittergraphen), euklidischer Abstand, . . .
  (benötigt geometrische Einbettung des Graphen)
- Distanzen zu Landmarken und Dreiecksungleichung
  (funktioniert ohne Einbettung)
- Erfahrungswerte
  (z.B. Bewertung von Spielsituationen)

Beispiel

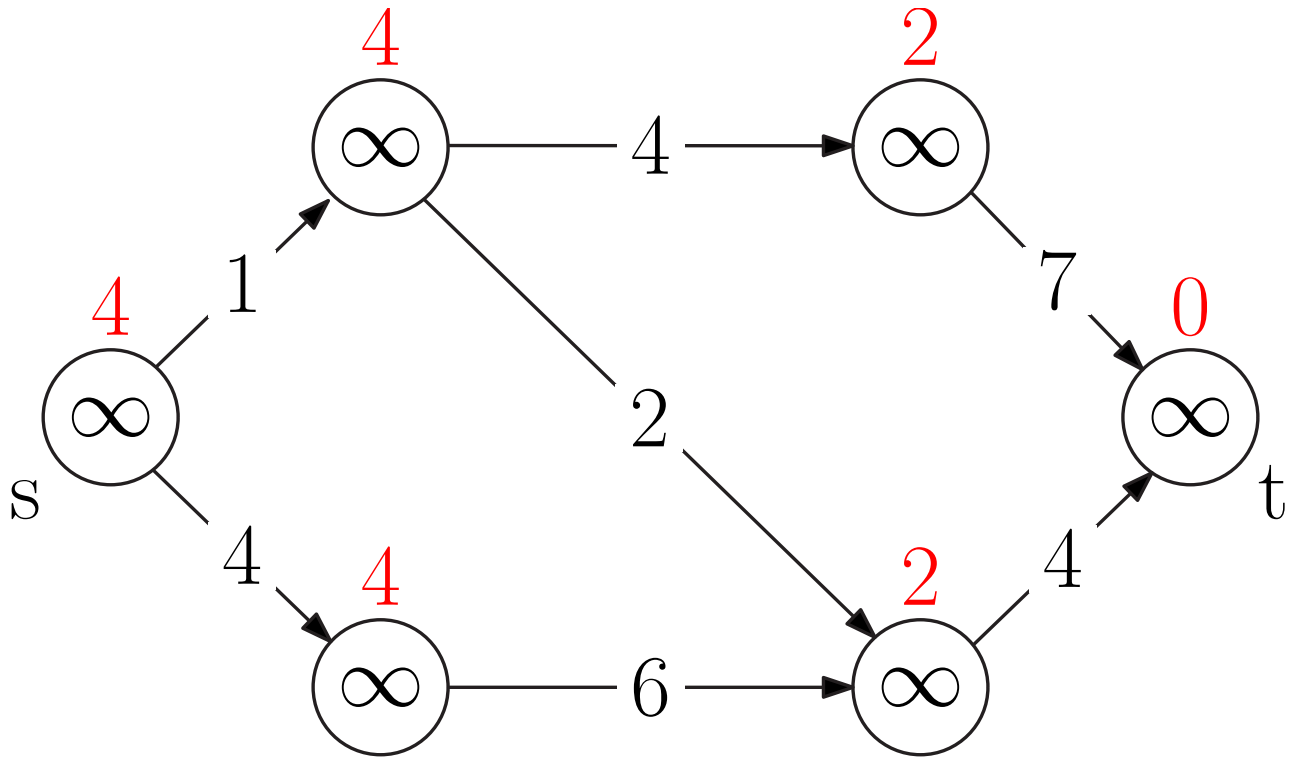


Suche in Graphen

A*-Suche

Beispiel

- Suche mit reduzierten Kantengewichten

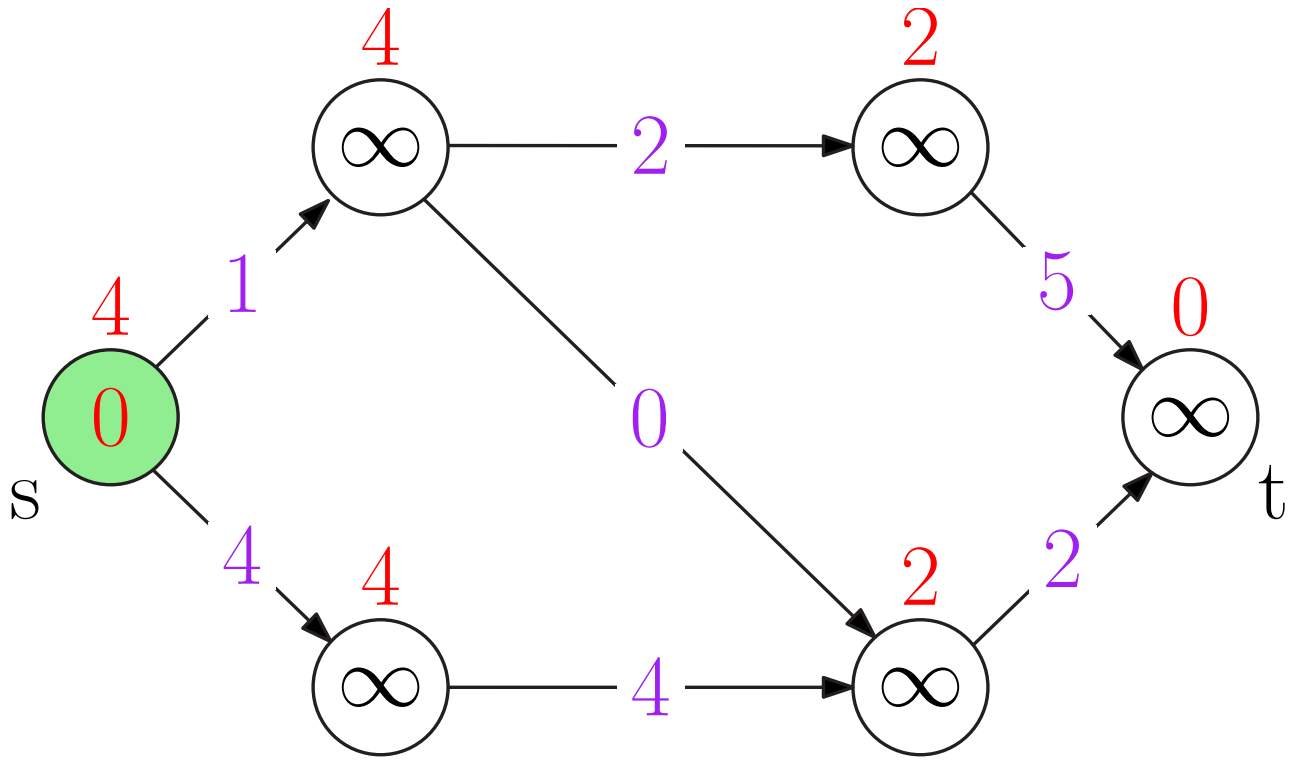


Suche in Graphen

A*-Suche

Beispiel

- Suche mit reduzierten Kantengewichten

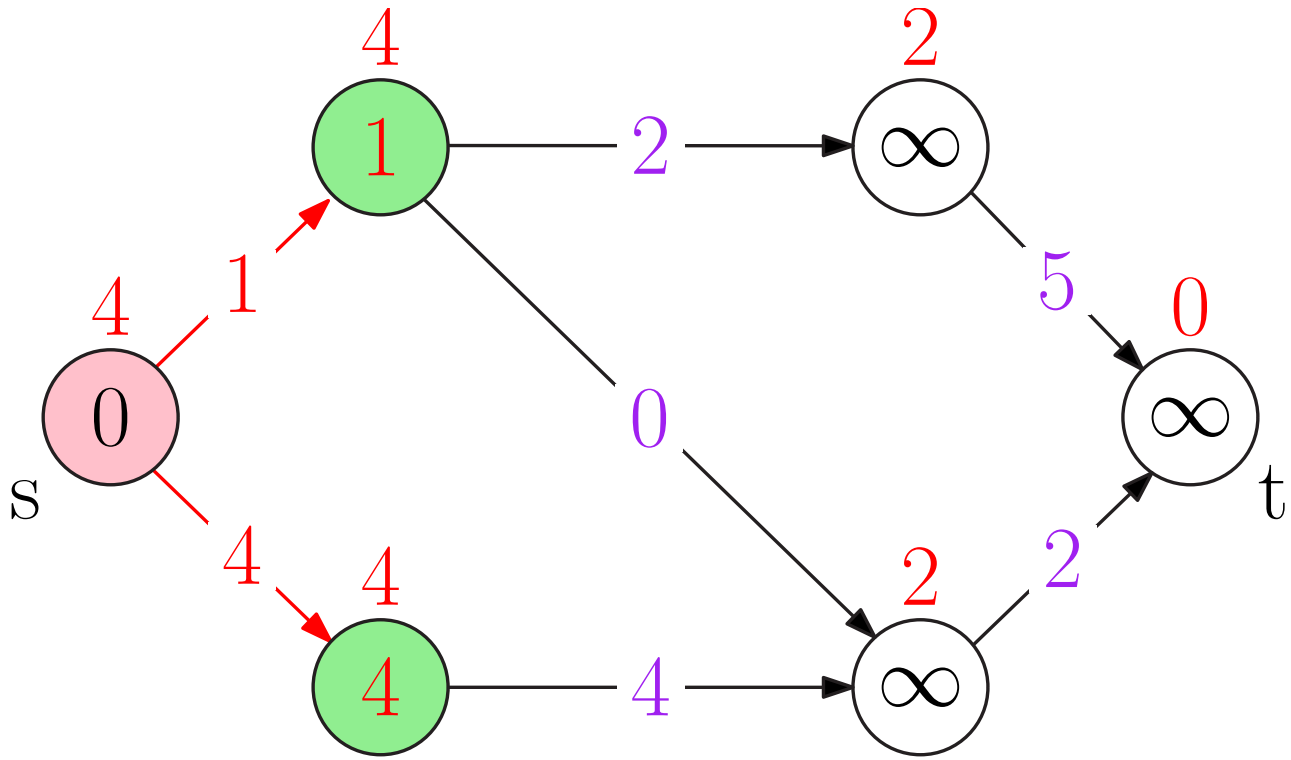


Suche in Graphen

A*-Suche

Beispiel

- Suche mit reduzierten Kantengewichten

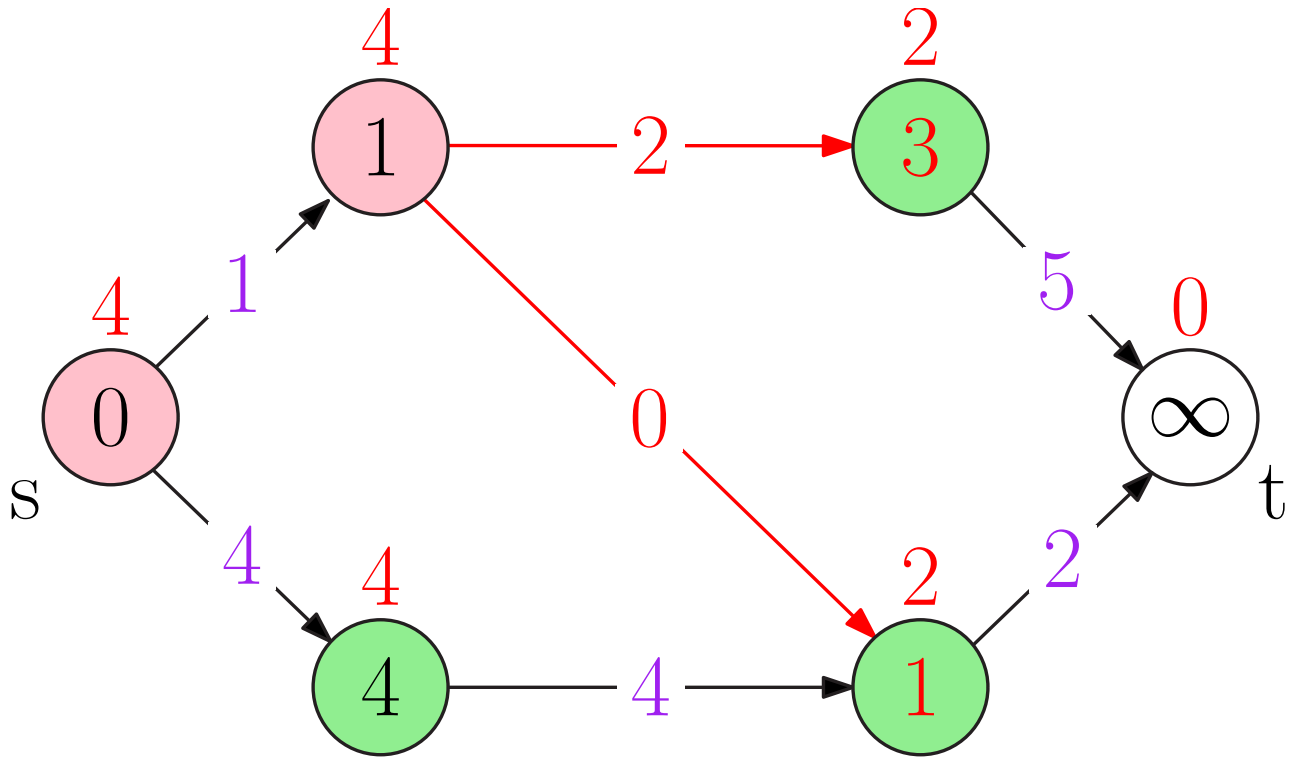


Suche in Graphen

A*-Suche

Beispiel

- Suche mit reduzierten Kantengewichten

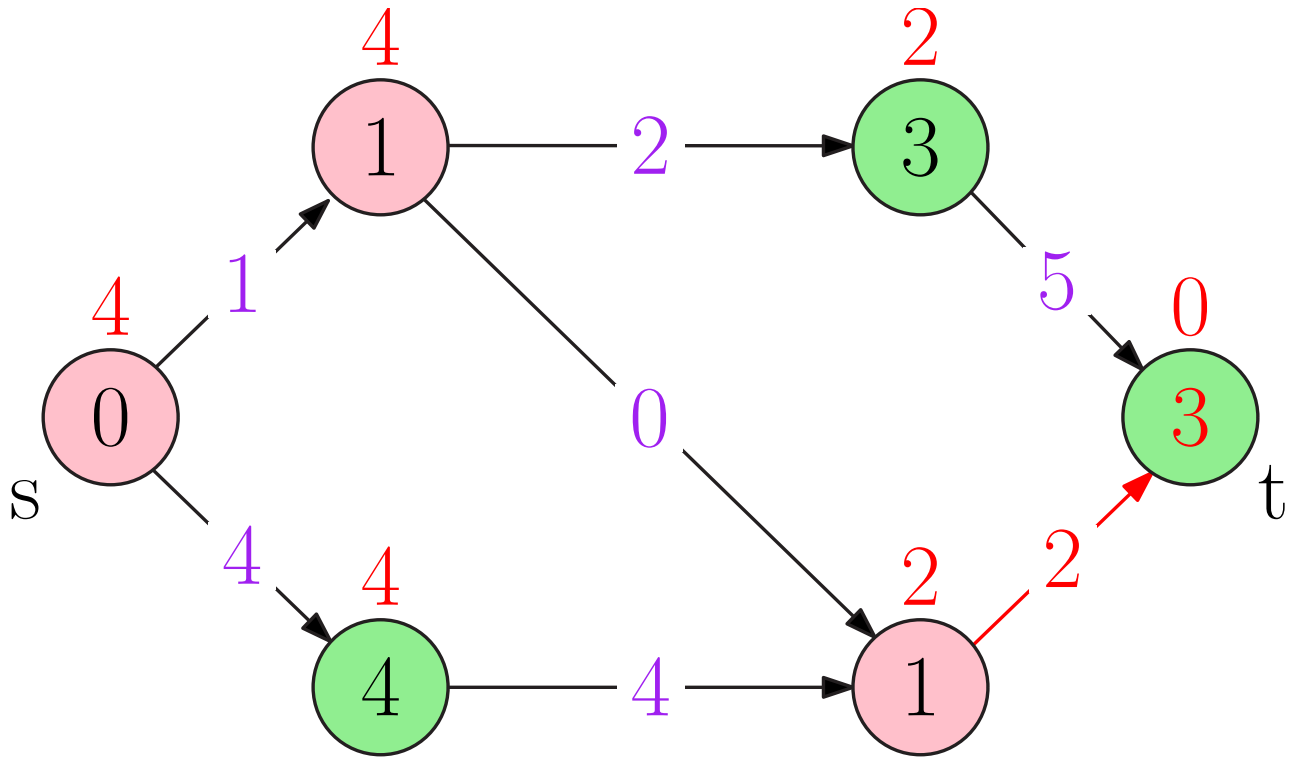


Suche in Graphen

A*-Suche

Beispiel

- Suche mit reduzierten Kantengewichten

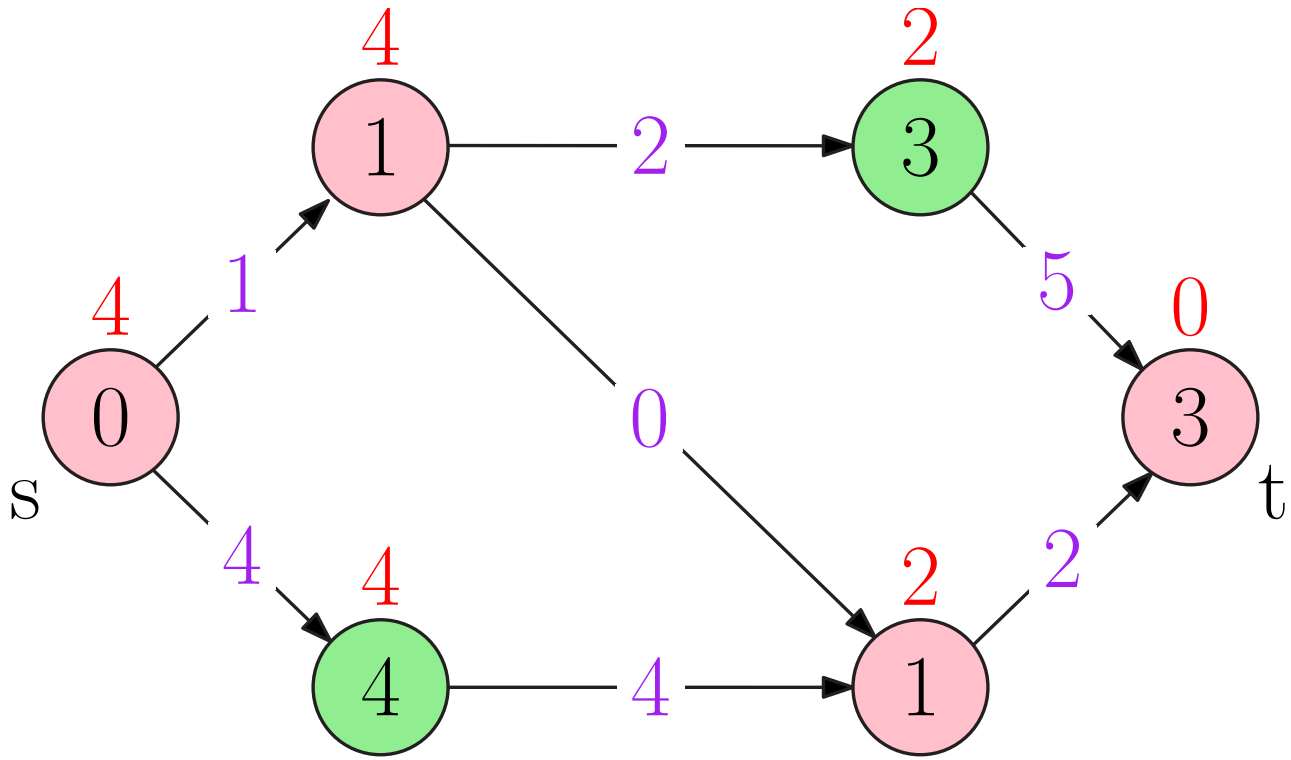


Suche in Graphen

A*-Suche

Beispiel

- Suche mit reduzierten Kantengewichten

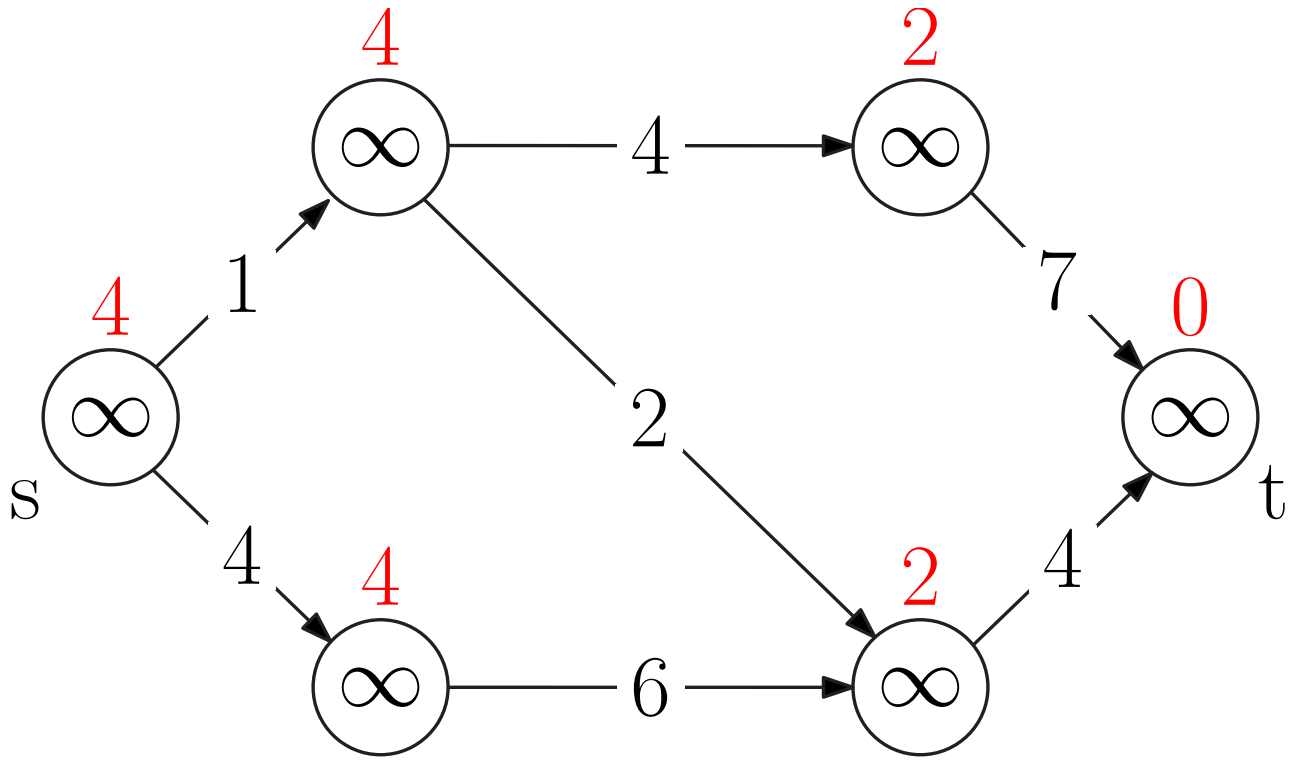


Suche in Graphen

A*-Suche

Beispiel

- Suche mit geänderten Schlüsseln in *Priority Queue*

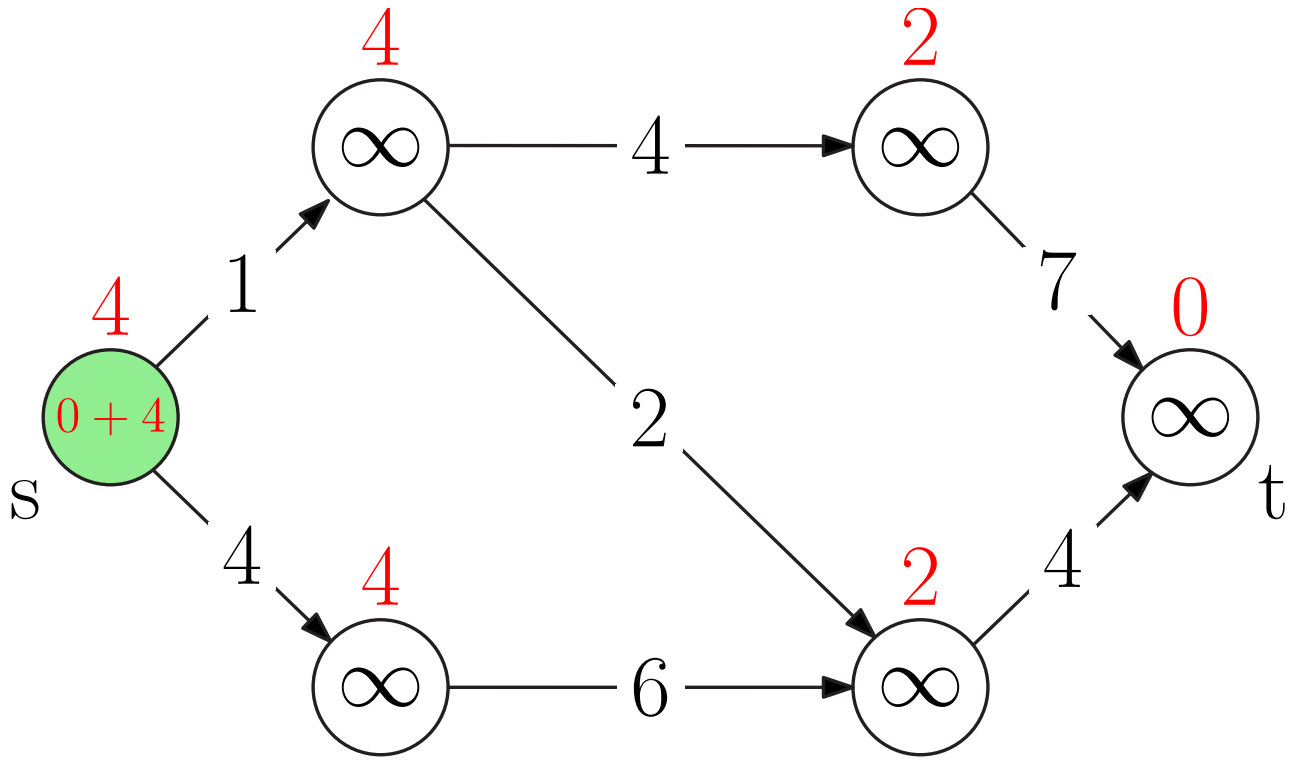


Suche in Graphen

A*-Suche

Beispiel

- Suche mit geänderten Schlüsseln in *Priority Queue*

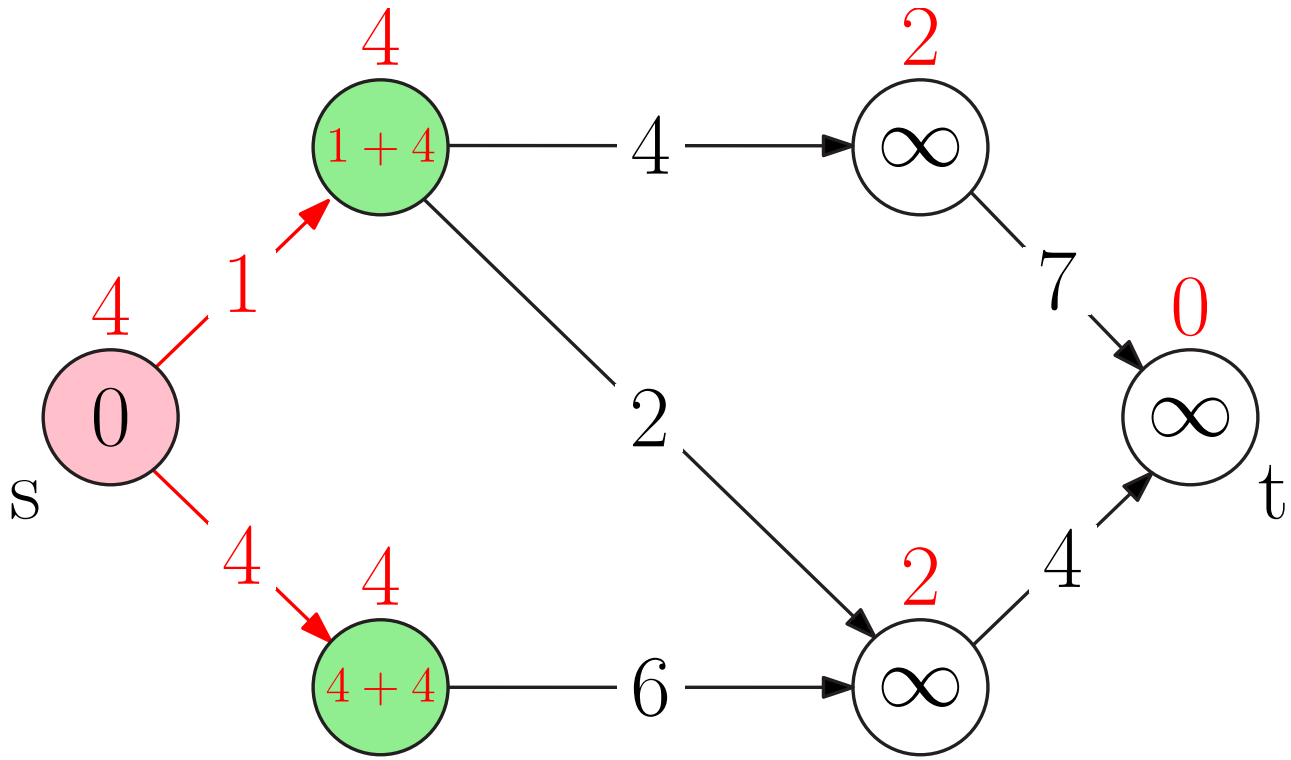


Suche in Graphen

A*-Suche

Beispiel

- Suche mit geänderten Schlüsseln in *Priority Queue*

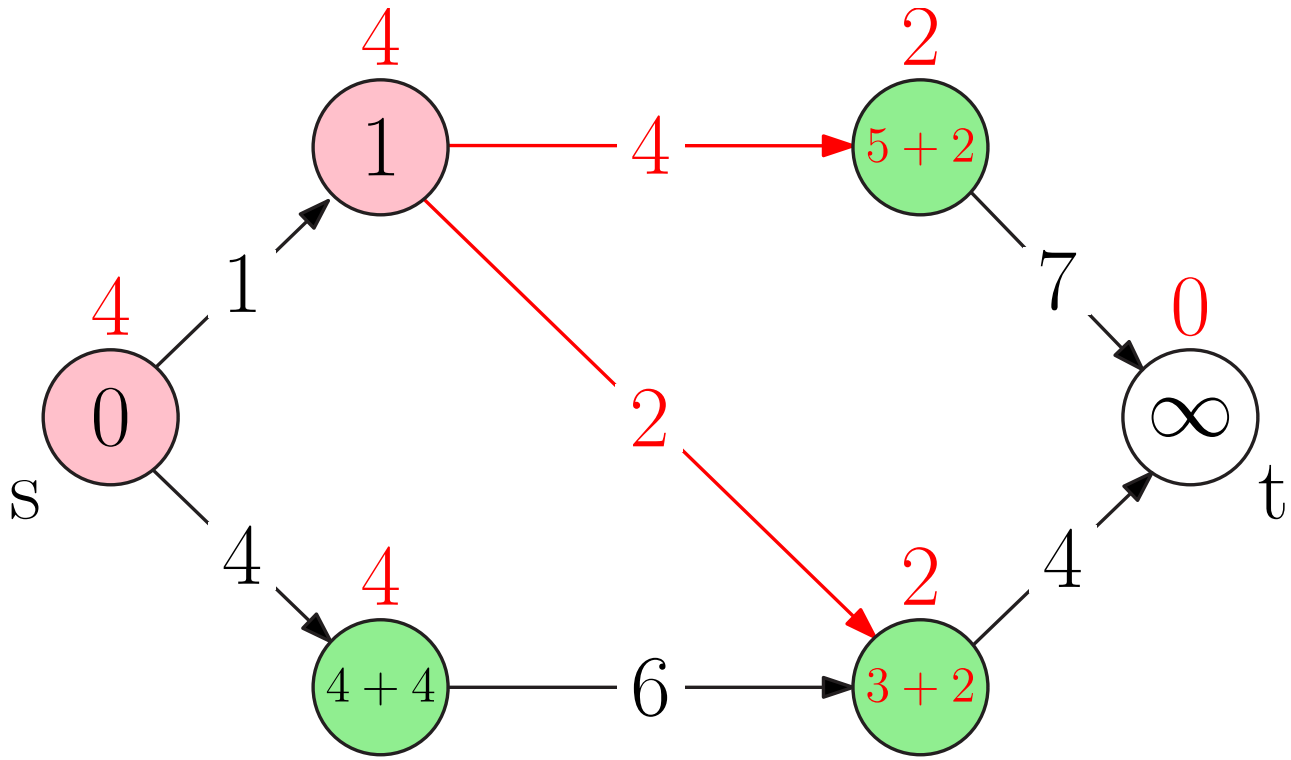


Suche in Graphen

A*-Suche

Beispiel

- Suche mit geänderten Schlüsseln in *Priority Queue*

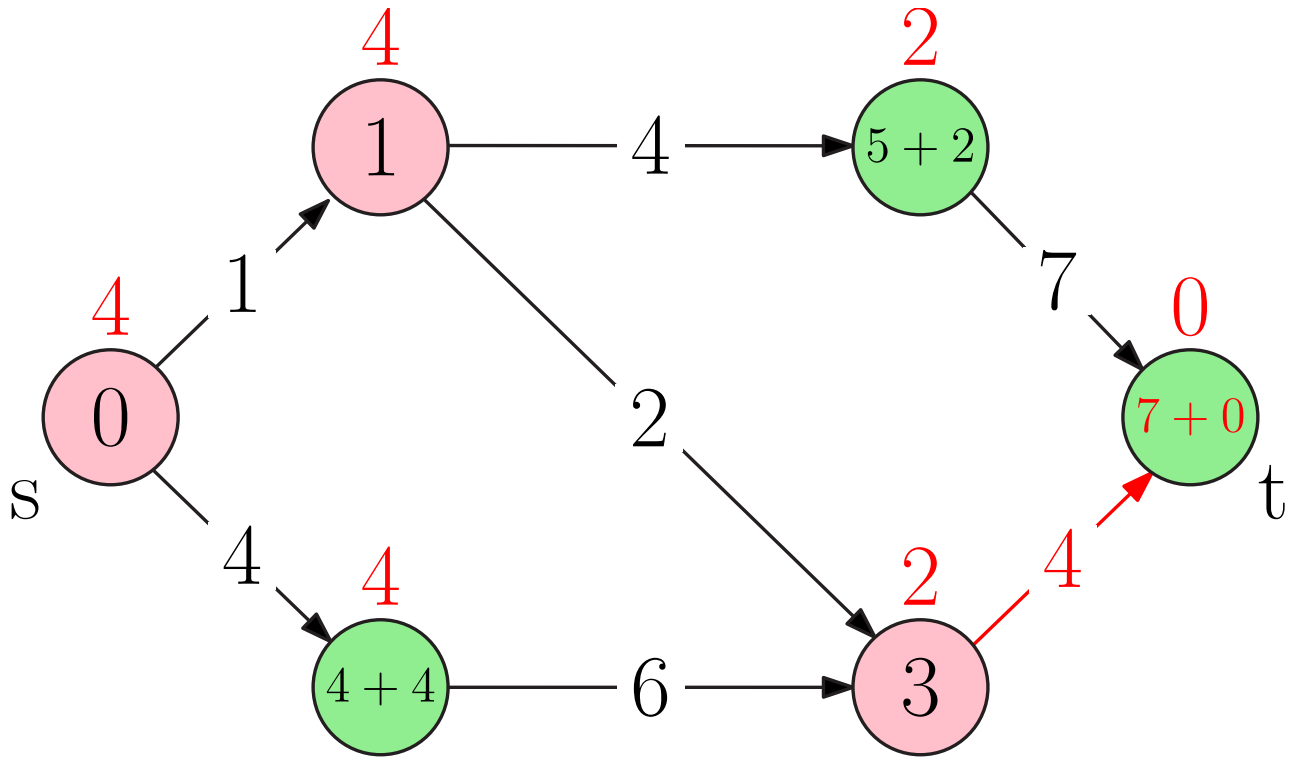


Suche in Graphen

A*-Suche

Beispiel

- Suche mit geänderten Schlüsseln in *Priority Queue*

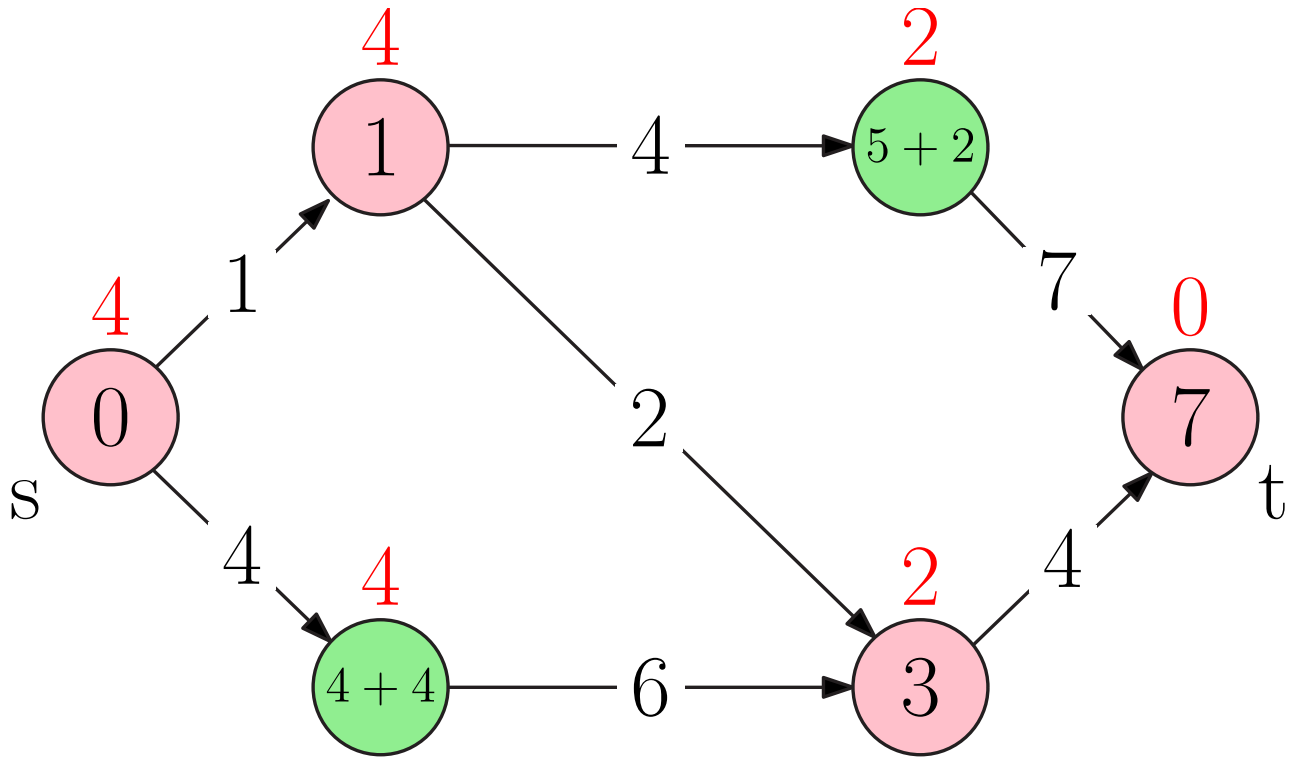


Suche in Graphen

A*-Suche

Beispiel

- Suche mit geänderten Schlüsseln in *Priority Queue*



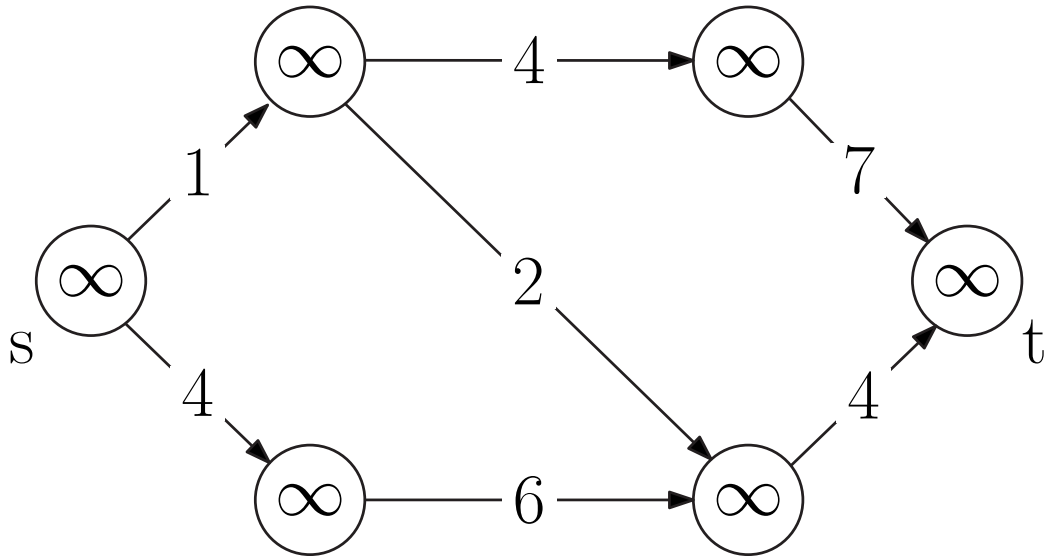
Suche in Graphen

A*-Suche – Landmarken

Erster Ansatz

- $\text{pot}(v) := \mu(v, t)$ sei optimales Potential
  - Algorithmus besucht nur Knoten auf kürzesten Pfaden
  - zu hoher Speicherverbrauch $\rightarrow$ benötigt alle kürzesten Distanzen!

Beispiel



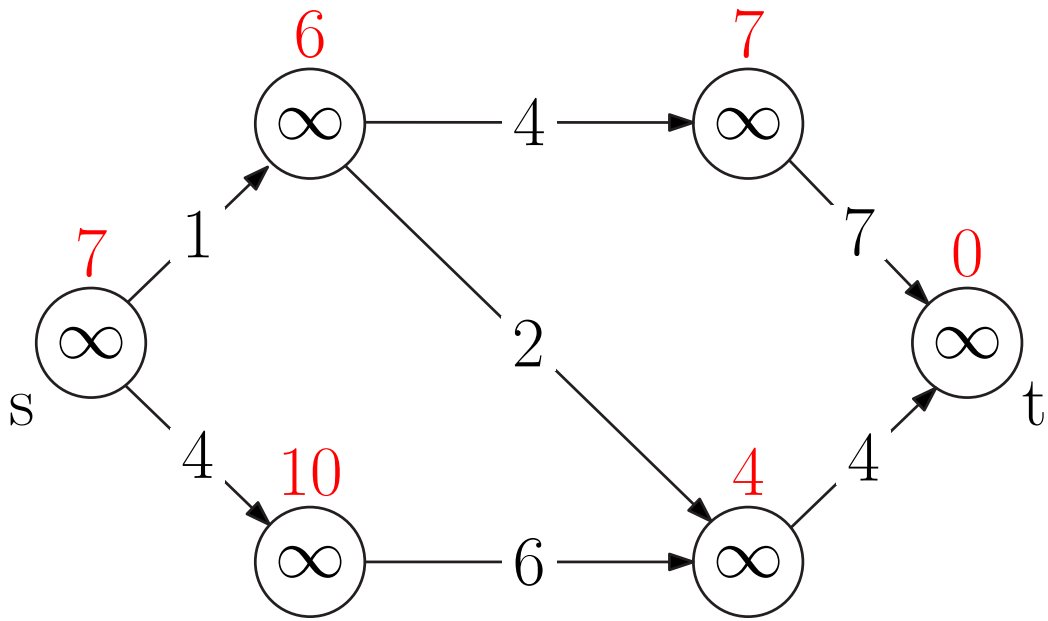
Suche in Graphen

A*-Suche – Landmarken

Erster Ansatz

- $\text{pot}(v) := \mu(v, t)$ sei optimales Potential
  - Algorithmus besucht nur Knoten auf kürzesten Pfaden
  - zu hoher Speicherverbrauch $\rightarrow$ benötigt alle kürzesten Distanzen!

Beispiel



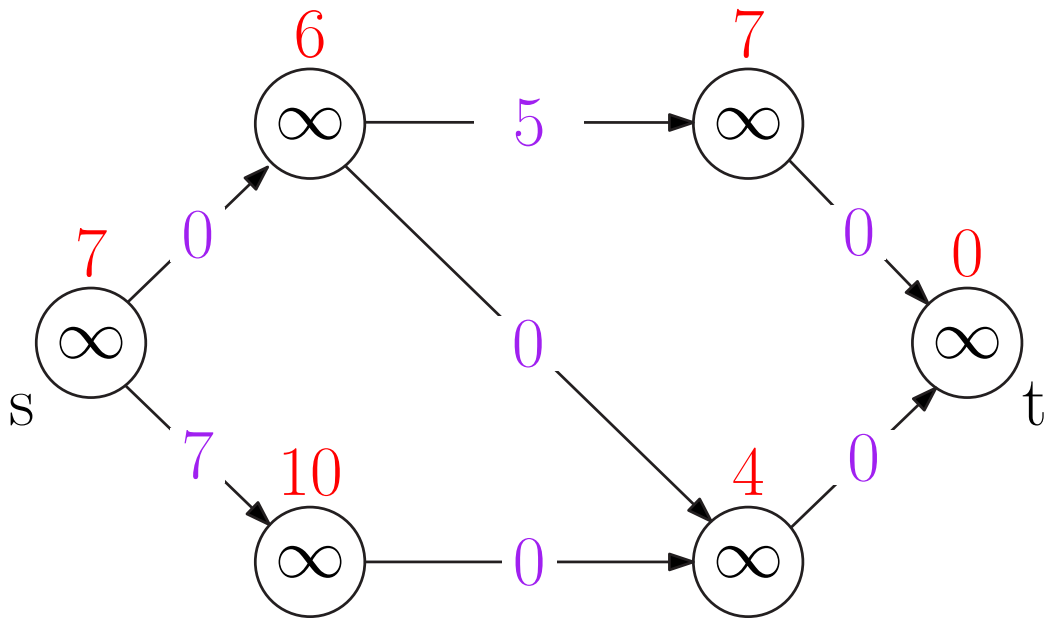
Suche in Graphen

A*-Suche – Landmarken

Erster Ansatz

- $\text{pot}(v) := \mu(v, t)$ sei optimales Potential
  - Algorithmus besucht nur Knoten auf kürzesten Pfaden
  - zu hoher Speicherverbrauch $\rightarrow$ benötigt alle kürzesten Distanzen!

Beispiel



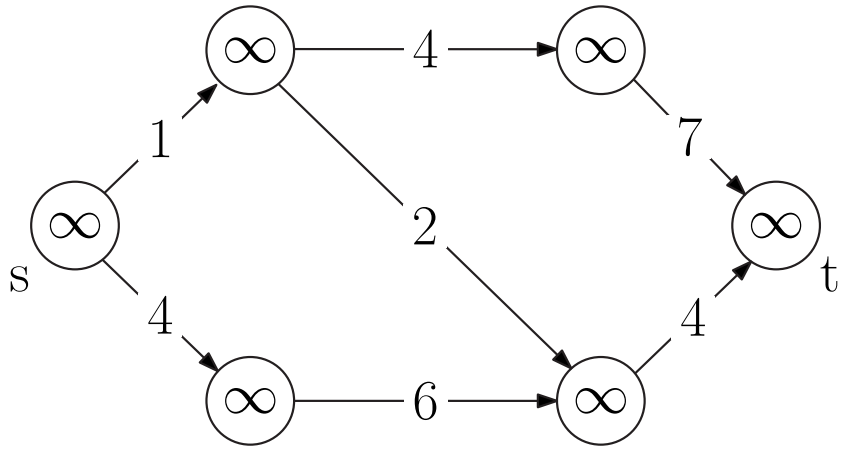
Suche in Graphen

A*-Suche – Landmarken

Kompromiss

- berechne Potential für einige Knoten l (Landmarken)
- bei konkreter Anfrage:
  - wähle Landmarke l hinter dem Ziel t (heuristisch)
  - verwende Potential $\text{pot}(v) := \mu(v, l) - \mu(t, l)$

Beispiel



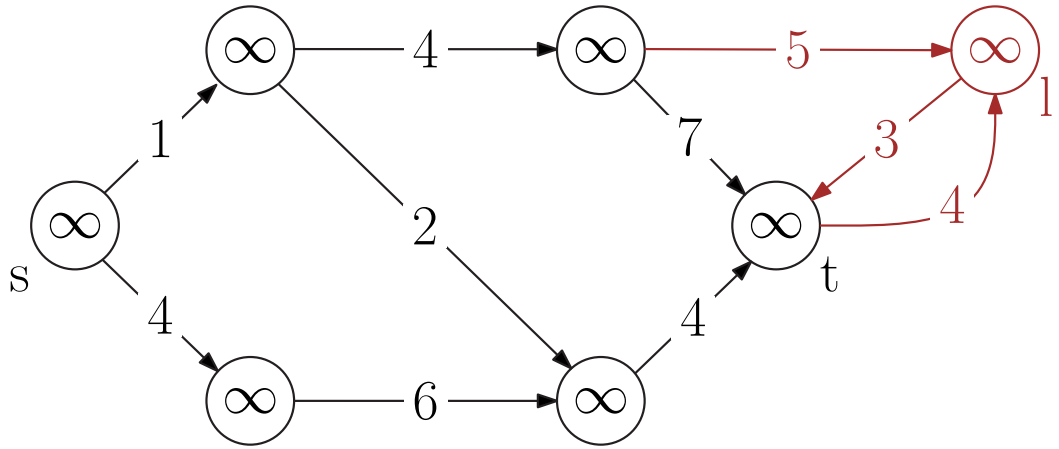
Suche in Graphen

A*-Suche – Landmarken

Kompromiss

- berechne Potential für einige Knoten l (Landmarken)
- bei konkreter Anfrage:
  - wähle Landmarke l hinter dem Ziel t (heuristisch)
  - verwende Potential $\text{pot}(v) := \mu(v, l) - \mu(t, l)$

Beispiel



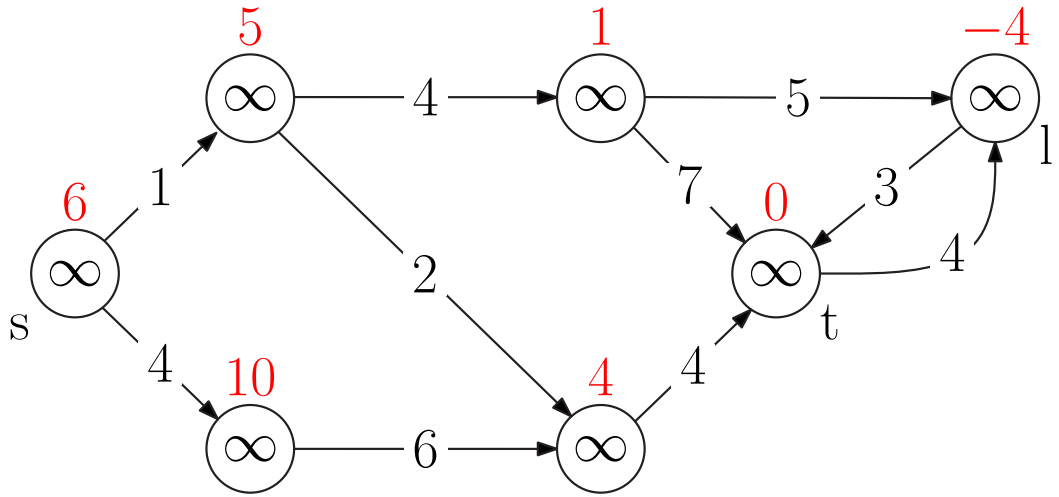
Suche in Graphen

A*-Suche – Landmarken

Kompromiss

- berechne Potential für einige Knoten *l* (Landmarken)
- bei konkreter Anfrage:
  - wähle Landmarke *l* hinter dem Ziel *t* (heuristisch)
  - verwende Potential $\text{pot}(v) := \mu(v, l) - \mu(t, l)$

Beispiel



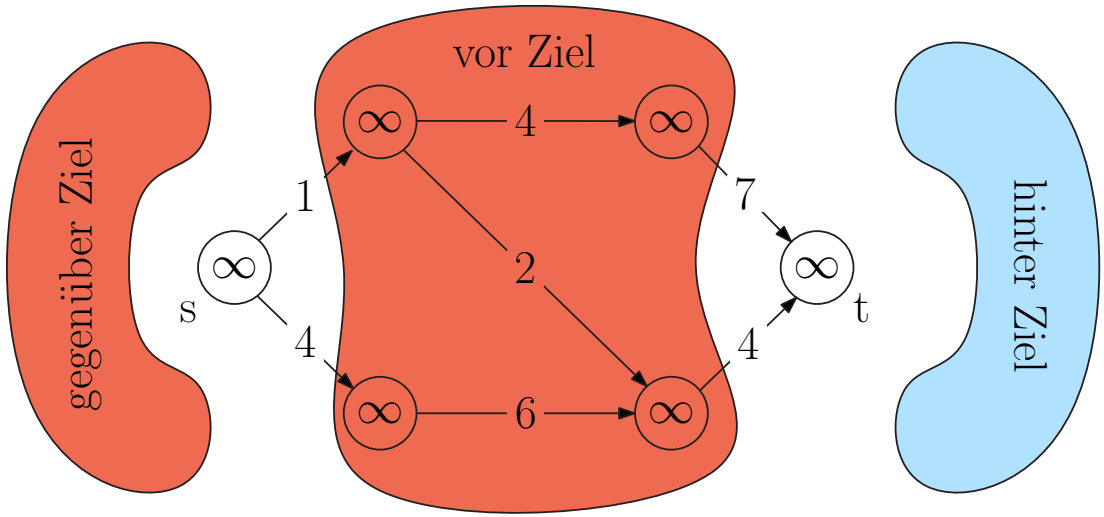
Suche in Graphen

A*-Suche – Landmarken

Qualität der Landmarken

- Was passiert bei schlechter Landmarke (vor/gegenüber Ziel)?
  → Korrektheit?
  → Laufzeit?

Beispiel

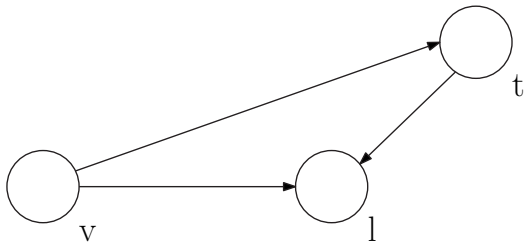


Suche in Graphen

A*-Suche – Landmarken

Qualität der Landmarken

- Korrektheit:
  immer korrekt dank Dreiecksungleichung
  - untere Schranke für Distanz zum Ziel t
    $\text{pot}(v) = \mu(v,l) - \mu(t,l) \leq \mu(v,t) \quad \Leftrightarrow \quad \mu(v,l) \leq \mu(v,t) + \mu(t,l)$
  - nicht-negative reduzierte Kantengewichte
    $$\begin{aligned}\overline{c}(u,v) &= c(u,v) + \text{pot}(v) - \text{pot}(u) \\ &= c(u,v) + \mu(v,l) - \mu(t,l) - \mu(u,l) + \mu(t,l) \\ &= c(u,v) + \mu(v,l) - \mu(u,l) \geq 0 \quad \forall (u,v) \in E\end{aligned}$$

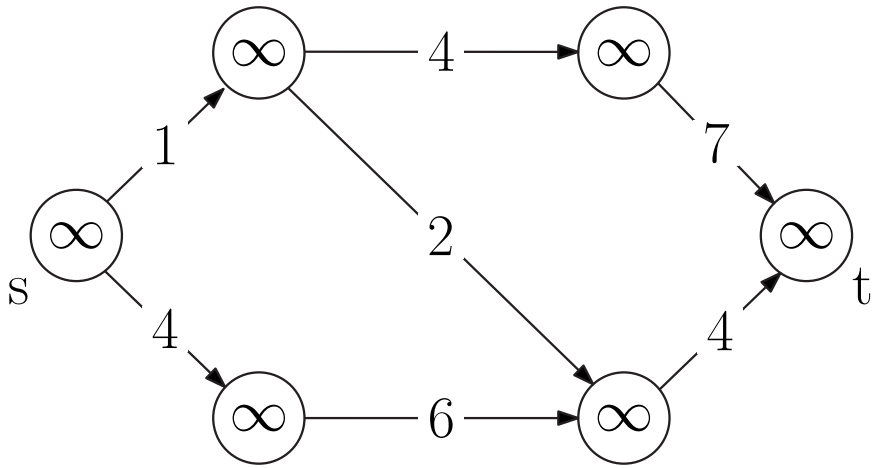


Suche in Graphen

A*-Suche – Landmarken

Qualität der Landmarken

- Laufzeit:
  Algorithmus sucht in falscher Richtung → langsam!
- Beispiel

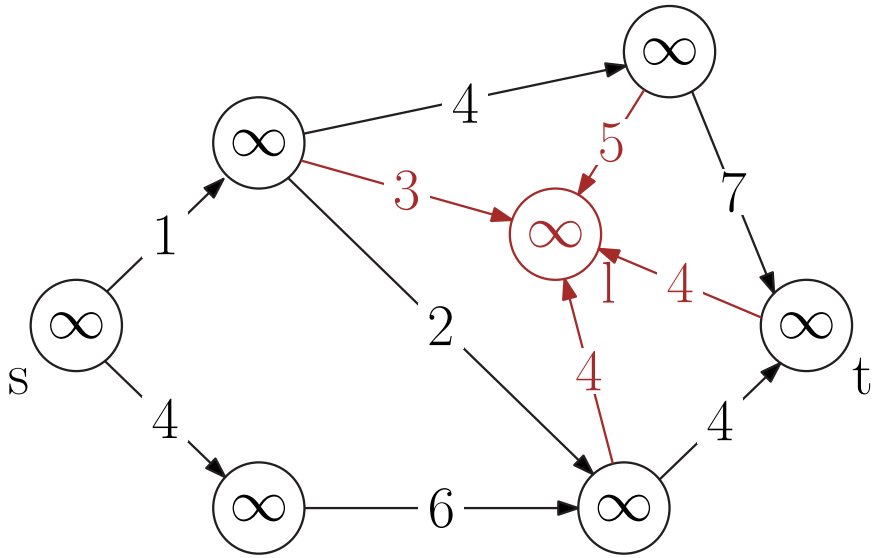


Suche in Graphen

A*-Suche – Landmarken

Qualität der Landmarken

- Laufzeit:
  Algorithmus sucht in falscher Richtung → langsam!
- Beispiel

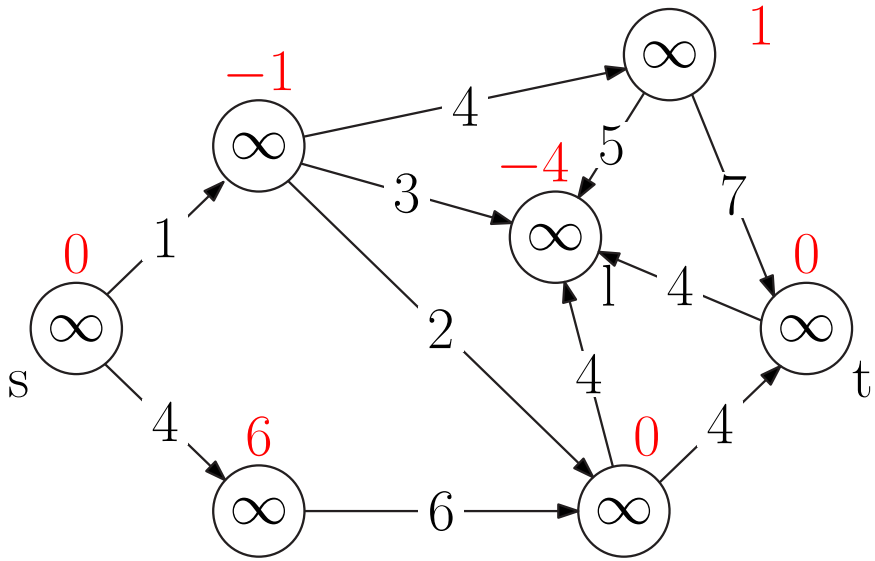


Suche in Graphen

A*-Suche – Landmarken

Qualität der Landmarken

- Laufzeit:
  Algorithmus sucht in falscher Richtung → langsam!
- Beispiel

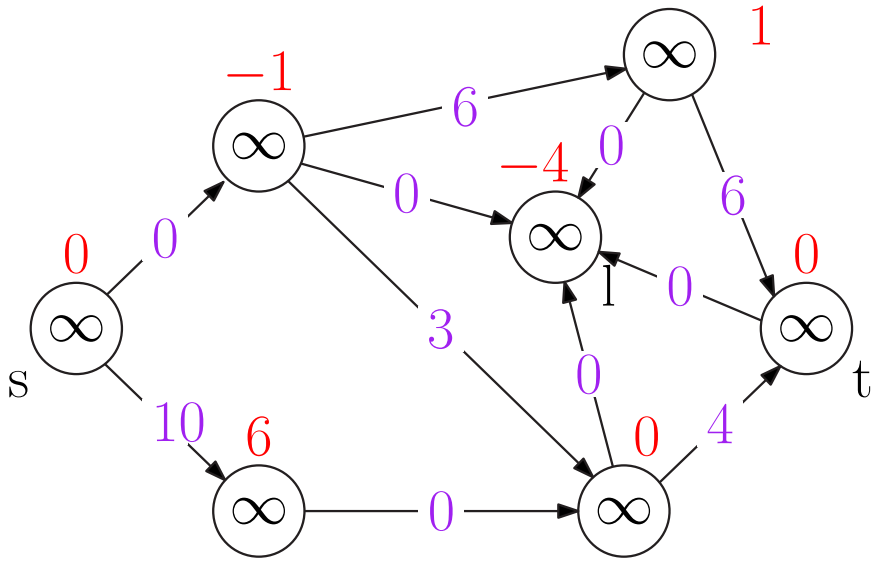


Suche in Graphen

A*-Suche – Landmarken

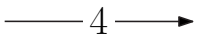
Qualität der Landmarken

- Laufzeit:
  Algorithmus sucht in falscher Richtung → langsam!
- Beispiel



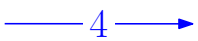
Legende

Karlsruher Institut für Technologie

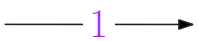


Kante mit Gewicht 4

Kante mit Gewicht 4, wird gerade in Vorwärtsrichtung betrachtet



Kante mit Gewicht 4, wird gerade in Rückwärtsrichtung betrachtet



Kante mit reduziertem Gewicht 1

neu eingefügte Kante mit Gewicht 4

Knoten v mit oberer Schranke für Distanz vom Start $d[v] = \infty$ und Potential $pot[v] = 2$

erreichter Knoten v mit oberer Schranke für Distanz vom Start $d[v] = 2$ (in *Priority Queue*)

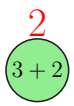
erreichter Knoten v mit oberer Schranke für Distanz vom Start $d[v] = 1$ (in *Priority Queue*)
(obere Schranke gerade verkleinert)

gescannter Knoten v mit exakter Distanz vom Start $d[v] = 1$ (aus *Priority Queue* entfernt)

Knoten in Vorwärtsrichtung gescannt mit exakter Distanz vom Start $d_{fwd}[v] = 1$,
in Rückwärtsrichtung erreicht mit oberer Schranke für Distanz zum Ziel $d_{bwd}[v] = 3$

Knoten in Vorwärtsrichtung gescannt mit $d_{fwd}[v] = 1$, in Rückwärtsrichtung erreicht mit $d_{bwd}[v] = 2$
(obere Schranke in Rückwärtsrichtung gerade verkleinert)

aktueller Treffpunkt von Vorwärts- und Rückwärtssuche mit minimalem $d_{fwd}[v] + d_{bwd}[v] = 3$



erreichter Knoten v mit oberer Schranke für Distanz vom Start $d[v] = 2$ (in *Priority Queue*)
(Schlüssel in *Priority Queue* ist $d[v] + \text{pot[v]} = 3 + 2$)

Zeiger auf Vorgänger
(in Vorwärtsrichtung)

Zeiger auf Vorgänger
(in Rückwärtsrichtung)

neu eingefügter Knoten

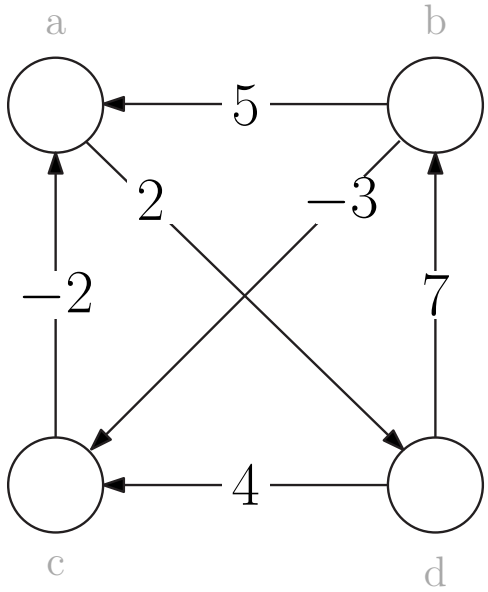
Suche in Graphen

All-to-all

- n-mal Dijkstra
- Negative Kantengewichte (aber keine negativen Kreise)
  - n-mal Bellman-Ford
    - Langsam $O(n^2 m)$
  - $\rightarrow$ Kombiniere Dijkstra, Bellman-Ford und Knotenpotentiale
  - $O(nm + n^2 \log n)$

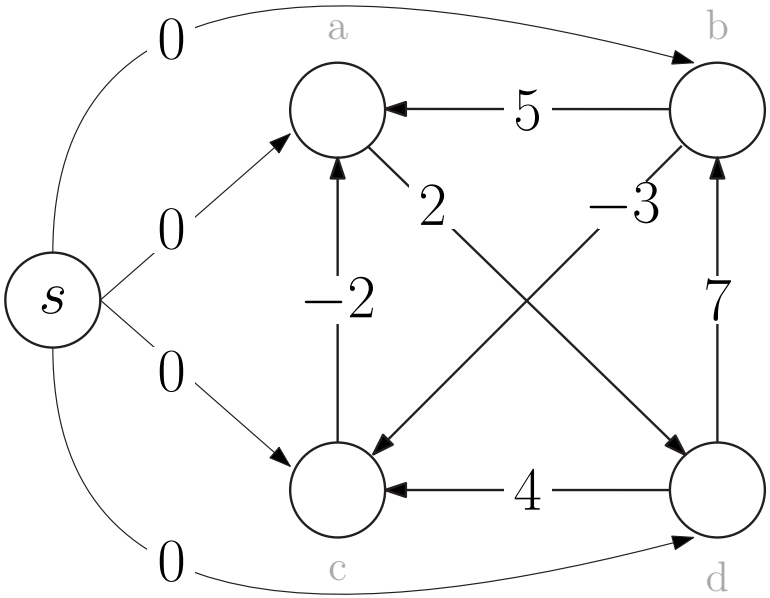
Suche in Graphen

All-to-all



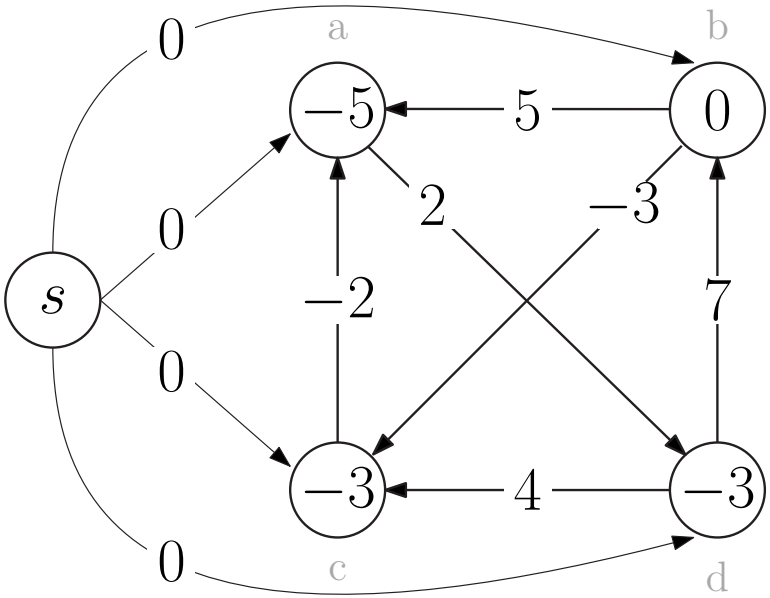
Suche in Graphen

All-to-all



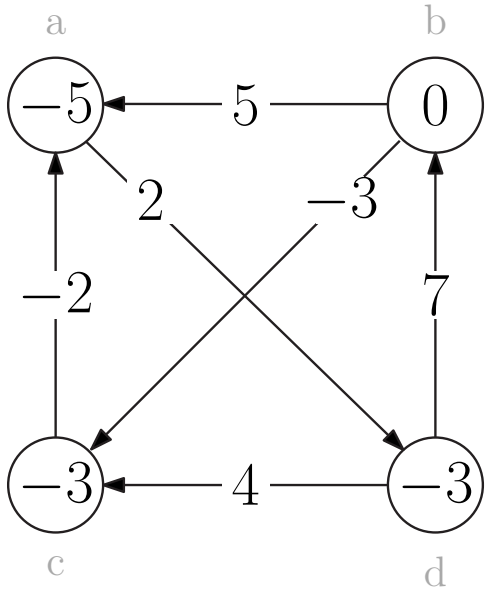
Suche in Graphen

All-to-all



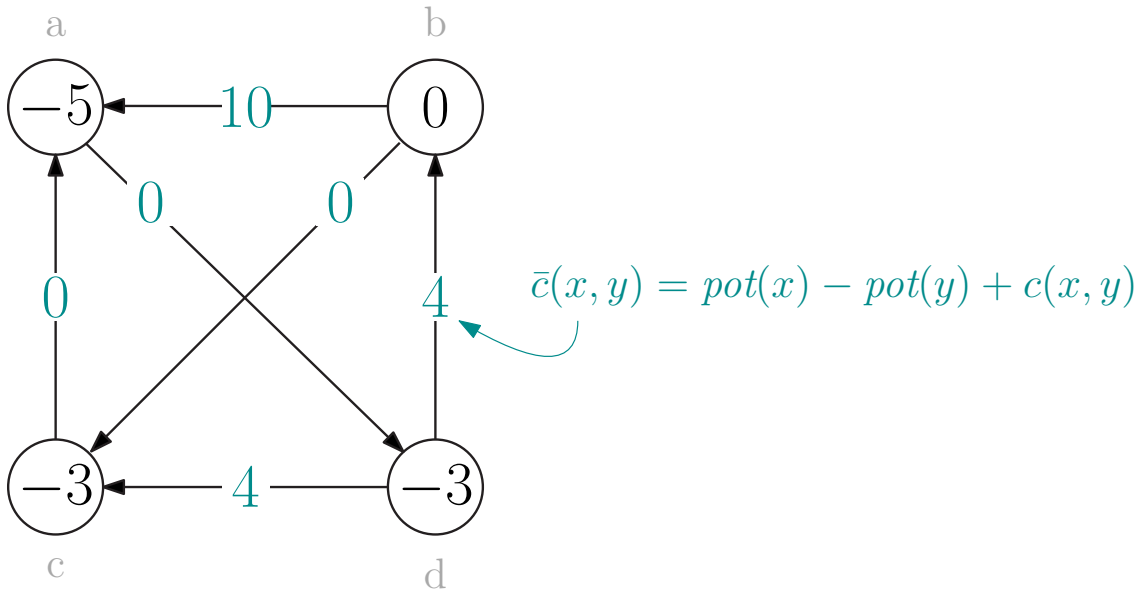
Suche in Graphen

All-to-all



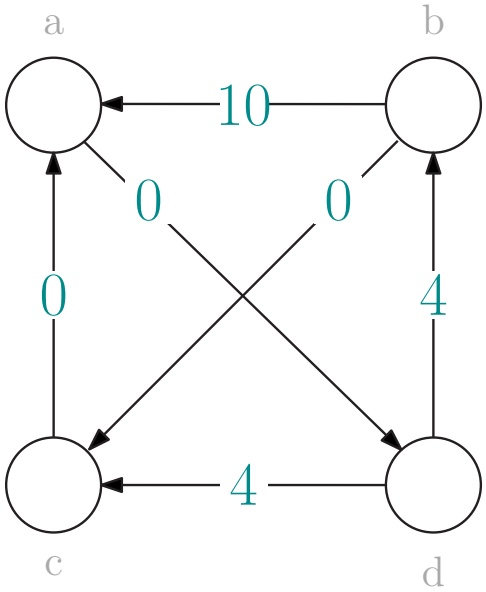
Suche in Graphen

All-to-all



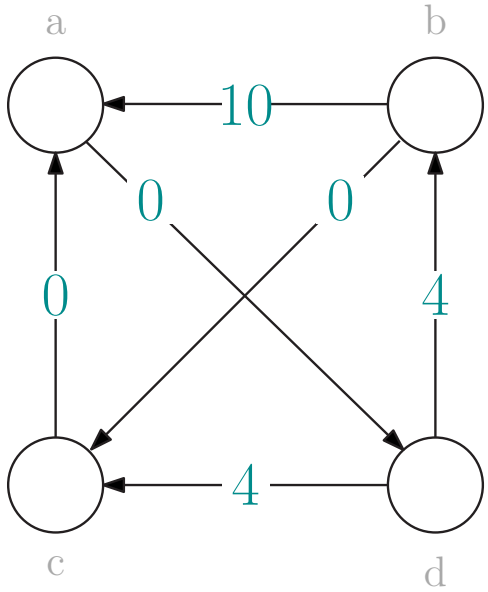
Suche in Graphen

All-to-all



Suche in Graphen

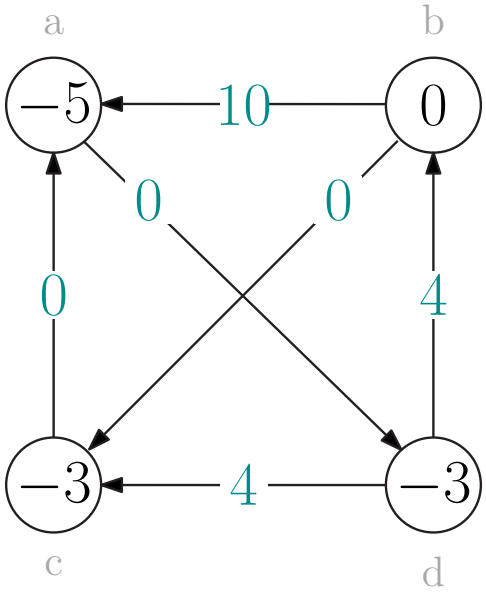
All-to-all



$$\bar{\mu}(a,b) = 4$$
$$\bar{\mu}(b,a) = 0$$
$$\bar{\mu}(c,d) = 0$$
$$\ldots$$

Suche in Graphen

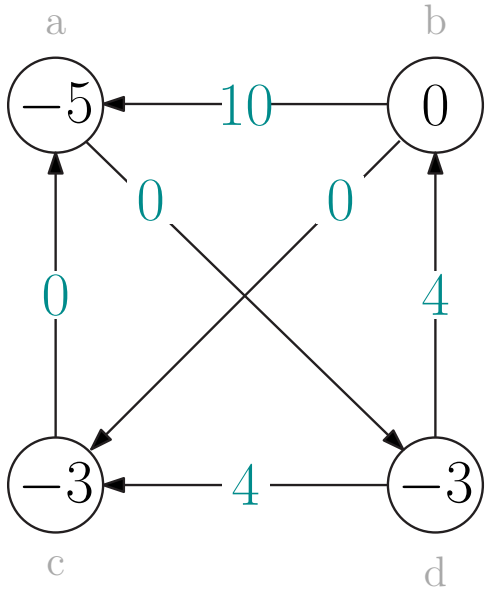
All-to-all



$$\bar{\mu}(a,b) = 4$$
$$\bar{\mu}(b,a) = 0$$
$$\bar{\mu}(c,d) = 0$$
$$\dots$$

Suche in Graphen

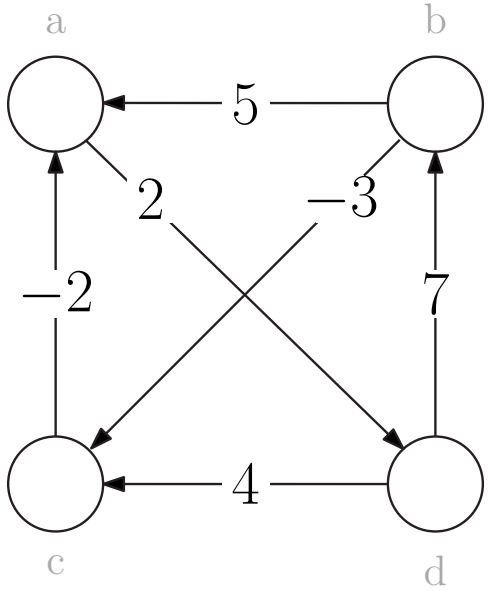
All-to-all



$$\mu(a,b) = \bar{\mu}(a,b) + pot(b) - pot(a) = 9$$
$$\mu(b,a) = -5$$
$$\bar{\mu}(c,d) = 0$$
$$\dots$$

Suche in Graphen

All-to-all



$$\mu(a,b) = \bar{\mu}(a,b) + pot(b) - pot(a) = 9$$
$$\mu(b,a) = -5$$
$$\bar{\mu}(c,d) = 0$$
$$\dots$$

Ende!

Feierabend!