Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 18 (9.7.2014)

Prioritätswartenschlangen II
Kürzeste Wege: Dijkstras Algorithmus

Fabian Kuhn

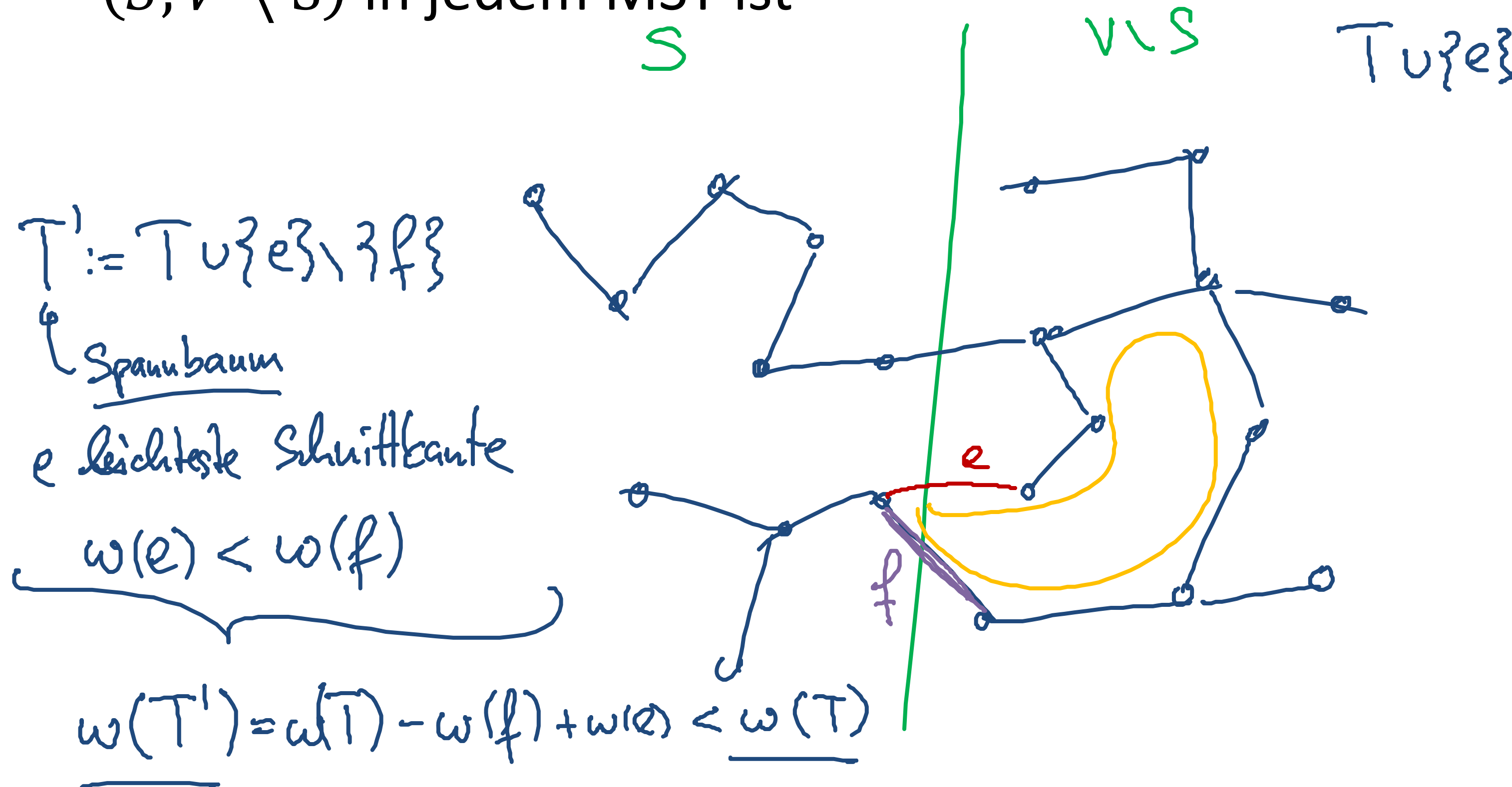
Algorithmen und Komplexität

UNI FREIBURG

MST Eindeutigkeit

Satz: Bei paarweise verschiedenen Kantengewichten ist der MST eindeutig.

- Wir zeigen zuerst, dass die leichteste Schnittkante jedes Schnitts $(S, V \setminus S)$ in jedem MST ist



MST Eindeutigkeit

Satz: Bei paarweise verschiedenen Kantengewichten ist der MST eindeutig.

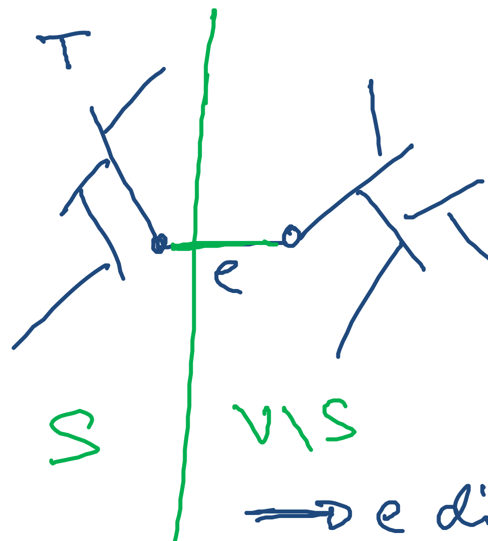
- Wir zeigen zuerst, dass die leichteste Schnittkante jedes Schnitts $(S, V \setminus S)$ in jedem MST ist

MST nicht eindeutig

T, T' : min. Spannbäume

$e \in T$

$e \notin T'$



$e \notin T' \rightarrow T'$ kein MST

$\Rightarrow$ e die leichteste Kante über $(S, V \setminus S)$

Prioritätswarteschlange

Heap / Priority Queue:

- Verwaltet eine Menge von *(key,value)*-Paaren

Operationen:

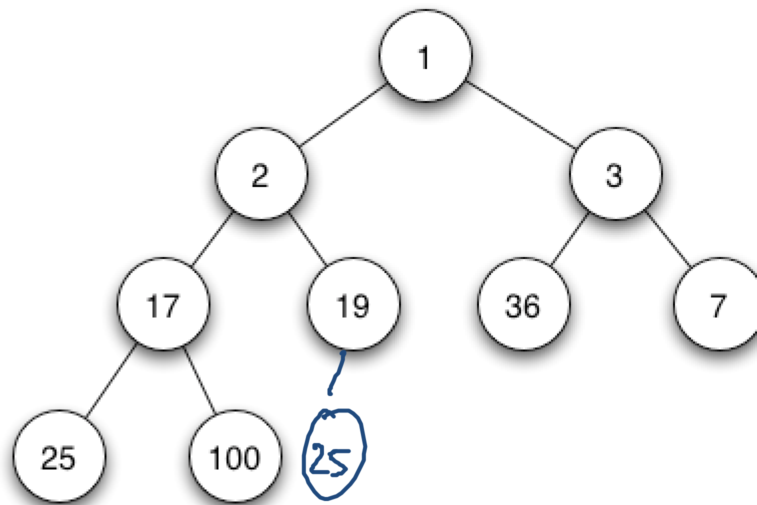
- *create* : erzeugt einen leeren Heap
- *H.insert(x, key)* : fügt Element *x* mit Schlüssel *key* ein
- *H.getMin()* : gibt Element mit kleinstem Schlüssel zurück
- *H.deleteMin()* : löscht Element mit kleinstem Schlüssel (gibt Element mit kleinstem Schlüssel zurück)
- *H.decreaseKey(x, newkey)* : Falls *newkey* kleiner als der aktuelle Schlüssel von *x* ist, wird der Schlüssel von *x* auf *newkey* gesetzt

Prioritätswarteschlangen

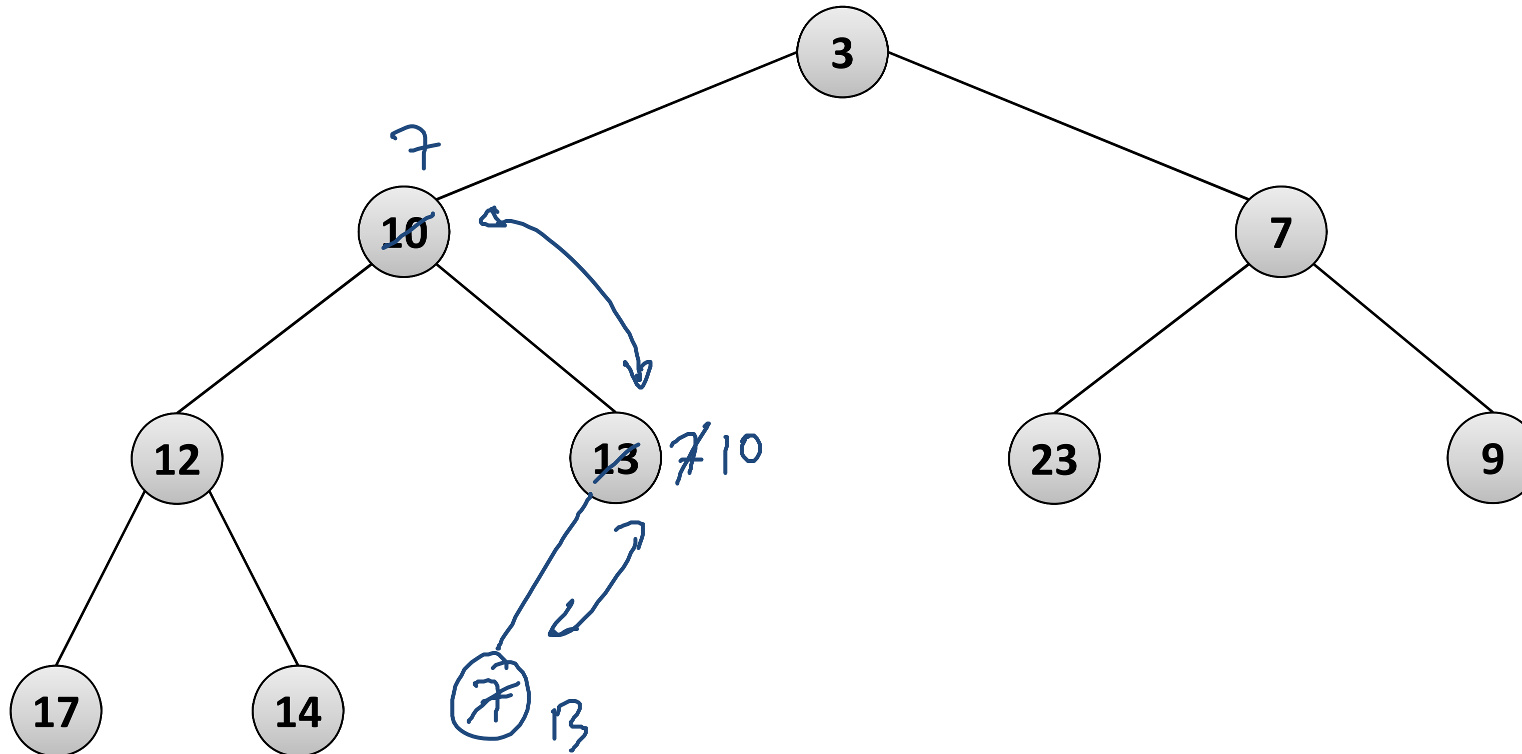
UNI FREIBURG

Implementierung als Binärbaum mit Min-Heap Eigenschaft

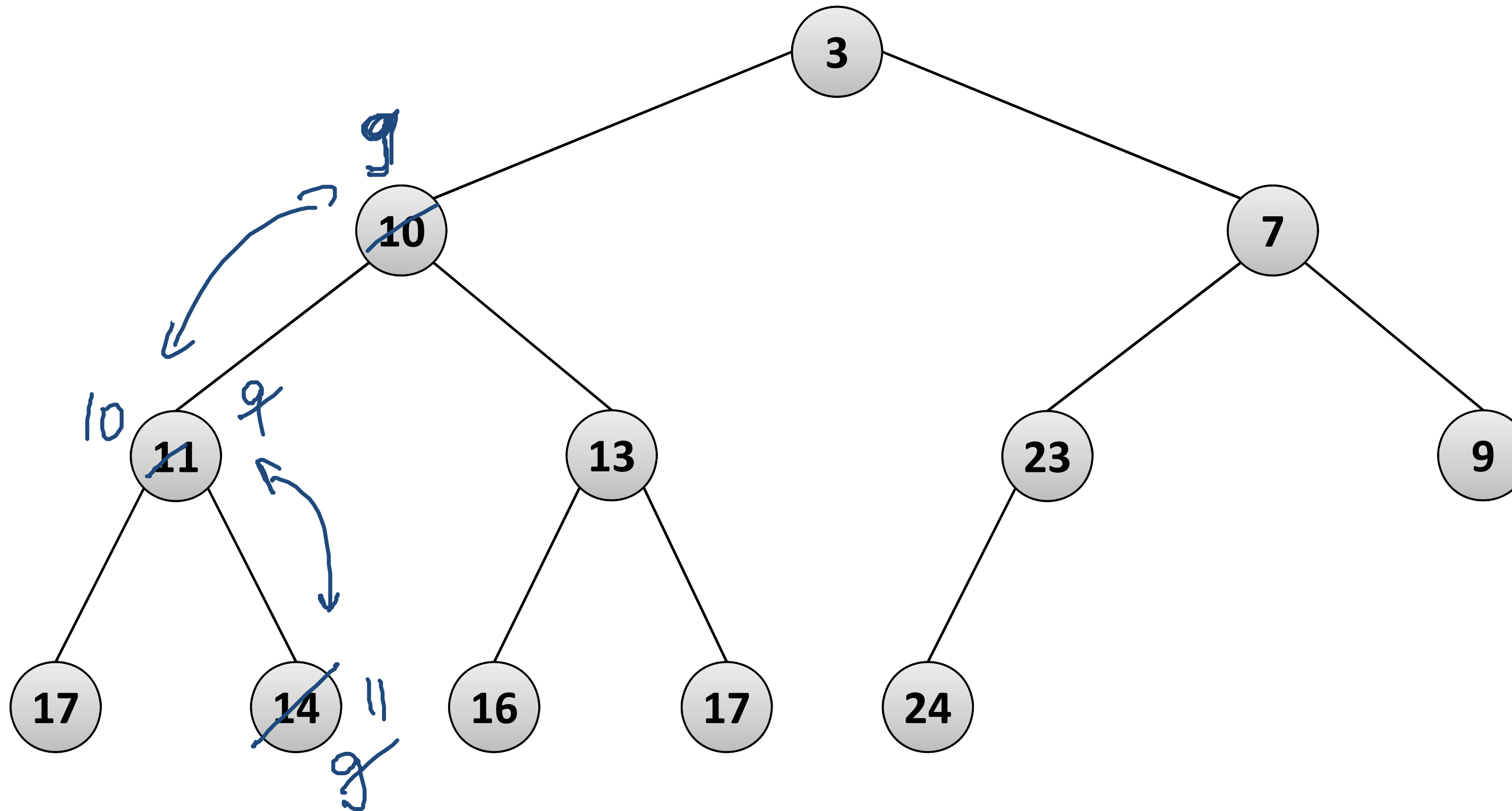
- Die Datenstruktur heisst deshalb oft auch Heap
- Ein Baum hat die Min-Heap Eigenschaft, falls in jedem Teilbaum, die Wurzel den kleinsten Schlüssel hat
- getMin-Operation: trivial!
- Baum wird immer so balanciert, wie möglich gehalten



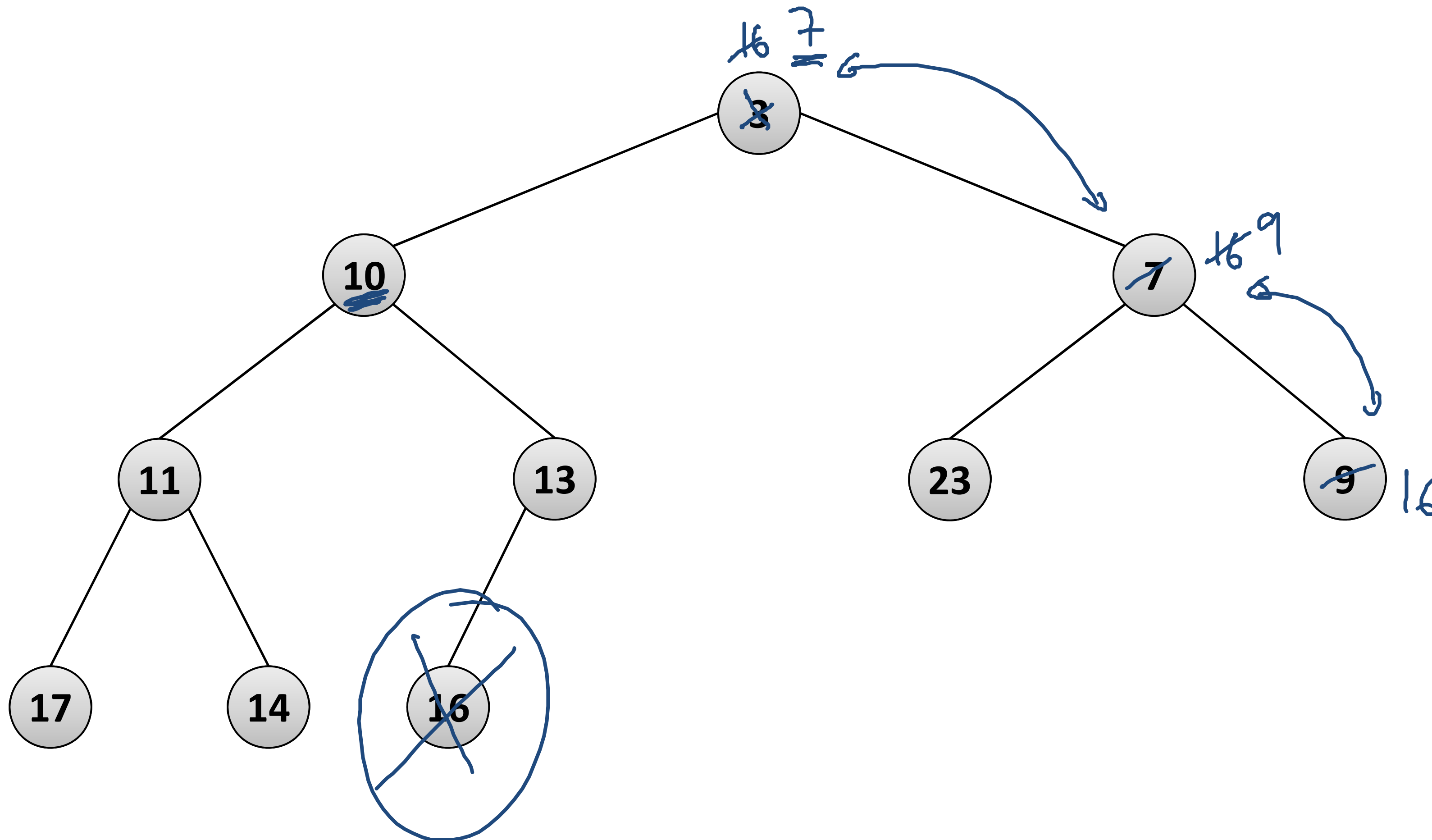
Prioritätswarteschlangen: Einfügen



Prioritätswarteschlangen: Decrease-Key



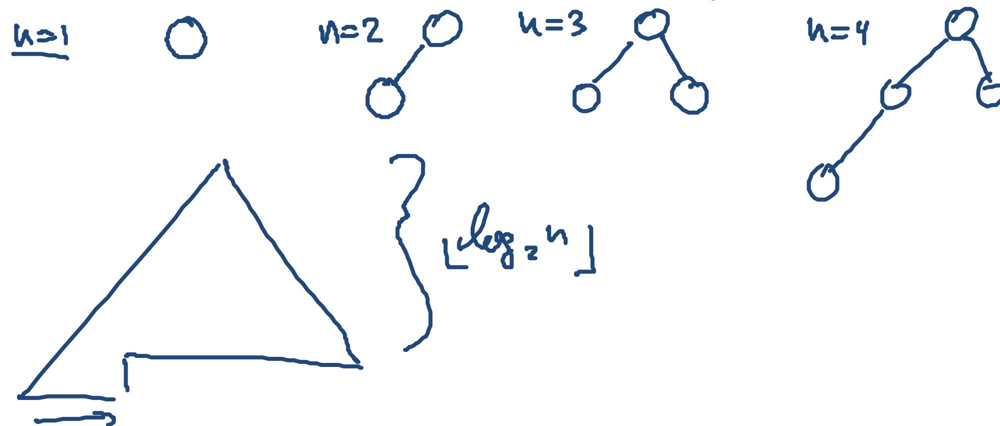
Prioritätswarteschlangen: Delete-Min



Prioritätswarteschlangen: Analyse

UNI FREIBURG

- Die besprochene Variante heisse auch **binärer Heap**
  - durch einen Binärbaum mit Min-Heap-Eigenschaft implementiert
- **Tiefe des Baums** ist immer genau $\lfloor \log_2 n \rfloor$

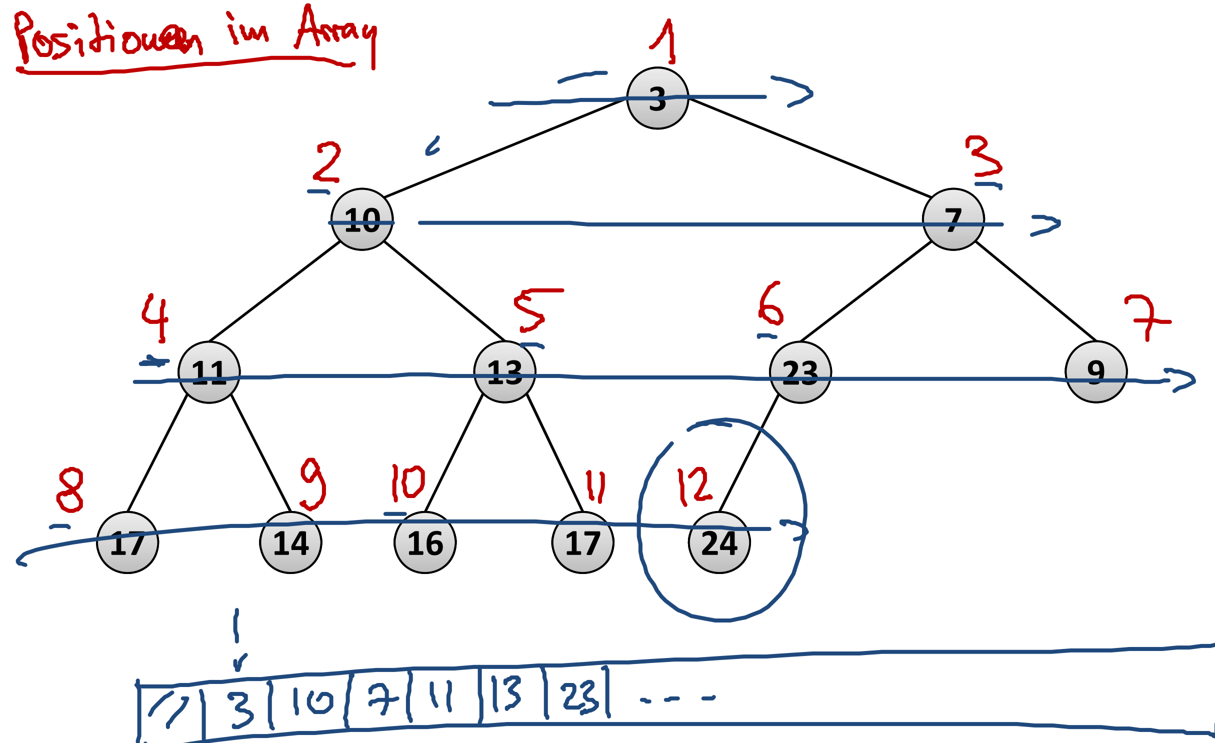


- **Laufzeit aller Operationen:** $O(\log n)$
  - wenn man den Binärbaum irgendwie vernünftig implementiert
  - man muss immer höchsten einmal den Binärbaum hoch (bei insert, decreaseKey) oder runter (bei deleteMin)
  - Wir werden gleich eine elegante Art sehen, den binären Heap zu impl.

Binäre Heaps, Array-Implementierung

Idee: Speichere **alles in ein Array**

- Das geht, weil der Baum perfekt balanciert ist



Binäre Heaps, Array-Implementierung

UNI FREIBURG

Positionen im Array (bei n Elementen):

- Wurzel: 1, letzter Eintrag: n, Parent von i: $\lfloor i/2 \rfloor$
- linkes Kind von i: $2i$, rechtes Kind von i: $2i + 1$

Pseudocode am Beispiel von Insert

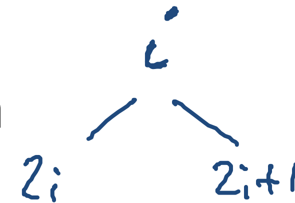
- Elemente sind in Array A and Positionen $1, \dots, n$ gespeichert

```
insert(x):
  n = n + 1
  A[n] = x
  i = n
  while (i > 1) and (A[i] < A[i/2]) do
      swap(A[i], A[i/2])
      i = i/2
```

i/2 : Ganzzahldiv.

Binäre Heaps: Pseudocode Delete-Min

```
smallest(i):    // returns index of smallest key among i and children
  j = i
  if (2*i < n) and (A[2*i] < A[j]) then
    j = 2*i
  if (2*i+1 < n) and (A[2*i+1] < A[j]) then
    j = 2*i + 1
  return j
```




```
deleteMin():
  A[1] = A[n]; n = n – 1;
  i = 1; j = smallest(i)
  while i ≠ j do
      swap(A[i], A[j])
      i = j
      j = smallest(i)
```

Wurzel

Heapsort

- Die Array-Implementierung von Heaps (Prioritätswarteschlangen) gibt auch einen weiteren effizienten Sortieralgorithmus

Heapsort (H ist ein binärer Heap, sortiere Array A)

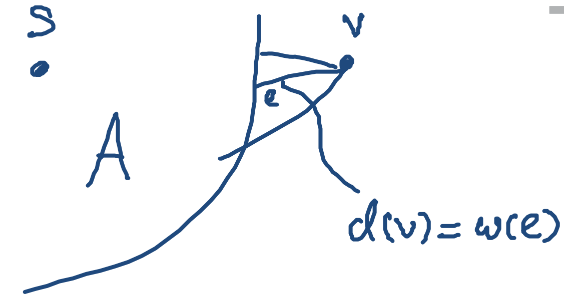
```
H = new BinaryHeap()
for i = 0 to n - 1 do
    H.insert(A[i])        ← O(log n)
for i = 0 to n - 1 do
    A[i] = H.deleteMin()  O(log n)
```

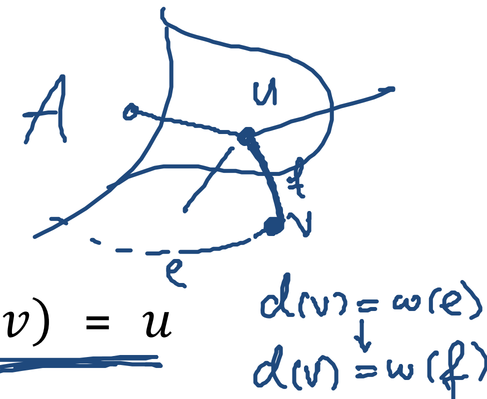
- **Laufzeit:** $O(n \log n)$

UNI FREIBURG

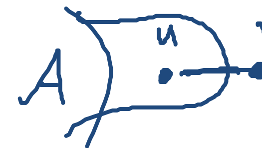
Prim's Algorithmus mit binären Heaps

```
H = new BinaryHeap(); A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); α(u) = NULL
H.insert(s, 0)
while H is not empty do
    u = H.deleteMin()
    for all neighbors v of u do
        if w({u,v}) < d(v) then
            H.decreaseKey(v, w({u,v})); α(v) = u
    if u ≠ s then A = A ∪ {{u,α(u)}}
```





d(v) = w(e)
↓
d(v) = w(f)



Laufzeit: $O(m \cdot \log n)$

- $n \leq m + 1$ insert-Operationen und deleteMin-Operationen
- $\leq m$ decreaseKey-Operationen

$O(m)$ Heap-Op.

Prim's Algorithmus: Bessere Laufzeit

UNI FREIBURG

Laufzeit mit binären Heaps: $O(m \cdot \log n)$

- $n \leq m + 1$ insert-Operationen und deleteMin-Operationen
- $\leq m$ decreaseKey-Operationen

Beste Implementierung von Prioritätswarteschlangen:

- Fibonacci Heaps (siehe Vorlesung Algorithmentheorie)
- Laufzeit der Operationen (deleteMin, decreaseKey amortisiert)

insert: $O(1)$, **deleteMin:** $O(\log n)$, **decreaseKey:** $O(1)$

n n ≤ m

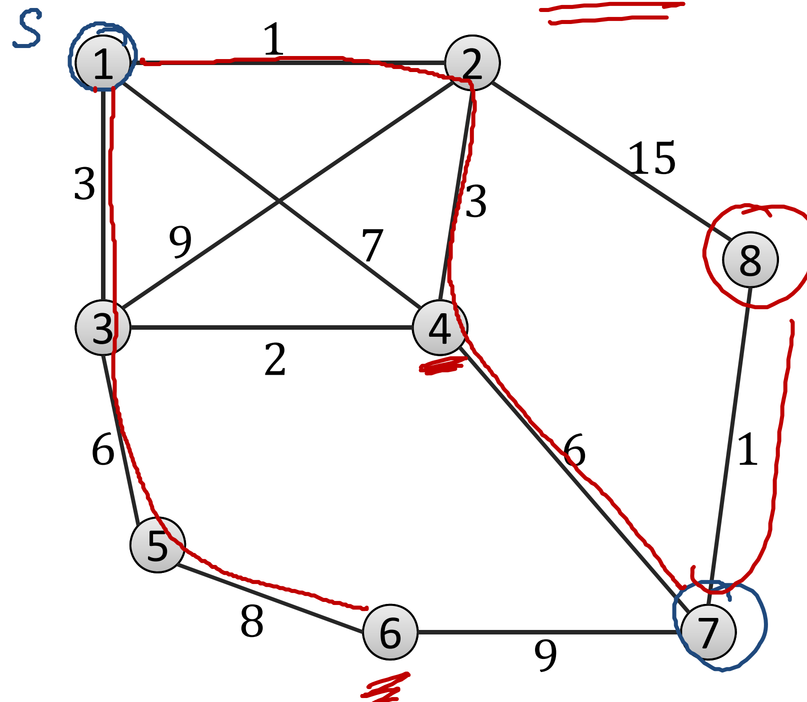
Laufzeit mit Fibonacci Heaps: $O(m + n \cdot \log n)$

- $n \leq m + 1$ insert-Operationen und deleteMin-Operationen
- $\leq m$ decreaseKey-Operationen

Kürzeste Wege

Problem

- Gegeben: gewichteter Graph $G = (V, E, w)$, Startknoten $s \in V$
  - Wir bezeichnen Gewicht einer Kante (u, v) als $w(u, v)$
  - Annahme: $\forall e \in E: w(e) \geq 0$
- Ziel: Finde kürzeste Pfade / Distanzen von s zu allen Knoten
  - Distanz von s zu v: $d_G(s, v)$ (Länge eines kürzesten Pfades)



single source shortest paths

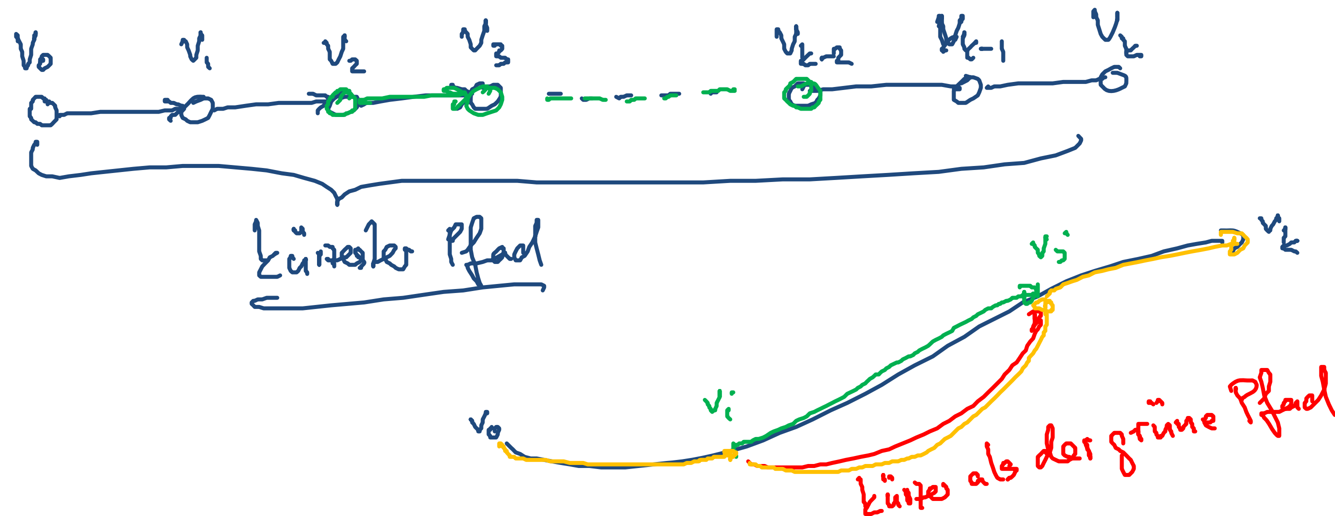
$d_G(s,v)$

$w(u,v) = w((u,v))$
$w(\{u,v\})$

Optimalität von Teilpfaden

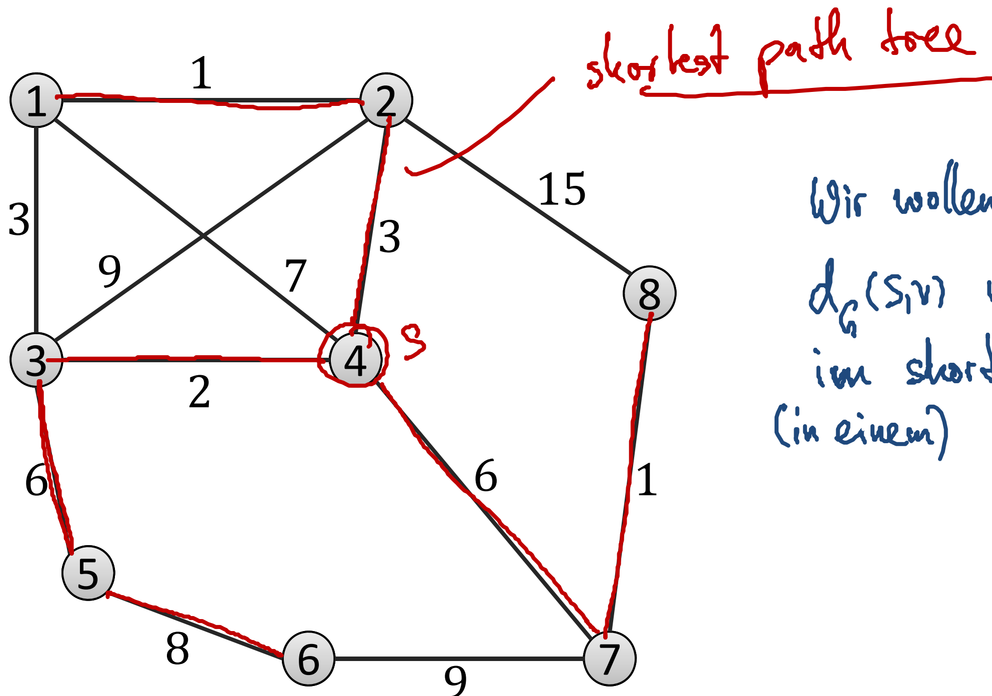
Lemma: Falls $v_0, v_1, \ldots, v_k$ ein kürzester Pfad von v_0 nach v_k ist, dann gilt für alle $0 \leq i \leq j \leq k$, dass der Teilpfad $v_i, v_{i+1}, \ldots, v_j$ ein kürzester Pfad von v_i nach v_j ist.

- gilt auch bei negativen Kantengewichten...



Shortest-Path Tree

- Im Knoten s gewurzelter Spannbaum, welcher kürzeste Pfade von s zu allen Knoten enthält.
  - Einen solchen Baum gibt immer (folgt insb. aus der Opt. der Teilpfade)
- Bei ungewichteten Graphen: BFS-Spannbaum
- **Ziel:** Finde einen "Shortest Path Tree"



Wir wollen für alle Knoten v $d_G(s,v)$ und den Parent von v im shortest-path Tree finden. (in einem)

Dijkstras Algorithmus: Idee

UNI FREIBURG

- Algorithmus von Edsger W. Dijkstra (1959 publiziert)

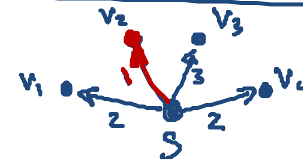
Annahme: $w(e) \geq 0$

Idee:

- Wir starten bei s und bauen schrittweise den Spannbaum auf

> **Invariante:**
> Algorithmus hat zu jeder Zeit einen bei s gewurzelten Teilbaum eines "Shortest Path Tree".

- Ziel: In jedem Schritt des Algorithmus einen Knoten hinzufügen
  - Am Anfang: Teilbaum besteht nur aus s (erfüllt Invariante trivialerweise...)

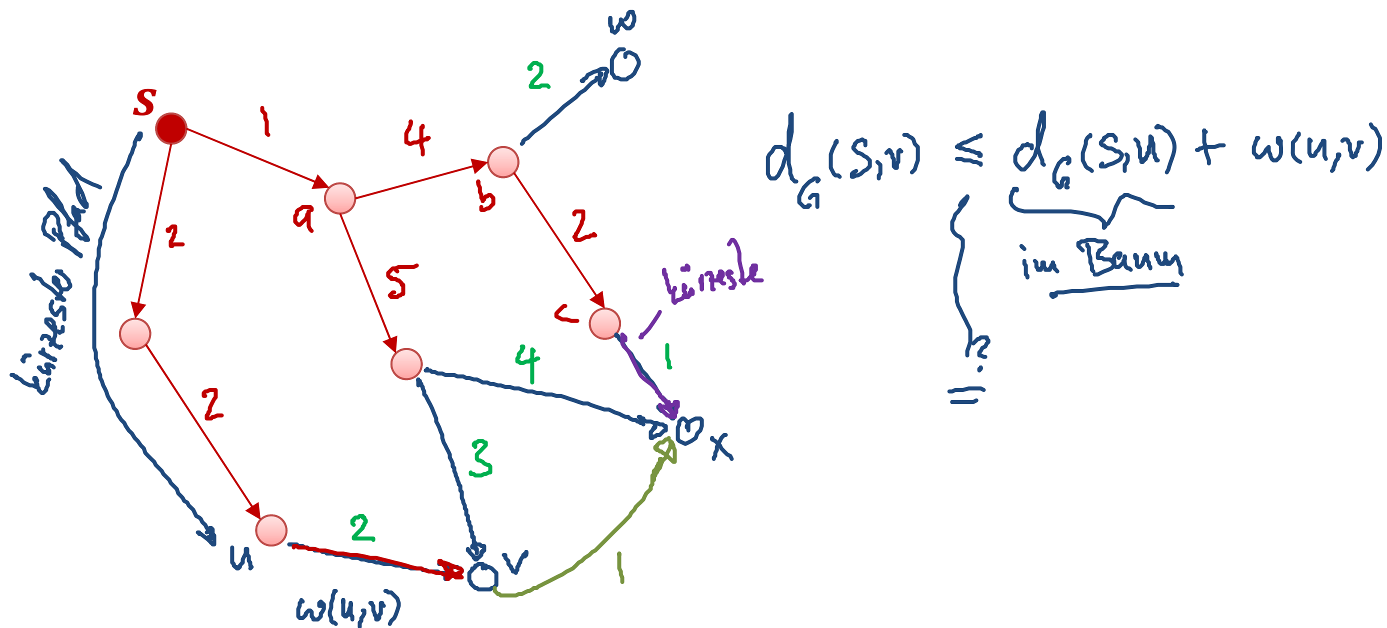


  - 1. Schritt: Wegen der Optimalität der Teilpfade, muss es einen kürzesten Pfad bestehend aus nur einer Kante geben...
  - Füge Knoten mit kleinstem Abstand zu s zum Baum hinzu

Dijkstras Algorithmus: Ein Schritt

Gegeben: Einen in s gewurzelten T, so dass T Teilbaum eines "Shortest Path Tree" von s in G ist.

Wie können wir T um einen Knoten erweitern?



Dijkstras Algorithmus: Ein Schritt

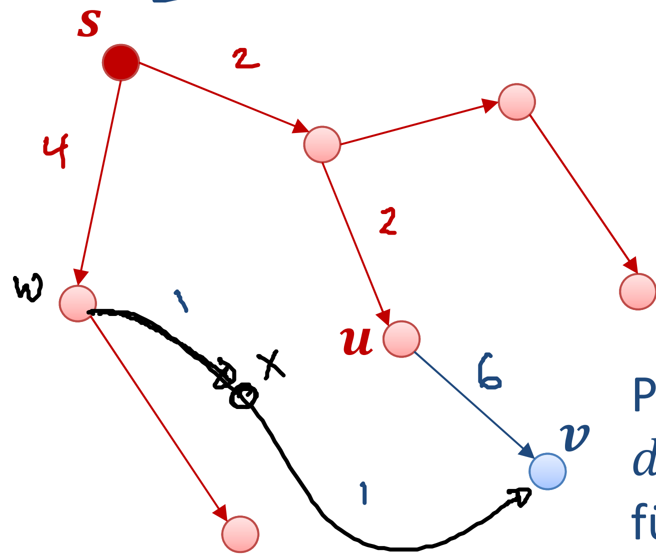
UNI FREIBURG

Gegeben: T ist Teilbaum eines "Shortest Path Tree" von s in G ist.

Lemma: Für die Kante (u, v) mit $u \in T$ und $v \notin T$, welche $d_G(s, u) + w(u, v)$ minimiert, gilt:

in T

$$\boldsymbol{d_G(s, v) = d_G(s, u) + w(u, v)}$$



funktioniert nur mit nicht-negativen Kantengewichten

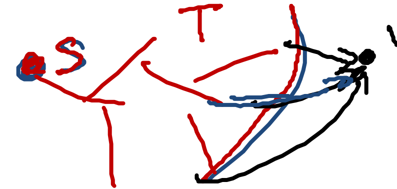
Paar (u, v) minimiert
$d_G(s, u) + w(u, v)$
für $u \in T$, $v \notin T$

$$d_G(s,w) + w(w,x) < d_G(s,u) + w(u,v)$$

Dijkstras Algorithmus

UNI FREIBURG

Invariante:
Algorithmus hat zu jeder Zeit einen bei s gewurzelten Teilbaum eines "Shortest Path Tree" T.



- Am Anfang ist $T = (\{s\}, \emptyset)$
- Für jeden Knoten $v \notin T$ berechnet man zu jedem Zeitpunkt

$$\delta(s,v) := \min_{u \in T \cap N_{\text{in}}(v)} d_G(s,u) + w(u,v)$$

  – sowie den Eingangsnachbar $u =: \alpha(v)$, welcher den Ausdruck minimiert...

- Invariante $\Longrightarrow \delta(s,v) \geq d_G(s,v)$
- Lemma auf letzter Folie:

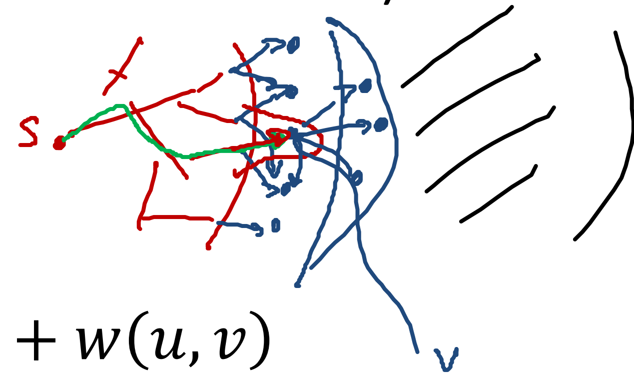
  Für das minmale $\boldsymbol{\delta(s,v)}$ gilt: $\boldsymbol{\delta(s,v) = d_G(s,v)}$

  über all $v \notin T$

Dijkstras Algorithmus

Initialisierung $T = (\emptyset, \emptyset)$

- $\delta(s,s) = 0$,   sowie $\delta(s,v) = \infty$ für alle $v \in V \setminus S$
- $\alpha(v) = \text{NULL}$ für alle $v \in V \setminus S$ (braucht's nicht für s)



Interationsschritt

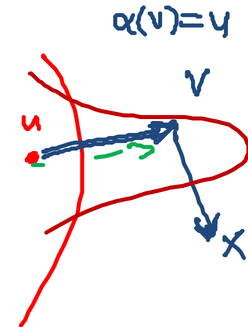
- Wähle Knoten v mit kleinstem

$$\delta(s,v) := \min_{u \in T \cap N_{\text{in}}(v)} d_G(s,u) + w(u,v)$$

- Gehe durch die Ausgangsnachbarn $x \in V \setminus T$ und setze

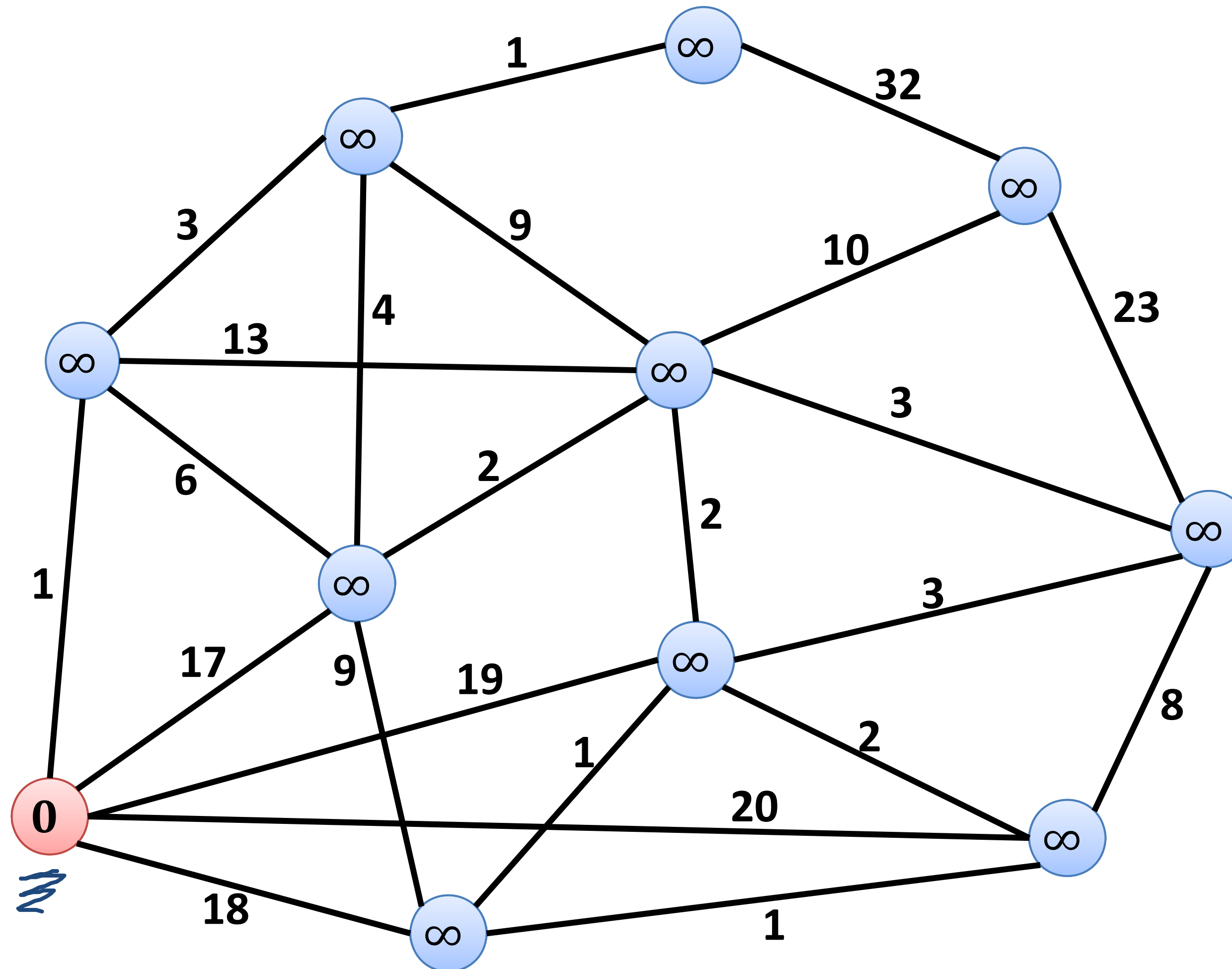
$$\delta(s,x) := \min\{\delta(s,x), \delta(s,v) + w(v,x)\}$$

  - Falls nötig, setze auch $\alpha(x) = v$

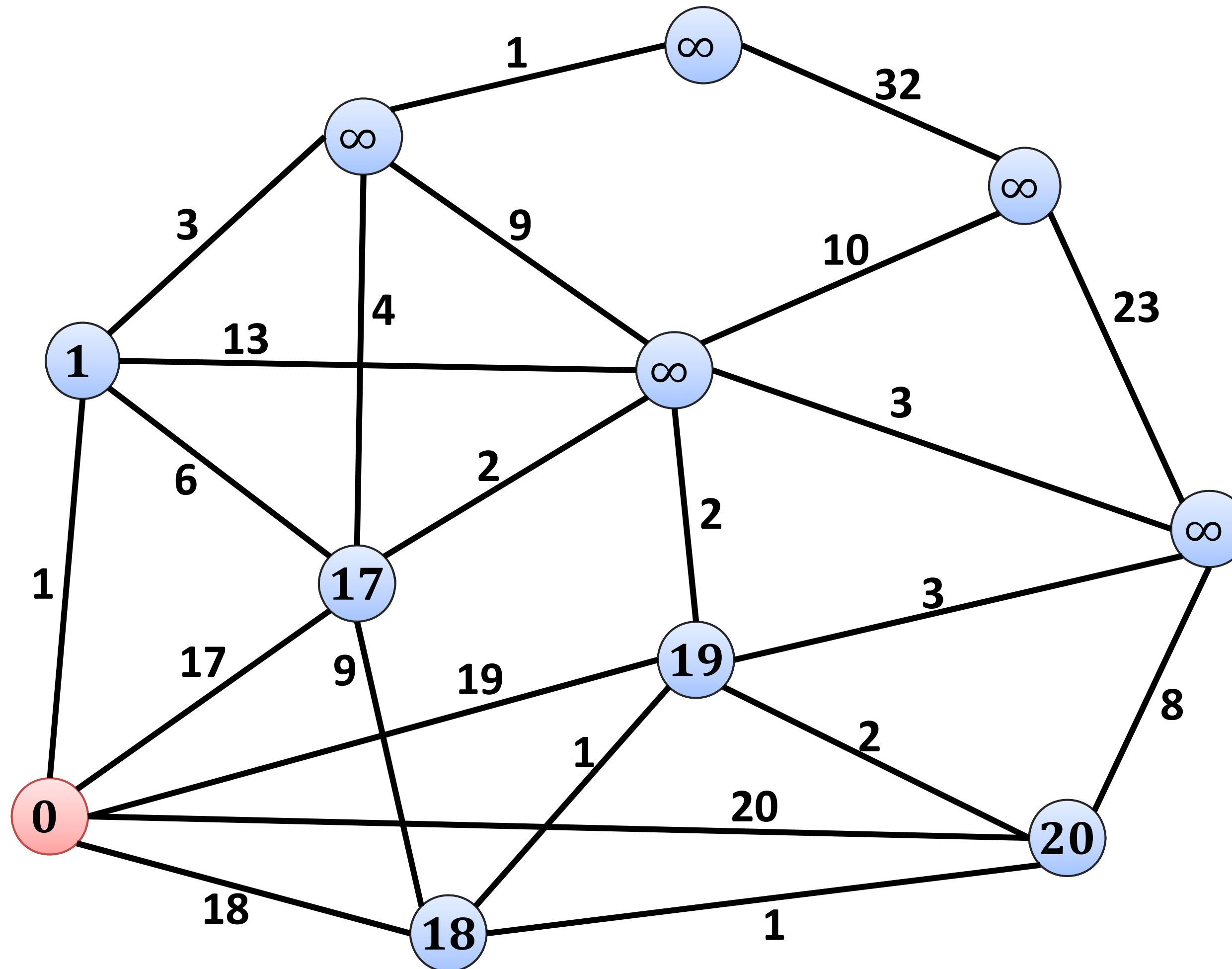


- Füge Kante $(\alpha(v), v)$ zum Baum T hinzu.

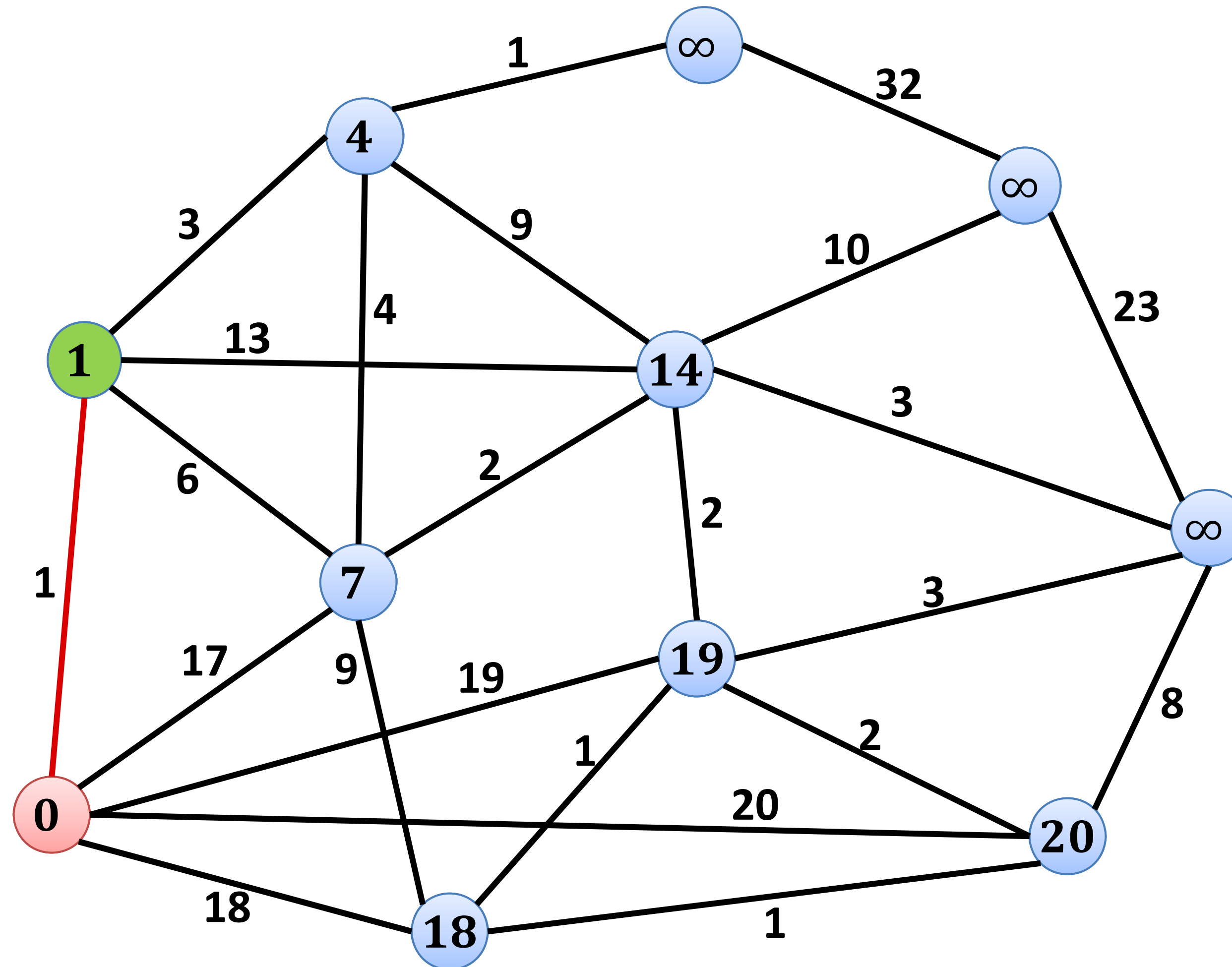
Dijkstras Algorithmus: Beispiel



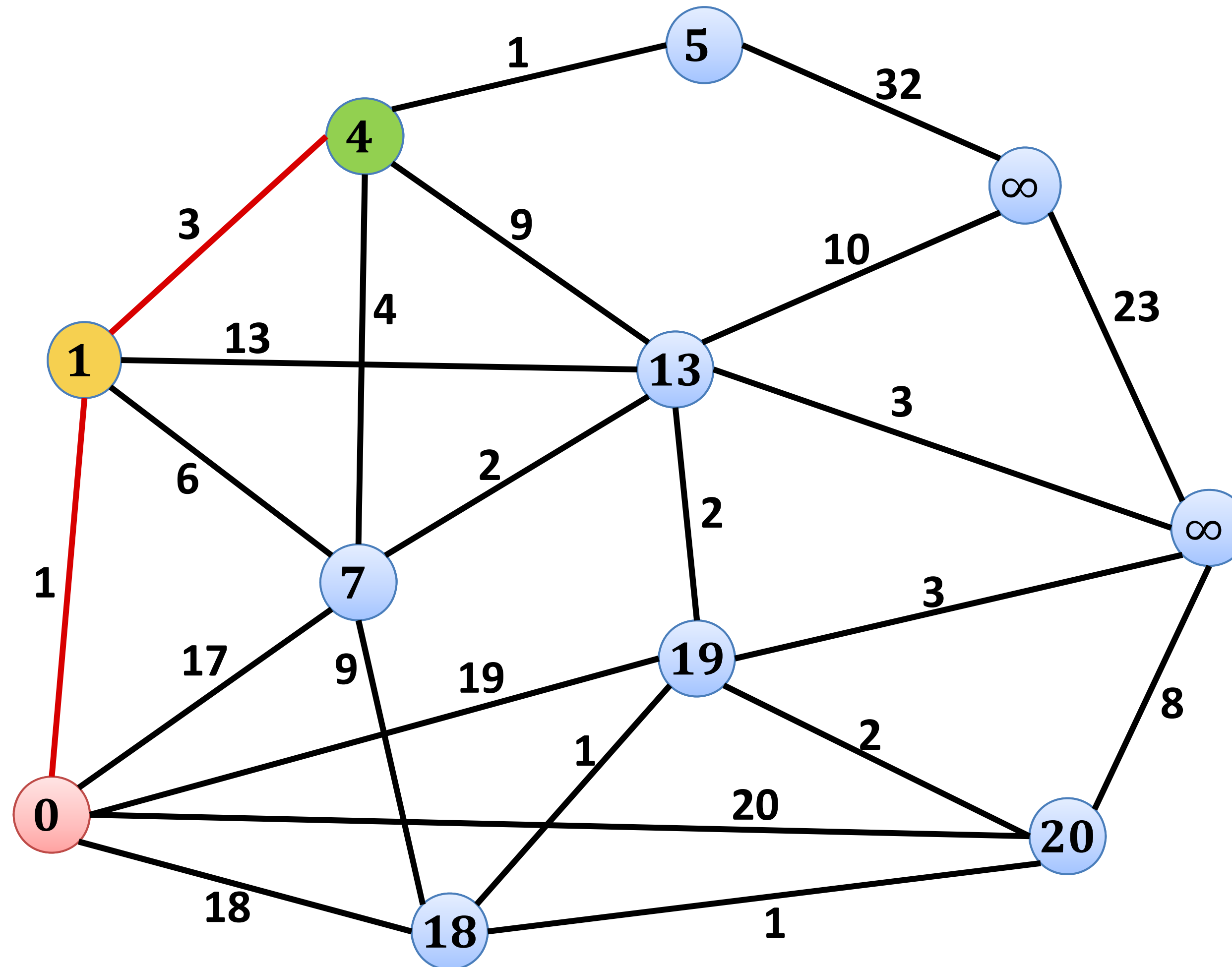
Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Beispiel

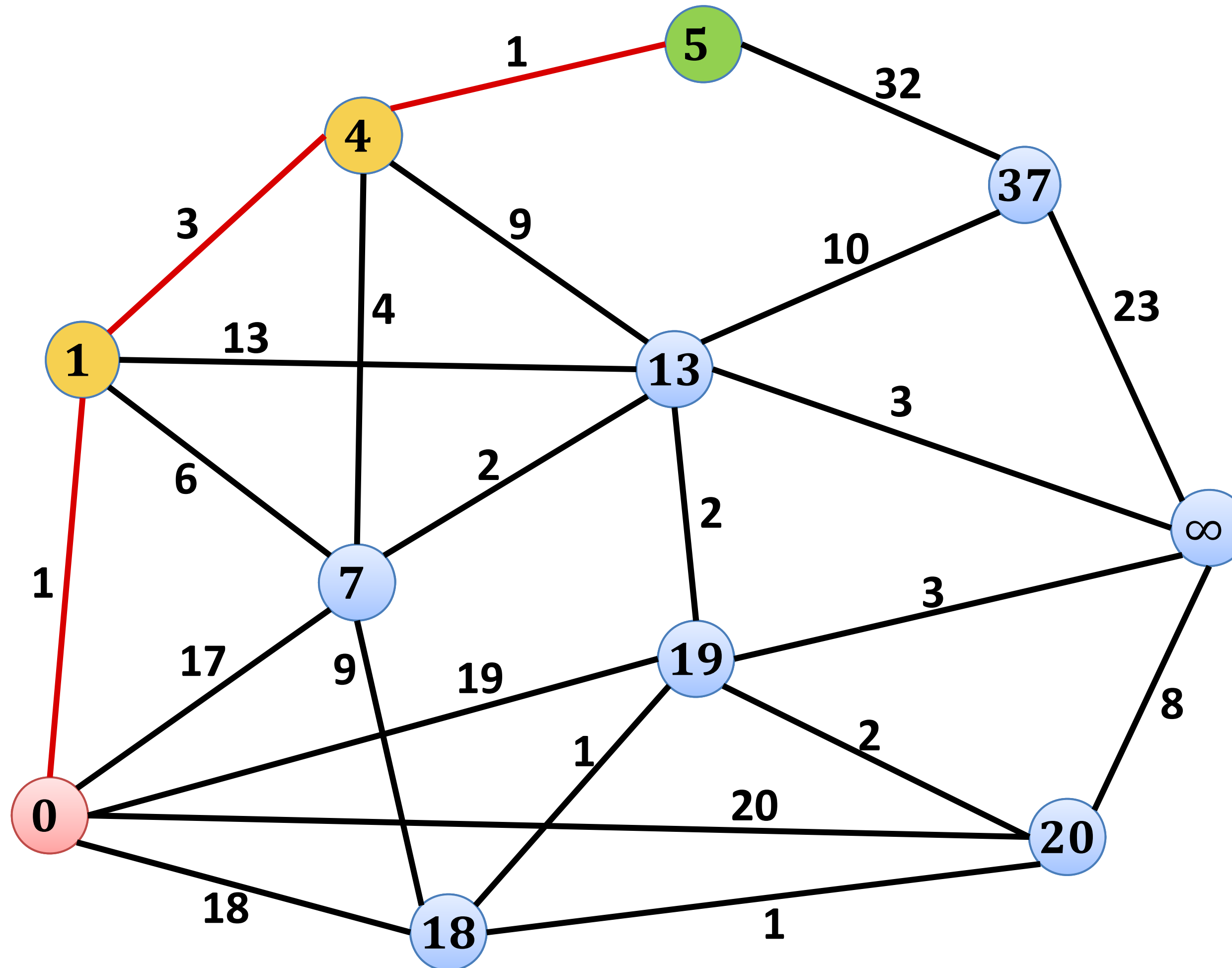


Dijkstras Algorithmus: Beispiel

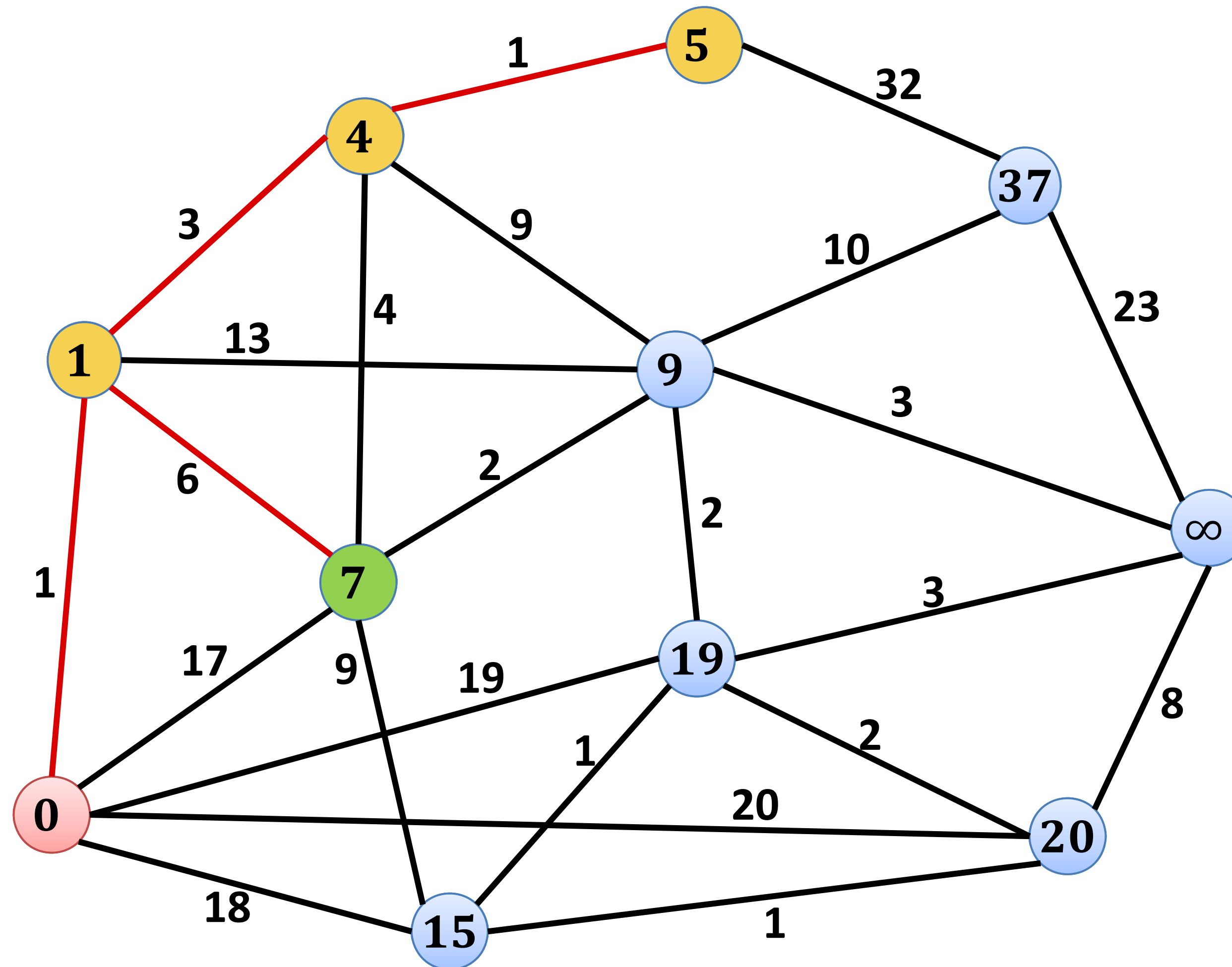


Dijkstras Algorithmus: Beispiel

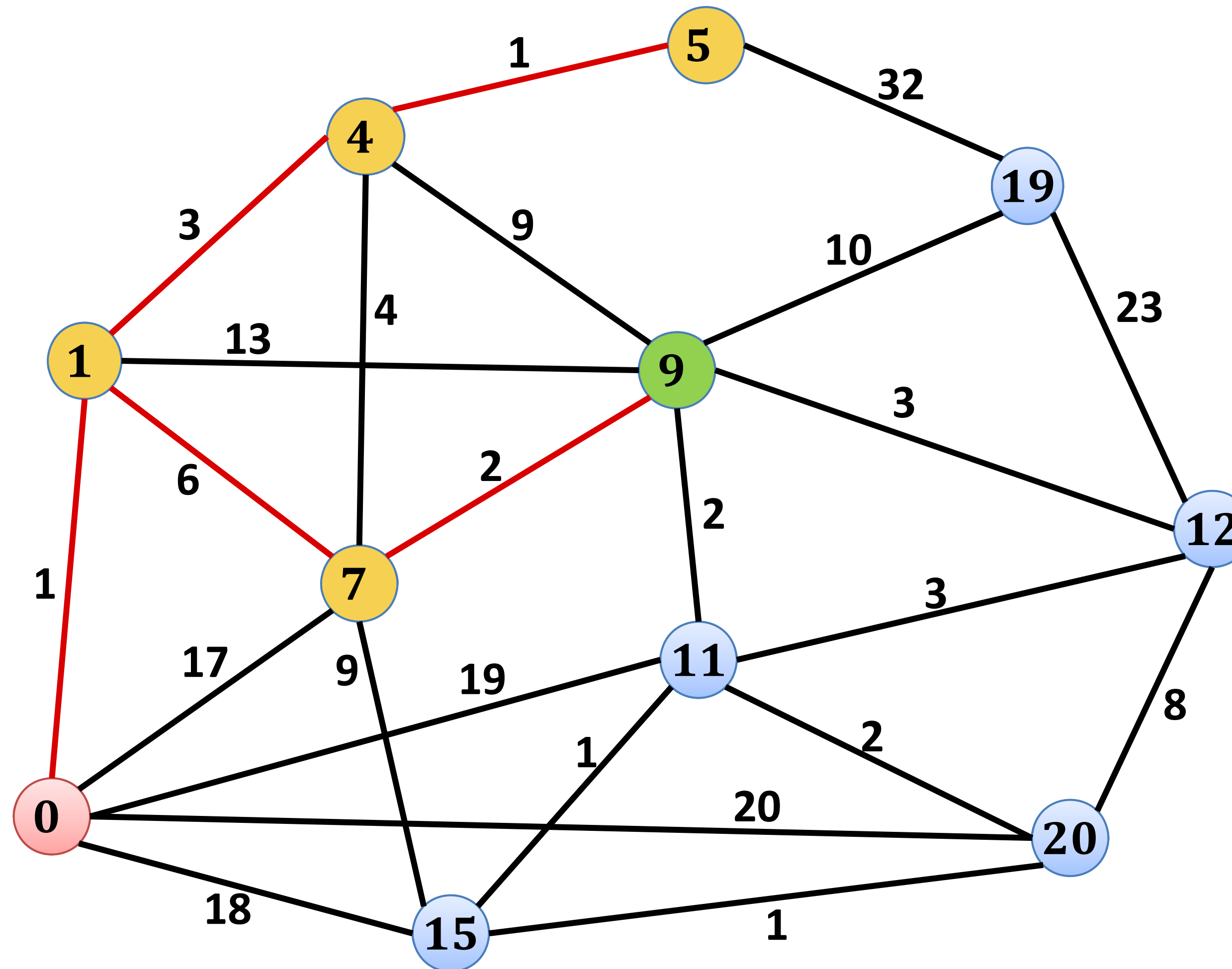
UNI FREIBURG



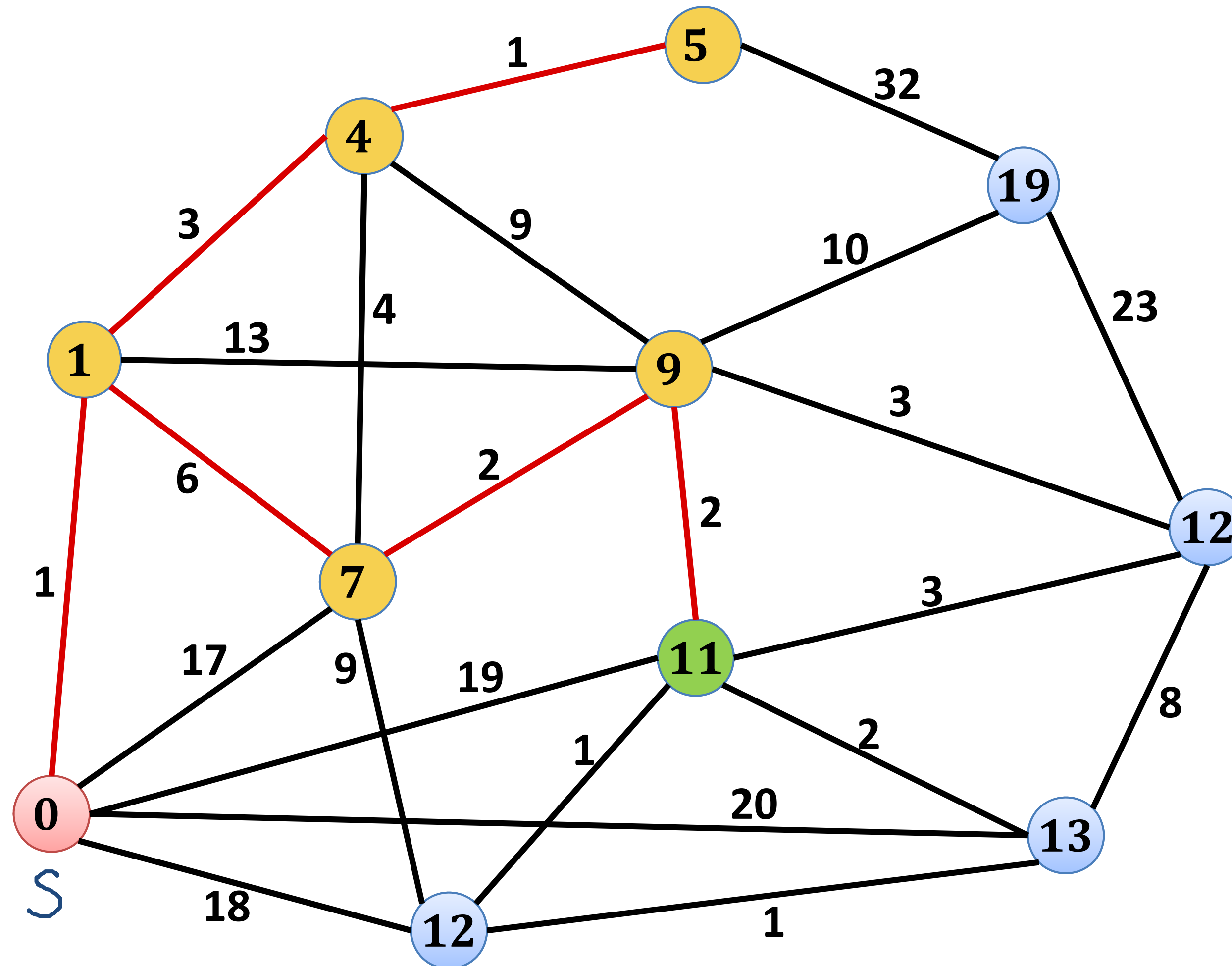
Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Implementierung

UNI FREIBURG

Initialisierung $\boldsymbol{T} = (\emptyset, \emptyset)$

- $\delta(s,s) = 0$, sowie $\delta(s,v) = \infty$ für alle $v \in V \setminus S$
- $\alpha(v) = \text{NULL}$ für alle $v \in V \setminus S$ (braucht's nicht für s)

Interationsschritt

- Wähle Knoten v mit kleinstem

$$\delta(s,v) \coloneqq \min_{u \in T \cap N_{\text{in}}(v)} d_G(s,u) + w(u,v)$$

- Gehe durch die Ausgangsnachbarn $x \in V \setminus T$ und setze

$$\delta(s,x) \coloneqq \min\{\delta(s,x), \delta(s,v) + w(v,x)\}$$

  – Falls nötig, setze auch $\alpha(x) = v$

- Füge Kante $(\alpha(v), v)$ zum Baum T hinzu.

Dijkstras Algorithmus: Implementierung

UNI FREIBURG

```
H = new priority queue; A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); δ(s,u) = ∞; α(u) = NULL
H.insert(s, 0)

while H is not empty do
    u = H.deleteMin()
    for all neighbors v of u do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) = δ(s,u) + w(u,v);
            α(v) = u
            H.decreaseKey(v, δ(s,u))
    if u ≠ s then A = A ∪ {{u, α(u)}}
```

d(u)

nicht in T, ausgehende Kanten

Dijkstas Algorithmus: Laufzeit

UNI FREIBURG

- Algorithmus-Implementierung ist fast identisch, wie diejenige von Prims MST Algorithmus

- **Anzahl Heap-Operationen:**

  create: 1, insert: n, deleteMin: n, decreaseKey: $\leq m$

- **Laufzeit mit binären Heaps:**

$$O(m \log n)$$

- **Laufzeit mit Fibonacci Heaps:**

$$O(m + n \log n)$$

Vorlesung nächste Woche

- Ich werde nächste Woche im Ausland sein...
- Wir werden übernächste Woche das Thema Graphen abschliessen.
- **Thema nächste Woche:**
  Editierdistanz und dynamische Programmierung
- Anstatt Vorlesung: Videos der letztjährigen Vorlesung
  - Details werden auf der Webseite und im Forum noch bekanntgegeben