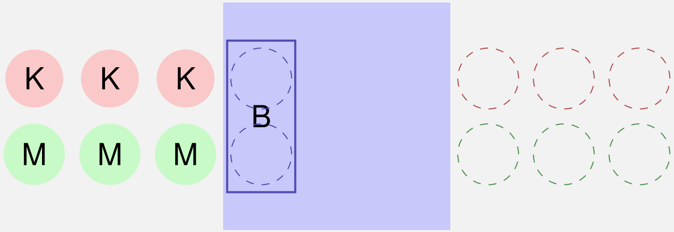
13. Shortest Paths

River Crossing (Missionaries and Cannibals)

Problem: Three cannibals and three missionaries are standing at a river bank. The available boat can carry two people. At no time may at any place (banks or boat) be more cannibals than missionaries. How can the missionaries and cannibals cross the river as fast as possible? [9]

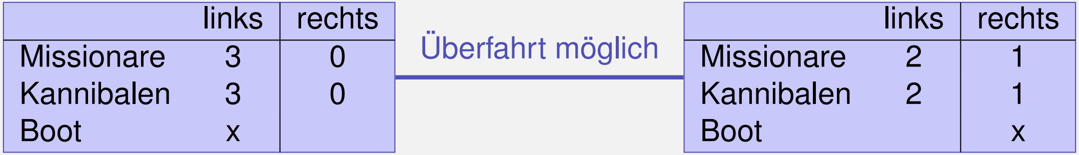


[9]There are slight variations of this problem. It is equivalent to the jealous husbands problem.

Problem as Graph

Enumerate permitted configurations as nodes and connect them with an edge, when a crossing is allowed. The problem then becomes a shortest path problem.

Example



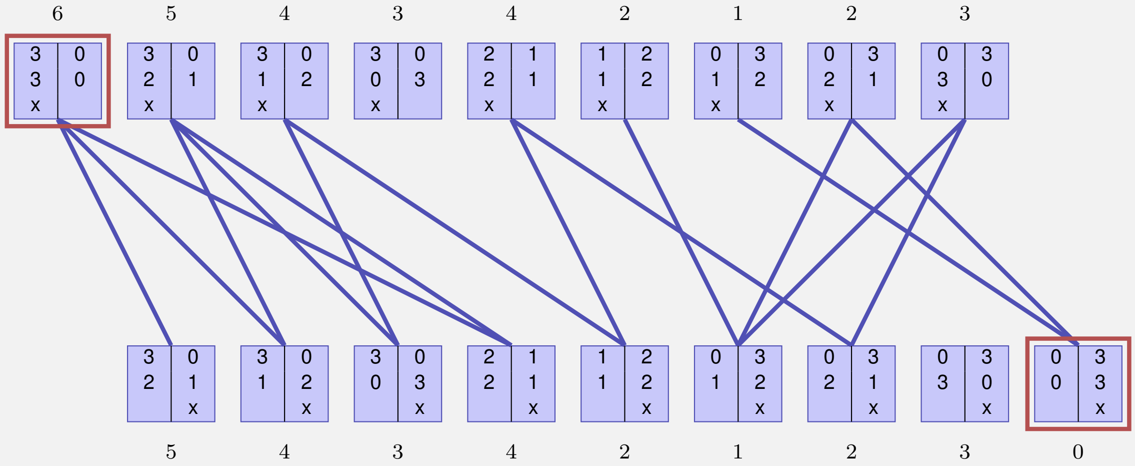
	links	rechts
Missionare	3	0
Kannibalen	3	0
Boot	x	

6 Personen am linken Ufer

	links	rechts
Missionare	2	1
Kannibalen	2	1
Boot		x

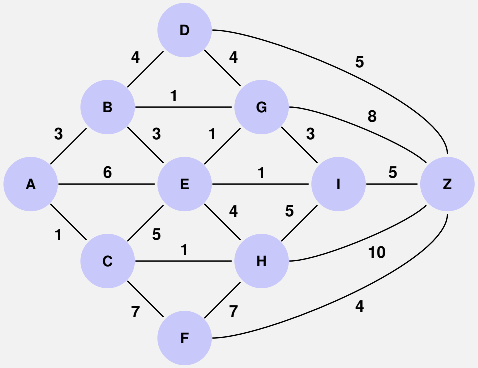
4 Personen am linken Ufer

The whole problem as a graph



Route Finding

Provided cities A - Z and Distances between cities.

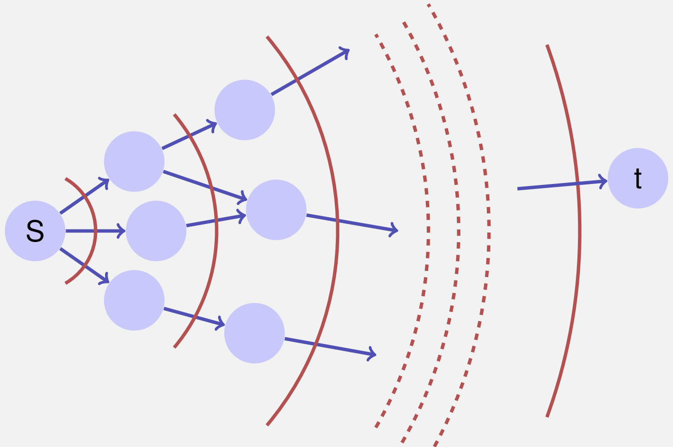


What is the shortest path from A to Z?

Simplest Case

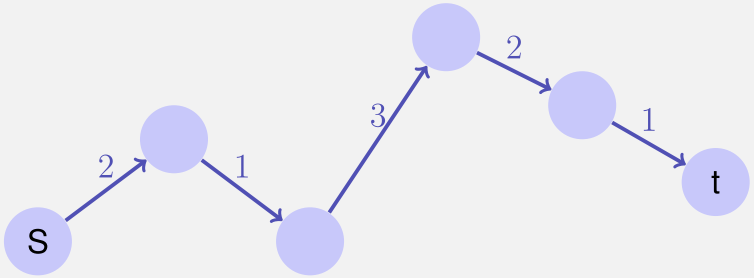
Constant edge weight 1 (wlog)

Solution: Breadth First Search



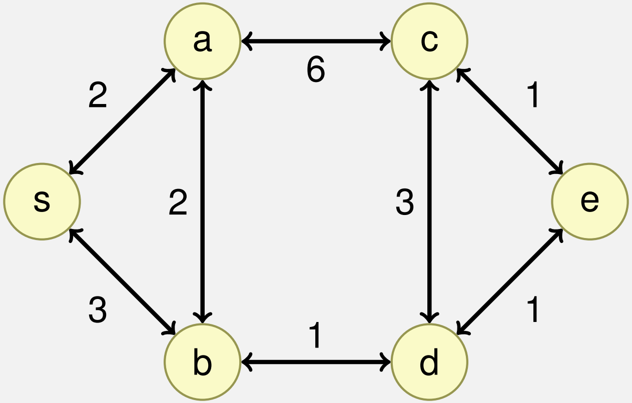
Weighted Graphs

Given: $G = (V, E, c)$, $c : E \to \mathbb{R}$, $s, t \in V$.
Wanted: Length (weight) of a shortest path from s to t.
Path: $p = \langle s = v_0, v_1, \dots, v_k = t \rangle$, $(v_i, v_{i+1}) \in E$ $(0 \leq i < k)$
Weight: $c(p) := \sum_{i=0}^{k-1} c((v_i, v_{i+1}))$.



Path with weight 9

Prelliminary Assumption



All weights of G are *positive*.

Shortest Paths

Weight of a shortest path from u to v:

$$\delta(u,v) = \begin{cases} \infty & \text{no path from } u \text{ to ävä} \\ \min\{c(p) : u \overset{p}{\rightsquigarrow} v\} & \text{sonst} \end{cases}$$

Ingredients of an Algorithm

Wanted: shortest paths from a starting node s.

- Weight of the shortest path found so far

$$d_s : V \to \mathbb{R}$$

At the beginning: $d_s[v] = \infty$ for all $v \in V$.
Goal: $d_s[v] = \delta(s, v)$ for all $v \in V$.

- Predecessor of a node

$$\pi_s : V \to \mathbb{R}$$

Zu Beginn $\pi_s[v]$ undefiniert für jeden Knoten $v \in V$

General Algorithm

1. Initialise d_s and π_s
2. Set $d_s[s] \leftarrow 0$
3. Choose an edge $(u, v) \in E$

```
Relaxiere $(u, v)$:
        if $d_s[v] > d[u] + c(u, v)$ then
                $d_s[v] \leftarrow d_s[u] + c(u, v)$
                $\pi_s[v] \leftarrow u$
```

4. Repeat 3 until nothing can be relaxed any more.
   (until $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$)

Triangle Inequality

For all $s, u, v \in V$:

$$\delta(s, v) \leq \delta(s, u) + \delta(u, v)$$

Triangle Inequality

For all $s, u, v \in V$:

$$\delta(s,v) \leq \delta(s,u) + \delta(u,v)$$

(A shortest path from s to v cannot be longer than a shortest path from s to v via u)

It is Safe to Relax

At any time in the algorithm above it holds

$$d_s[v] \geq \delta(s, v) \quad \forall v \in V$$

It is Safe to Relax

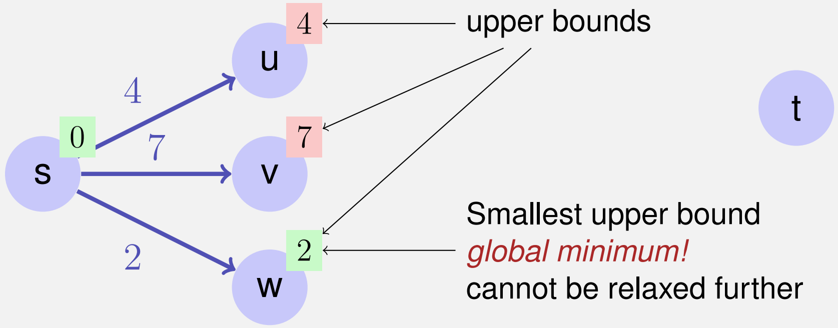
At any time in the algorithm above it holds

$$d_s[v] \geq \delta(s,v) \quad \forall v \in V$$

Inductive:

$$\begin{aligned} & \delta(S,v) \leq \delta(S,u) + \delta(u,v) && \text{[Triangle Inequality].} \\ & \delta(s,u) \leq d_s[u] && \text{[Induction Hypothesis].} \\ & \delta(u,v) \leq c(u,v) && \text{[Minimality of } \delta\text{]} \\ \Rightarrow \quad & \delta(S,v) \leq d_s[u] + c(u,v) \end{aligned}$$

Observation (Dijkstra)



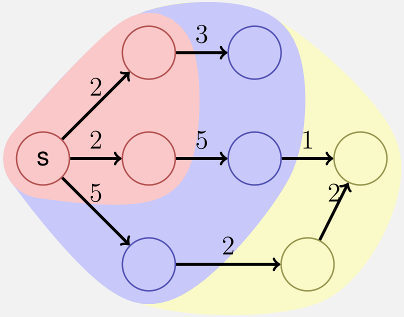
upper bounds

Smallest upper bound
global minimum!
cannot be relaxed further

Basic Idea

Set V of nodes is partitioned into

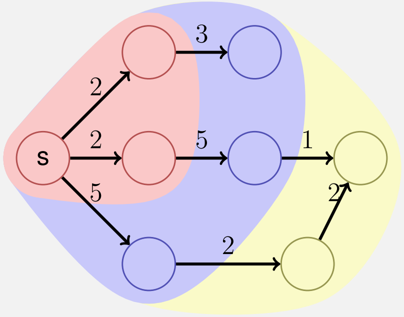
- the set M of nodes for which a shortest path from s is already known,
- the set $R = \bigcup_{v \in M} N^+(v) \setminus M$ of nodes where a shortest path is not yet known but that are accessible directly from M,
- the set $U = V \setminus (M \cup R)$ of nodes that have not yet been considered.



Induction

Induction over $|M|$: choose nodes from R with smallest upper bound. Add r to M and update R and U accordingly.

Correctness: if within the "wavefront" a node with minimal weight has been found then no path with greater weight over different nodes can provide any improvement.



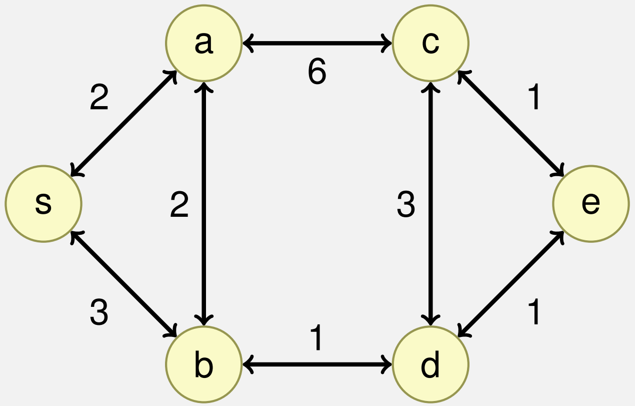
Algorithmus Dijkstra

Initial: $\mathrm{PL}(n) \leftarrow \infty$ für alle Knoten.

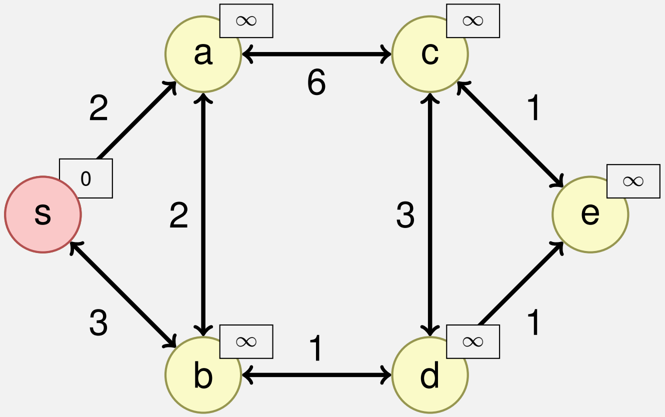
- Set $\mathrm{PL}(s) \leftarrow 0$
- Start with $M = \{s\}$. Set $k \leftarrow s$.
- While a new node k is added and this is not the target node

  1. For each neighbour node n of k:
     - compute path length x to n via k
     - If $\mathrm{PL}(n) = \infty$, than add n to R
     - If $x < \mathrm{PL}(n) < \infty$, then set $\mathrm{PL}(n) \leftarrow x$ and adapt R .
  2. Choose as new node k the node with smallest path length in R.

Example



Example

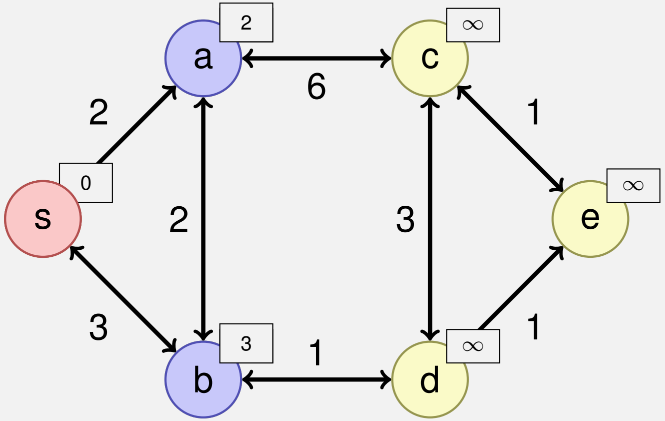


$$M = \{s\}$$

$$R = \{\}$$

$$U = \{a, b, c, d, e\}$$

Example

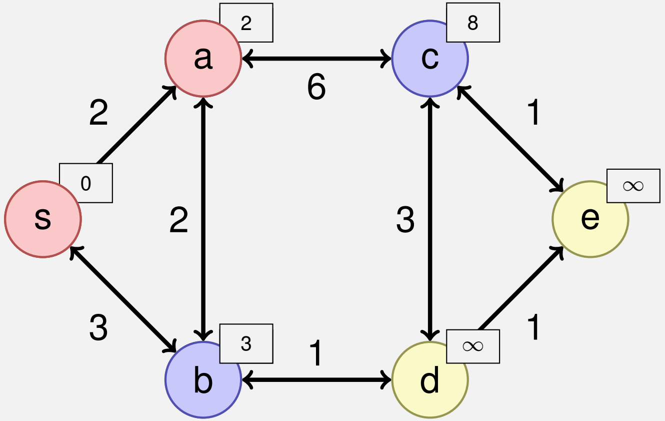


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Example

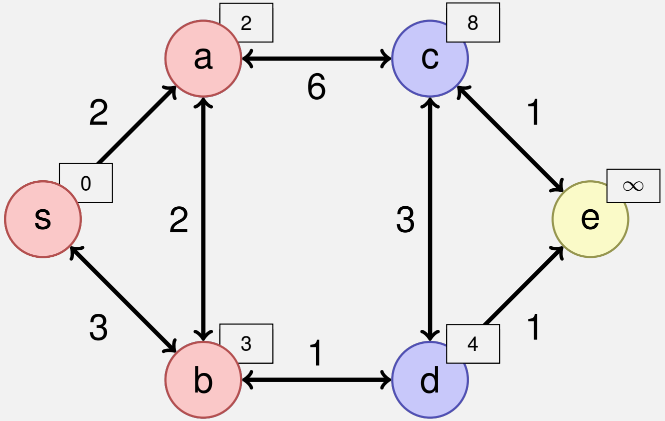


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Example

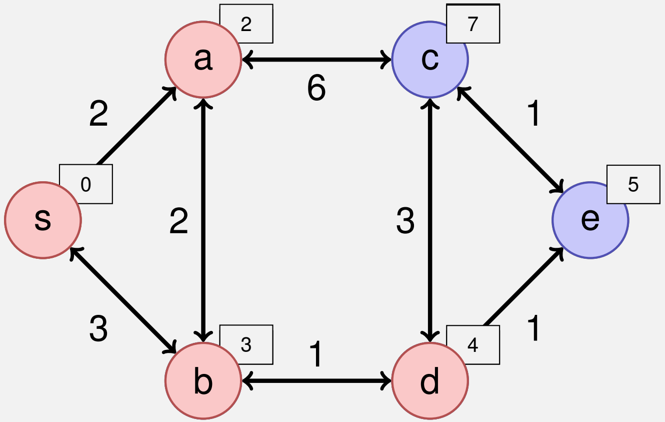


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Example

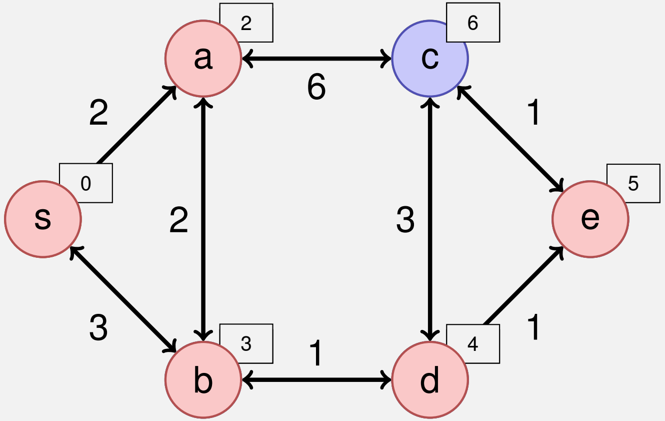


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Example

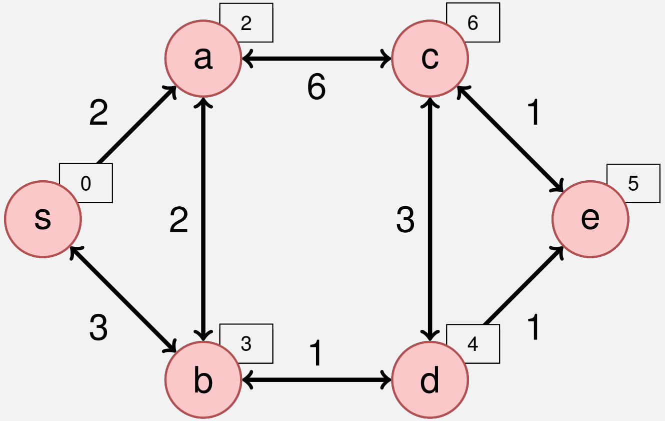


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Example



$$M = \{s, a, b, d, e, c\}$$

$$R = \{\}$$

$$U = \{\}$$

Implementation: Data Structure for R?

Required operations:

- Insert (add to R)
- ExtractMin (over R) and DecreaseKey (Update in R)

```
foreach $v \in N^+(m)$ do
    if $d(m) + c(m,v) < d(v)$ then
        $d(v) \leftarrow d(m) + c(m,v)$
        if $v \in R$ then
            DecreaseKey$(R, v)$          // Update of a $d(v)$ in the heap of $R$
        else
            $R \leftarrow R \cup \{v\}$          // Update of $d(v)$ in the heap of $R$
```

Implementation: Data Structure for R?

Required operations:

- Insert (add to R)
- ExtractMin (over R) and DecreaseKey (Update in R)

```
foreach $v \in N^+(m)$ do
    if $d(m) + c(m,v) < d(v)$ then
        $d(v) \leftarrow d(m) + c(m,v)$
        if $v \in R$ then
            DecreaseKey($R, v$)            // Update of a $d(v)$ in the heap of $R$
        else
            $R \leftarrow R \cup \{v\}$            // Update of $d(v)$ in the heap of $R$
```

MinHeap!

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap?

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap?
  - alternative (a): Store position at the nodes

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap?
  - alternative (a): Store position at the nodes
  - alternative (b): Hashtable of the nodes

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap?
    - alternative (a): Store position at the nodes
    - alternative (b): Hashtable of the nodes
    - alterantive (c): re-insert node after update-operation and mark it "deleted" once extracted (Lazy Deletion)

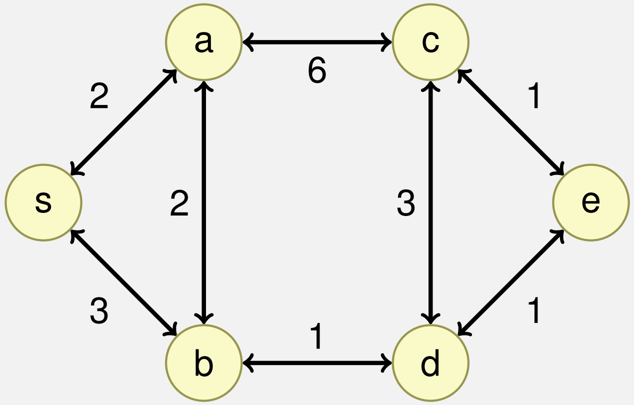
Runtime

- $|V| \times$ ExtractMin: $\mathcal{O}(|V| \log |V|)$
- $|E| \times$ Insert or DecreaseKey: $\mathcal{O}(|E| \log |V|)$
- $1 \times$ Init: $\mathcal{O}(|V|)$
- Overal: $\mathcal{O}(|E| \log |V|)$.

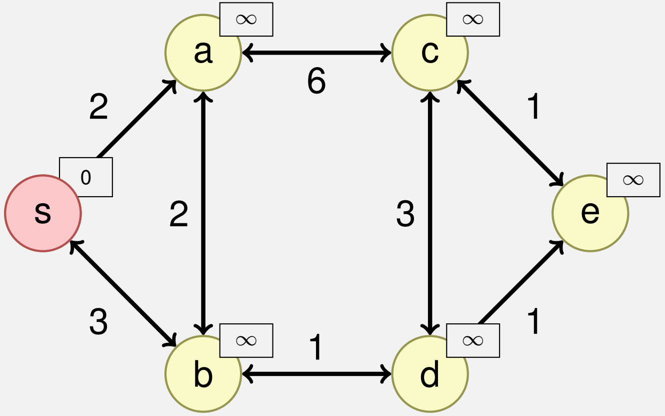
Reconstruct shortest Path

- Memorize best predecessor during the update step in the algorithm above. Store it with the node or in a separate data structure.
- Reconstruct best path by traversing backwards via best predecessor

Example



Example

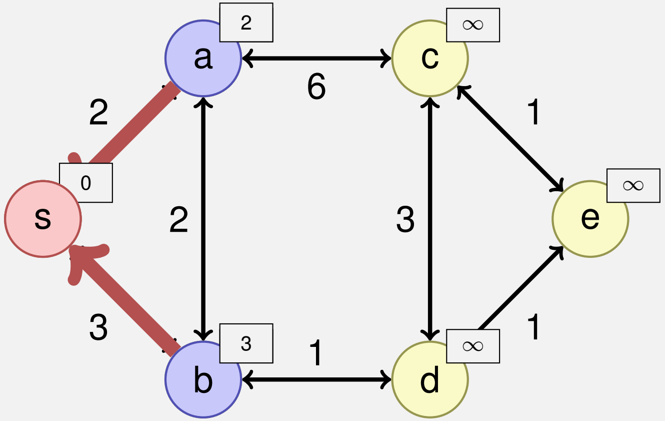


$$M = \{s\}$$

$$R = \{\}$$

$$U = \{a, b, c, d, e\}$$

Example

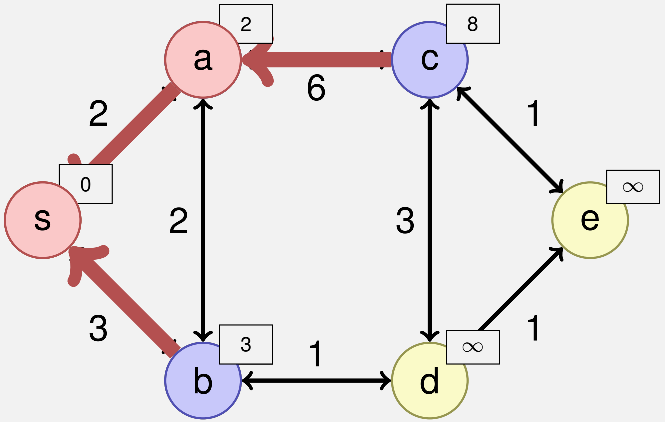


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Example

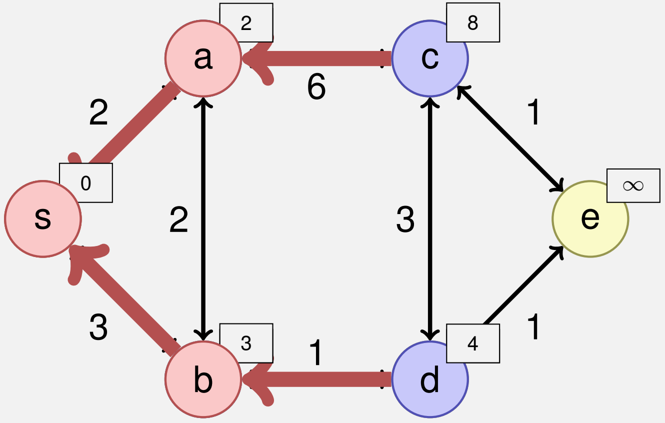


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Example

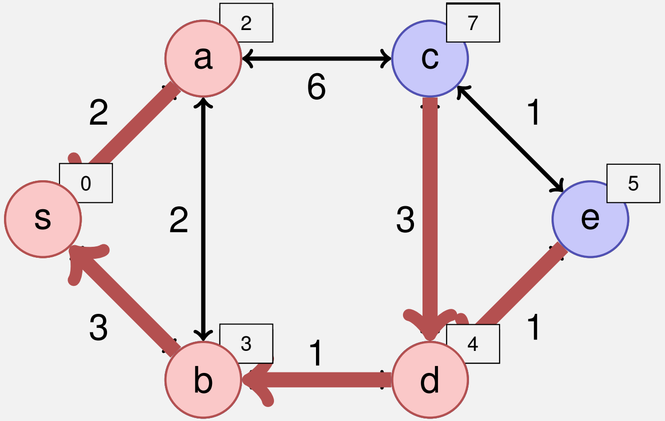


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Example

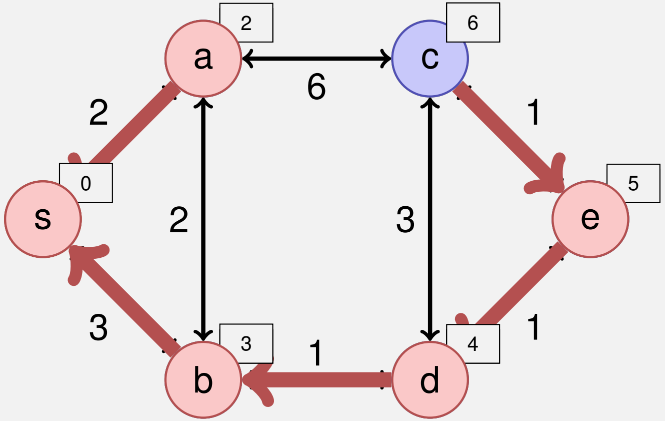


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Example

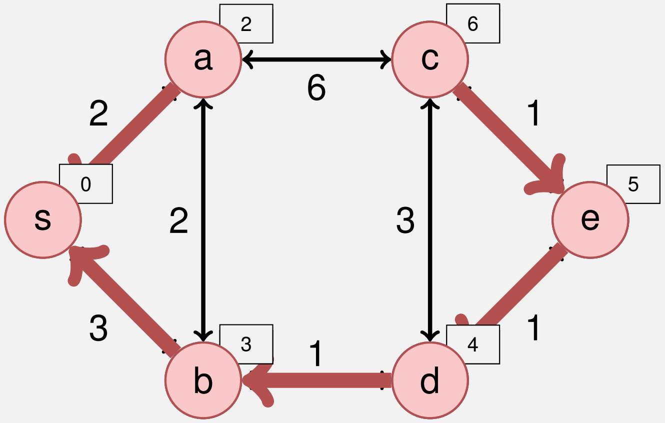


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Example



$$M = \{s, a, b, d, e, c\}$$

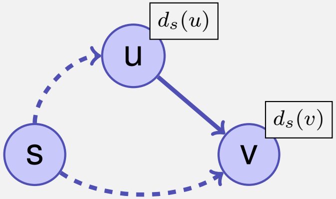
$$R = \{\}$$

$$U = \{\}$$

General Weighted Graphs

Relaxing Step as with Dijkstra:

```
Relax($u, v$) ($u, v \in V$, $(u, v) \in E$)
if $d_s(v) > d_s(u) + c(u, v)$ then
    $d_s(v) \leftarrow d_s(u) + c(u, v)$
    return true
return false
```



Problem: cycles with negative weights can shorten the path, a shortest path is not guaranteed to exist.

Observations

- *Observation 1:* Sub-paths of shortest paths are shortest paths. Let $p = \langle v_0, \dots, v_k \rangle$ be a shortest path from v_0 to v_k. Then each of the sub-paths $p_{ij} = \langle v_i, \dots, v_j \rangle$ $(0 \le i < j \le k)$ is a shortest path from v_i to v_j.
  Proof: if not, then one of the sub-paths could be shortened which immediately leads to a contradiction.
- *Observation:* If there is a shortest path then it is simple, thus does not provide a node more than once.
  Immediate Consequence of observation 1.

Dynamic Programming Approach (Bellman)

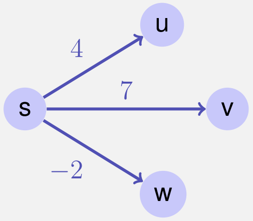
Induction over number of edges $d_s[i,v]$: Shortest path from s to v via maximally i edges.

$$d_s[i,v] = \min\{d_s[i-1,v], \min_{(u,v)\in E}(d_s[i-1,u] + c(u,v))$$

$$d_s[0,s] = 0, d_s[0,v] = \infty \; \forall v \neq s.$$

Dynamic Programming Approach (Bellman)

	s	$\cdots$	v	$\cdots$	w
0	0	∞	∞	∞	∞
1	0	∞	7	∞	-2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	0	$\cdots$	$\cdots$	$\cdots$	$\cdots$



Algorithm: Iterate over last row until the relaxation steps do not provide any further changes, maximally $n-1$ iterations. If still changes, then there is no shortest path.

Algorithm Bellman-Ford(G, s)

Input : Graph $G = (V, E, c)$, starting point $s \in V$
Output : If return value true, minimal weights d for all shortest paths from s, otherwise no shortest path.

```
$d(v) \leftarrow \infty \; \forall v \in V$; $d(s) \leftarrow 0$
for $i \leftarrow 1$ to $|V|$ do
    $f \leftarrow$ false
    foreach $(u, v) \in E$ do
        $f \leftarrow f \vee \mathrm{Relax}(u, v)$
    if $f =$ false then return true
return false;
```

Runtime $\mathcal{O}(|E| \cdot |V|)$.