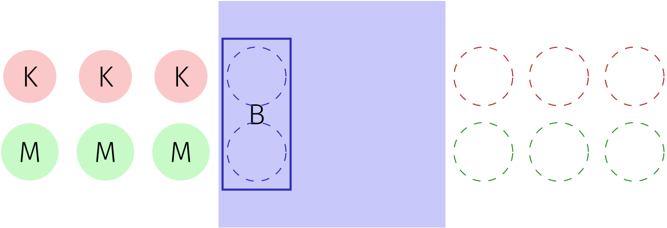
13. Shortest Paths

Motivation, Universal Algorithm, Dijkstra’s algorithm on distance graphs, [Ottman/Widmayer, Kap. 9.5.1-9.5.2 Cormen et al, Kap. 24.1-24.3]

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? [16]



---

[16] 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	

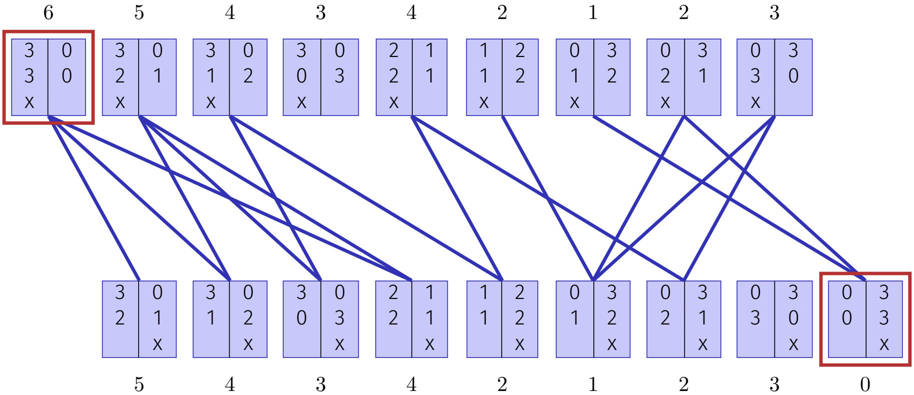
6 Personen am linken Ufer

Überfahrt möglich

	links	rechts
Missionare	2	1
Kannibalen	2	1
Boot		x

4 Personen am linken Ufer

The whole problem as a graph



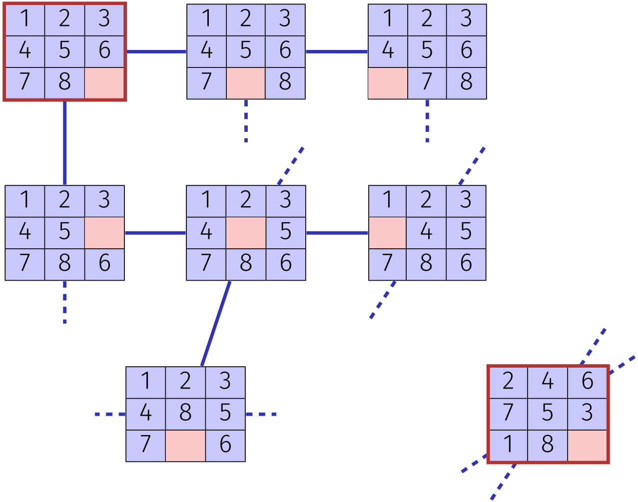
Another Example: Mystic Square

Want to find the fastest solution for

2	4	6
7	5	3
1	8	

1	2	3
4	5	6
7	8	

Problem as Graph



1	2	3
4	5	6
7	8	

1	2	3
4	5	6
7		8

1	2	3
4	5	6
	7	8

1	2	3
4	5	
7	8	6

1	2	3
4		5
7	8	6

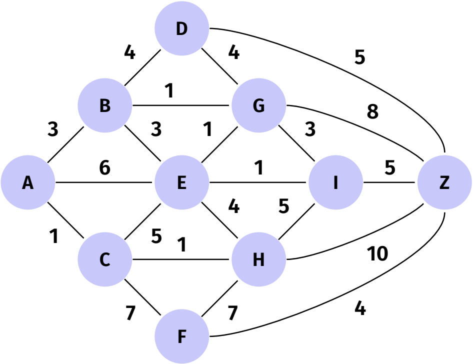
1	2	3
	4	5
7	8	6

1	2	3
4	8	5
7		6

2	4	6
7	5	3
1	8	

Route Finding

Provided cities A - Z and Distances between cities.

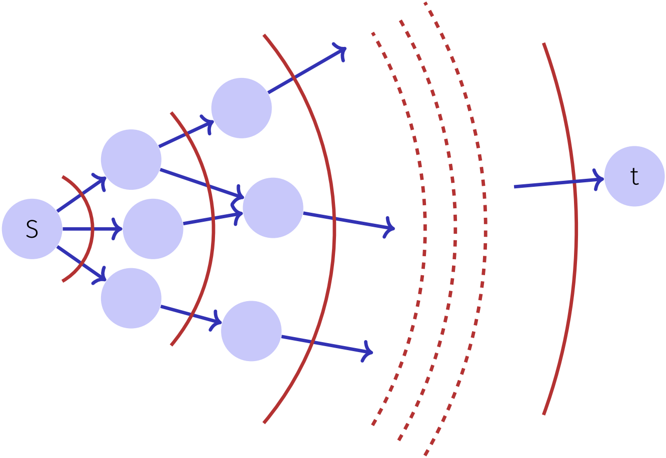


What is the shortest path from A to Z?

Simplest Case

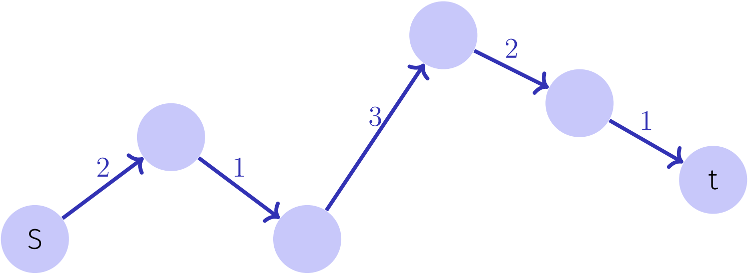
Constant edge weight 1 (wlog)

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: $p = \langle s = v_0, v_1, \ldots, v_k = t\rangle$, $(v_i, v_{i+1}) \in E$ $(0 \leq i < k)$
Weight: $c(p) := \sum_{i=0}^{k-1} c((v_i, v_{i+1}))$.



Path with weight 9

Shortest Paths

Notation: we write

$$u \overset{p}{\rightsquigarrow} v \qquad \text{oder} \qquad p : u \rightsquigarrow v$$

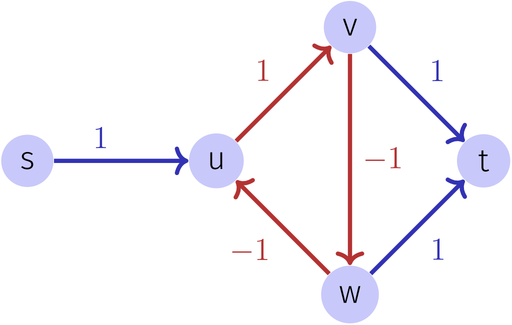
and mean a path p from u to v

Notation: $\delta(u, v)$ = weight of a shortest path from u to v:

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

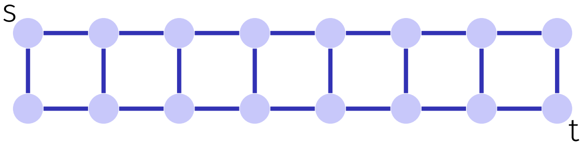
Observations (1)

It may happen that a shortest paths does not exist: negative cycles can occur.



Observations (2)

There can be exponentially many paths.



(at least $2^{|V|/2}$ paths from s to t)

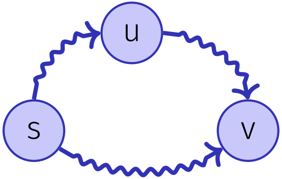
⇒ To try all paths is too inefficient

Observations (3)

Triangle Inequality

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

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

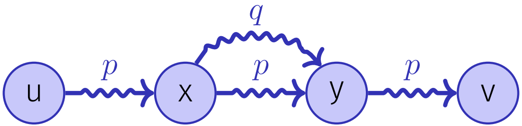


A shortest path from s to v cannot be longer than a shortest path from s to v that has to include u

Observations (4)

Optimal Substructure

Sub-paths of shortest paths are shortest paths. Let $p = \langle v_0, \ldots, v_k \rangle$ be a shortest path from v_0 to v_k. Then each of the sub-paths $p_{ij} = \langle v_i, \ldots, v_j \rangle$ $(0 \leq i < j \leq k)$ is a shortest path from v_i to v_j.



If not, then one of the sub-paths could be shortened which immediately leads to a contradiction.

Observations (5)

Shortest paths do not contain cycles

1. Shortest path contains a negative cycle: there is no shortest path, contradiction
2. Path contains a positive cycle: removing the cycle from the path will reduce the weight. Contradiction.
3. Path contains a cycle with weight 0: removing the cycle from the path will not change the weight. Remove the cycle (convention).

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

  $$\pi_s : V \to V$$

  Initially $\pi_s[v]$ undefined for each node $v \in V$

General Algorithm

1. Initialise d_s and π_s: $d_s[v] = \infty$, $\pi_s[v] =$ null for each $v \in V$
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 $d_s[v] \leq d_s[u] + c(u,v) \quad \forall (u,v) \in E$)

It is Safe to Relax

At any time in the algorithm above it holds

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

In the relaxation step:

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

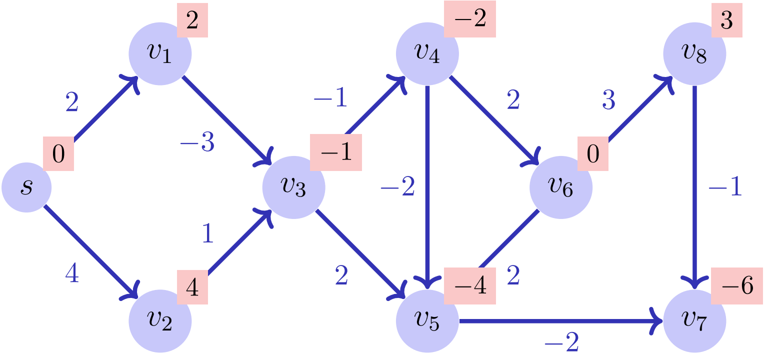
$$\Rightarrow \min\{d_s[v], d_s[u] + c(u,v)\} \geq \delta(s,v)$$

Central Question

How / in which order should edges be chosen in above algorithm?

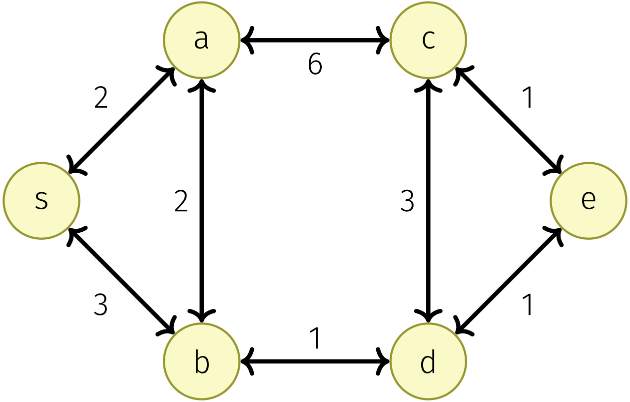
Special Case: Directed Acyclic Graph (DAG)

DAG ⇒ topological sorting returns optimal visiting order



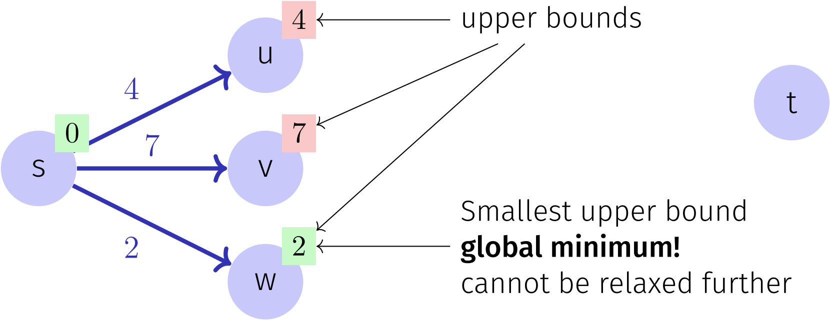
Top. Sort: ⇒ Order $s, v_1, v_2, v_3, v_4, v_6, v_5, v_8, v_7$.

Assumption (preliminary)



All weights of G are **positive**.

Observation (Dijkstra)



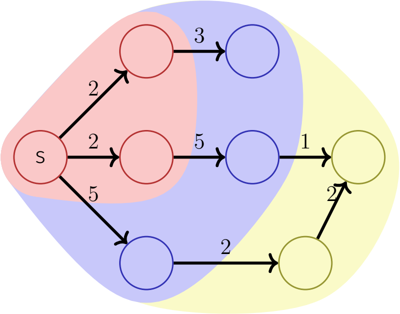
upper bounds

Smallest upper bound
global minimum!
cannot be relaxed further

Basic Idea

Set V of nodes is partitioned into

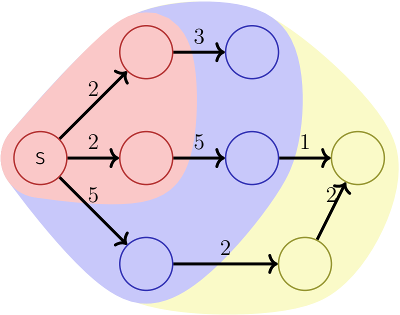
- the set M of nodes for which a shortest path from s is already known,
- the set $R = \bigcup_{v \in M} N^+(v) \setminus M$ of nodes where a shortest path is not yet known but that are accessible directly from M,
- the set $U = V \setminus (M \cup R)$ of nodes that have not yet been considered.



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 w has been found then no path over later nodes (providing weight $\geq d$) can provide any improvement.

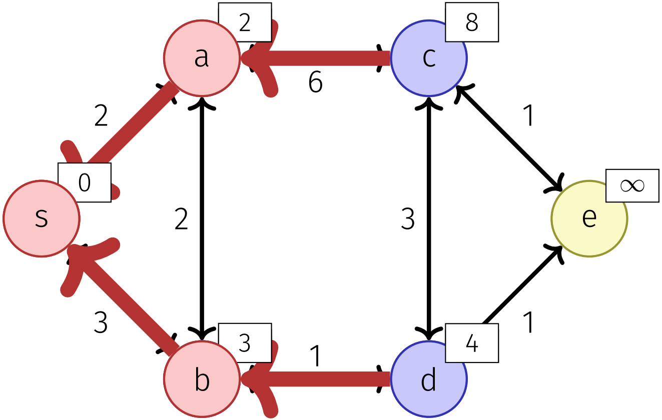


Algorithm Dijkstra(G, s)

Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$,
Output: Minimal weights d of the shortest paths and corresponding predecessor node for each node.

```
foreach $u \in V$ do
    $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow$ null
$d_s[s] \leftarrow 0$; $R \leftarrow \{s\}$
while $R \neq \emptyset$ do
    $u \leftarrow$ ExtractMin($R$)
    foreach $v \in N^+(u)$ do
        if $d_s[u] + c(u, v) < d_s[v]$ then
            $d_s[v] \leftarrow d_s[u] + c(u, v)$
            $\pi_s[v] \leftarrow u$
            $R \leftarrow R \cup \{v\}$
```

Example



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

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

$$U = \{e\}$$

Implementation: Data Structure for R?

Required operations:

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

```
foreach $v \in N^+(u)$ do
    if $d_s[u] + c(u,v) < d_s[v]$ then
        $d_s[v] \leftarrow d_s[u] + c(u,v)$
        $\pi_s[v] \leftarrow u$
        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?
  - alternative (a): Store position at the nodes
  - alternative (b): Hashtable of the nodes
  - alterantive (c): re-insert node after successful relax operation and mark it "deleted" once extracted (Lazy Deletion).[17]

---

[17] For lazy deletion a pair of egde (or target node) and distance is required.

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: $\mathcal{O}(|E|\log|V|)$.