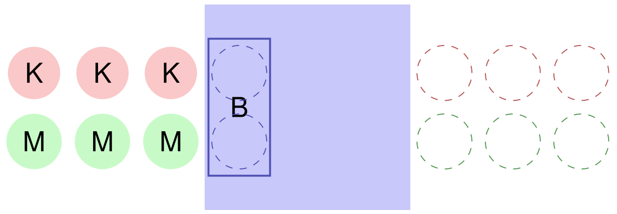
24. Shortest Paths

Motivation, Dijkstra's algorithm on distance graphs, Bellman-Ford Algorithm, Floyd-Warshall Algorithm

[Ottman/Widmayer, Kap. 9.5 Cormen et al, Kap. 24.1-24.3, 25.2-25.3]

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? [46]



[46] 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	

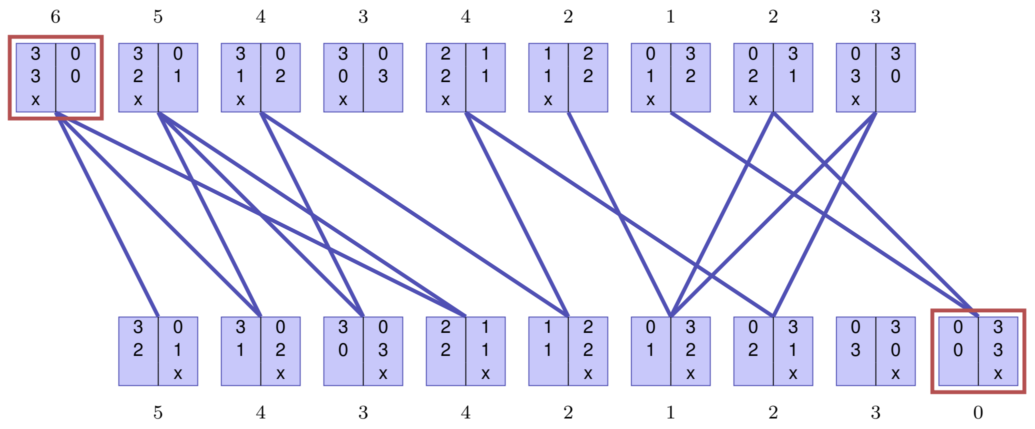
Überfahrt möglich

	links	rechts
Missionare	2	1
Kannibalen	2	1
Boot		x

6 Personen am linken Ufer

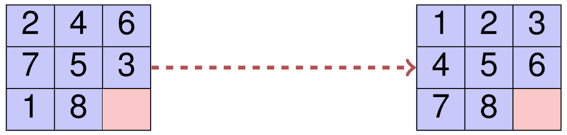
4 Personen am linken Ufer

The whole problem as a graph

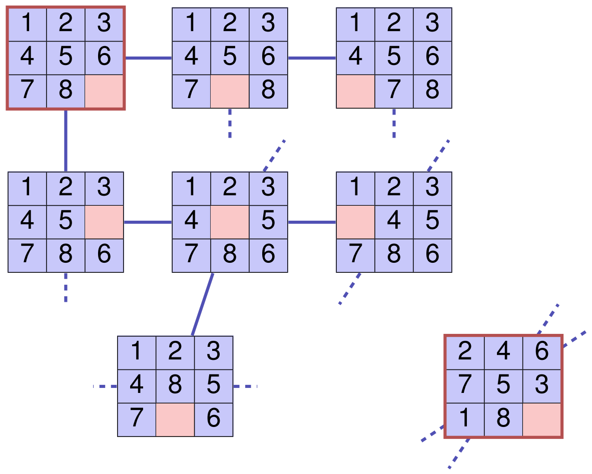


Another Example: Mystic Square

Want to find the fastest solution for

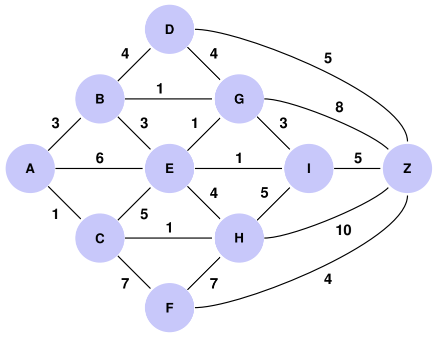


Problem as Graph



Route Finding

Provided cities A - Z and Distances between cities.

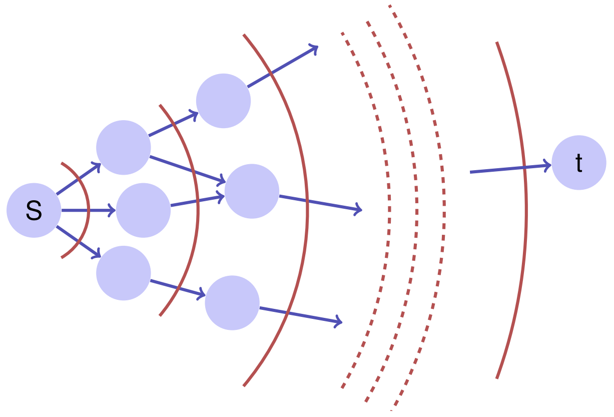


What is the shortest path from A to Z?

Simplest Case

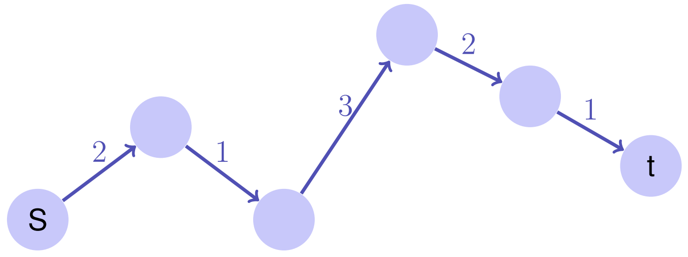
Constant edge weight 1 (wlog)

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



Path with weight 9

Shortest Paths

Notation: we write

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

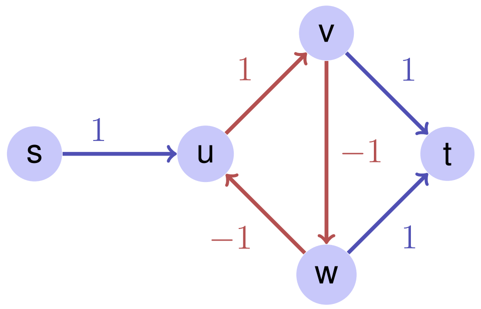
and mean a path p from u to v

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

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

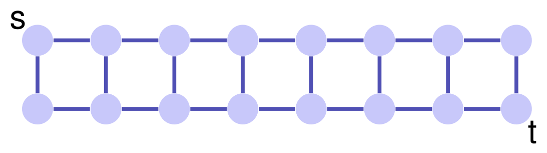
Observations (1)

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)

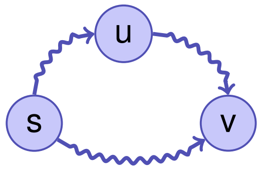
$\Rightarrow$ To try all paths is too inefficient

Observations (3)

Triangle Inequality

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

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

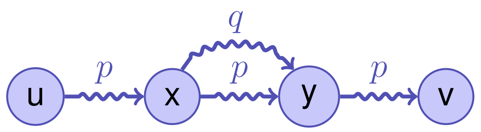


A shortest path from s to v cannot be longer than a shortest path from s to v that has to include u

Observations (4)

Optimal Substructure

Sub-paths of shortest paths are shortest paths. Let $p = \langle v_0, \dots, v_k \rangle$ 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.



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:

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

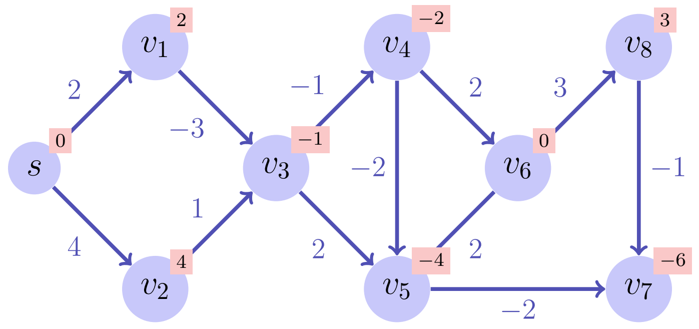
$$\Rightarrow \min\{d_s[v], d_s[u] + c(u, v)\} \geq \delta(s, v)$$

Central Question

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

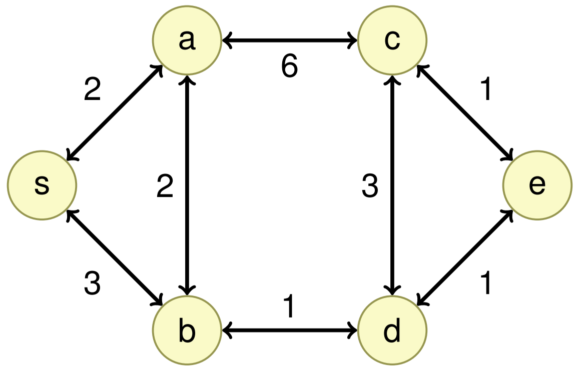
Special Case: Directed Acyclic Graph (DAG)

DAG $\Rightarrow$ topological sorting returns optimal visiting order



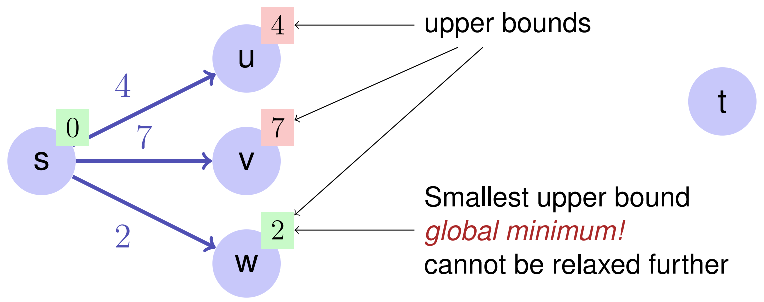
Top. Sort: $\Rightarrow$ Order $s, v_1, v_2, v_3, v_4, v_6, v_5, v_8, v_7$.

Assumption (preliminary)



All weights of G are *positive*.

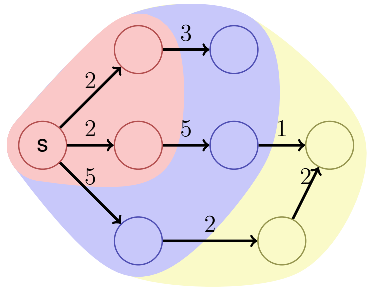
Observation (Dijkstra)



Basic Idea

Set V of nodes is partitioned into

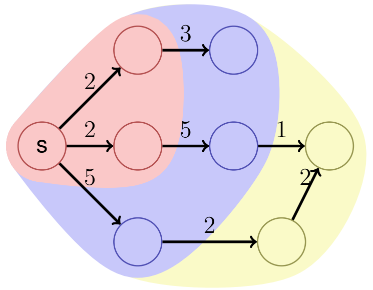
- the set M of nodes for which a shortest path from s is already known,
- the set $R = \bigcup_{v \in M} N^+(v) \setminus M$ of nodes where a shortest path is not yet known but that are accessible directly from M,
- the set $U = V \setminus (M \cup R)$ of nodes that have not yet been considered.



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.

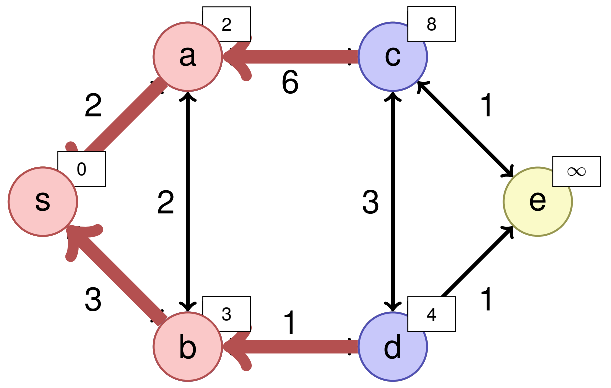


Algorithm Dijkstra(G, s)

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).[47]

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

Runtime

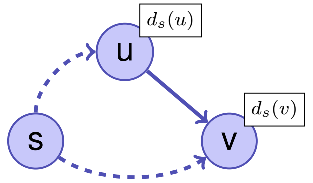
- $|V|\times$ ExtractMin: $\mathcal{O}(|V|\log|V|)$
- $|E|\times$ Insert or DecreaseKey: $\mathcal{O}(|E|\log|V|)$
- $1\times$ Init: $\mathcal{O}(|V|)$
- Overal: $\mathcal{O}(|E|\log|V|)$.

Can be improved when a data structure optimized for ExtractMin and DecreaseKey ist used (Fibonacci Heap), then runtime $\mathcal{O}(|E|+|V|\log|V|)$.

General Weighted Graphs

Relaxing Step as before but with a return value:

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



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

Dynamic Programming Approach (Bellman)

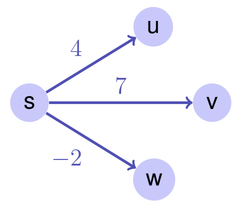
Induction over number of edges $d_s[i, v]$: Shortest path from s to v via maximally i edges.

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

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

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.

```
foreach $u \in V$ do
    $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow$ null
$d_s[s] \leftarrow 0$;
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;
```

All shortest Paths

Compute the weight of a shortest path for each pair of nodes.

- $|V| \times$ Application of Dijkstra's Shortest Path algorithm $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$ (with Fibonacci Heap: $\mathcal{O}(|V|^2 \log |V| + |V| \cdot |E|)$)
- $|V| \times$ Application of Bellman-Ford: $\mathcal{O}(|E| \cdot |V|^2)$
- There are better ways!

Induction via node number[48]

Consider weights of all shortest paths S^k with intermediate nodes in $V^k := \{v_1, \dots, v_k\}$, provided that weights for all shortest paths S^{k-1} with intermediate nodes in V^{k-1} are given.

- v_k no intermediate node of a shortest path of $v_i \rightsquigarrow v_j$ in V^k: Weight of a shortest path $v_i \rightsquigarrow v_j$ in S^{k-1} is then also weight of shortest path in S^k.
- v_k intermediate node of a shortest path $v_i \rightsquigarrow v_j$ in V^k: Sub-paths $v_i \rightsquigarrow v_k$ and $v_k \rightsquigarrow v_j$ contain intermediate nodes only from S^{k-1}.

[48]like for the algorithm of the reflexive transitive closure of Warshall

DP Induction

$d^k(u, v)$ = Minimal weight of a path $u \rightsquigarrow v$ with intermediate nodes in V^k

Induktion

$$d^k(u,v) = \min\{d^{k-1}(u,v), d^{k-1}(u,k) + d^{k-1}(k,v)\} (k \geq 1)$$
$$d^0(u,v) = c(u,v)$$

DP Algorithm Floyd-Warshall(G)

```
**Input:** Acyclic Graph $G = (V, E, c)$
**Output:** Minimal weights of all paths $d$
$d^0 \leftarrow c$
for $k \leftarrow 1$ to $|V|$ do
    for $i \leftarrow 1$ to $|V|$ do
        for $j \leftarrow 1$ to $|V|$ do
            $d^k(v_i, v_j) = \min\{d^{k-1}(v_i, v_j), d^{k-1}(v_i, v_k) + d^{k-1}(v_k, v_j)\}$
```

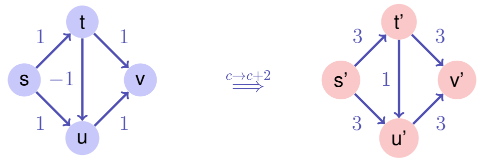
Runtime: $\Theta(|V|^3)$

Remark: Algorithm can be executed with a single matrix d (in place).

Reweighting

Idea: Reweighting the graph in order to apply Dijkstra's algorithm.

The following does *not* work. The graphs are not equivalent in terms of shortest paths.



Reweighting

Other Idea: "Potential" (Height) on the nodes

- $G = (V, E, c)$ a weighted graph.
- Mapping $h : V \to \mathbb{R}$
- New weights

$$\tilde{c}(u, v) = c(u, v) + h(u) - h(v),\ (u, v \in V)$$

Reweighting

Observation: A path p is shortest path in in $G = (V, E, c)$ iff it is shortest path in in $\tilde{G} = (V, E, \tilde{c})$

$$\begin{aligned}\tilde{c}(p) &= \sum_{i=1}^{k} \tilde{c}(v_{i-1}, v_i) = \sum_{i=1}^{k} c(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i) \\ &= h(v_0) - h(v_k) + \sum_{i=1}^{k} c(v_{i-1}, v_i) = c(p) + h(v_0) - h(v_k)\end{aligned}$$

Thus $\tilde{c}(p)$ minimal in all $v_0 \rightsquigarrow v_k \iff c(p)$ minimal in all $v_0 \rightsquigarrow v_k$.

Weights of cycles are invariant: $\tilde{c}(v_0, \dots, v_k = v_0) = c(v_0, \dots, v_k = v_0)$

Johnson's Algorithm

Add a new node $s \notin V$:

$$\begin{aligned} G' &= (V', E', c') \\ V' &= V \cup \{s\} \\ E' &= E \cup \{(s,v) : v \in V\} \\ c'(u,v) &= c(u,v),\ u \neq s \\ c'(s,v) &= 0 (v \in V) \end{aligned}$$

Johnson's Algorithm

If no negative cycles, choose as height function the weight of the shortest paths from s,

$$h(v) = d(s,v).$$

For a minimal weight d of a path the following triangular inequality holds:

$$d(s,v) \leq d(s,u) + c(u,v).$$

Substitution yields $h(v) \leq h(u) + c(u,v)$. Therefore

$$\tilde{c}(u,v) = c(u,v) + h(u) - h(v) \geq 0.$$

Algorithm Johnson(G)

Input: Weighted Graph $G = (V, E, c)$
Output: Minimal weights of all paths D.

```
New node $s$. Compute $G' = (V', E', c')$
if BellmanFord($G'$, $s$) = false then return "graph has negative cycles"
foreach $v \in V'$ do
    $h(v) \leftarrow d(s,v)$ // $d$ aus BellmanFord Algorithmus
foreach $(u,v) \in E'$ do
    $\tilde{c}(u,v) \leftarrow c(u,v) + h(u) - h(v)$
foreach $u \in V$ do
    $\tilde{d}(u,\cdot) \leftarrow$ Dijkstra($\tilde{G}'$, $u$)
    foreach $v \in V$ do
        $D(u,v) \leftarrow \tilde{d}(u,v) + h(v) - h(u)$
```

Analysis

Runtimes

- Computation of G': $\mathcal{O}(|V|)$
- Bellman Ford G': $\mathcal{O}(|V| \cdot |E|)$
- $|V| \times$ Dijkstra $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$
  (with Fibonacci Heap: $\mathcal{O}(|V|^2 \log |V| + |V| \cdot |E|)$)

Overal $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$
($\mathcal{O}(|V|^2 \log |V| + |V| \cdot |E|)$)