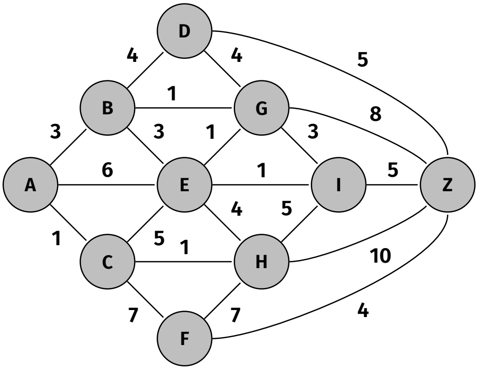
26. Shortest Paths

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

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

Route Finding

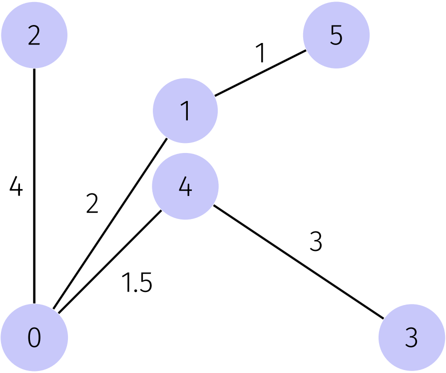
Provided cities A - Z and distances between cities



What is the shortest path from A to Z?

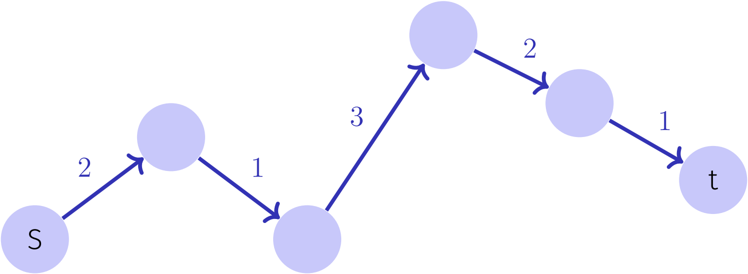
Notation

A **weighted graph** $G = (V, E, c)$ is a graph $G = (V, E)$ with an **edge weight function** $c : E \to \mathbb{R}$. $c(e)$ is called **weight** of the edge e.



Weighted Paths

Given: $G = (V, E, c)$, $c : E \to \mathbb{R}$, $s, t \in V$.
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

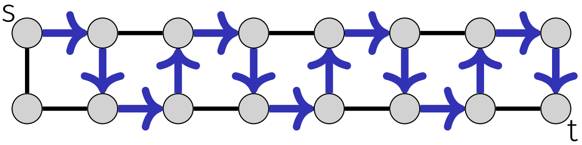
Wanted: $\delta(u, v)$ = minimal weight of a path from u to v:

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

In the following we call a path with minimal weight simply a **shortest path**.

Trivial algorithm?

Try out all paths?

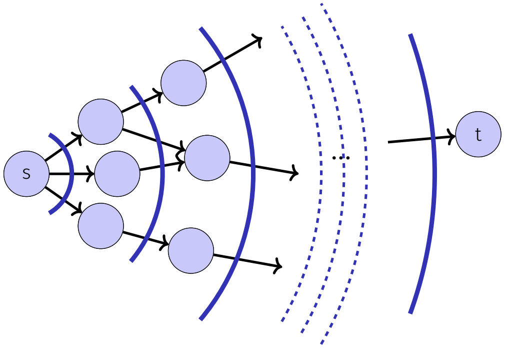


(at least $2^{|V|/2}$ paths from s to t)

$\Rightarrow$ Inefficient. There can be exponentially many paths.

Simplest Case

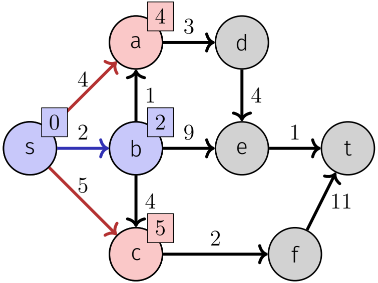
Constant edge weight (every edge has weight 1)

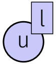


$\Rightarrow$ **Solution:** Breadth First Search $\mathcal{O}(|V| + |E|)$

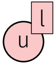
Dijkstra’s Algorithm: Observation

important assumption: all weights are positive.





Shortest path $s \rightsquigarrow u$ has length l (exactly).

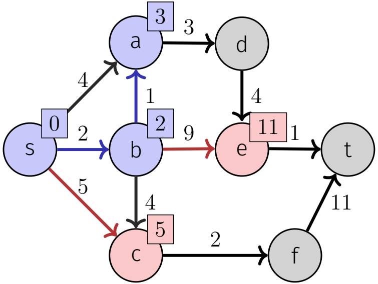


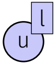
Upper bound:
Shortest path $s \rightsquigarrow u$ has length at most l.

Observation: Shortest outgoing edge (s, u) is the shortest path from s to this node u.

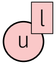
Dijkstra's Algorithm: Observation

important assumption: all weights are positive.





Shortest path $s \rightsquigarrow u$ has length l (exactly).



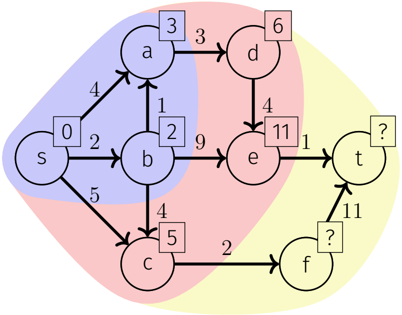
Upper bound:
Shortest path $s \rightsquigarrow u$ has length at most l.

General Observation: The smallest upper bound of a(n orange) node u constitutes the exact length of the shortest path from s to u.

Dijkstra's Algorithm: Basic Idea (Greedy)

V is split into:

- $\mathbf{K}$: nodes with known shortest path
- $\mathbf{N} = \bigcup_{v \in K} N^+(v) \setminus K$: successors of K
  $\Rightarrow$ an upper bound is known
- $\mathbf{R} = V \setminus (K \cup N)$: remaining nodes
  $\Rightarrow$ nothing is known yet



Greedy:

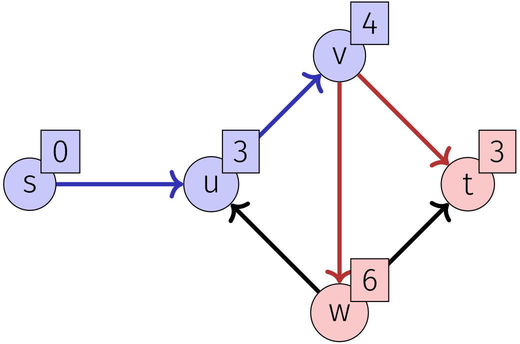
Starting with $\mathbf{N} = \{s\}$, until $\mathbf{N} = \emptyset$: node from $\mathbf{N}$ with smallest upper bound joins $\mathbf{K}$, and its neighbors join $\mathbf{N}$.

Invariants:

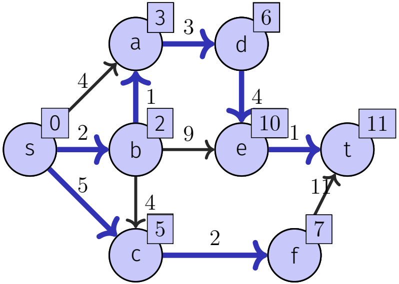
- after i steps: shortest paths to i nodes known ($|K| = i$).
- for all nodes in $\mathbf{v} \in N$: the upper bound is the (exact) length of shortest path $\mathbf{s} \rightsquigarrow \bullet \rightarrow v$ from $\mathbf{s}$ to $\mathbf{v}$ with nodes only from $\mathbf{K} \cup \{\mathbf{v}\}$.

Quiz

Is the following constellation of upper bounds possible?



Example

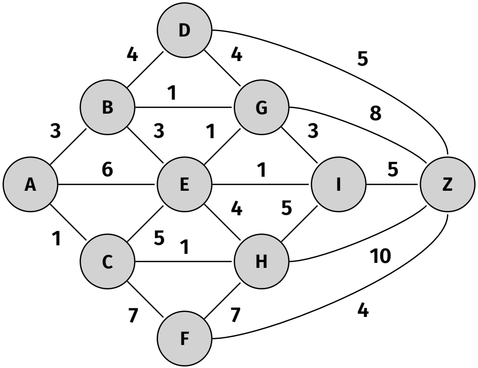


$\mathbf{K} = \{s, b, a, c, d, f, e, t\}$
$\mathbf{N} = \{\}$
$\mathbf{R} = \{\}$

Known shortest paths from s:

$s \rightsquigarrow s$: 0
$s \rightsquigarrow b$: 2
$s \rightsquigarrow a$: 3
$s \rightsquigarrow c$: 5
$s \rightsquigarrow d$: 6
$s \rightsquigarrow f$: 7
$s \rightsquigarrow e$: 10
$s \rightsquigarrow t$: 11

Quiz



Which nodes are in K (known shortest paths) after six steps of Dijkstra's algorithm with starting node A?

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 $v \in V$

Algorithm: Dijkstra(G, s)

Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$,

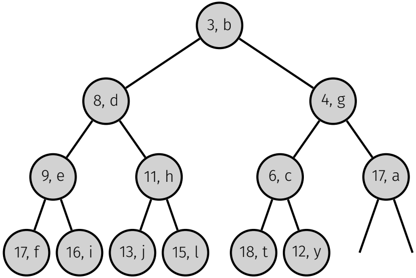
Output: Length d_s of the shortest paths from s and predecessor π_s for each node

```
foreach u ∈ V do
    d_s[u] ← ∞; π_s[u] ← null
d_s[s] ← 0; N ← {s}
while N ≠ ∅ do
    u ← arg min_{u∈N} d_s[u]; N ← N \ {u}
    foreach v ∈ N⁺(u) do
        if d_s[u] + c(u,v) < d_s[v] then
            d_s[v] ← d_s[u] + c(u,v)
            π_s[v] ← u
            N ← N ∪ {v}
```

Implementation: Data Structure for N?

Required operations:

- **`Insert((p, k))`**: $\mathcal{O}(\log |V|)$
  add key (node) k
  with value (upper bound) p
- **`ExtractMin()`**: $\mathcal{O}(\log |V|)$
  remove element with smallest value
- **`DecreaseKey((p, k))`**: $\mathcal{O}(\log |V|)$
  update the value of key k to p

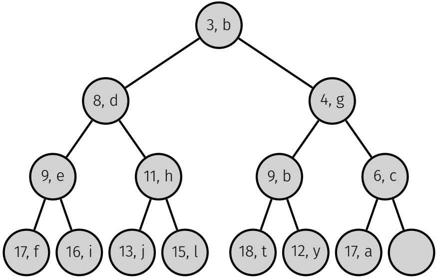


$\Rightarrow$ MinHeap with nodes from N as keys and with upper bounds as value

DecreaseKey

Two possibilities:

- tracking position: store at nodes or external
- or avoid DecreaseKey: with Lazy Deletion



Lazy Deletion:

- Re-insert node with smaller upper bound
- Mark nodes "deleted" once extracted

$\Rightarrow$ Memory consumption of heap can grow to $\Theta(|E|)$ instead of $\Theta(|V|)$

$\Rightarrow$ Because $|E| \leq |V|^2$: Insert and ExtractMin still in $\mathcal{O}(\log |V|^2) = \mathcal{O}(\log |V|)$

Algorithm: Dijkstra(G, s) with Lazy Deletion

Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$,
Output: Length d_s of the shortest paths from s and predecessor π_s for each node

```
foreach u ∈ V do
    d_s[u] ← ∞;  π_s[u] ← null
K = {}; d_s[s] ← 0; N ← {s}
while N ≠ ∅ do
    d, u ← ExtractMin(N)
    if u ∉ K then
        K ← K ∪ {u}
        foreach v ∈ N⁺(u) do
            if d + c(u, v) < d_s[v] then
                d_s[v] ← d + c(u, v); π_s[v] ← u
                Insert((d + c(u, v), v))
```

Running time:
Initialization: $\mathcal{O}(|V|)$
$(|V| + |E|)$ times ExtractMin: $\mathcal{O}((|V| + |E|) \cdot \log |V|)$;
$(|E| + 1)$ times Insert: $\mathcal{O}(|E| \cdot \log |V|)$;
$\Rightarrow$ Overall: $\mathcal{O}((|V| + |E|) \cdot \log |V|)$

Runtime of Dijkstra (without Lazy Deletion)

- $|V|\times$ ExtractMin: $\mathcal{O}(|V|\log|V|)$
- $|E|\times$ Insert or DecreaseKey: $\mathcal{O}(|E|\log|V|)$
- $1\times$ Init: $\mathcal{O}(|V|)$
- Overal[39] [40] :

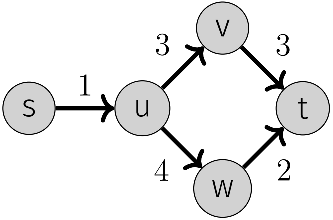
$$\mathcal{O}((|V|+|E|)\log|V|)$$

[39] For connected graphs: $\mathcal{O}(|E|\log|V|)$

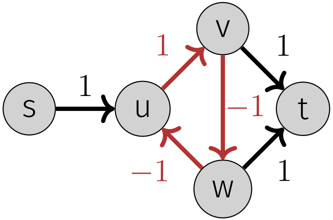
[40] Can be improved when a data structure optimized for ExtractMin and DecreaseKey ist used (Fibonacci Heap), then runtime $\mathcal{O}(|E|+|V|\log|V|)$.

Observations

- Is the shortest path always unique? No!



  Dijkstra's algorithm finds one (any) shortest path.
- Is there always at least one shortest path? No! Negative cycles.



26.3 General Algorithm

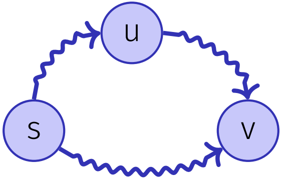
Why Dijkstra is correct and how to generalize.

Observations (1)

Triangle Inequality

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

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

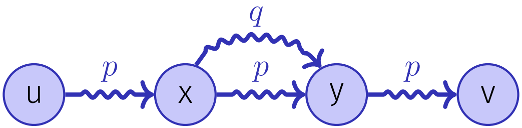


A shortest path from s to v cannot be longer than a shortest path from s to v that has to include u

Observations (2)

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 $p_{ij} = \langle v_i, \ldots, v_j \rangle$ $(0 \le i < j \le k)$ 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 (3)

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).

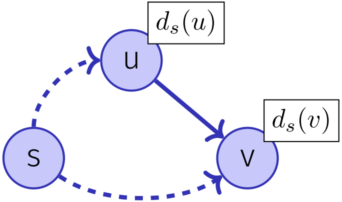
General Algorithm

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 $(u, v) \in E$

Relax(u, v):

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



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:

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

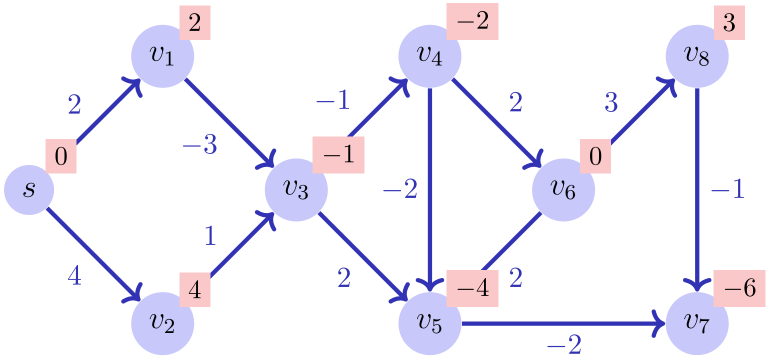
$$\Rightarrow \min\{d_s[v], d_s[u] + c(u,v)\} \geq \delta(s,v)$$

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 $s, v_1, v_2, v_3, v_4, v_6, v_5, v_8, v_7$.

Other Cases

- Special case: $c \equiv = 1 \Rightarrow$ BFS
- Special Case: Positive Edge Weights $\Rightarrow$ Dijkstra ☺.
- General Weighted Graphs: cycles with negative weights can shorten the path, a shortest path is not guaranteed to exist.

Dynamic Programming Approach (Bellman)

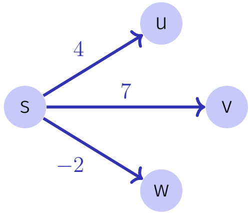
Induction over number of edges $d_s[i,v]$: Shortest path from s to v via maximally i edges.

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

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

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**
    $f \leftarrow f \vee \text{Relax}(u, v)$
  if $f =$ false **then return** true
return false;

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