Graphs Algorithms - Shortest Paths

CS313E - Elements of Software Design

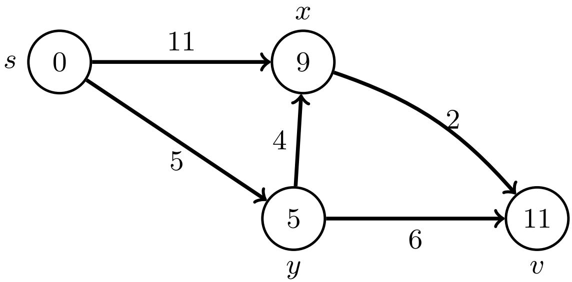
Kia Teymourian

05/11/2022

Agenda

Shortest Paths

- An example application of it is to find the shortest way to drive from A to B (for example using a navigation system).
- We have weighted Graph $G(V, E)$ with weights on edges $W : E \to \mathbb{R}$



Shortest Paths

- A weighted Graph $\boldsymbol{G(V, E)}$
- With weights on edges $\boldsymbol{W : E \rightarrow \mathbb{R}}$

We learn about two algorithms for Single-Source Shortest-Paths Problem:

- **Dijkstra** $\boldsymbol{O(V lg(V) + E)}$ which assumes **none-negative edge weights**.
- **Bellman Ford** $\boldsymbol{O(VE)}$ which is a general algorithm.

Shortest Path Model

Our model as weighted graph $G(V, E)$, $W : E \to \mathbb{R}$

- **V** are vertices (For example all street intersections)
- **E** edges , directed edges are for example one-way streets
- $\boldsymbol{W(U, V)}$ weight of edges from u to v

$$\textbf{Path } p = \langle v_0, v_1, \ldots, v_k \rangle$$

$$(v_i, v_{i+1}) \in E \textbf{ for } 0 \leq i < k$$

$$w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

Single Source Shortest Paths

- Given a graph $\boldsymbol{G = (G, V)}$, weights w and a start source vertex S,
- Find $\boldsymbol{\delta(S, V)}$ (the best path) from S to each vertex $v \in V$.

$$
\begin{aligned}
\boldsymbol{d[v]} &= \begin{cases} \mathbf{0} & \textbf{if } \boldsymbol{v = s} \\ \boldsymbol{\infty} & \textbf{otherwise} \end{cases} \\
&= \boldsymbol{\delta(s, v)} \\
\boldsymbol{d[v]} &\geq \boldsymbol{\delta(s, v)} \ \textbf{always}
\end{aligned}
$$

- When we find a better path from S to v, $\boldsymbol{d[v]}$ **will decrease** and show that there is better path available.
- The **predecessor on the best path to v** is given by $\boldsymbol{\Pi[v]}$, initially we have $\Pi[v] = NIL$

Initial State for the Single Source Shortest Path

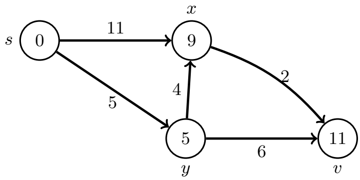
- All vertices $\boldsymbol{v.d = \infty}$
- All vertices $\boldsymbol{v.\pi = NILL}$
- Distance to source is zero $\boldsymbol{s.d = 0}$

Relaxing an Edge

Relaxing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $v.d$ and $v.\pi$.

Algorithm 1 RELAX(u,v,w)

if $v.d > u.d + w(u, v)$ **then**
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
end if

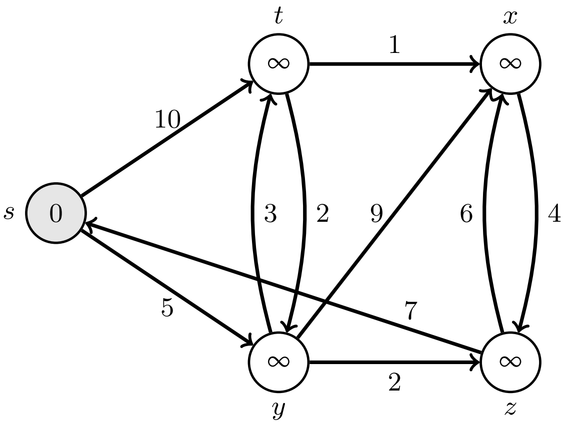


Dijkstra's Algorithm

Dijkstra's Algorithm searches for each edge $(u, v) \in E$.

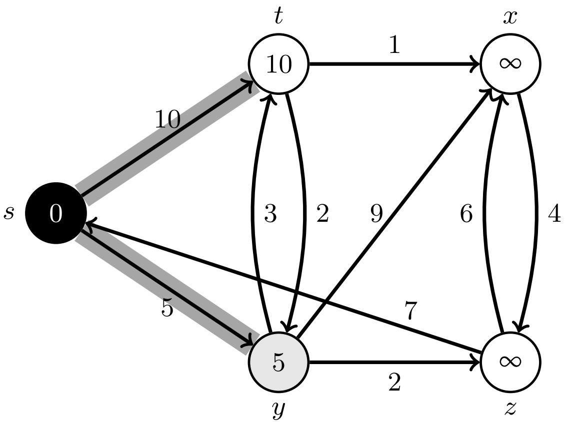
- It assumes assume $w(u, v) \geq 0$ weights are positives.
- It maintains a set S of vertices whose final shortest path weights have been determined.
- It repeatedly selects $u \in (V - S)$ with the **minimum shortest path** estimate, and add u to S, then relaxes all edges out of u.
- The main strategy of **Dijkstra is a greedy algorithm**. Repeatedly choose closes vertex $u \in (V - S)$ to add it into the set S

Example Execution of Dijkstra's Algorithm - Step-1 Start/Initialization



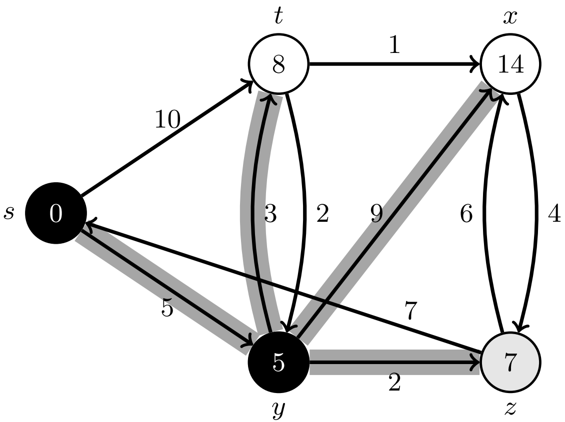
Step	s	t	y	x	z
1	0	∞	∞	∞	∞

Example Execution of Dijkstra's Algorithm, Step-2



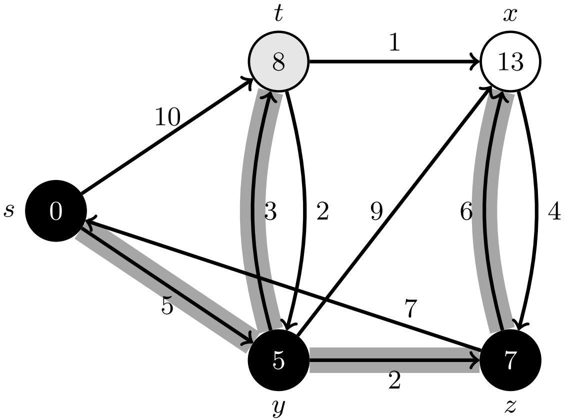
Step	**s**	**t**	**y**	**x**	**z**
2	0	10	5	∞	∞

Example Execution of Dijkstra's Algorithm, Step-3



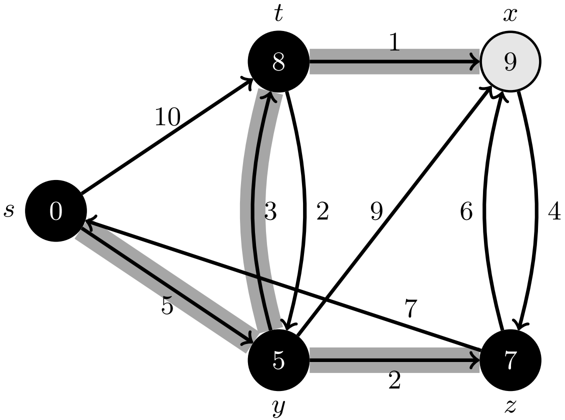
Step	s	t	y	x	z
3	0	8	5	14	7

Example Execution of Dijkstra's Algorithm, Step-4



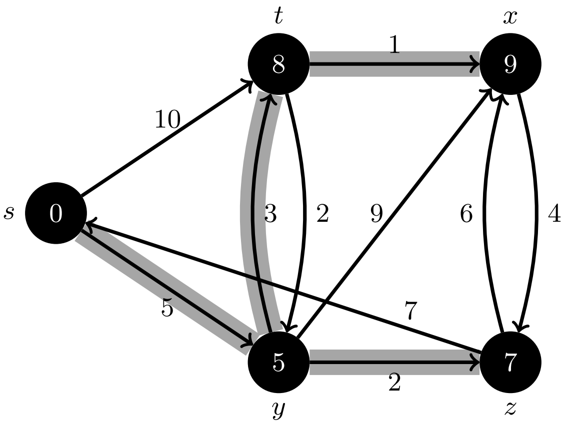
Step	s	t	y	x	z
4	0	8	5	13	7

Example Execution of Dijkstra's Algorithm, Step-5



Step	**s**	**t**	**y**	**x**	**z**
5	0	8	5	9	7

Example Execution of Dijkstra's Algorithm, Step-6



Step	**s**	**t**	**y**	**x**	**z**
6	0	8	5	9	7

Algorithm 2 $DIJKSTRA(G, w, s)$

▷ Uses a priority queue

$INITIALIZE-SINGLE-SOURCE(G, s)$ ▷ Initialize it
$S = \varnothing$
$Q = G.V$ ▷ Insert into prioroty queue
while $Q \neq \varnothing$ **do**
    $u = EXTRACT-MIN(Q)$ ▷ Deletes u from Q
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$ **do**
        $RELAX(u, v, w)$ ▷ This will descrease distances
    end for
end while

Dijkstra Complexity

So we have the following costs:

- $\Theta(V)$ inserts into priority queue.
- $\Theta(V)$ EXTRACT-MIN operations
- $\Theta(E)$ DECREASE-KEY operations inside RELAX operation

Dijkstra Complexity - Implementation

Using Array

An simple Array implementation of Dijkstra Algorithm would cause the following costs:

- $\Theta(V)$ times for EXTRACT-MIN
- $\Theta(1)$ for decrease each key

Sum: $\Theta(V \times V + E \times 1) = \Theta(V^2 + E) = \Theta(V^2)$

Heap Structure

Using a Heap Structure and extract-mean of Heap.

- $\Theta(lg(V))$ times for EXTRACT-MIN
- $\Theta(lg(V))$ for decrease each key

Sum: $\Theta(V \times lg(V) + E \times lg(V)) = \Theta(V lg(V) + E lg(V))$

You can improve it when you use **Fibonacci heap to $\boldsymbol{\Theta(V lg(V) + E)}$**

Dijkstra's Algorithm Visualization

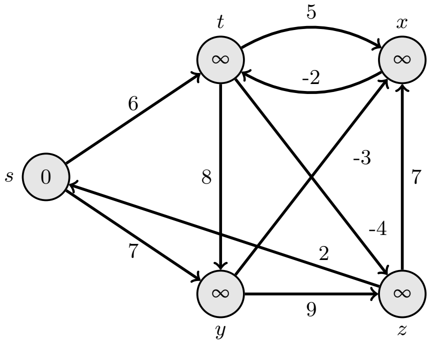
- https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html
- https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_en.html
- https://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/dijkstra.html

Bellman-Ford algorithm

Single-Source Shortest Paths (SSSP) problem

The Bellman-Ford algorithm

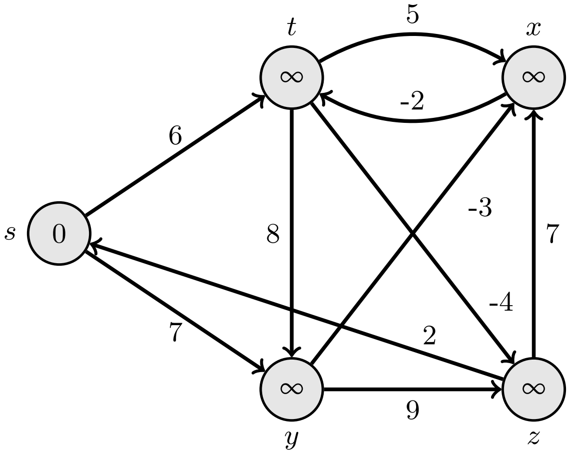
- Bellman-Fort is a more general algorithm for single-source shortest path problems.
- The weights can be negative.
- You can proof that if $G = (V, E)$ that contains no negative weight cycles, then after Bellman-Ford algorithm execution you have $d[v] = \delta(s, v)$ for all $v \in V$.



Algorithm 3 BELLMAN-FORD

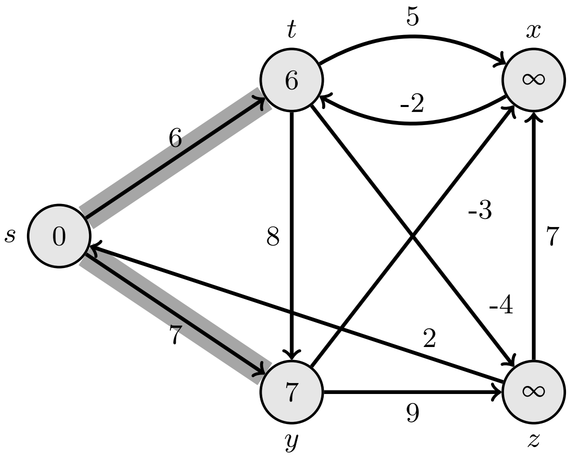
```
INITIALIZE − SINGLE − SOURCE(G, s)
for i = 1 to |G.V| − 1 do                ▷ Iterate over all vertices (none start vertex)
    for each edge (u, v) ∈ G.E do        ▷ Iterate over all edges
        RELAX(u, v, w)                   ▷ Relax the edge
    end for
end for
for each edge (u, v) ∈ G.E do            ▷ Iterate over all edges for a check
    if v.d > u.d + w(u, v) then
        return FALSE                     ▷ If so report that a negative-weight cycle exists
    end if
end for
return TRUE
```

Example Execution of Bellman-Ford algorithm - Initialization



s	t	y	x	z
0	∞	∞	∞	∞

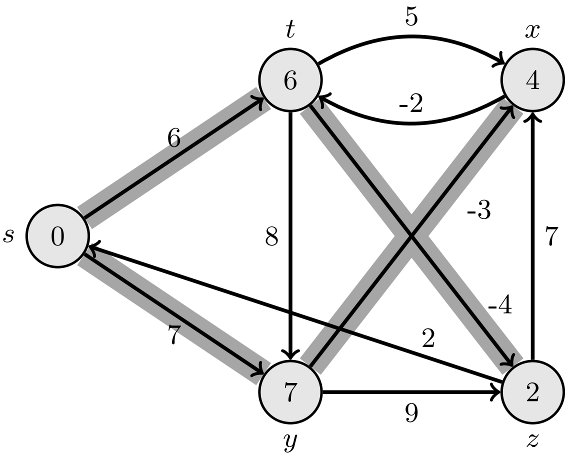
Example Execution of Bellman-Ford algorithm



s	t	y	x	z
0	6	7	∞	∞

- ▷ Situation after 1st successive pass over the edges
- ▷ Shaded edges indicate predecessor values

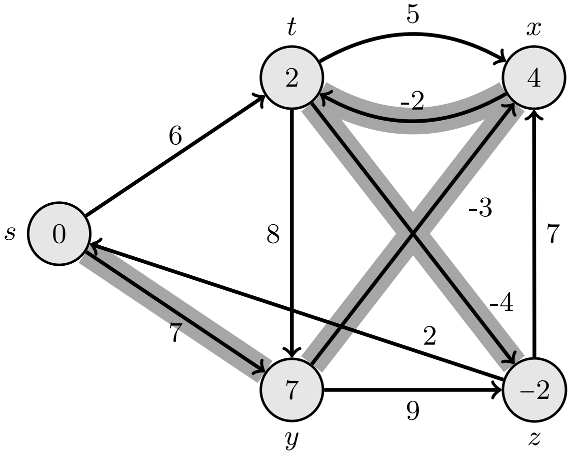
Example Execution of Bellman-Ford algorithm



s	t	y	x	z
0	6	7	4	-2

- Situation after 2nd successive pass over the edges
- Shaded edges indicate predecessor values

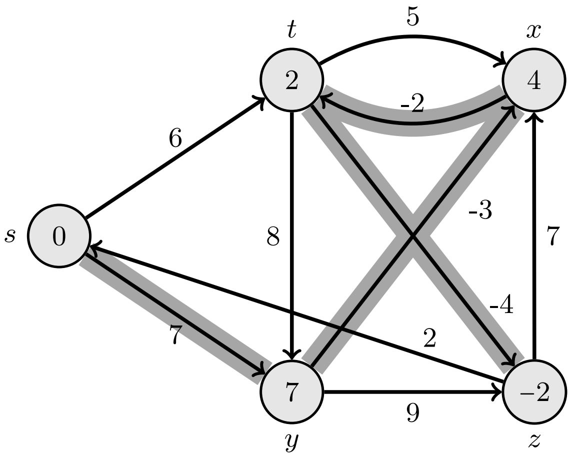
Example Execution of Bellman-Ford algorithm



s	t	y	x	z
0	2	7	4	-2

- ▷ Situation after 3rd successive pass over the edges
- ▷ Shaded edges indicate predecessor values

Example Execution of Bellman-Ford algorithm



s	t	y	x	z
0	2	7	4	- 2

- Situation after 4th successive pass over the edges
- Shaded edges indicate predecessor values

Bellman-Ford algorithm Complexity

So we have the following costs:

- The Bellman-Ford algorithm runs in time $\Theta(V)$ for initialization
- Each of the $|V| - 1$ passes over the edges takes $\Theta(E)$
- And for the loop lines at the end to check, takes $O(E)$

The Bellman-Ford algorithm runs in time $O(VE)$.

Bellman-Ford Algorithm Visualization

- https://www-m9.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html
- https://visualgo.net/en/sssp

Readings from CLRS Book (Introduction to Algorithms, 3rd Edition)

- Chapter 22 Graphs
- Chapter 24 Single-Source Shortest Paths
- Section 24.1 The Bellman-Ford algorithm
- Section 24.3 Dijkstra's algorithm