Dynamic Programming

- Single Source Shortest Paths Problem
- Dynamic Programming Formulation
- Bellman-Ford Algorithm

Imdad ullah Khan

SSSP Problem

Input: A weighted graph G and a source vertex $s \in V$
Output: Shortest paths from s to all vertices $v \in V$

This assumes no negative weight cycle in G

$\triangleright$ Actually, no negative cycle reachable from the source vertex s is sufficient

Input: A weighted graph G and a source vertex $s \in V$
Output: Shortest paths from s to all $v \in V$ or output a negative cycle

SSSP Problem: Dynamic Programming Formulation

Input: A weighted graph G and a source vertex $s \in V$
Output: Shortest paths from s to all $v \in V$ or output a negative cycle

$\textsc{opt}(v, i)$: minimum weight of a $s - v$ path that uses at most i edges

$$\textsc{opt}(v, i) = \min \begin{cases} \infty & \text{if } i = 0 \wedge s \neq v \\ 0 & \text{if } i = 0 \wedge s = v \\ \textsc{opt}(v, i-1) & \\ \min\limits_{u \in N(v)} \textsc{opt}(u, i-1) + w(uv) & \end{cases}$$

Maintain a table of solution for each v and $1 \leq i \leq n$ (memo)

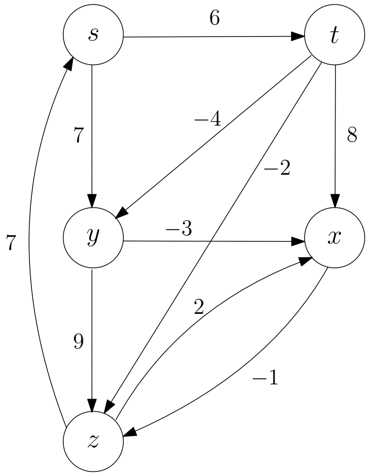
A negative cycle is detected if $\textsc{opt}(v, n)$ improves upon $\textsc{opt}(v, n-1)$

SSSP Problem: Bellman Ford Algorithm

Algorithm 1 Bellman Ford Algorithm (G, s)

```
for each vertex v in G do
    d[v] ← ∞                          ▷ d[v] maintains OPT(v, ·)
    p[v] ← nil      ▷ p[v] is the predecessor of v on shortest path from s
    d[s] ← 0                          ▷ The distance from s to itself is zero
for i = 1 to |V| − 1 do               ▷ Relax all the edges |V| − 1 time
    for each edge (u, v) in G do
        if d[v] > d[u] + w(u, v) then  ▷ If the distance can be improved
            d[v] ← d[u] + w(u, v)      ▷ Update the distance
            p[v] ← u                   ▷ Update the predecessor
for each edge (u, v) in G do
    if d[v] > d[u] + w(u, v) then      ▷ If the distance can be improved
        return "Negative cycle detected – The problem has no solution"
return d, p
```

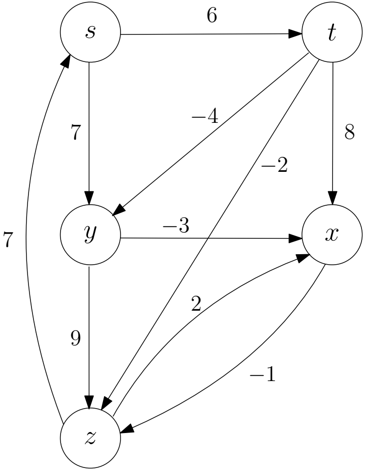
SSSP Problem: Bellman Ford Algorithm



After Initialization

v	$d[v]$	$\pi[v]$
s	0	NIL
t	∞	NIL
x	∞	NIL
y	∞	NIL
z	∞	NIL

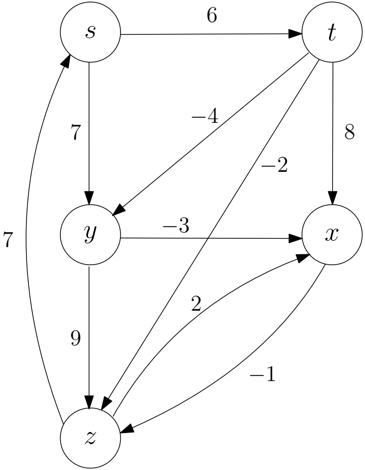
SSSP Problem: Bellman Ford Algorithm



After the first relaxation iteration

v	$d[v]$	$\pi[v]$
s	0	NIL
t	6	s
x	−1	y
y	2	t
z	4	t

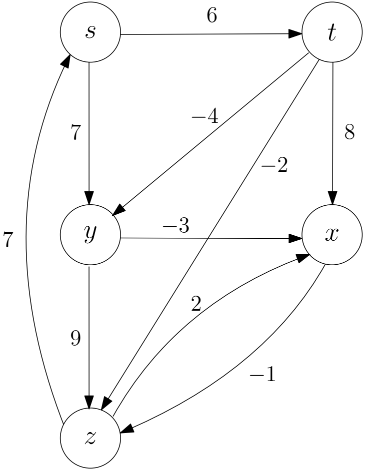
SSSP Problem: Bellman Ford Algorithm



After the second relaxation iteration

v	$d[v]$	$\pi[v]$
s	0	NIL
t	6	s
x	-1	y
y	2	t
z	-2	x

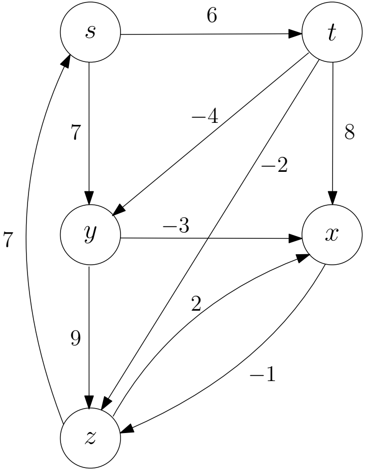
SSSP Problem: Bellman Ford Algorithm



After the third relaxation iteration

v	$d[v]$	$\pi[v]$
s	0	NIL
t	6	y
x	−1	y
y	2	t
z	−2	t

SSSP Problem: Bellman Ford Algorithm

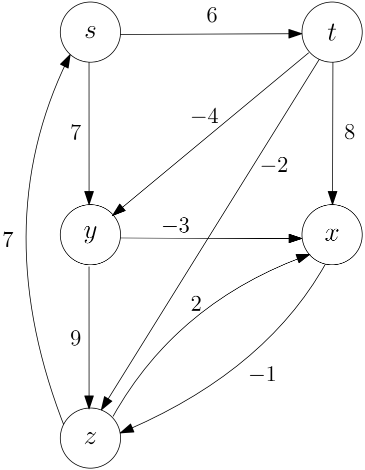


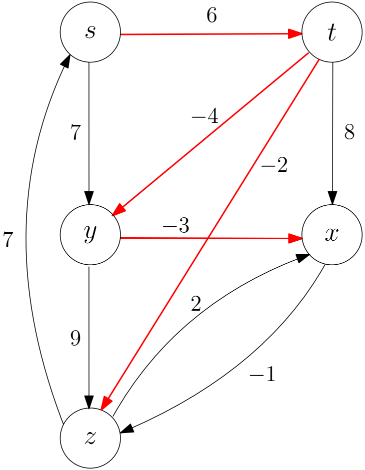
After the fourth (final) relaxation iteration

v	$d[v]$	$\pi[v]$
s	0	NIL
t	6	s
x	−1	y
y	2	t
z	−2	x

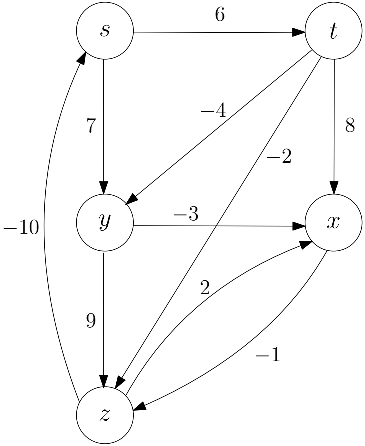
The fifth relaxation does not change it

SSSP Problem: Bellman Ford Algorithm





SSSP Problem: Bellman Ford Algorithm

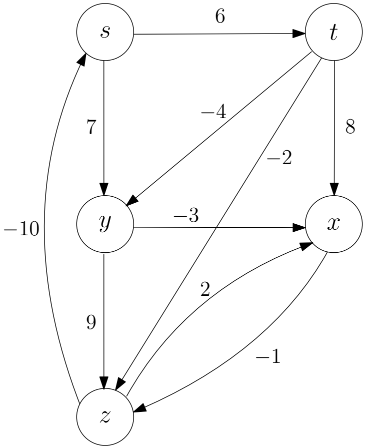


$w(\langle s, t, z, s \rangle) = -6$

In 4 edge relaxations we get

v	$d[v]$	$\pi[v]$
s	-24	z
t	-12	s
x	-19	y
y	-16	t
z	-14	t

SSSP Problem: Bellman Ford Algorithm



$w(\langle s, t, z, s \rangle) = -6$

After fifth edge relaxations we can still improve using the edge (z, s)

v	$d[v]$	$\pi[v]$
s	-30	z
t	-18	s
x	-25	y
y	-22	t
z	-20	t

Bellman-Ford Algorithm Finding Negative Cycle

- To find the negative cycle, trace back the predecessor chain from the vertex whose $d[\cdot]$ decreased in the last step, until we reach a vertex that has already been seen in the chain
- e.g. starting from s, we have the following predecessor chain: $s, x, z, x, \ldots$
- We stop when we see x for the second time, and we output the cycle $\langle x, z, s, x \rangle$
- Note this cycle is a subcycle of the original negative cycle in the graph

Bellman-Ford Algorithm Finding Negative Cycle

- $O(n^2)$ subproblems
- $O(deg(v))$ time to compute each entry of the table

Runtime is $O(n \cdot \sum_v deg(v)) = O(nm)$

The algorithm uses $O(n)$ space to store the distance and predecessor arrays