CS583 Lecture 08

Single Source Shortest Path

Jyh-Ming Lien

some materials here are based on Prof. Shehu, and Prof. Wang's past lecture notes

Path Weight

- ***Weight of path*** $p = \langle v_0, v_1, \ldots, v_k \rangle$

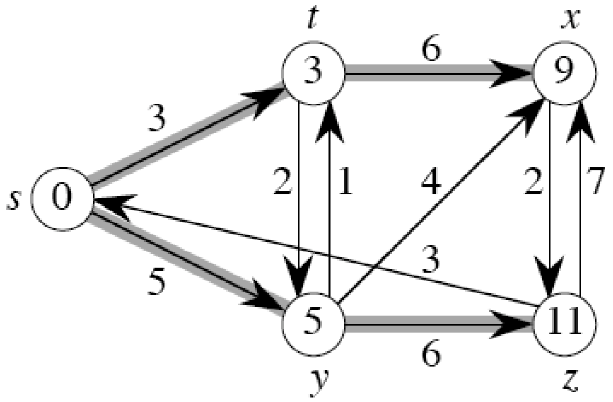
$$\begin{aligned} &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ &= \text{sum of edge weights on path } p \ . \end{aligned}$$

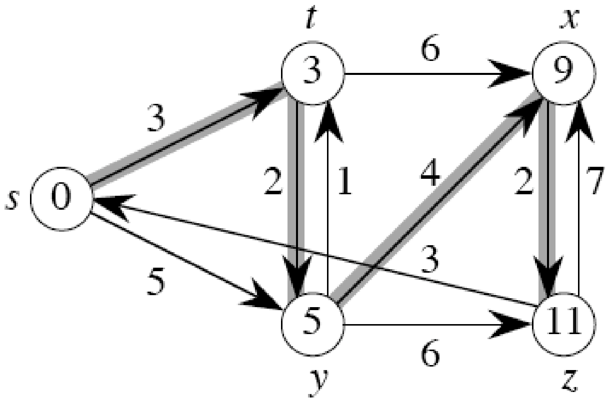
- ***Shortest-path weight*** u to v:

$$\delta(u, v) = \begin{cases} \min \left\{ w(p) : u \overset{p}{\leadsto} v \right\} & \text{if there exists a path } u \leadsto v \ , \\ \infty & \text{otherwise} \ . \end{cases}$$

Shortest Path

- Shortest path u to v is any path p such that $w(p) = \delta(u, v)$.
- may not be unique





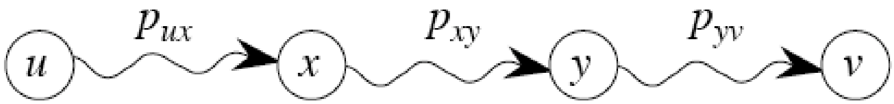
- the shortest paths from one vertex to all the other vertices form a **tree**

Shortest Path

- Variants
  - single source
  - single destination
  - single pair
  - all pairs (next week)
- negative edge weight
  - assume there are no negative cycles

Optimal Structure

- Any subpath of a shortest path is a shortest path



  - proof

Cycles

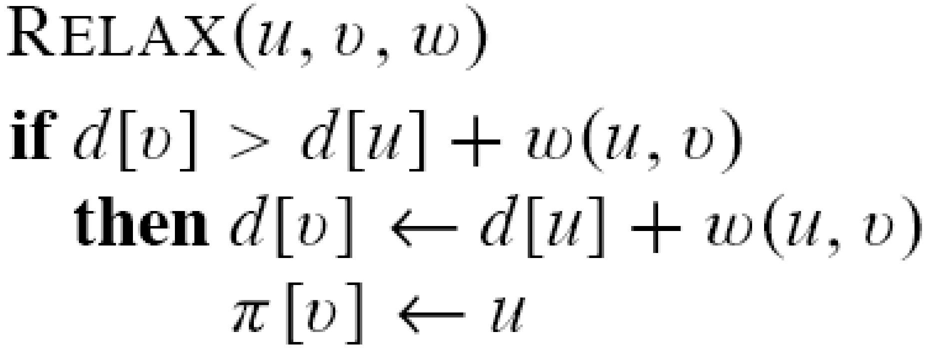
- Shortest path cannot contain cycles (assume that no negative cycles)
    - proof

Shortest Path

- output of the shortest path algorithm for each vertex *v* from source *s*
  - d[*v*], distance from *s*
    - this is initially $+\infty$
    - reduces as algorithm progress
  - π[*v*], the predecessor of *v* on a shortest path from *s*
    - this is initially null

The RELAX function

- we can alway improve the value of d[*v*] for *v* by **relaxing** an edge (*u*,*v*)



```
RELAX($u, v, w$)
if $d[v] > d[u] + w(u, v)$
  then $d[v] \leftarrow d[u] + w(u, v)$
       $\pi[v] \leftarrow u$
```

- RELAX will never hurt your solution!

A Framework

- A framework for single-source shortest paths (SSSP) algorithms
  - make all d[*v*]= ∞ and π[*v*]=null
  - call RELAX
- The algorithms differ in
  - the order RELAX is called
  - the number of times RELAX is called

Important Properties

- triangle inequality $\delta(s, v) \leq \delta(s, u) + w(u, v)$
- upper-bound property

Always have $d[v] \geq \delta(s, v)$ for all v. Once $d[v] = \delta(s, v)$, it never changes.

- no-path property If $\delta(s, v) = \infty$, then $d[v] = \infty$ always.
- convergence property

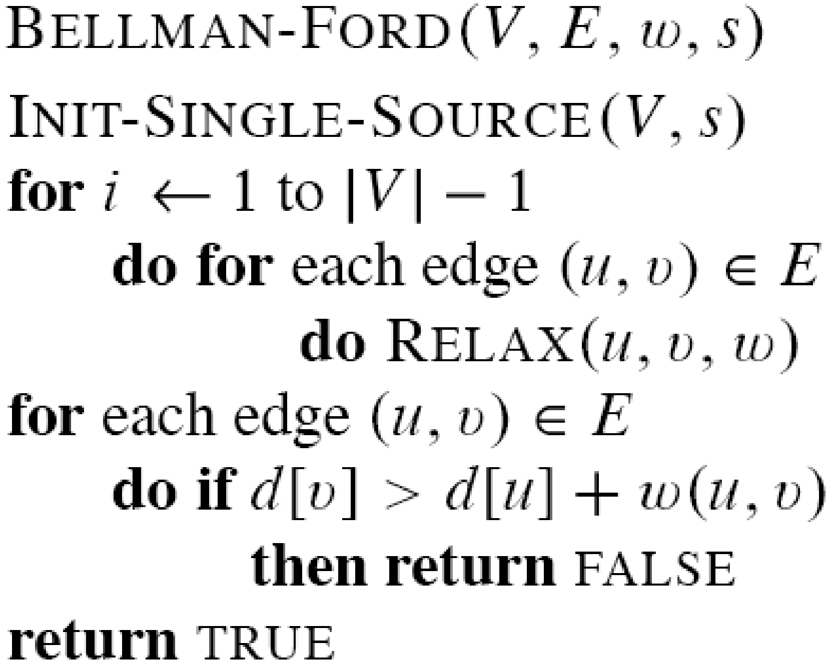
if $d[u] = \delta(s, u)$, then after RELAX(u,v,w), $d[v] = \delta(s, v)$

- path relaxation property

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k. If we relax, *in order*, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$.

bellman-ford algorithm

- allow negative edge weights
- can detect negative cycles



```
BELLMAN-FORD($V, E, w, s$)
INIT-SINGLE-SOURCE($V, s$)
for $i \leftarrow 1$ to $|V| - 1$
    do for each edge $(u, v) \in E$
        do RELAX($u, v, w$)
for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
        then return FALSE
return TRUE
```

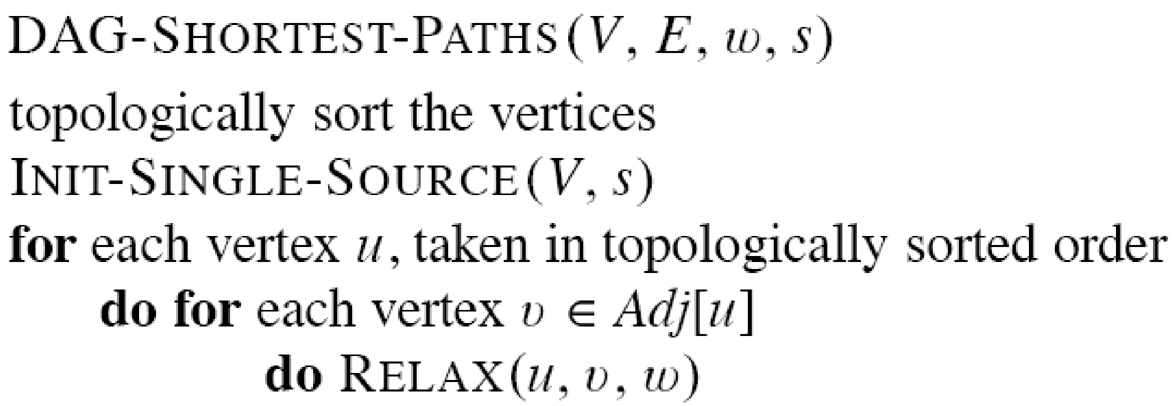
bellman-ford algorithm

- Example

- Time complexity? Why does it work?

DAG

- algorithm



```
DAG-SHORTEST-PATHS($V$, $E$, $w$, $s$)
topologically sort the vertices
INIT-SINGLE-SOURCE($V$, $s$)
for each vertex $u$, taken in topologically sorted order
    do for each vertex $v \in Adj[u]$
           do RELAX($u$, $v$, $w$)
```

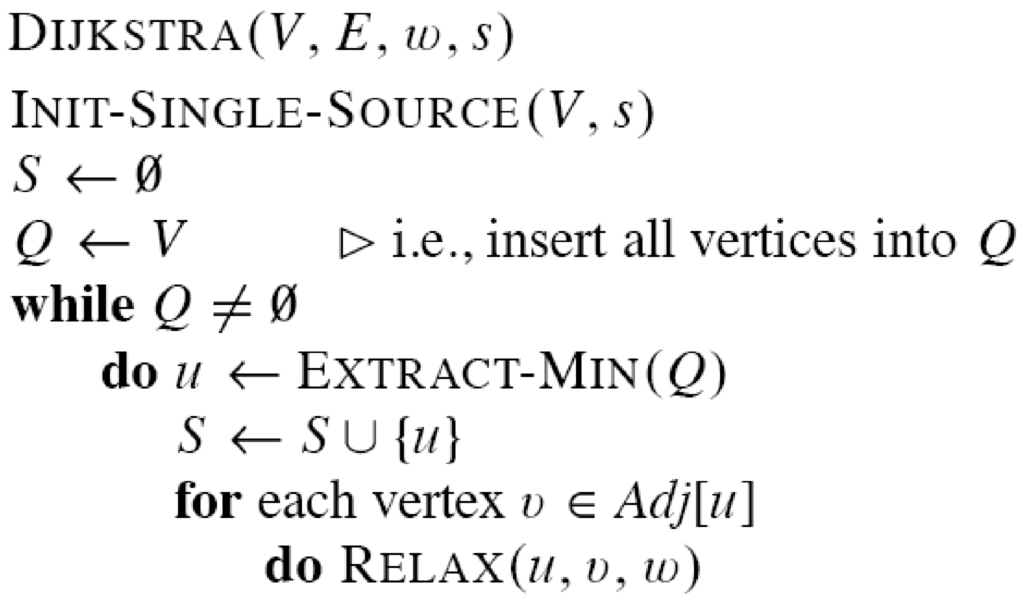
- Example:
- Time complexity? Why does it work?

Dijkstra’s Algorithm

- no negative weights
- a weighted version of BFS
    - instead of a queue, uses a priority queue
    - keys are d[*v*]
- very similar to Prim’s algorithm
    - greedy
    - iteratively expand the shortest path tree
    - differences?

Dijkstra's algorithm

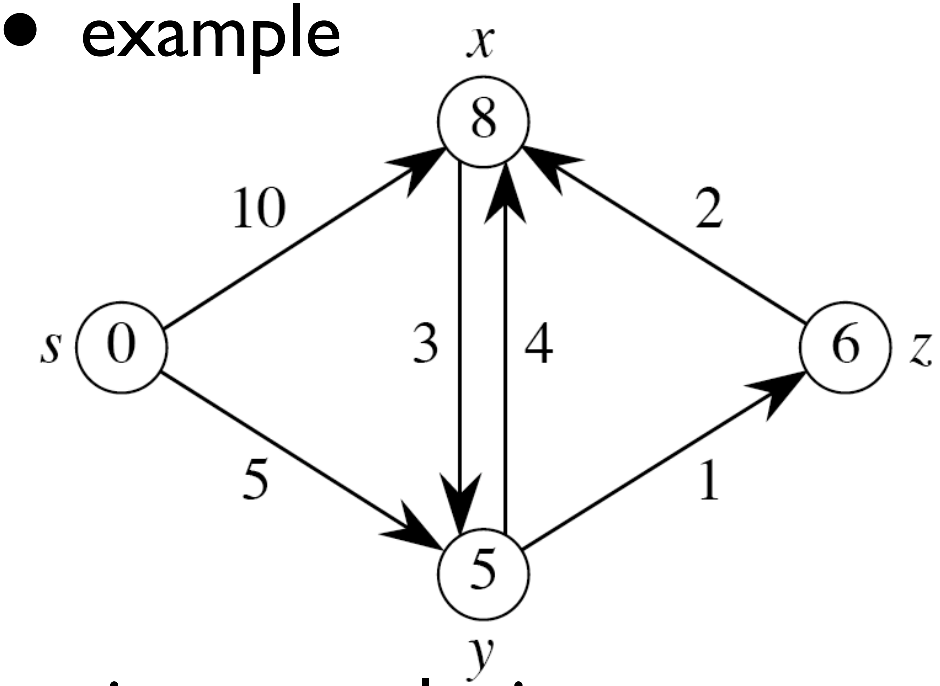
- algorithm



```
DIJKSTRA($V, E, w, s$)
INIT-SINGLE-SOURCE($V, s$)
$S \leftarrow \emptyset$
$Q \leftarrow V$          ▷ i.e., insert all vertices into $Q$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN($Q$)
        $S \leftarrow S \cup \{u\}$
        for each vertex $v \in Adj[u]$
            do RELAX($u, v, w$)
```

Dijkstra's algorithm

- example



- time complexity
- why does it work?

Other Algorithms

- Best-first search
- A* (and D*) search
    - Both best-first search and Dijsktra's algorithm are a special case of A*
- Iterative deepening depth-first search
- We will also look at algorithms that find all-pairs shortest-paths next week