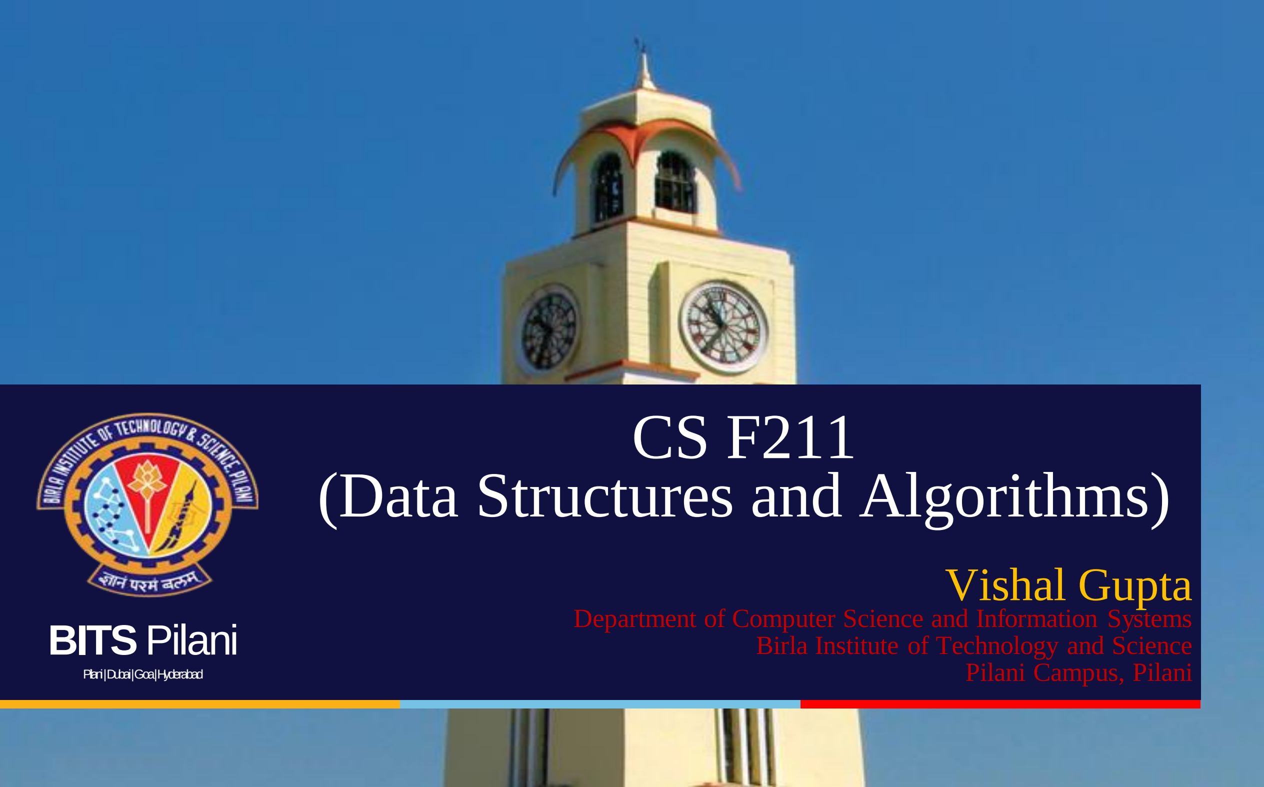


BITS Pilani

Pilani|Dubai|Goa|Hyderabad

CS F211 (Data Structures and Algorithms)

Vishal Gupta
Department of Computer Science and Information Systems
Birla Institute of Technology and Science
Pilani Campus, Pilani

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Agenda: Single Source Shortest Paths

Slides Source: **http://jupiter.math.nctu.edu.tw/~huilan/2011Spring/10Single-Source%20Shortest%20Paths.pdf**

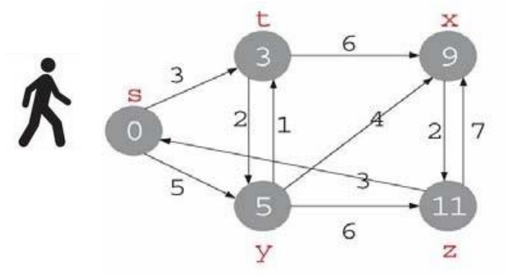
Overview

- Let $G = (V, E)$ be a **weighted directed graph** with weight function w.
- The weight of a path (directed) $p = < v_0, v_1, .., v_k >$ is $\sum_{i=1}^{k} w(v_{i-1}, v_i)$.
- Define

$$\delta(u, v) = \begin{cases} \min\{ w(p) \mid p \text{ is a path from } u \text{ to } v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases} .$$

- A shortest path from u to v is any path p with weight $w(p) = \delta(u, v)$.

- Example.



- The **breadth-first-search algorithm** is a shortest-paths algorithm that works on **unweighted graphs** (or each edge have unit weight).

Variants

- ***Single-source shortest-paths problem***: find a shortest path from a given ***source*** vertex *s* to each vertex $v \in V$.
  Many other problems can be solved by the algorithm for the single-source problem, including the following variants.

- ***Single-destination shortest-paths problem*****:** Find a shortest path to a given ***destination*** vertex t from each vertex v.

- ***Single-pair shortest-path problem*****:** Find a shortest path from u to v for given vertices u and v.
  □No algorithms for this problem are known that run asymptotically faster than the best single-source algorithms in the worst case.

- ***All-pairs shortest-paths problem*****:** Find a shortest path from u to v for every pair of vertices u and v.
  □ It can usually be solved faster (Chapter 25).

Optimal substructure of a shortest path

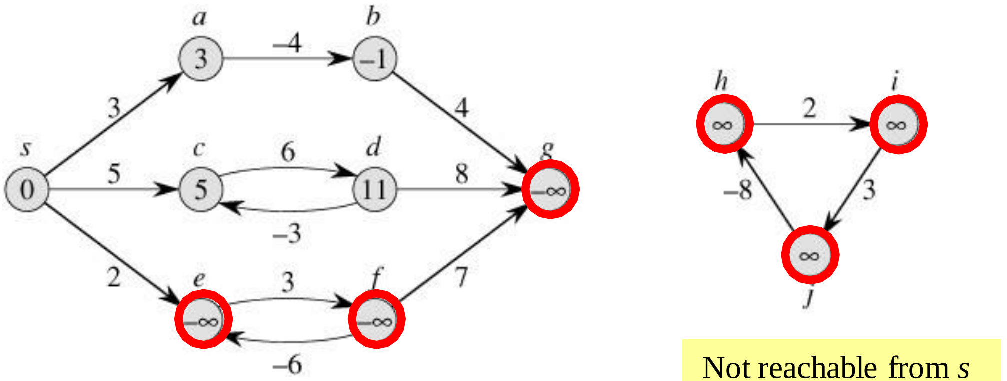
- Optimal-substructure property: **A shortest path between two vertices contains other shortest paths within it.**

Lemma 24.1: (Subpaths of shortest paths are shortest paths)
Given a weighted, directed graph $G = (V, E)$ with weight function w, let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a shortest path from v_1 to v_k, then the subpath $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a shortest path from v_i to v_j.

- **Dijkstra's algorithm** is a greedy algorithm.
- **Floyd-Warshall algorithm** for a***ll-pairs shortest-paths problem*** is a dynamic-programming algorithm.

Negative-weight edges

- ▸ If $G = (V, E)$ contains no negative-weight cycles reachable from the source s, then for all $v \in V$, the shortest-path weight $\delta(s, v)$ remains well defined, even if it has a negative value.
- ▸ If there is a negative-weight cycle on some path from s to v, $\delta(s, v) = -\infty$.



A negative-weight cycle reachable from *s*

Not reachable from *s*

- **Dijkstra's algorithm** assumes that <u>all edge weights are nonnegative</u>. **Bellman-Ford algorithm** allows negative-weight edges and produces a correct answer as long as <u>no negative-weight cycles</u> are reachable from the source.

Cycles

- Can a shortest path contain a cycle

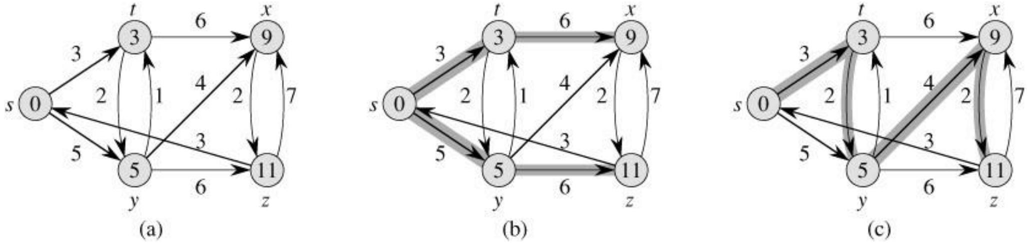
  - It cannot contain a negative-weight cycle.
  - Nor can it contain a positive-weight cycle (it can be removed!).
  - We can remove a 0-weight cycle from any path to produce another path whose weight is the same.

  Therefore, without loss of generality we can assume that when we are finding shortest paths, they have no cycles.

- Since any acyclic path in a graph $G = (V, E)$ contains at most $|V|$ distinct vertices, it also contains at most $|V| - 1$ edges. Thus, we can restrict our attention to shortest paths of at most $|V| - 1$ edges.

Representing shortest paths

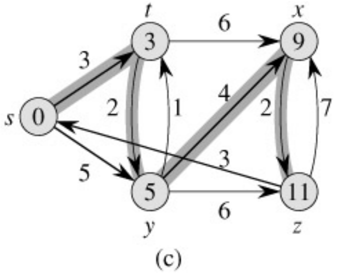
- Besides finding the shortest path weight, we wish to find the vertices on shortest paths.



- Given a graph $G = (V, E)$, $\pi[v]$ denotes the ***predecessor*** of $v \in V$ (as BFS does!). **PRINT-PATH(*G, s, v*)** can print a shortest path from *s* to *v*. Suppose $\pi[v]$ for every *v*.

```
PRINT-PATH($G$, $s$, $v$)
1   if $v = s$
2       then print $s$
3       else if $\pi[v]$ = NIL
4               then print "no path from" $s$ "to" $v$ "exists"
5               else PRINT-PATH($G$, $s$, $\pi[v]$)
6                    print $v$
```

Predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ induced by the π values, where $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$ and $E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$.



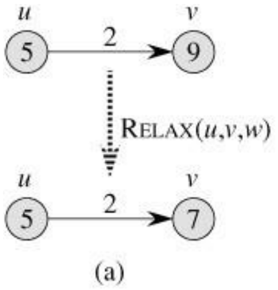
- A ***shortest-paths tree*** rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that
  1. V' is the set of vertices reachable from s in G,
  2. G' forms a rooted tree with root s, and
  3. for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.

Relaxation

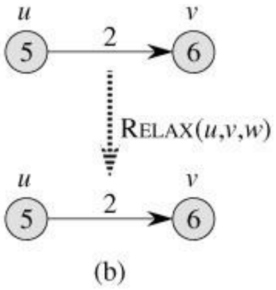
- For each vertex v, $d[v]$ is an upper bound on the weight of a shortest path from s to v and is called a shortest-path estimate.

```
INITIALIZE-SINGLE-SOURCE(G, s)
1    for each vertex $v \in V[G]$
2            do $d[v] \leftarrow \infty$
3               $\pi[v] \leftarrow$ NIL
4    $d[s] \leftarrow 0$
```

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



(a)



(b)

- The process of ***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 $d[v]$ and $\pi[v]$.

  - Dijkstra's algorithm and the shortest-paths algorithm for directed acyclic graphs: <u>each edge is relaxed exactly once</u>.
  - Bellman-Ford algorithm: <u>each edge is relaxed many times</u>.

<u>Properties of shortest paths and relaxation</u>

- **Triangle inequality** (Lemma 24.10)
  For any edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$

- **Upper-bound property** (Lemma 24.11)
  $d[v] \geq \delta(s, v)$ for all vertices $v \in V$, and once $d[v]$ achieves the value $\delta(s, v)$, it never changes.

- **No-path property** (Corollary 24.12)
  If there is no path from s to v, then we always have $d[v] = \delta(s, v) = \infty$.

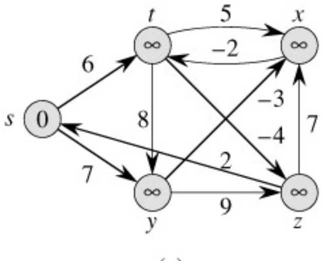
- **Convergence property** (Lemma 24.14)
  If $s \rightsquigarrow u \rightarrow v$ is a **shortest path** and $d[u] = \delta(s, u)$ prior to relaxing edge (u, v), then $d[v] = \delta(s, v)$ at all times afterward.

- **Path-relaxation property** (Lemma 24.15)
  If $p = \langle s, v_1,\ldots, v_k \rangle$ is a shortest path from s to v_k, and the edges of p are
  relaxed in the order $(s, v_1), (v_1, v_2),\ldots, (v_{k-1}, v_k)$, then $d[v_k] = \delta(s, v_k)$.

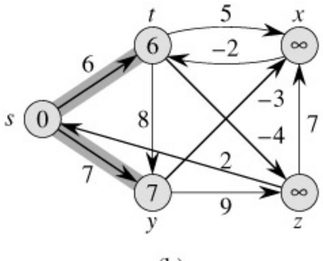
- **Predecessor-subgraph property** (Lemma 24.17)
  Once $d[v] = \delta(s, v)$ for all $v \in V$, the **predecessor subgraph** is a shortest-paths tree rooted at s.

1. The Bellman-Ford algorithm

- The ***Bellman-Ford algorithm*** solves the single-source shortest-paths problem (edge weights may be negative)
- If there is a negative-weight cycle that is reachable from the source, the algorithm indicates that no solution exists.
- ***Bellman-Ford algorithm***:



(a)



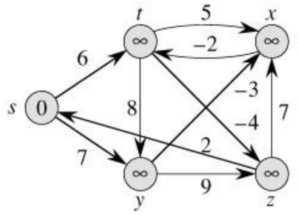
(b)

INITIALIZE-SINGLE-SOURCE(G, s)

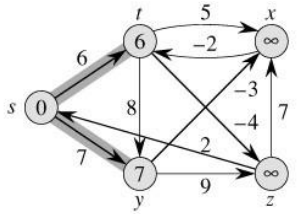
```
for each edge $(u, v) \in E[G]$
    do RELAX($u, v, w$)
```

```
BELLMAN-FORD($G, w, s$)

1   INITIALIZE-SINGLE-SOURCE($G, s$)

2   for $i \leftarrow 1$ to $|G.V| - 1$
3           do for each edge $(u, v) \in G.E$
4                   do RELAX($u, v, w$)

5   for each edge $(u, v) \in G.E$
6       do if $v.d > u.d + w(u, v)$
7           then return FALSE
8   return TRUE
```

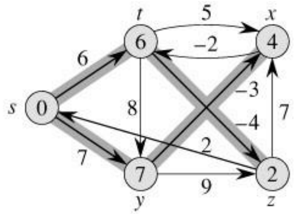
- After initializing the d and π values of all vertices in line 1, the algorithm **makes $|V| - 1$ passes over the edges of the graph**.



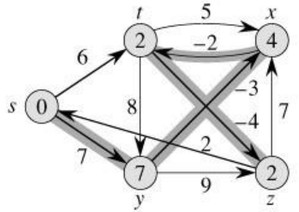
(a)



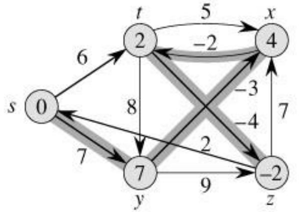
(b)



(c)



(d)



(e)

- Each pass relaxes the edges in the order (*t*, *x*), (*t*, *y*), (*t*, *z*), (*x*, *t*), (*y*, *x*), (*y*, *z*), (*z*, *x*), (*z*, *s*), (*s*, *t*), (*s*, *y*). There are **four** passes!!

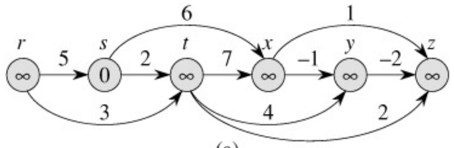
► **Analysis of running time:**

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

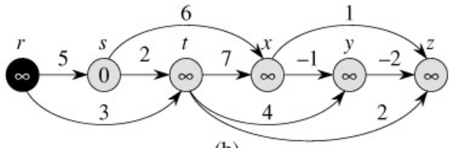
- The initialization: $\Theta(V)$ time
- Each passe: $\Theta(E)$ time
- The **for** loop of lines 5-7 takes $O(E)$ time.

2. Single-source shortest paths in directed acyclic graphs

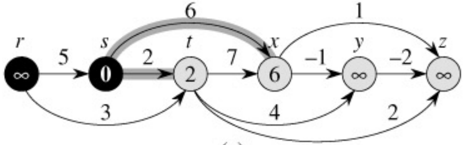
```
DAG-SHORTEST-PATHS($G$, $w$, $s$)

topologically sort the vertices of $G$
INITIALIZE-SINGLE-SOURCE($G$, $s$)
for each vertex $u$, taken in topologically sorted order
    do for each vertex $v \in Adj[u]$
        do RELAX($u$, $v$, $w$)
```



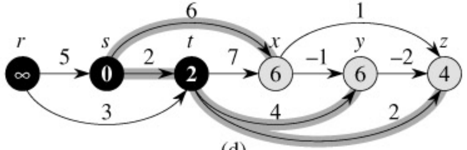
(a)



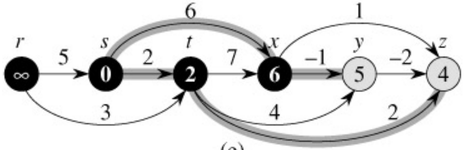
(b)



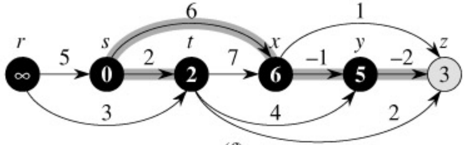
(c)



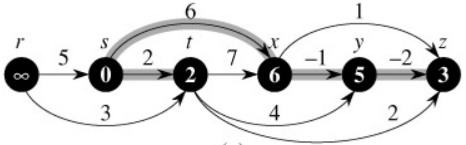
(d)



(e)



(f)



(g)

- Running time: $\Theta(V + E)$ time.

Theorem 24.5

If a weighted, directed graph $G = (V, E)$ has source vertex s and **no cycles,** DAG-SHORTEST-PATHS returns $d[v] = \delta(s, v)$ for all vertices $v \in V$, and G_π is a shortest-paths tree.

Proof For a shortest path $p = \langle v_0, v_1,\ldots, v_k$

the edges on p are relaxed in the order $(v_0, v_1), (v_1, v_2),\ldots, (v_{k-1}, v_k)$.

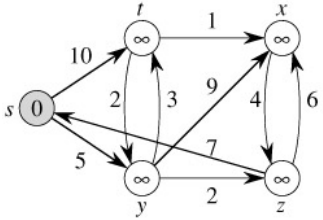
The **path-relaxation property** implies that $d[v_i] = \delta(s, v_i)$ at termination for $i = 0, 1,\ldots, k$. ▓

DIJKSTRA’s ALGORITHM

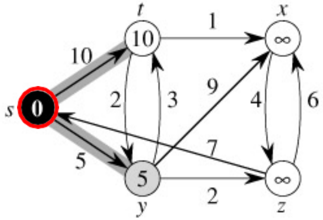
```
DIJKSTRA (G, w, s)
   INITIALIZE-SINGLE-SOURCE (G,s)
   S = $\phi$
   Q = G.V
   while Q $\neq \phi$
      u = EXTRACT-MIN (Q)
      S = S U {u}
      for each vertex v $\in$ G.Adj[u]
         RELAX (u, v, w)
```

3. Dijkstra's algorithm

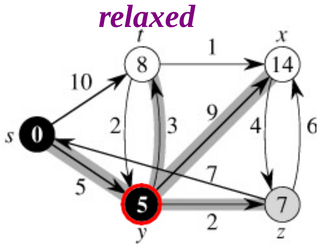
- Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph in which **all edge weights are nonnegative**.



(a) $Q=V$



(b) $Q=V$-{0}



(c) $Q=V$-{0,5}

- Dijkstra's algorithm uses a **greedy strategy** (always chooses the "lightest" or "closest" vertex in V - S to add to set S).

- The key of the c**orrectness of Dijkstra's algorithm** is each time a vertex u is added to set S, we have $d[u] = \delta(s, u)$.

Theorem 24.6: (Correctness of Dijkstra's algorithm)
Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $d[u] = \delta(s, u)$ for all vertices $u \in V$.

Proof Claim: For each vertex $u \in V$, we have $d[u] = \delta(s, u)$ at the time when u is added to set S.
Suppose this is not true. Let u be the first vertex for which $d[u] \neq \delta(s,u)$ when it is added to S. Let P be a shortest path from s to u.
Let y be the first vertex along P such that $y \notin S$, and let $x \in S$ be y's predecessor.