Single-Source Shortest Paths

Dr. G P Gupta

single-source shortest-paths problem

- **single-source shortest-paths problem:**
- Given a graph G = (V, E), we want to find a shortest path from a given source vertex
  $s \in V$ to each vertex $v \in V$.
- ***The algorithm for the single-source problem can solve many other problems, including the following variants.***
  - Single-destination shortest-paths problem
  - Single-pair shortest-path problem
  - All-pairs shortest-paths problem

Outline

- **Bellman-Ford algorithm**.
  - – uses dynamic programming
- Single-source shortest paths in directed acyclic graphs
- **Dijkstra's algorithm**
  - – uses the greedy approach

optimal-substructure property

Lemma 24.1: Let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a SP from v_1 to v_k. Then, $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a SP from v_i to v_j, where $1 \le i \le j \le k$.

So, we have the optimal-substructure property.

Bellman-Ford's algorithm uses dynamic programming.

Dijkstra's algorithm uses the greedy approach.

Let $\delta(u, v)$ = weight of SP from u to v.

Corollary: Let p = SP from s to v, where $p = s \xrightarrow{p'} u \to v$. Then, $\delta(s, v) = \delta(s, u) + w(u, v)$.

Lemma 24.10: Let $s \in V$. For all edges $(u,v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u,v)$.

Relaxation

- For each vertex $v \in V$, we maintain an attribute $v.d$
- ***v*.d :**
  - is an upper bound on the weight of a shortest path from source s to *v*.
- ***v*.*π* :**
  - a predecessor vertex of *v*.

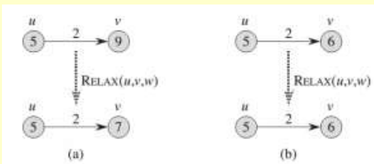
Relaxation

- Algorithms keep track of $d[v]$, $\pi[v]$.
- **Initialized** as follows:

```
INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

Relaxation

- 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 $v.d$ and $v.\pi$.



Relaxation

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

a relaxation step on edge (u, v) in $O(1)$ time

Properties of Relaxation

- d[v], if not ∞, is the length of *some* path from s to v.
- d[v] either stays the same or decreases with time
- Therefore, if d[v] = δ(s, v) at any time, this holds thereafter
- Note that d[v] $\geq$ δ(s, v) always
- After i iterations of relaxing on all (u,v), if the shortest path to v has i edges, then d[v] = δ(s, v).

Properties of Relaxation

Consider any algorithm in which d[v], and π[v] are first initialized by calling Initialize(G, s) [s is the source], and are only changed by calling Relax. We have:

Lemma 24.11: ($\forall$ v:: d[v] $\geq$ δ(s, v)) is an invariant.

Implies d[v] doesn't change once d[v] = δ(s, v).

Proof:
Initialize(G, s) establishes invariant. If call to Relax(u, v, w) changes d[v], then it establishes:

d[v] = d[u] + w(u, v)
$\geq$ δ(s, u) + w(u, v) , invariant holds before call.
$\geq$ δ(s, v) , by Lemma 24.10.

Corollary 24.12: If there is no path from s to v, then d[v] = δ(s, v) = ∞ is an invariant.

- Bellman-Ford returns a compact representation of the set of shortest paths from s to all other vertices in the graph reachable from s. This is contained in the predecessor subgraph.

Predecessor Subgraph

Lemma 24.16: Assume given graph G has no negative-weight cycles reachable from s. Let G_π = predecessor subgraph. G_π is always a tree with root s (i.e., this property is an invariant).

Proof:
Two proof obligations:

(1) G_π is acyclic.
(2) There exists a unique path from source s to each vertex in V_π.

Proof of (1):

Suppose there exists a cycle c = ‹v_0, v_1, …, v_k›, where $v_0 = v_k$.
We have $\pi[v_i] = v_{i-1}$ for i = 1, 2, …, k.

Assume relaxation of (v_{k-1}, v_k) created the cycle.
We show cycle has a negative weight.

Note: Cycle must be reachable from s. (Why?)

Bellman-Ford Algorithm

- Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which ***edge weights may be negative.***
- returns ***a boolean value indicating*** whether or not there is a negative-weight cycle that is reachable from the source.
- ***If there is such a cycle***, the algorithm indicates that *no solution exists.*
- ***If there is no such cycle***, the algorithm produces the shortest paths and their weights.

Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

Bellman-Ford Algorithm

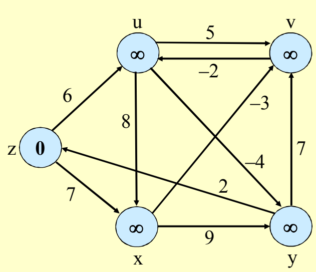
```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

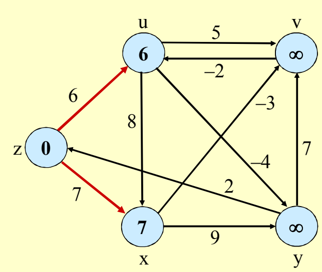
Bellman-Ford Algorithm

- So if Bellman-Ford has not converged after V(G) - 1 iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.

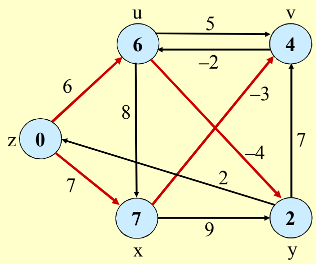
Example



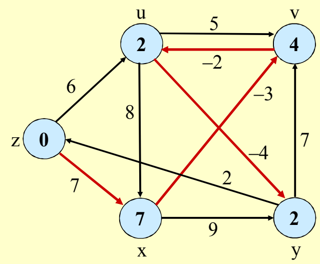
Example



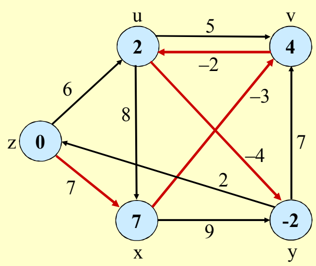
Example



Example



Example



dynamic programming

Note: This is essentially **dynamic programming**.

Let d(i, j) = cost of the shortest path from s to i that is at most j hops.

$$
d(i, j) = \begin{cases} 0 & \textbf{if } i = s \wedge j = 0 \\ \infty & \textbf{if } i \neq s \wedge j = 0 \\ \min(\{d(k, j-1) + w(k, i) : i \in \mathrm{Adj}(k)\} \\ \qquad \cup \ \{d(i, j-1)\}) & \textbf{if } j > 0 \end{cases}
$$

i →

j ↓	z 1	u 2	v 3	x 4	y 5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

Analysis of Bellman-Ford Algorithm

The Bellman-Ford algorithm runs in time $O(VE)$, since the initialization in line 1 takes $\Theta(V)$ time, each of the $|V| - 1$ passes over the edges in lines 2–4 takes $\Theta(E)$ time, and the **for** loop of lines 5–7 takes $O(E)$ time.

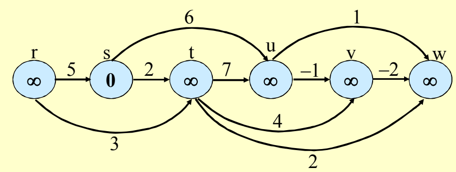
Single-source shortest paths in directed acyclic graphs

Single-source shortest paths in DAGs

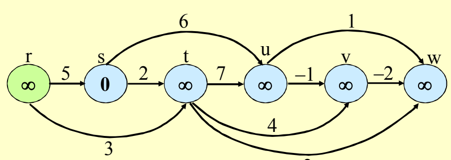
•Shortest paths are always well defined in a dag, since even **if there are negative-weight edges, no negative-weight cycles can exist**

```
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
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

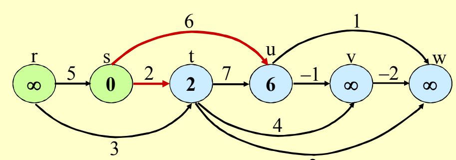
Example



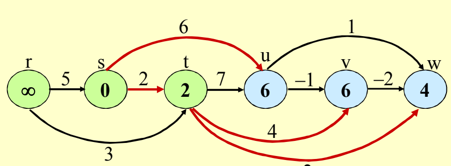
Example



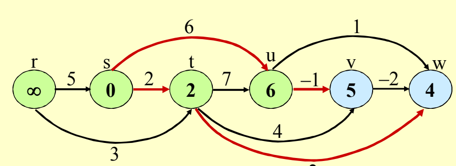
Example



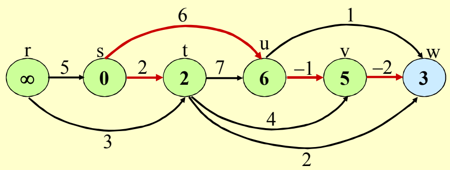
Example



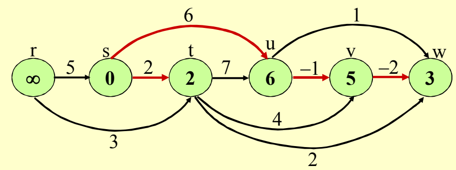
Example



Example



Example



• can compute shortest paths from a single source in **Θ (V+E) time**.

Single-source shortest paths in DAGs

■ Analysis

topological sort of line 1 takes $\Theta(V + E)$ time

• The for loop of lines 3–5 makes one iteration per vertex.

Altogether, the for loop of lines 4–5 relaxes each edge exactly once.

• Because each iteration of the inner for loop takes O(1) time, the total running time is $\Theta(V + E)$

• which ***is linear in the size of an adjacency-list*** representation of the graph.

Dijkstra's Algorithm

Dijkstra's Algorithm

• Assumes **no negative-weight edges**.

•**Greedy approach**

• Maintains a set **S** of vertices whose ***Shortest Path from s has been determined***.

• Repeatedly selects u in **V–S** with minimum ***Shortest Path*** estimate (greedy choice).

• Store V–S in priority queue Q.

Dijkstra's Algorithm

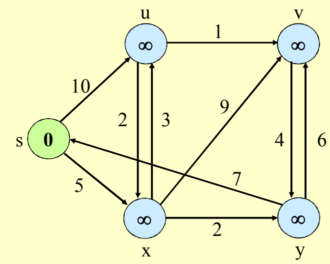
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = Ø
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Dijkstra's Algorithm

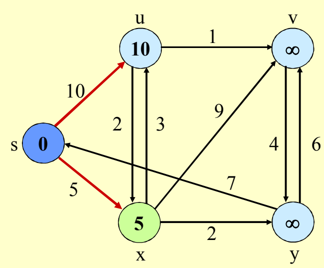
```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

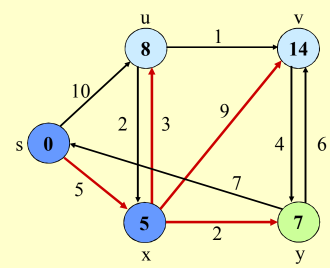
Example



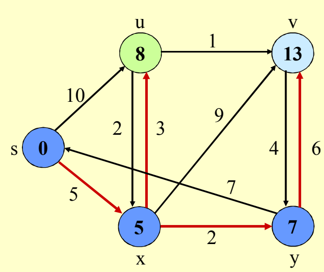
Example



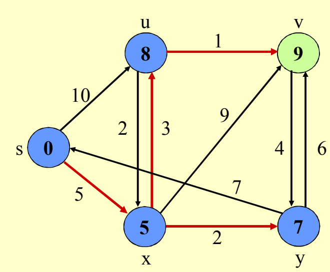
Example



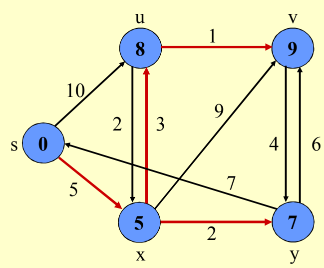
Example



Example



Example



Complexity

Running time is

$O(V^2)$ using linear array for priority queue.

$O((V + E) \lg V)$ using binary heap.

$O(V \lg V + E)$ using Fibonacci heap.

(See book.)