PROBLEM SOLVING 9

Shortest Paths: Bellman-Ford, Floyd-Warshall

CS340

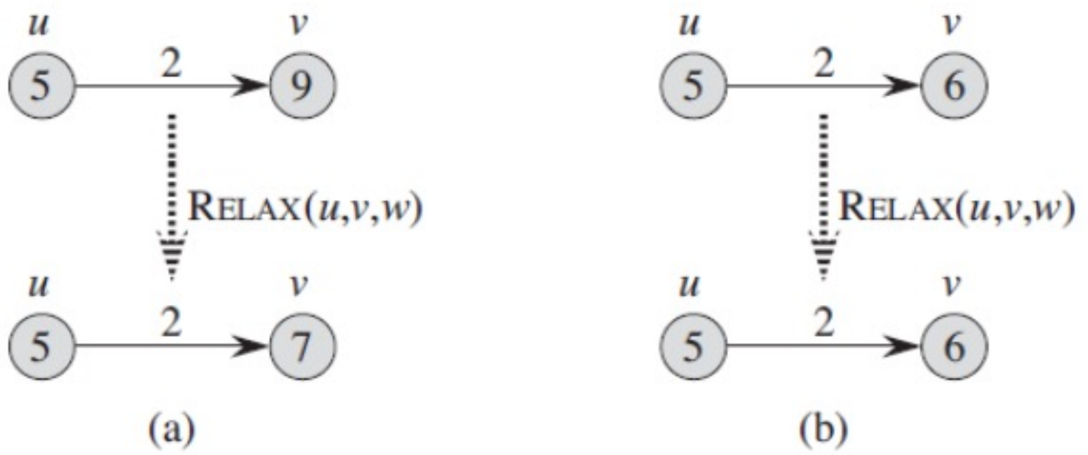
Initialization

- All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

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

Relaxing an edge

- Can we improve the shortest-path estimate for v by going through u and taking (u,v)?



Relaxing an edge

$\textsc{Relax}(u, v, w)$

if $v.d > u.d + w(u, v)$

$\quad v.d = u.d + w(u, v)$

$\quad v.\pi = u$

Properties of shortest paths and relaxation

- Path-relaxation property
  - If $p = <v_0, v_1,\ldots,v_k>$ i is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1,v_2),\ldots(v_{k-1},v_k)$. then $v_k.d = \delta(s, v_k)$. This property holds **regardless** of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Bellman-Ford Algorithm

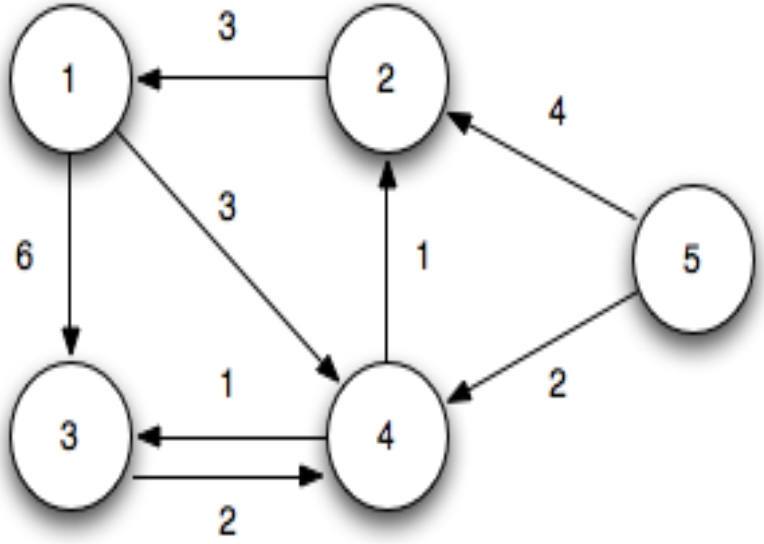
- Allows negative-weight edges.
- Computes v.d and π.d for all v ∈ V.
- Returns TRUE if no negative-weight cycles reachable from s, FALSE otherwise.

Bellman-Ford Algorithm

- Relies on the **Path Relaxation Property**
- Relaxes each edge |V|-1 times (why?)
- Relaxes each edge 1 more time to detect negative-weight cycles. Returns true if none.
- Time Complexity:
  nested for loop
  outer loop = |v|-1
  inner loop = E
  Total = Θ(VE)
- It's a loop!

```
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 (source = 5)



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

Interview Questions

- What is the approach taken by the Bellman-Ford algorithm?
  - Divide and Conquer
  - Greedy
  - Dynamic Programming
  - Search Tree
  - Other

Interview Questions

- Give 2 similarities between Bellman-Ford and Dijkstra’s algorithm.
- Give 2 differences

Interview Questions

- What is the time complexity of Bellman-Ford on a complete graph of n vertices?

Interview Questions

- Given a graph G = (V, E) with positive edge weights, can the Bellman-Ford algorithm and Dijkstra's algorithm produce different shortest-path trees? Will they always produce the same shortest-path weights?
- (Why do we have a shortest paths tree?)
- What does it mean if a sequence of relaxation steps sets the source vertex s parent to a non-NIL value?

Interview Question

- After the ith relaxation of every edge, which paths have converged to the correct value where $v.d = \delta(s,v)$?
  - All paths of length i or less?
  - All paths i hops from source node?
  - Have any paths converged to δ or do they not converge until the algorithm is complete?

Single-source shortest paths in a dag

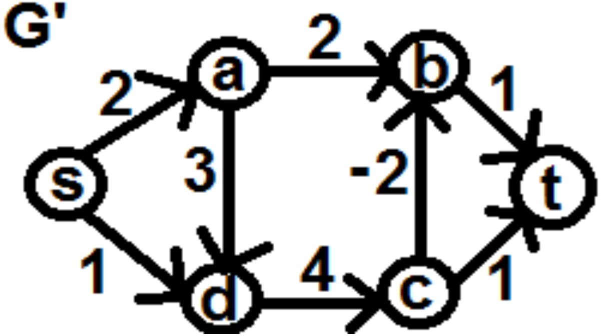
- Dag = Directed Acyclic Graph
- We’re guaranteed **no negative-weight cycles**
- **Negative-weight edges** are ok
- Relax edges according to a topological sort of vertices
- By the **Path Relaxation Property**, relaxing in topological order guarantees edges are relaxed in path order.

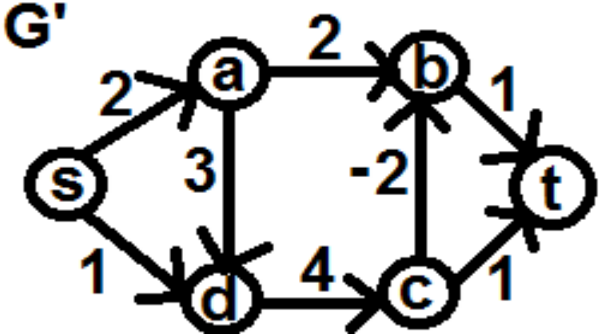
Single-source shortest paths in a dag

```
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)
```

Time complexity Θ(V+E)
This is one of our time-saving ideas: make the problem less general.

Single-source shortest paths in a dag





Interview Questions

- Suppose we change DAG-SHORTEST-PATHS to process only the first $|V|-1$ vertices, taken in topologically sorted order. Would the algorithm remain correct?
- Can a directed acyclic graph have more than one topological ordering?
- Give a $\Theta(V+E)$ algorithm for finding the longest paths from s in a weighted directed acyclic graph G. Does your solution work when G is not acyclic?

All-Pairs shortest paths

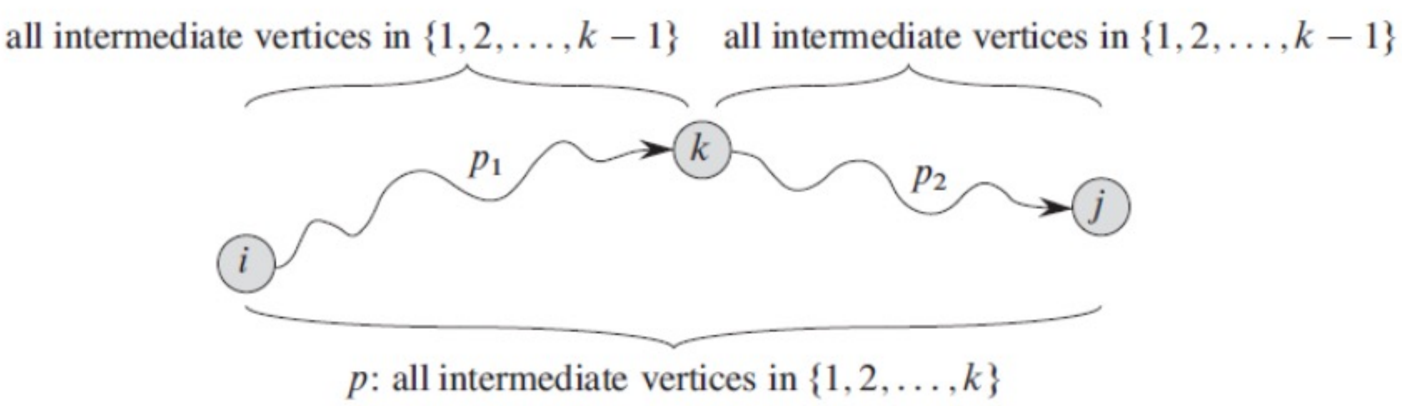
- As opposed to single source, we want to find distances between all pairs of vertices.
- Output: A matrix of shortest path distances
- Could run BELLMAN-FORD once from each vertex
  - $O(V^2E)$ which is $O(V^4)$ if the graph is dense ($E = \Theta(V^2)$).
- If no negative-weight edges, could run Dijkstra's algorithm once from each vertex
  - $O(VE \lg V)$ with binary heap → $O(V^3 \lg V)$ if dense,
  - $O(V^2 \lg V + VE)$ with Fibonacci heap → $O(V^3)$ if dense.

Floyd-Warshall algorithm

- A completely different approach
- Negative-weight edges may be present
- No negative-weight cycles

Floyd-Warshall algorithm

- Looking for a path from i to j, does it pass through vertex k?



- Use a matrix to iterate through every possibility.
- How many possibilities are there?
- Check out this week's video on Rod Cutting

Floyd-Warshall algorithm

FLOYD-WARSHALL(W)

```
n = W.rows
D^(0) = W
for k = 1 to n
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    for i = 1 to n
        for j = 1 to n
            d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

What is the time complexity?

Floyd- Warshall

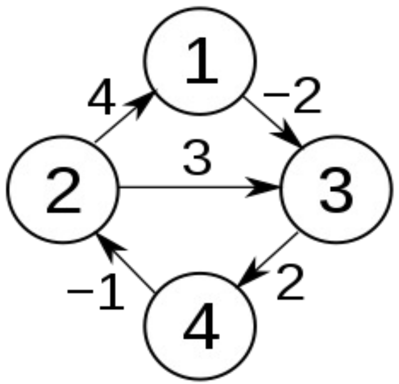
K=0	j			
i		1	2	3	4
	1				
	2				
	3				
	4				

K=1	j			
i		1	2	3	4
	1				
	2				
	3				
	4				

K=2	j			
i		1	2	3	4
	1				
	2				
	3				
	4				

K=3	j			
i		1	2	3	4
	1				
	2				
	3				
	4				

K=4	j			
i		1	2	3	4
	1				
	2				
	3				
	4				



Interview Questions

- What is the approach taken by the Floyd-Warshall algorithm?
  - Divide and Conquer
  - Greedy
  - Dynamic Programming
  - Search Tree
  - Other