Graph Algorithms II

SCC and Shortest Paths

Single Source Shortest Paths

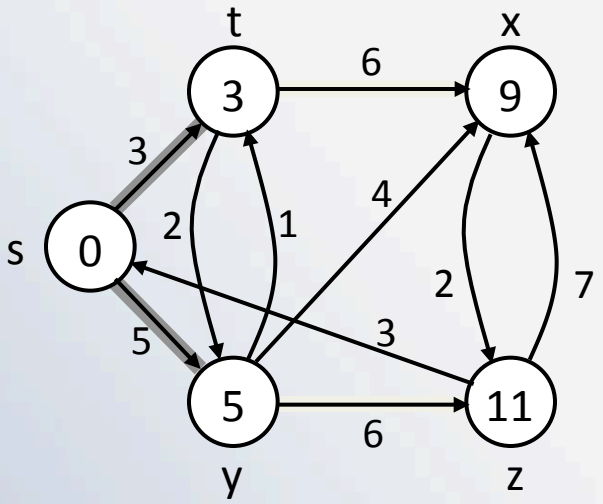
- Given a weighted, directed graph, a shortest path from **vertex u** to **vertex v** is defined as the path with minimum weight from *u* to *v*.
- The single-source shortest path algorithms gives the shortest path to from *u* to every other *v*.
  - Single-destination shortest paths
  - Single-pair shortest paths
  - All-pairs shortest paths
- Assume (for now) all weights non-negative
- CLRS Ch. 24

Single Source Shortest Paths: Variants

- **Single-destination shortest paths**
  - Same as single source, with edges reversed
- **Single-pair shortest paths**
  - Problem solved if we solve single-source problem.
  - No algorithms are known that run asymptotically faster than single-source shortest paths.
- **All-pairs shortest paths**
  - Run Single-Source for every source vertex
  - Faster: Specialized algorithms (Ch 25)

Shortest Path Problems

- **Input:**
  - Directed graph G = (V, E)
  - Non-negative weight function w



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

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

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

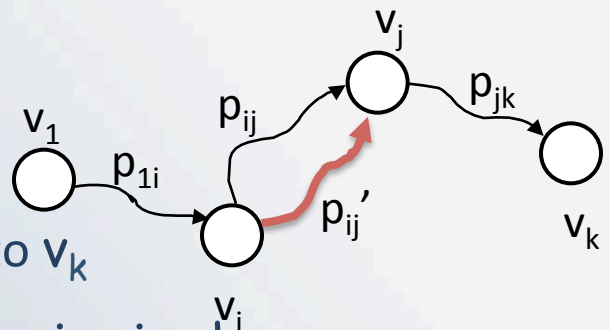
$$\delta(u, v) = \min \begin{cases} w(p) : u \overset{p}{\rightsquigarrow} v & \text{if there exists a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- Shortest path u to v is any path p such that $w(p) = \delta(u, v)$

Optimal Substructure of Shortest Paths

Given:

- A weighted, directed graph G = (V, E)
- A **weight function** w: E → **R**,
- A **shortest path** $p = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A **subpath** of p: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \le i \le j \le k$



Then: p_{ij} is a shortest path from v_i to v_j

Proof: $p = v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$$

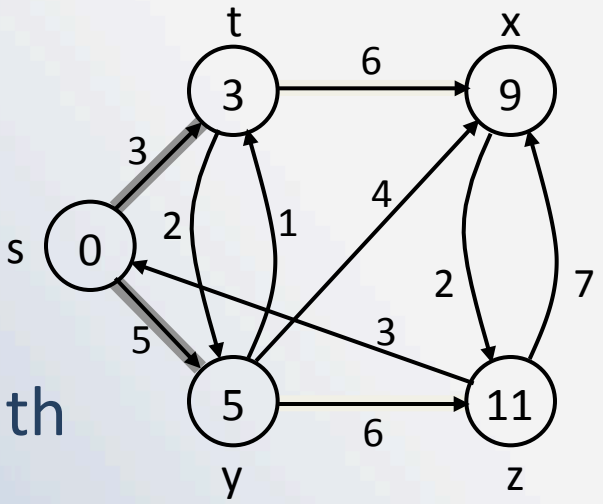
Assume $\exists\ p_{ij}'$ from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$ contradiction!

Shortest-Path Representation

For each vertex $v \in V$:

- d[v] is a **shortest-path estimate**
  - **Initially, d[v] = ∞**
  - Reduces as algorithms progress
  - At the end d[v] = δ(s, v)



- **π[v]** = **predecessor** of v on a shortest path from ***s***
  - If no predecessor, π[v] = NIL
  - π induces a tree—**shortest-path tree**
- Shortest paths & shortest path trees are not unique (and neither are spanning trees)

Shortest-Path Representation

- We build a predecessor array (each node points to its parent, all the way back to the source)

- We calculate the distance from some **source** vertex.

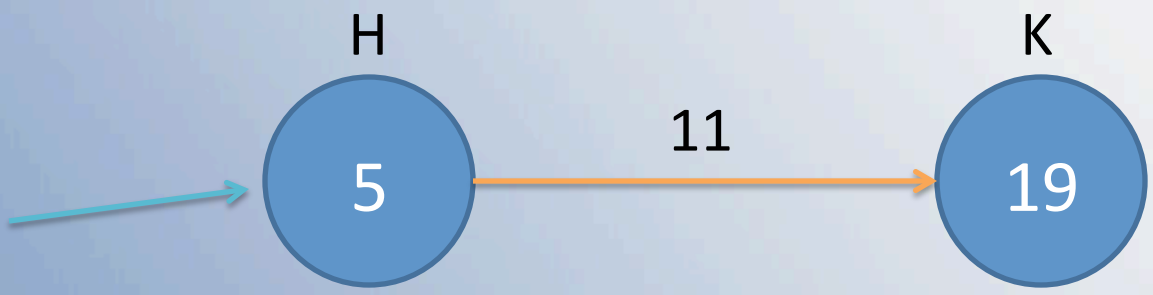
Initialization

INITIALIZE-SINGLE-SOURCE(V, s)

1. **for** each $v \in V$
2. **do** $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

Relaxation

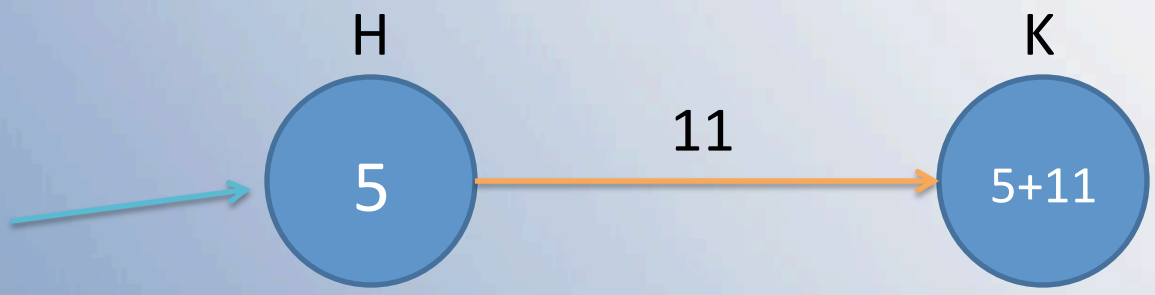


Given: Two nodes with shortest path estimates of 5 and 19.

We want to consider whether the orange edge coming into node K from node H can be used to reduce K's current estimate.

Can it?

Relaxation



Given: Two nodes with shortest path estimates of 5 and 19.

We want to consider whether the orange edge coming into node K from node H can be used to reduce K's current estimate.

Can it?

YES. (5+11) = 16 < 19. So d[K] is now 16, and π[K] is now H.

RELAX(u, v, w)

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

- All the single-source shortest-paths algorithms
  - start by calling INIT-SINGLE-SOURCE
  - then relax edges

- The algorithms differ in the order and how many times they relax each edge.

Dijkstra's Algorithm

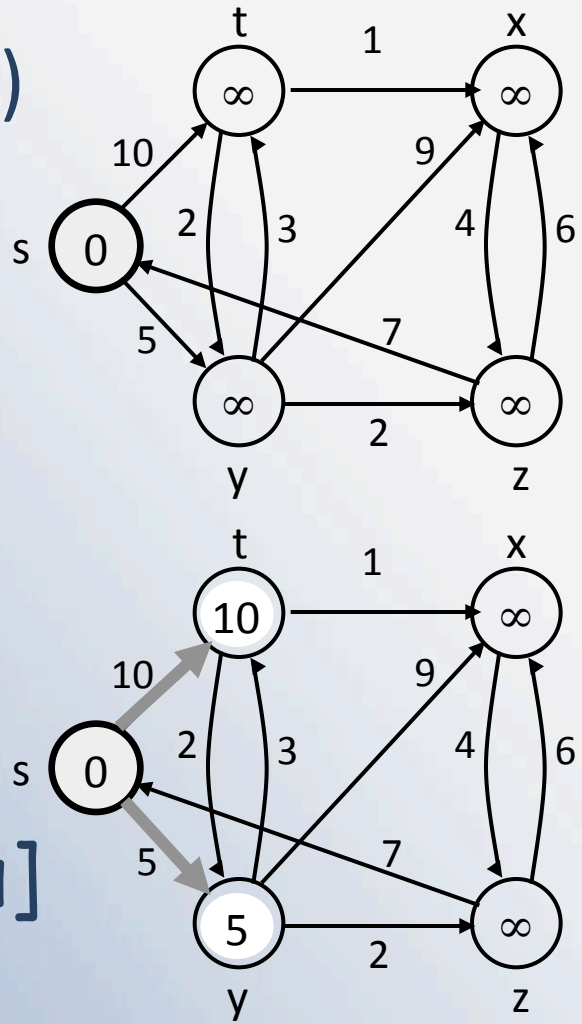
- Assume we have no negative weight edges...

Dijkstra's Algorithm

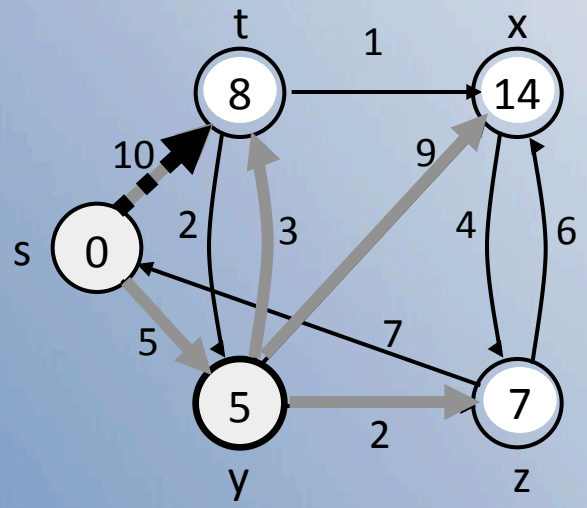
- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0 \; \forall \; (u, v) \in E$
- Maintains two sets of vertices:
  - S = vertices whose final shortest-path weights have already been determined
  - Q = vertices in V – S: min-priority queue
    - Keys in Q are estimates of shortest-path weights (d[v])
- Repeatedly select a vertex $u \in V - S$, with the minimum shortest-path estimate d[u]

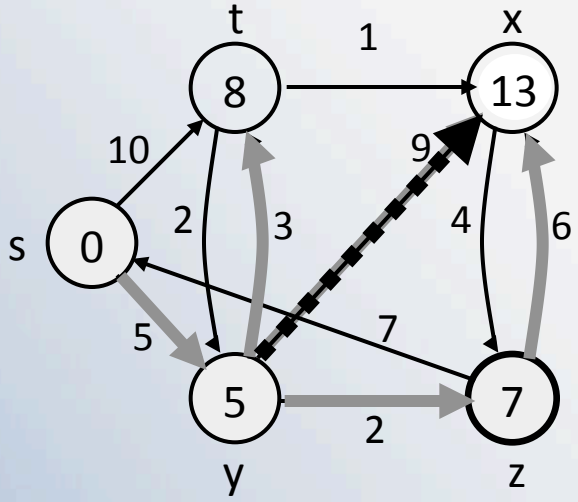
Dijkstra (G, w, s)

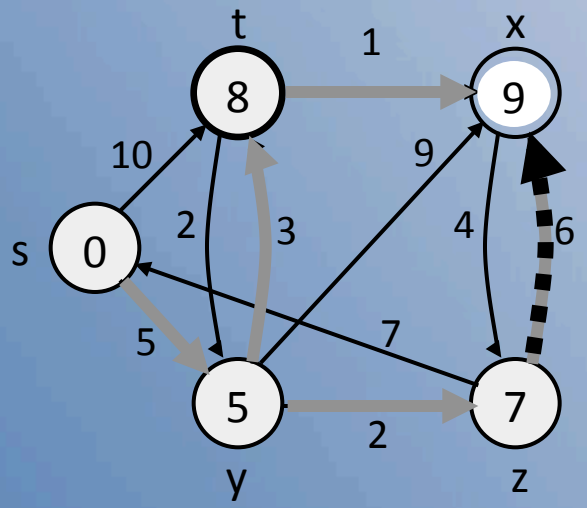
1. INITIALIZE-SINGLE-SOURCE(V, s)
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **while** $Q \neq \emptyset$
5. **do** $u \leftarrow$ EXTRACT-MIN(Q)
6. $S \leftarrow S \cup \{u\}$
7. **for** each vertex $v \in Adj[u]$
8. **do** RELAX(u, v, w)

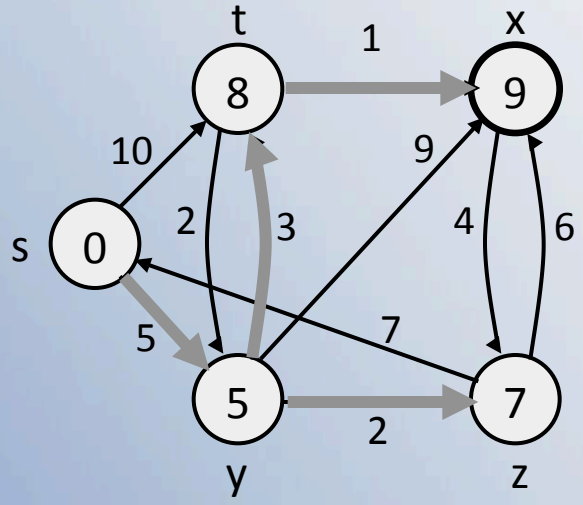


Example


t
x
1
8
14
10
9
s
0
2
3
4
6
5
7
5
7
2
y
z


t
x
1
8
13
10
9
s
0
2
3
4
6
5
7
5
7
2
y
z


t
x
1
8
9
10
9
s
0
2
3
4
6
5
7
5
7
2
y
z

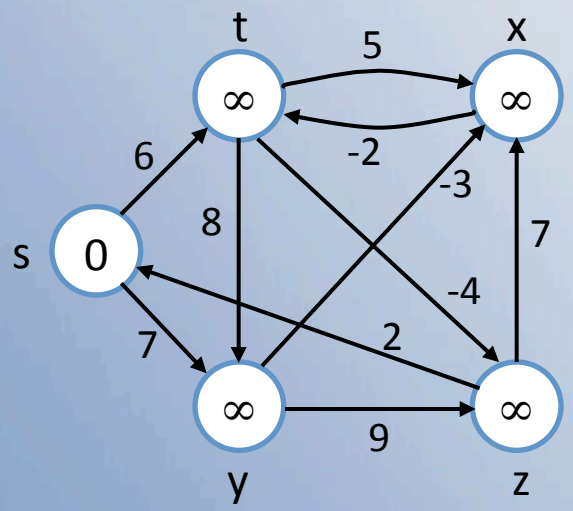

t
x
1
8
9
10
9
s
0
2
3
4
6
5
7
5
7
2
y
z

Bellman-Ford Algorithm

- Single-source shortest paths problem
  - Computes d[v] and π[v] for all v $\in$ V
- Allows *negative edge weights*
- Returns:
  - **TRUE** if no negative-weight cycles are reachable from the source s
  - **FALSE** otherwise $\Rightarrow$ no solution exists
- Idea:
  - Traverse all the edges |V – 1| times, every time performing a relaxation step of each edge

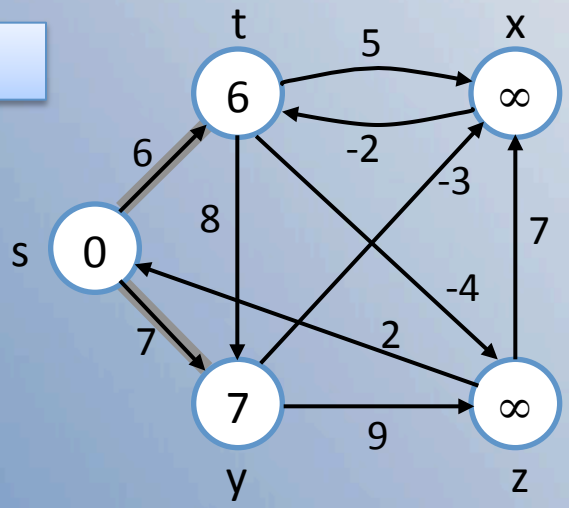
Bellman-Ford Idea

1. INITIALIZE-SINGLE-SOURCE(V, s)
2. Relax every edge, |V|-1 times

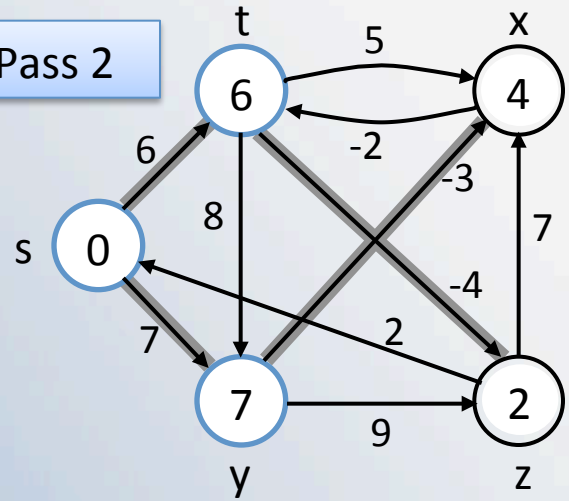


Example

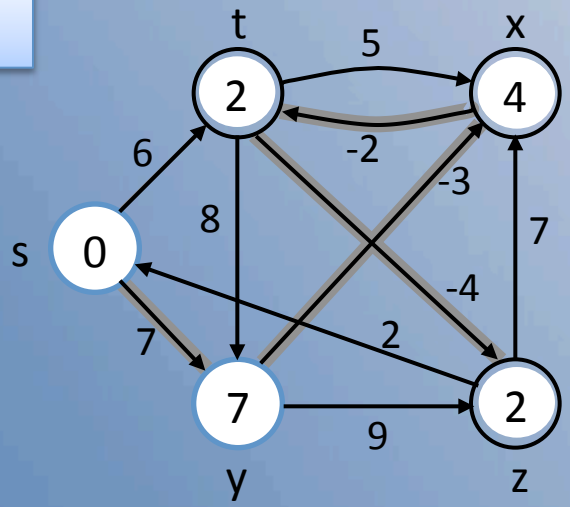
Pass 1



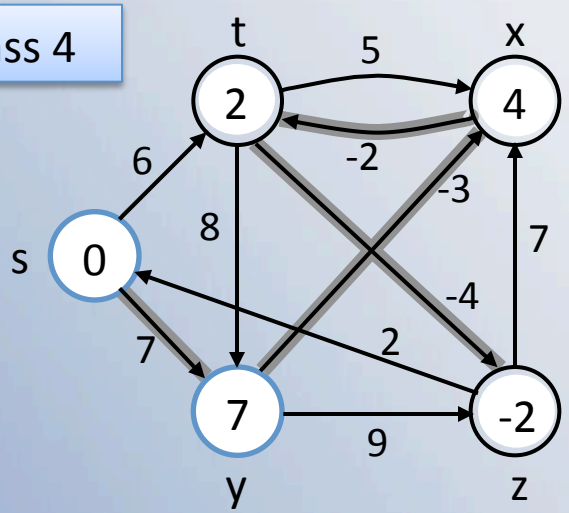
Pass 2



Pass 3



Pass 4



Why does this work

- Consider a shortest path from u to v

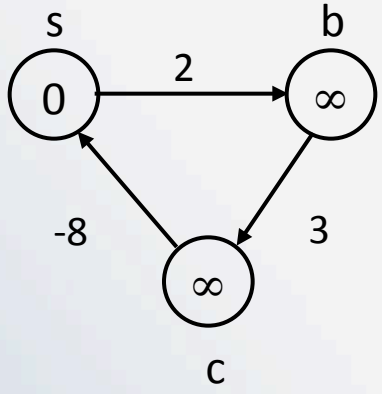
Bellman-Ford

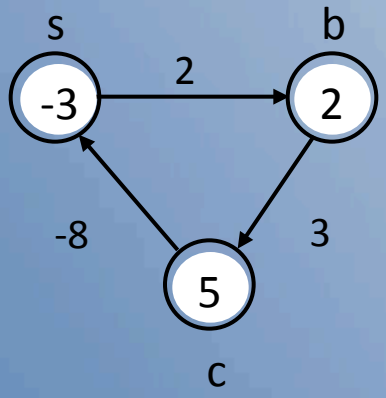
```
1.  INITIALIZE-SINGLE-SOURCE(V, s)
2.  for i ← 1 to |V| - 1
3.        for each edge (u, v) ∈ E
4.              RELAX(u, v, w)
5.  for each edge (u, v) ∈ E
6.        if d[v] > d[u] + w(u, v)
7.              then return FALSE
8.  return TRUE
```

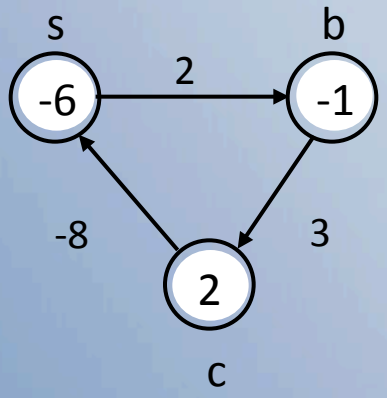
Cycle Detection

Detecting Negative Cycles

- **for** each edge (u, v) ∈ E
- **if** d[v] > d[u] + w(u, v)
- **then return** FALSE
- **return** TRUE







Look at edge (s, b):

d[b] = -1
d[s] + w(s, b) = -4

⇒ d[b] > d[s] + w(s, b)

Shortest Path in a DAG : Idea

- If there are no cycles, then there are better algorithms.

Analysis?

$\Theta(V+E)$

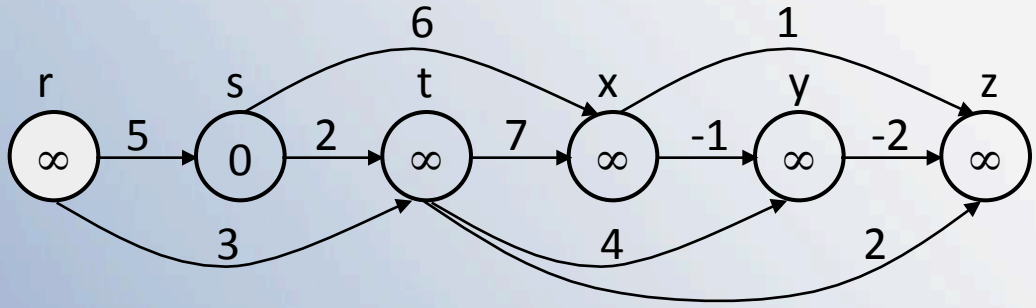
Algorithm

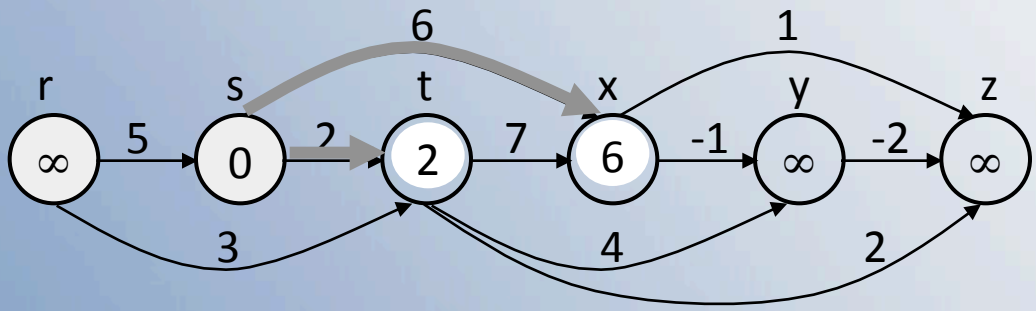
Topological Sort the DAG

Relax the edges in this sorted order

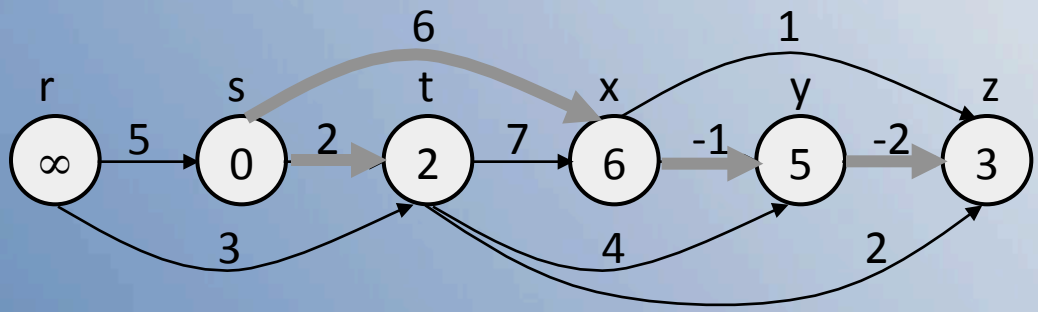
Why does this work?

Example DAG


r
s
t
x
y
z
∞
0
∞
∞
∞
∞
5
2
7
-1
-2
6
1
3
4
2

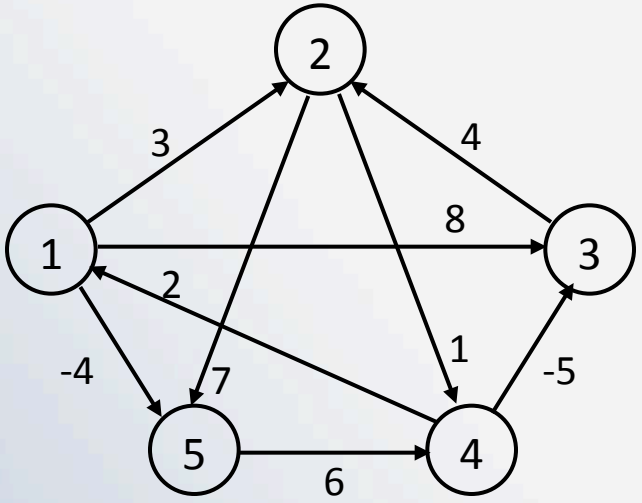

r
s
t
x
y
z
∞
0
2
6
∞
∞
5
2
7
-1
-2
6
1
3
4
2

...


r
s
t
x
y
z
∞
0
2
6
5
3
5
2
7
-1
-2
6
1
3
4
2

All-Pairs Shortest Paths

- **Given:**
  - Directed graph G = (V, E)
  - Weight function w : E → **R**
- **Compute:**
  - The shortest paths between all pairs of vertices in a graph
  - Representation of the result: an n × n matrix of shortest-path distances δ(u, v)

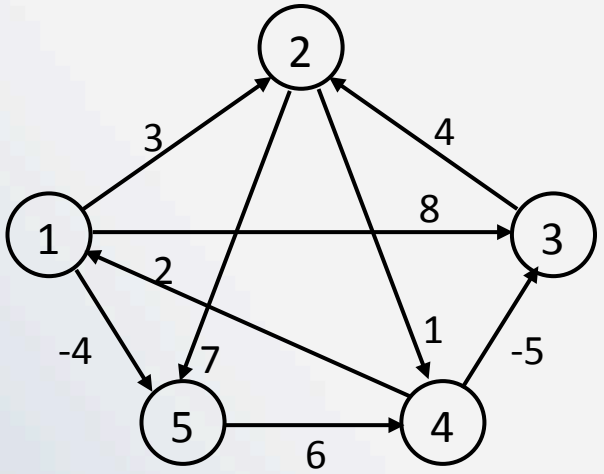


All-Pairs Shortest Paths - Solutions

- 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 the graph is dense
- We can solve the problem in $O(V^3)$, with no elaborate data structures

All-Pairs Shortest Paths

- Assume the graph (G) is given as
  adjacency matrix of weights
  - $W = (w_{ij})$, n x n matrix, $|V| = n$
  - Vertices numbered 1 to n



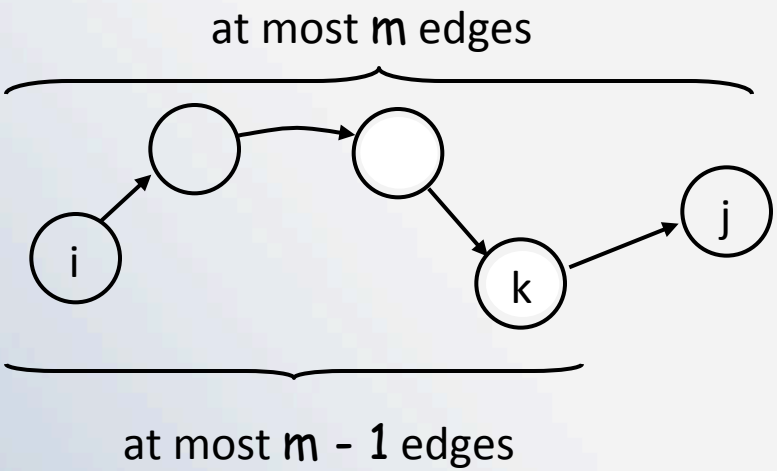
$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of } (i, j) & \text{if } i \neq j \text{ , } (i, j) \in E \\ \infty & \text{if } i \neq j \text{ , } (i, j) \notin E \end{cases}$$

- Output the result in an n x n matrix

  $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$
- Solve the problem using **dynamic programming**

Optimal Substructure of a Shortest Path

- All subpaths of a shortest path are also shortest paths
- Let p: a shortest path p from vertex i to j that contains at most m edges
- If i = j
  - w(p) = 0 and p has no edges

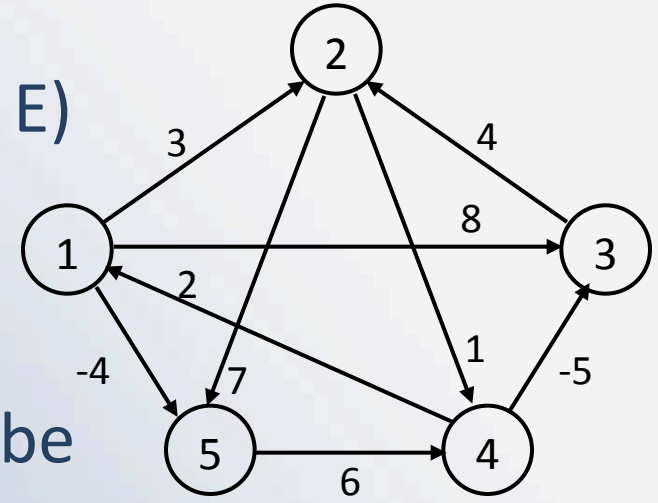


- If $i \neq j$: $p = i \overset{p'}{\rightsquigarrow} k \rightarrow j$
  - p' has at most m-1 edges
  - p' is a shortest path

$$\delta(i, j) = \delta(i, k) + w_{kj}$$

The Floyd-Warshall Algorithm

- **Given:**
  - Directed, weighted graph G = (V, E)
  - Negative-weight edges may be present
  - No negative-weight cycles could be present in the graph



- **Compute:**
  - The shortest paths between all pairs of vertices in a graph

Intermediate Vertices

Without loss of generality, we will assume that $V=\{1,2,\ldots,n\}$, i.e., that the vertices of the graph are numbered from 1 to n.

Given a path $p=(v_1, v_2,\ldots, v_m)$ in the graph, we will call the vertices v_k with index k in $\{2,\ldots,m-1\}$ the intermediate vertices of p.

Key Definition

The key to the Floyd-Warshall algorithm is the following definition:

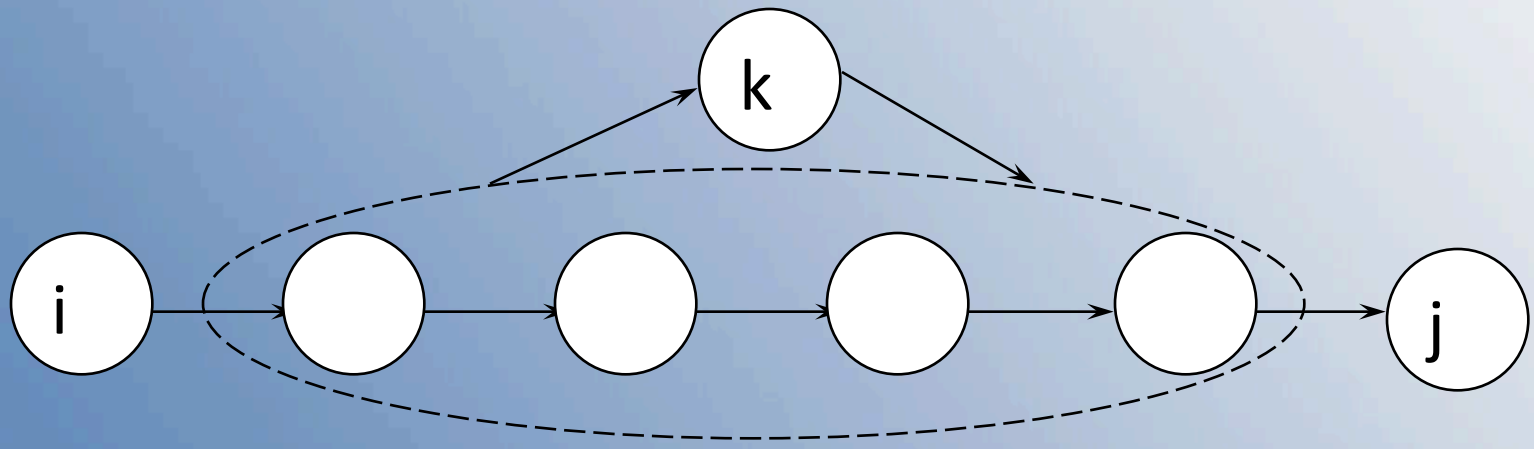
Let $d_{ij}^{(k)}$ denote the length of the shortest path from i to j such that all intermediate vertices are contained in the set {1,…,k}.

Remark 1

A shortest path does not contain any vertex twice, as this would imply that the path contains a cycle. By assumption, cycles in the graph have a positive weight, so removing the cycle would result in a shorter path, which is impossible.

Floyd Warshall Algorithm

- **How it works**
  - Let the vertices in a graph be numbered from 1 … n.
  - Consider the subset {1,2,…, k} of these n vertices.
  - Imagine finding the shortest path from vertex i to vertex j that uses vertices in the set {1,2,…,k} only.
  - There are two situations:
    1) k is an intermediate vertex on the shortest path.
    2) k is not an intermediate vertex on the shortest path.



Remark 2

Consider a shortest path **p** from **i** to **j** such that the intermediate vertices are from the set {**1**, …,**k**}.

If the vertex **k** is not an intermediate vertex on **p**, then $d_{ij}^{(k)} = d_{ij}^{(k-1)}$

If the vertex **k** is an intermediate vertex on **p**, then $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Remark 2

Therefore, we can conclude that

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)} , d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

Recursive Formulation

If we do not use intermediate nodes, i.e., when k=0, then

$$d_{ij}^{(0)} = w_{ij}$$

If k>0, then

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)} , d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

Floyd Warshall Algorithm

- Let D start with D[i,j] = w(I,j) , and update D with the calculated shortest paths: bottom up.

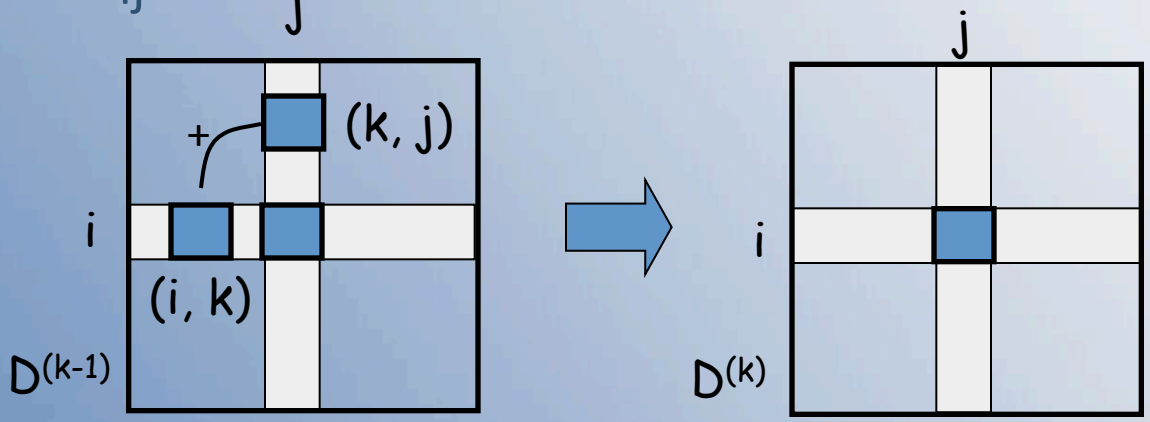
```
for k=1 to n {
    for i=1 to n {
        for j=1 to n
            D[i,j] = min(D[i,j],D[i,k]+D[k,j])
        }
    }
}
```

  - Why is k the outermost loop variable?
  - Does i/j order matter?

Computing the Shortest Path Weights

- $d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{if } k \geq 1 \end{cases}$
- The final solution: $D^{(n)} = (d_{ij}^{(n)})$:

$$d_{ij}^{(n)} = \delta(i, j) \; \forall \; i, j \in V$$



FLOYD-WARSHALL(W)

1. $n \leftarrow rows[W]$
2. $d^{(0)} \leftarrow W$
3. **for** $k \leftarrow 1$ **to** n
4. **do for** $i \leftarrow 1$ **to** n
5. **do for** $j \leftarrow 1$ **to** n
6. **do** $d_{ij}^{(k)} \leftarrow \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. **return** $d^{(n)}$

Running Time of Floyd-Warshall:

A. E^3

B. V^3

C. $EV\ log(V)$

D. V^2

E. EV^2

1. $n \leftarrow rows[W]$
2. $d^{(0)} \leftarrow W$
3. **for** $k \leftarrow 1$ **to** n
4. **do for** $i \leftarrow 1$ **to** n
5. **do for** $j \leftarrow 1$ **to** n
6. **do** $d_{ij}^{(k)} \leftarrow \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. **return** $d^{(n)}$