Faculty of Information Technology, Monash University

FIT2004: Algorithms and Data Structures

Week 9: Bellman Ford and Floyd-Warshall Algorithms

These slides are prepared by M. A. Cheema and are based on the material developed by Arun Konagurthu and Lloyd Allison.

Reminders/Things to note

- Assignment 3 has been released
  - Due 10-May-2019, 23:55:00
- When should Assignment 4 be released?
  - Due 12-May-2019

Recommended reading

- Unit notes: Chapter 13
- Cormen et al. Introduction to Algorithms.
  - Section 24.1 Bellman-Ford Algorithm

- http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Graph/

- http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Graph/Directed/

Outline

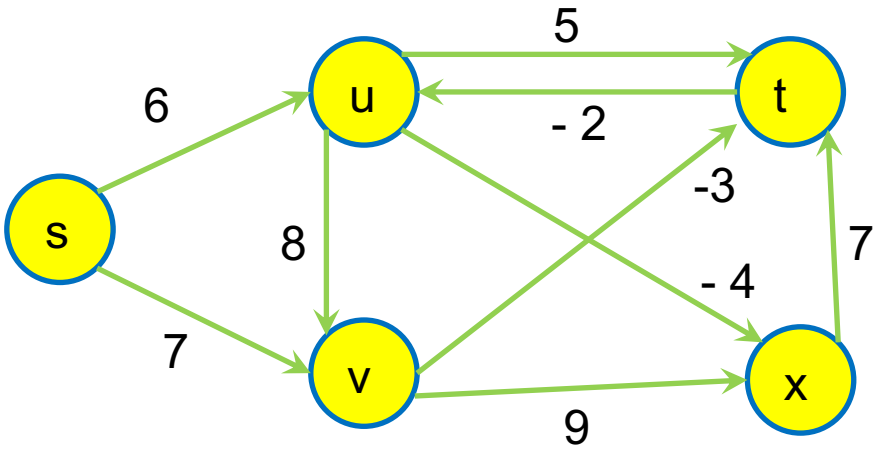
1. Shortest path in a graph with negative weights
2. All-pairs shortest paths
3. Transitive Closure

Shortest path (negative weights)

- What is the shortest distance from s to x in this graph?
- What will be the shortest distance from s to x if Dijkstra's algorithm is used on this graph?
- Dijkstra's algorithm cannot handle graph with negative weights.
- How to compute shortest paths on such graphs?
  - Bellman-Ford Algorithm

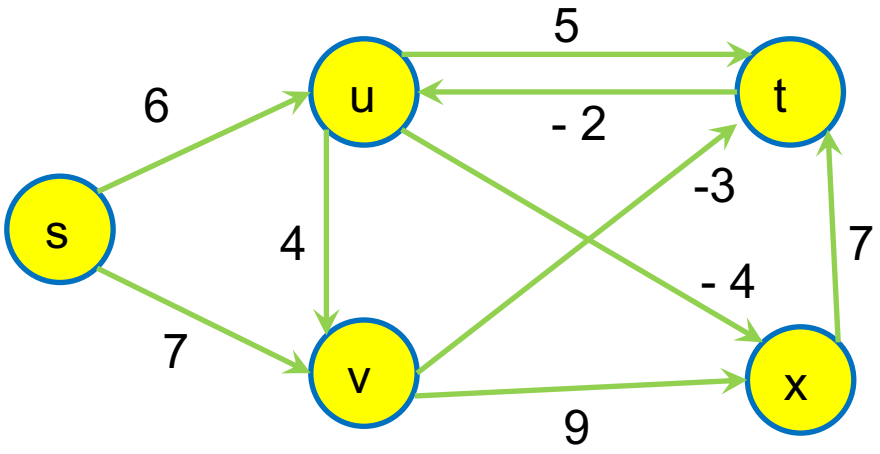
Dijkstra's

Graph with –ve weights



Shortest path (negative weights)

- What is the shortest distance from s to x on **this** graph?
- If there is a negative cycle in the graph, the notion of shortest path/distance does not make sense.
- Bellman-Ford algorithm returns
  - shortest distances from s to all vertices in the graph if there are no negative cycles
  - an error if there is a negative cycle reachable from s (i.e., can be used to detect negative cycles)



Bellman-Ford Algorithm

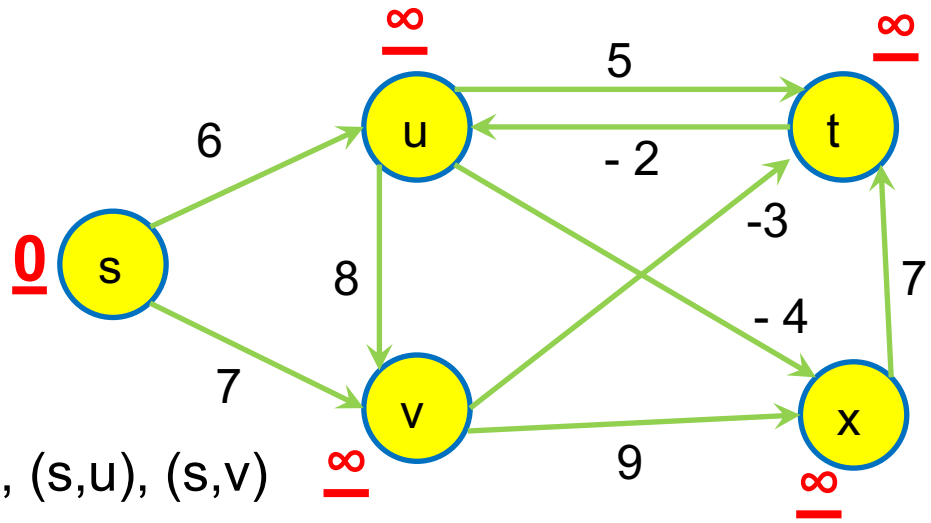
Initialize:

- For each vertex a in the graph
  - dist(s,a) = ∞
- dist(s,s) = 0

Consider the following operation:

- For each edge (a, b, w) in the graph // the order does not matter
  - dist(s, b) = min(dist(s,b) , dist(s,a) + w)

What is dist(s,u)?



Assume the following order:

(u,t), (u,v), (u,x), (t,u), (v,t), (v,x), (x,t), (s,u), (s,v)

Bellman-Ford Algorithm

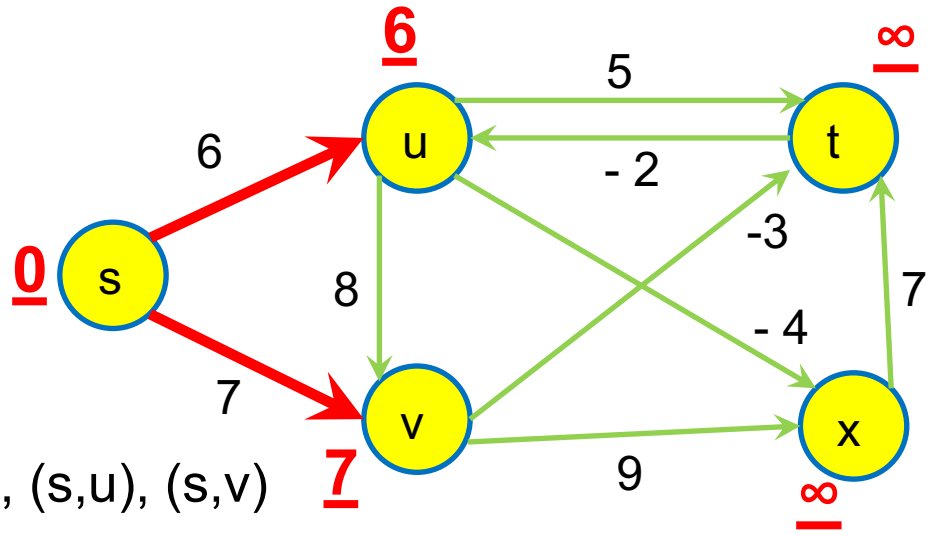
Initialize:

- For each vertex a in the graph
  - dist(s,a) = ∞
- dist(s,s) = 0

Consider the following operation:

- Repeat 2 times
  - For each edge (a, b, w) in the graph // the order does not matter
    - dist(s, b) = min(dist(s,b) , dist(s,a) + w)

What is dist(s,x)?



Assume the following order:

(u,t), (u,v), (u,x), (t,u), (v,t), (v,x), (x,t), (s,u), (s,v)

Bellman-Ford Algorithm

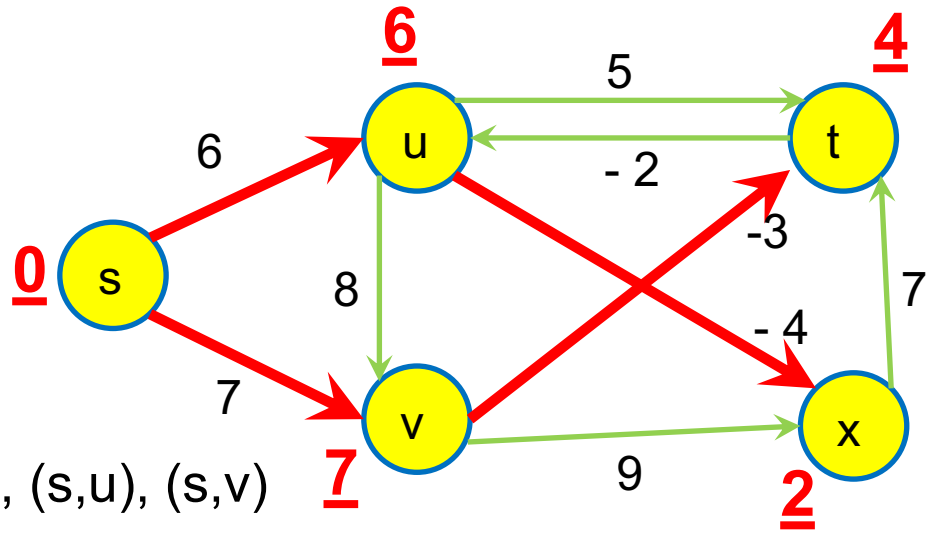
Initialize:

- For each vertex a in the graph
  - dist(s,a) = ∞
- dist(s,s) = 0

Consider the following operation:

- Repeat 3 times
  - For each edge (a, b, w) in the graph // the order does not matter
    - dist(s, b) = min(dist(s,b) , dist(s,a) + w)

What is dist(s,u)?



Assume the following order:

(u,t), (u,v), (u,x), (t,u), (v,t), (v,x), (x,t), (s,u), (s,v)

Bellman-Ford Algorithm

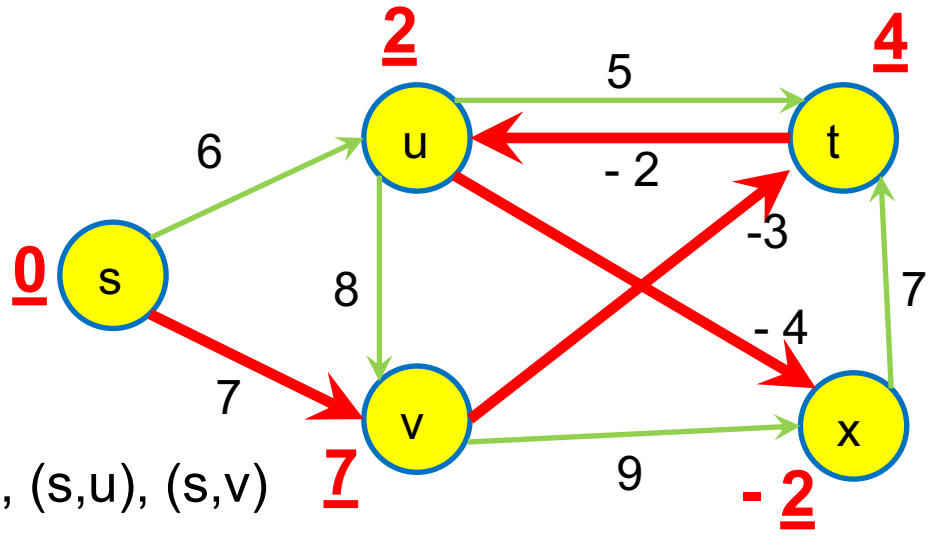
Initialize:

- For each vertex a in the graph
  - dist(s,a) = ∞
- dist(s,s) = 0

Consider the following operation:

- Repeat 4 times
  - For each edge (a, b, w) in the graph // the order does not matter
    - dist(s, b) = min(dist(s,b) , dist(s,a) + w)

What is dist(s,x)?



Assume the following order:

(u,t), (u,v), (u,x), (t,u), (v,t), (v,x), (x,t), (s,u), (s,v)

Bellman-Ford Algorithm

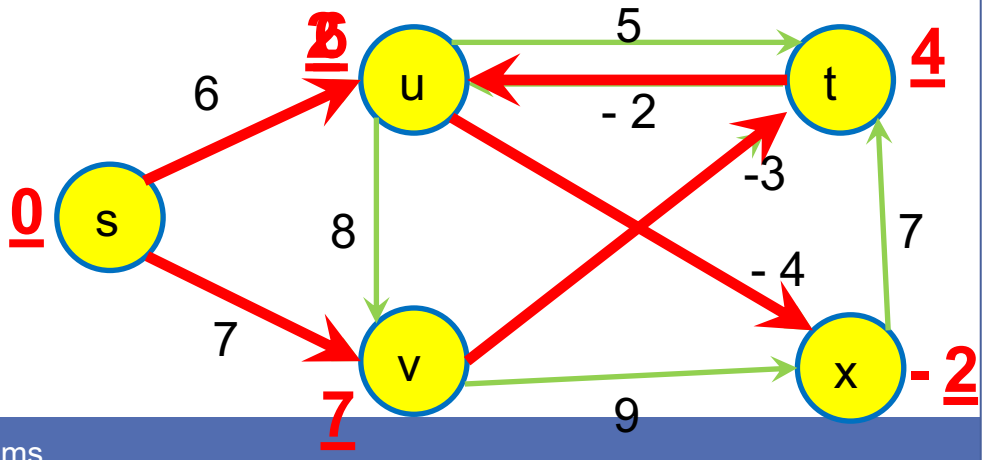
```
# STEP 1: Initializations
dist[1...V] = infinity
pred[1...V] = Null
dist[s] = 0
# STEP 2: Iteratively estimate dist[v] (from source s)
for i = 1 to V-1:
        for each edge <u,v,w> in the whole graph:
                est = dist[u] + w
                if est < dist[v]:
                        dist[v] = est
                        pred[v] = u

# STEP 3: Checks and returns false if a negative weight cycle
# is along the path from s to any other vertex
for each edge <u,v,w> in the whole graph:
        if dist[u]+w < dist[v] :
                return error; # negative edge cylce found in this graph

return dist[...], pred[...]
```

Time Complexity:

O(VE)

Bellman-Ford Algorithm: Correctness

- We established that the negative cycles do not make sense
- Can a shortest path from s to t have a positive cycle (i.e., with weight more than zero)?
  - No, because the path that avoids this cycle will have smaller distance
  - Also, if the path has a zero-weight cycle, it can be ignored
  - i.e., shortest distances can be computed by ignoring the non-negative cycles
- What is the maximum number of edges in a shortest path between two vertices ignoring zero-weight cycles?
  - V - 1
- **Invariant: After i-th iteration, for every vertex v in graph, there does not exist a path from s to v that contains at most i edges AND has distance smaller than dist[v] computed by the algorithm.**
  - The first iteration guarantees the shortest distances to all vertices considering paths of length at most 1 edge
  - The second iteration guarantees the shortest distances to all vertices considering paths of length at most 2 edges
  - …
  - The (V - 1)-th iteration guarantees the shortest distances to vertices considering paths of length at most (V-1) edges – thus, after (V-1)-th iteration, we must have found the shortest distances without –ve cycles
- If V-th iteration reduces the distance of a vertex, this means that there is a shortest path with at least V edges which implies that there is a negative cycle.

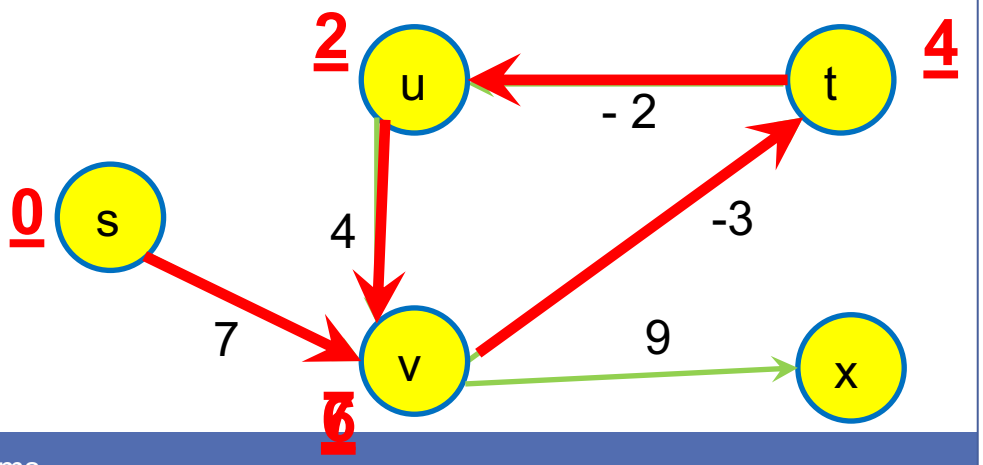


Bellman-Ford Algorithm: Negative Cycles

- If V-th iteration reduces the distance of a vertex, this means that there is a shortest path with at least V edges which implies that there is a negative cycle.
- Consider the graph with vertices s, u, v, and t and assume we have run (V-1 = 3) iterations.
- In the 4th iteration, the weight of at least one vertex will be reduced (due to the presence of a negative cycle).
- Important: Bellman-Ford Algorithm finds negative cycles only if such cycle is reachable from the source vertex
  - E.g., if x is the source vertex, the algorithm will not detect the negative cycle

Bellman-Ford algo

unreachable -ve cycle



All-Pairs Shortest

Problem

- Return shortest distances between **all** pairs of vertices in a connected graph.

For unweighted graphs:

- For each vertex v in the graph
  - Call Breadth-First Search for v

Time complexity:

$O(V(V+E)) = O(V^2 + EV) \rightarrow O(EV)$ [for connected graphs $O(V) \leq O(E)$]

For dense graphs: E is $O(V^2)$, therefore total cost is $O(V^3)$ for dense graphs

For weighted graphs (with non-negative weights):

- For each vertex v in the graph
  - Call Dijkstra's algorithm for v

Time complexity:

$O(V(E \log V)) = O(EV \log V)$

For dense graphs: $O(V^3 \log V)$

For weighted graphs (allowing negative weights):

- For each vertex v in the graph
  - Call Bellman-Ford algorithm for v

Time complexity:

$O(V(VE)) = O(V^2 E)$

For dense graphs: $O(V^4)$

Can we do better?

- Yes, Floyd-Warshall Algorithm returns all-pairs shortest distances in $O(V^3)$ for graphs allowing negative weights.

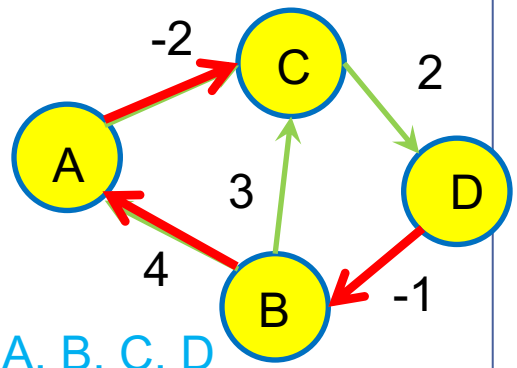
Outline

1. Shortest path in a graph with negative weights
2. All-pairs shortest paths
3. Transitive Closure

Floyd-Warshall Algorithm

- Initialize adjacency matrix called dist[][] considering adjacent edges only
- For each vertex k in the graph
  - For each pair of vertices i and j in the graph
    - If dist(i → k → j) is smaller than the current dist(i→j)
      - Update/create shortcut i →j with weight equal to dist(i→k→j)

      i.e., update dist[i][j] = dist[i][k] + dist[k][j]



Assume that the outer for-loop will access vertices in the order A, B, C, D

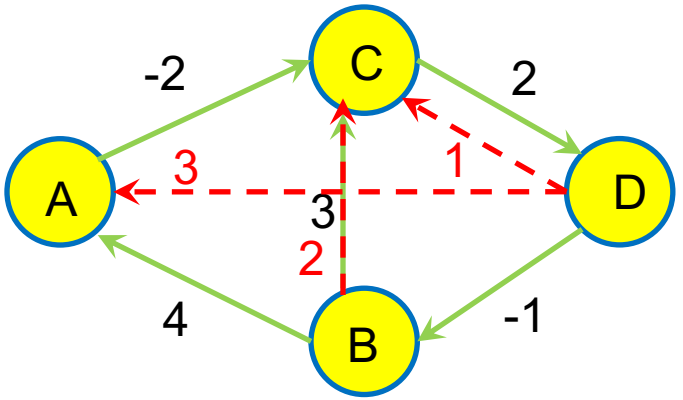
First iteration of outer loop (i.e., k is A):

Which shortcut(s) i→j is/are updated/created after the execution of the **inner** for-loop?

Second iteration of outer loop (i.e., k is B):

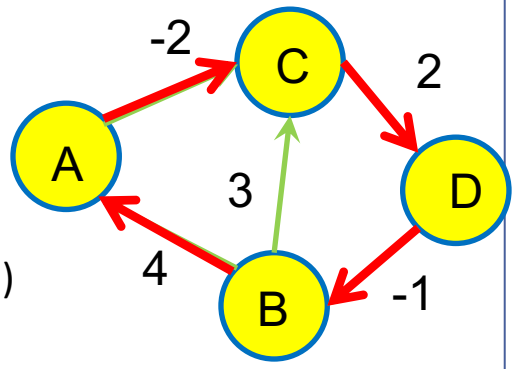
Which shortcut(s) i→j is/are updated/created after the execution of the **inner** for-loop?

	A	B	C	D
A	0	∞	-2	∞
B	4	0	2	∞
C	∞	∞	0	2
D	3	-1	1	0



Floyd-Warshall Algorithm

- Create the adjacency matrix called dist[][]
- For each vertex k in the graph
  - For each pair of vertices i and j in the graph
    - If dist(i → k → j) is smaller than the current dist(i→j)
      - Update/create shortcut i →j with weight equal to dist(i→k→j)
      - i.e., update dist[i][j] = dist[i][k] + dist[k][j]



Assume that the outer for-loop will access vertices in the order A, B, C, D

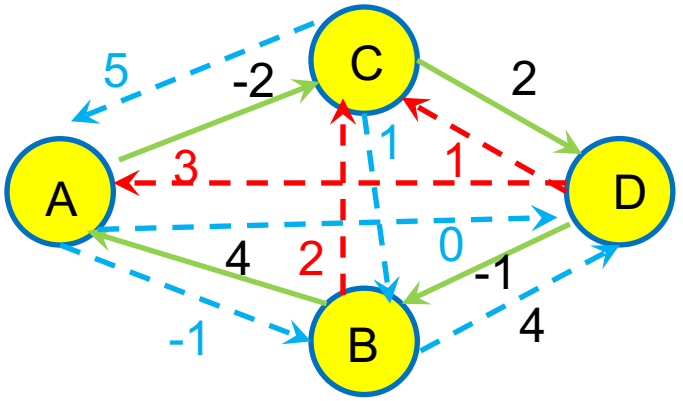
Third iteration of outer loop (i.e., k is C):

Which shortcut(s) i→j is/are updated after the execution of the **inner** for-loop?

Fourth iteration of outer loop (i.e., k is D):

Which shortcut(s) i→j is/are updated after the execution of the **inner** for-loop?

	A	B	C	D
A	0	~~∞~~ -1	-2	~~∞~~ 0
B	4	0	2	~~∞~~ 4
C	~~∞~~ 5	~~∞~~ -1	0	2
D	3	-1	1	0



Floyd-Warshall Algorithm

```
dist[][] = E #  Initialize adjacency matrix using E
for vertex k in 1..V:
   #Invariant: dist[i][j] corresponds to the shortest path from i
to j considering the intermediate vertices 1 to k-1
   for vertex i in 1..V:
      for vertex j in 1..V:
         dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Time Complexity:

$O(V^3)$

Space Complexity:

$O(V^2)$

Floyd-Warshall Algorithm: Correctness

```
Invariant: dist[i][j] corresponds to the shortest path from i to j
considering only intermediate vertices 1 to k-1
```

What is the shortest distance from i to j considering only intermediate vertices 1-3 (e.g., k = 4)

- 13 (i → 2 → 3 → j)

What is the shortest distance from i to 4 considering only intermediate vertices 1-3 (e.g., k =4)

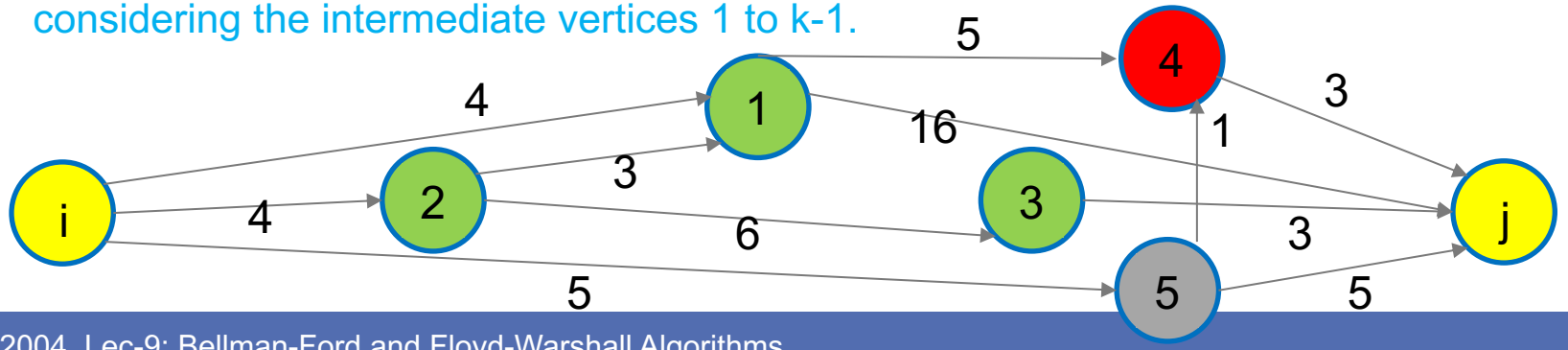
- 9 (i → 1 → 4)

Base Case (k=1):

- It is true because dist[][] is initialized based only on the adjacent edges

Inductive Step (example k = 4):

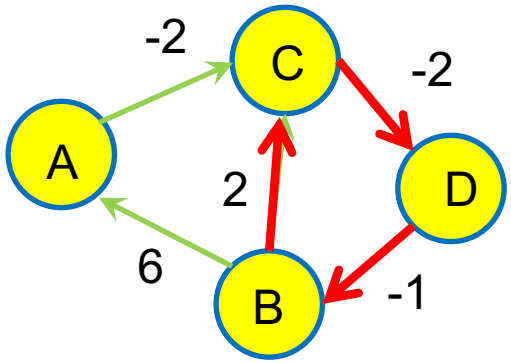
- Assume dist[i][j] is the shortest path from i to j considering only vertices 1 to k-1
- If k-th vertex is to improve on the known path from i to j, it can only be by going from i to k and then k to j (possibly via vertices in 1 to k-1)
  - Thus, minimum of dist(i→k→j) and dist(i→j) gives the minimum distance from i to j considering the intermediate vertices 1 to k-1.



Floyd-Warshall Algorithm: Negative Cycles

- If there is a negative cycle, there will be a vertex v such that dist[v][v] is negative.
- Look at the diagonal of the adjacency matrix and return error if a negative value is found

	A	B	C	D
A	0	-5	-3	-5
B	5	-1	1	-1
C	3	-3	-1	-3
D	4	-2	0	-2



Floyd-Warshall Algorithm: Negative Cycles

- Bellman-Ford algorithm may not find a negative cycle if it is not reachable from the source vertex
- Floyd-Warshall guarantees that it can find negative cycles

Bellman-Ford algo

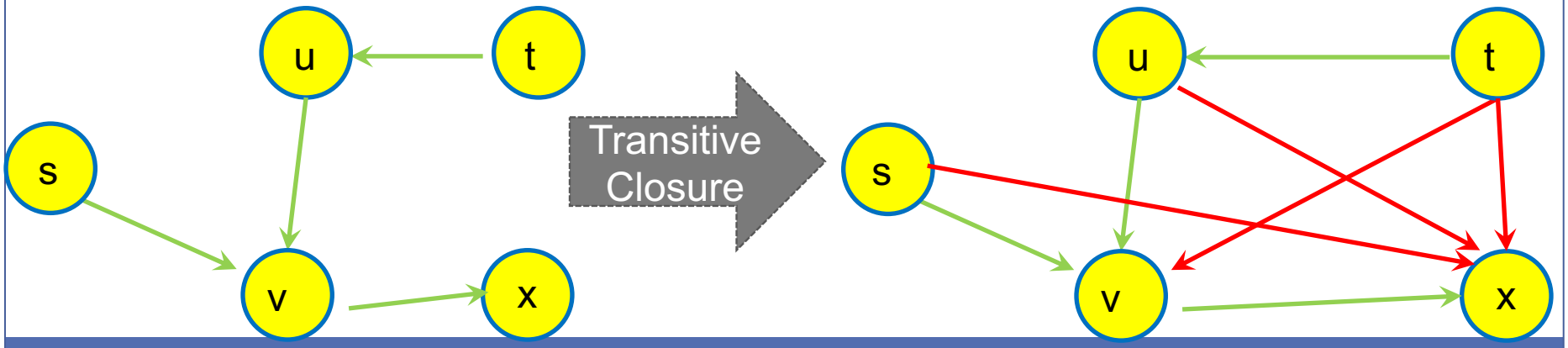
unreachable -ve cycle

Outline

1. Shortest path in a graph with negative weights
2. All-pairs shortest paths
3. Transitive Closure

Transitive Closure of a Graph

- Given a graph G = (V,E), its transitive closure is another graph (V,E') that contains the same vertices V but contains an edge from every u to v if there is a path from u to v in the original graph.
- **Applications:** What are the pairs of vertices (u,v) in the graph such that one can reach from u to v.
  - E.g., given flights between different cities, can I go from city A to city B (regardless of the number of flights I need to take), or where can/cannot I go from city A.
- **Solution:** Assign each edge a weight 1 and then apply Floyd-Warshall algorithm. If dist[i][j] is not infinity, this means there is a path from i to j in the original graph. (Or just maintain True and False as shown next)



Floyd-Warshall Algorithm for Transitive Closure

```
# Modify Floyd-Warshall Algorithm to compute Transitive Closure
# initialization
for vertex i in 1..V:
        for vertex j in 1..V:
            if there is an edge between i and j or i == j:
                TC[i][j] = True
            else:
                TC[i][j] = False
for vertex k in 1..V:
# Invariant: TC[i][j] corresponds to the existence of path from i to j considering the intermediate vertices 1 to k-1
    for vertex i in 1..V:
        for vertex j in 1..V:
            TC[i][j] = TC[i][j] or (TC[i][k] and TC[k][j])
```

Time Complexity:

$O(V^3)$

Space Complexity:

$O(V^2)$

Summary

Take home message

- Dijkstra's algorithm works only for graphs with non-negative weights
- Bellman-Ford computes shortest paths in graphs with negative weights in O(VE) and can also detect the negative cycles that are reachable
- Floyd-Warshall Algorithm computes all-pairs shortest paths and transitive closure in $O(V^3)$

Things to do (this list is not exhaustive)

- Go through recommended reading and make sure you understand why the algorithms are correct
- Implement Bellman-Ford and Floyd-Warshall Algorithms

Coming Up Next

- Minimum spanning trees