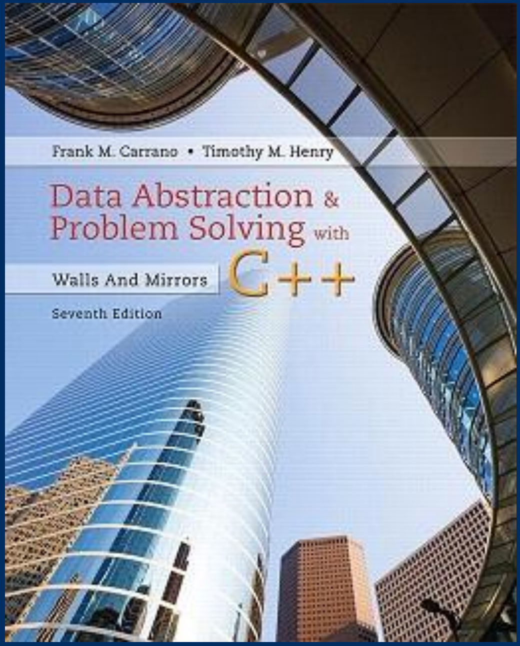
Graphs – Shortest Path

CS 302 - Data Structures

M. Abdullah Canbaz



Modified from authors' slides

Reminders

- Assignment 7 is available
  - Due Wednesday, May 7th at 2pm
  - TA
    - Athanasia Katsila,
      Email: *akatsila [at] nevada {dot} unr {dot} edu*,
      Office Hours: Thursdays, 10:30 am - 12:30 pm at SEM 211

- Assignment 8 is available
  - Due Wednesday, May 16th at 2pm
  - TA
    - Shehryar Khattak,
      Email: *shehryar [at] nevada {dot} unr {dot} edu*,
      Office Hours: Friday, 11:00 am - 1:00 pm at ARF 116

Graph Algorithms

- **Depth-first search**
  - Visit all the nodes in a branch to its deepest point before moving up
- **Breadth-first search**
  - Visit all the nodes on one level before going to the next level
- **Single-source shortest-path**
  - Determines the shortest path from a designated starting node to every other node in the graph

Single Source Shortest Path

Austin		Austin	
	160 miles		200 miles
Houston		Dallas	
	800 miles		780 miles
Atlanta		Denver	
	600 miles		1400 miles
Washington		Atlanta	
			600 miles
		Washington	
Total miles	1560 miles	Total Miles	2980 miles

Shortest-path problem

- There might be multiple paths from a source vertex to a destination vertex
- *Shortest path*: the path whose total weight (i.e., sum of edge weights) is minimum

Austin→Houston→Atlanta→Washington:

1560 miles

Austin→Dallas→Denver→Atlanta→Washington:

2980 miles

Variants of Shortest Path

- **Single-pair** shortest path
  - Find a shortest path from **u** to **v**
    - for given vertices **u** and **v**

- **Single-source** shortest paths
  - G = (V, E) $\Rightarrow$ find a shortest path from a given source vertex **s** to each vertex **v** $\in$ V

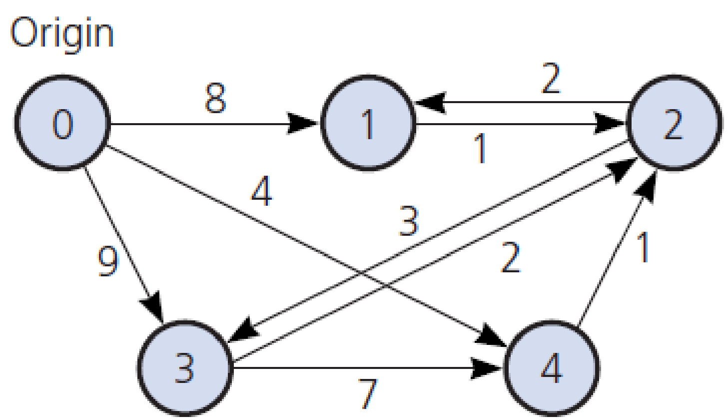
Variants of Shortest Paths (cont’d)

- **Single-destination** shortest paths
  - Find a shortest path to a given destination vertex **t** from each vertex v
  - Reversing the direction of each edge → single-source
- **All-pairs** shortest paths
  - Find a shortest path from u to v for every pair of vertices u and v

Shortest Paths

- The shortest path between two vertices in a weighted graph
  - Has the smallest edge-weight sum

(a)



(b)

	0	1	2	3	4
0	∞	8	∞	9	4
1	∞	∞	1	∞	∞
2	∞	2	∞	3	∞
3	∞	∞	2	∞	7
4	∞	∞	1	∞	∞

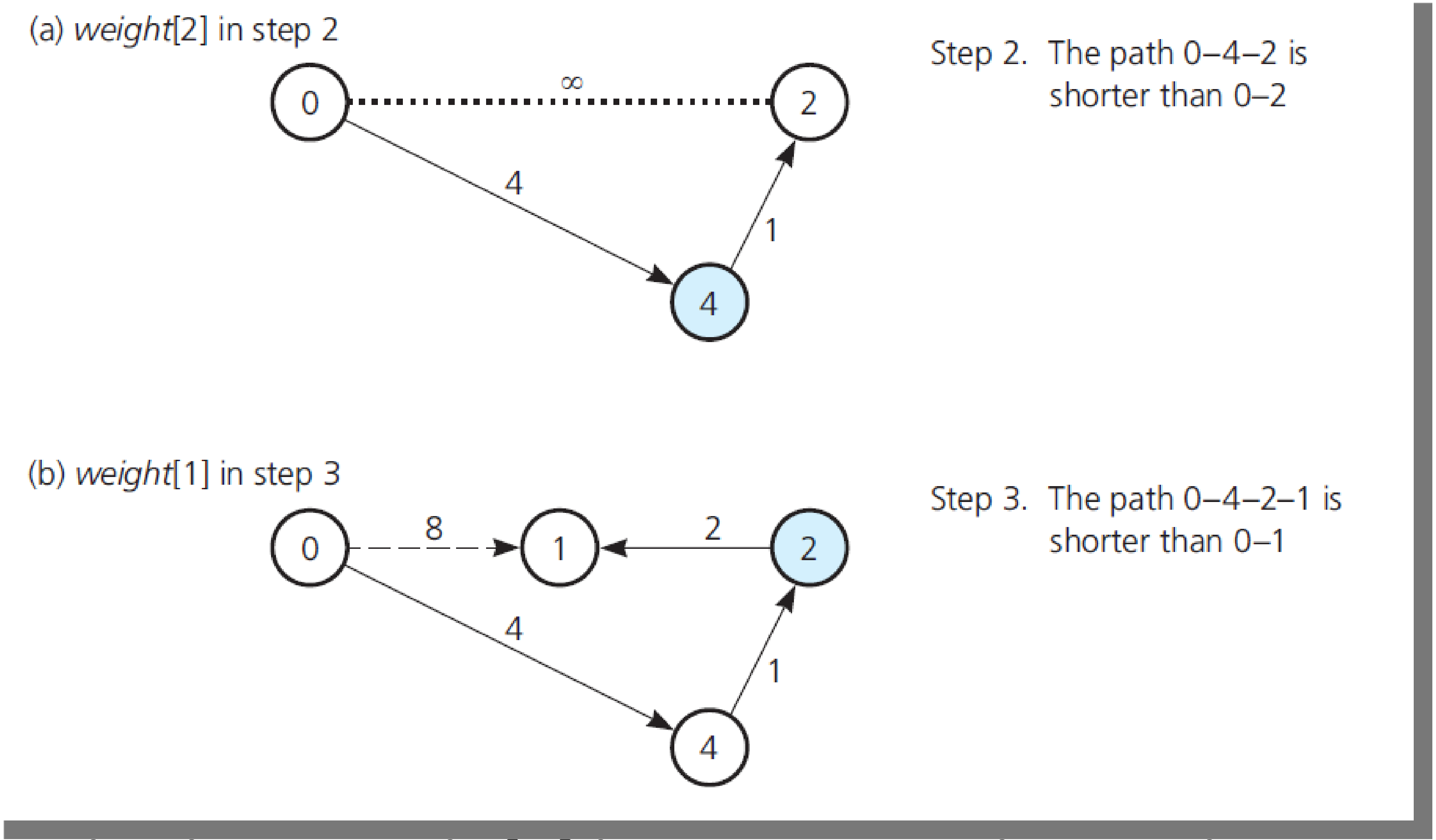
(a) A weighted directed graph and (b) its adjacency matrix

Shortest Paths

			weight				
Step	v	vertexSet	[0]	[1]	[2]	[3]	[4]
1	–	0	0	8	∞	9	4
2	4	0, 4	0	8	5	9	4
3	2	0, 4, 2	0	7	5	8	4
4	1	0, 4, 2, 1	0	7	5	8	4
5	3	0, 4, 2, 1, 3	0	7	5	8	4

- A trace of the shortest-path algorithm applied to the graph

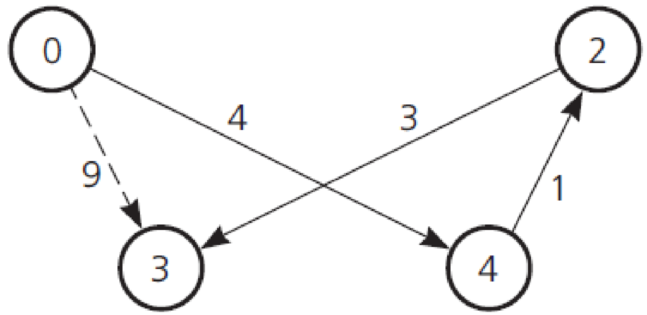
Shortest Paths



- Checking weight[u] by examining the graph: (a) weight[2] in step 2; (b) weight[1] in step 3;

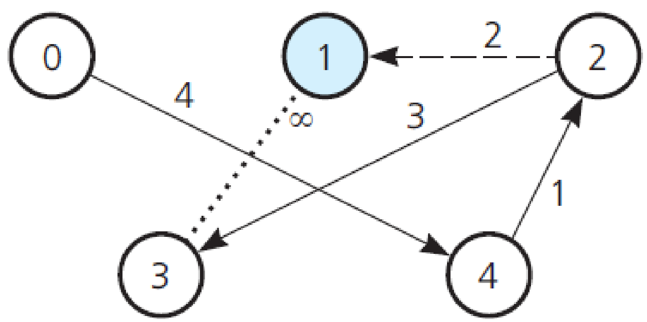
Shortest Paths

(c) *weight*[3] in step 3



Step 3 continued. The path 0–4–2–3 is shorter than 0–3

(d) *weight*[3] in step 4



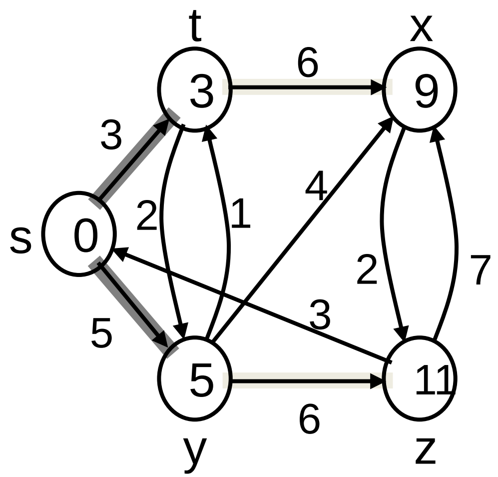
Step 4. The path 0–4–2–3 is shorter than 0–4–2–1–3

- Checking weight[u] by examining the graph: (c) weight[3] in step 3; (d) weight[3] in step 4

Notation

- **Weight of path** p = ⟨v_0, v_1, . . . , v_k⟩

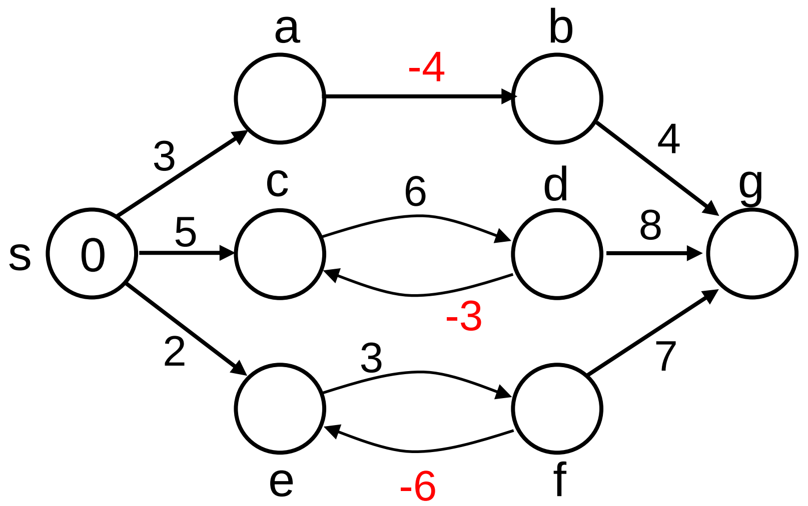
$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$



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

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

Negative Weights and Negative Cycles



- Negative-weight edges may form negative-weight cycles.

- If negative cycles are reachable from the source, the shortest path is **not well defined.**
  – i.e., keep going around the cycle, and get $w(s, v) = -\infty$ for all v on the cycle

Could shortest path solutions contain cycles?

- Negative-weight cycles
  - Shortest path is not well defined
- Positive-weight cycles:
  - By removing the cycle, we can get a shorter path
- Zero-weight cycles
  - No reason to use them; can remove them to obtain a path with same weight

Shortest-path algorithms

- Solving the shortest path problem in a brute-force manner requires enumerating all possible paths.
  - There are **O(V!)** paths between a pair of vertices in an acyclic graph containing V nodes.

- We will discuss two algorithms
  - **Dijkstra**'s algorithm
  - **Bellman-Ford**'s algorithm

Shortest-path algorithms

- Dijkstra's and Bellman-Ford's algorithms are "greedy" algorithms!
  - Find a **"globally"** optimal solution by making **"locally"** optimum decisions.
- Dijkstra's algorithm
  - Does not handle negative weights.
- Bellman-Ford's algorithm
  - Handles negative weights but not negative cycles reachable from the source.

Shortest-path algorithms (cont'd)

- Both Dijkstra's and Bellman-Ford's algorithms are iterative:
  - Start with a shortest path **estimate** for every vertex: $d[v]$
  - Estimates are updated iteratively until convergence:

$$d[v] \rightarrow \delta(v)$$

Shortest-path algorithms (cont'd)

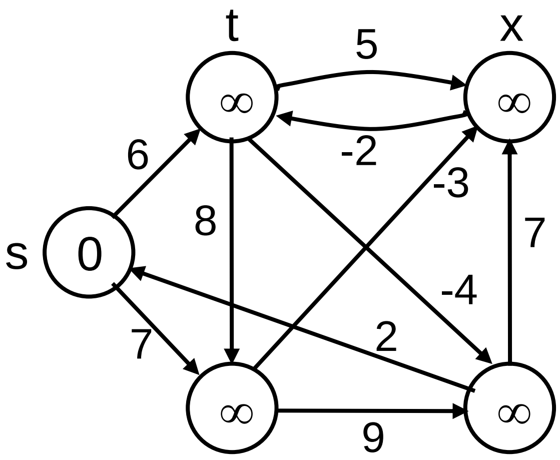
- Two common steps:

  (1) Initialization

  (2) Relaxation (i.e., update step)

Initialization Step

- Set d[s]=0
  - i.e., source vertex
- Set d[v]=∞ for $v \neq s$
  - i.e., large value



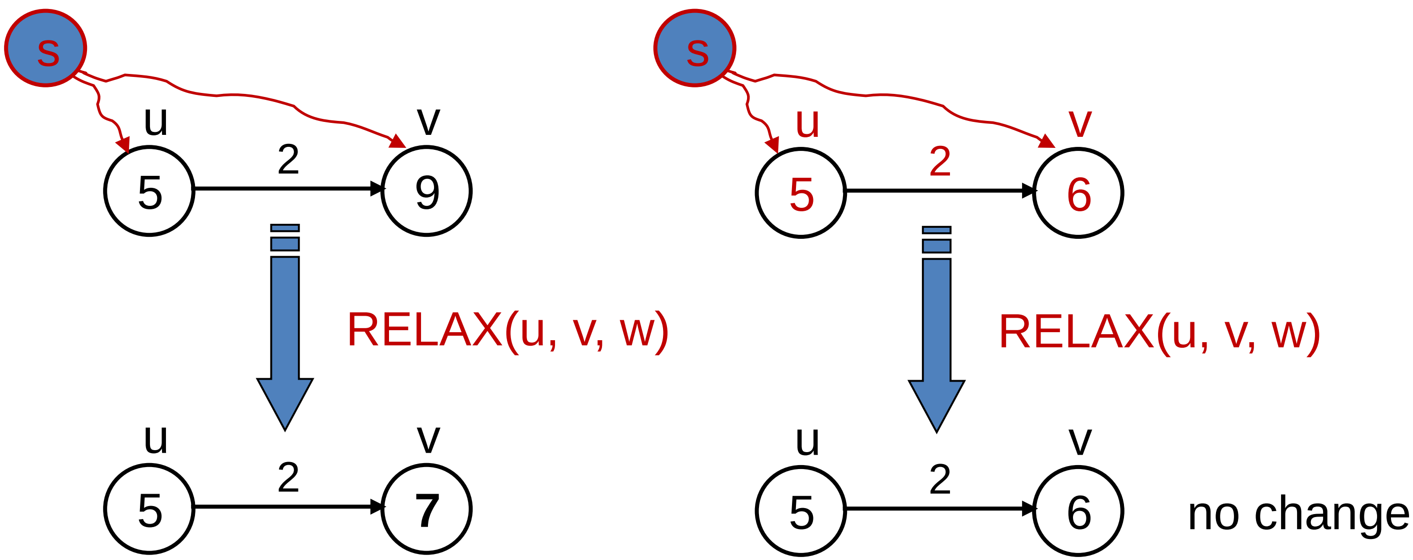
Relaxation Step

Relaxing an edge (u, v) implies testing whether we can improve the shortest path to v found so far by going through u:

If d[v] > d[u] + w(u, v)

we can improve the shortest path to v

$\Rightarrow$ d[v]=d[u]+w(u,v)



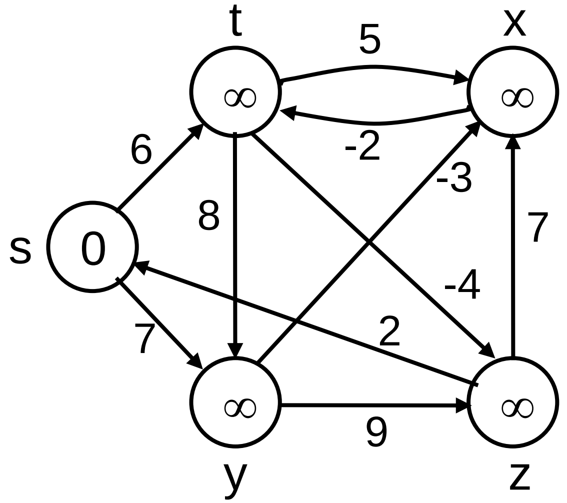
Bellman-Ford Algorithm

- Can handle negative weights
- Detects negative cycles reachable from the source
- Returns FALSE if negative-weight cycles are reachable from the source s $\Rightarrow$ **no solution**

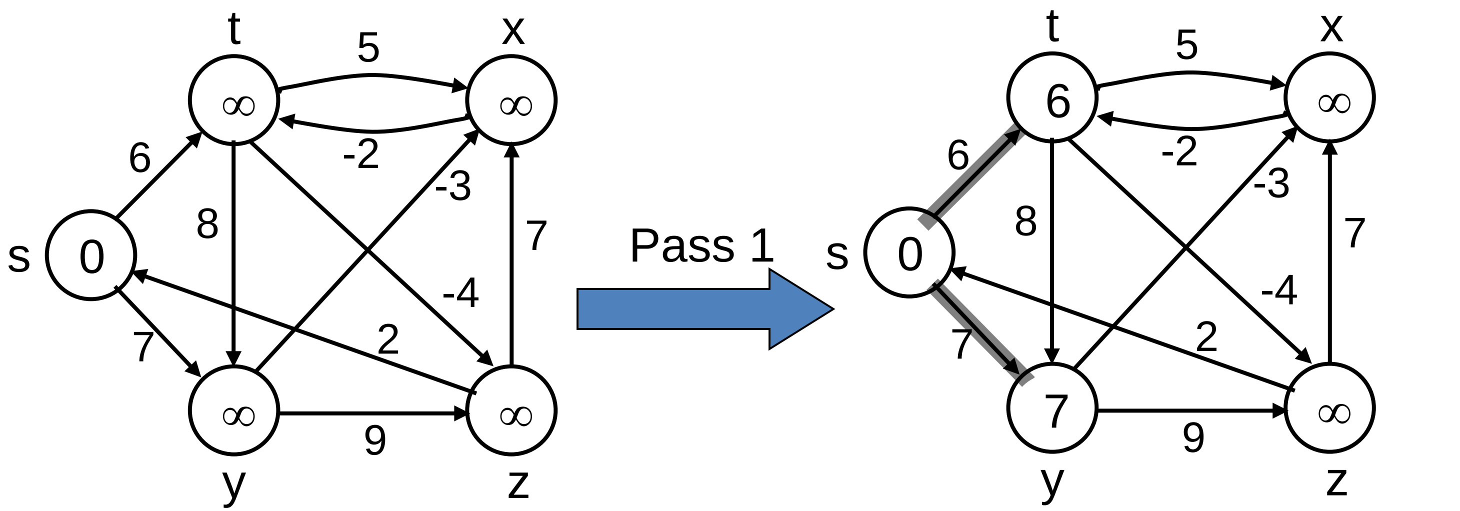
Bellman-Ford Algorithm (cont' d)

- Each edge is relaxed |V–1| times by making |V-1| passes over the whole edge set
  – to make sure that each edge is relaxed exactly |V – 1| times
- it puts the edges in an unordered list and goes over the list |V – 1| times

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)



Example

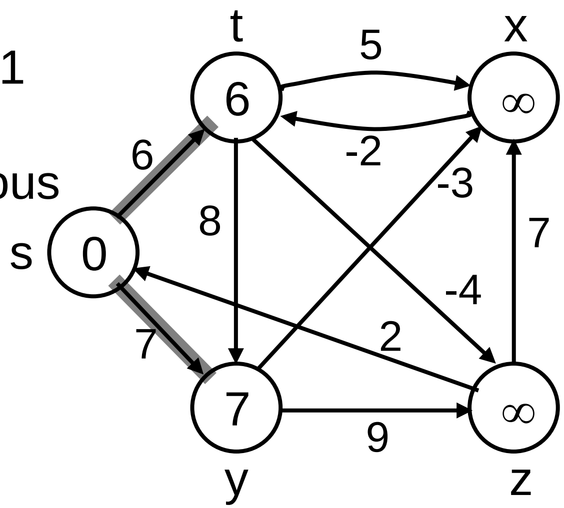


E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

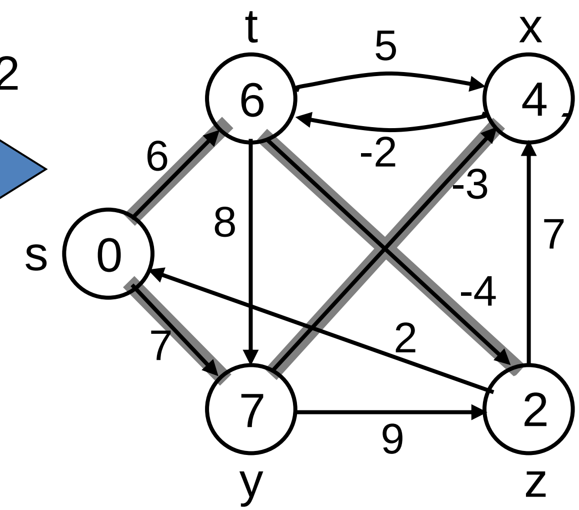
Example

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

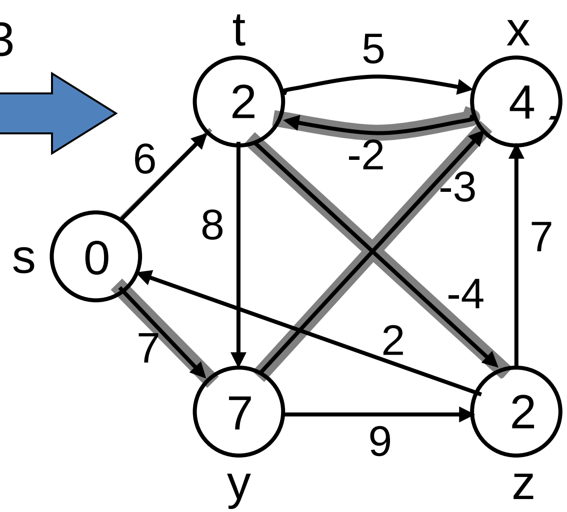
Pass 1 (from previous slide)



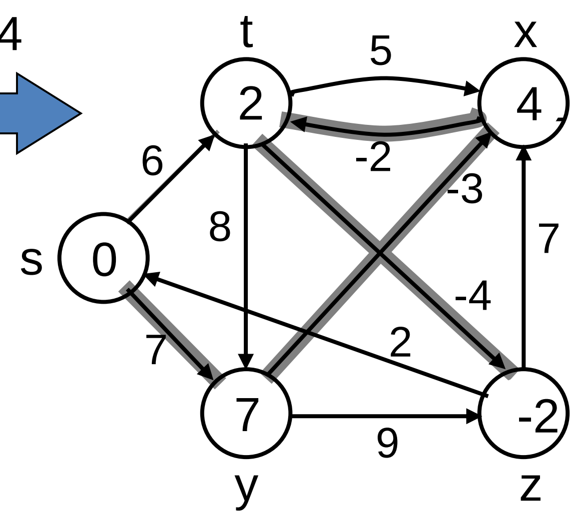
Pass 2



Pass 3



Pass 4



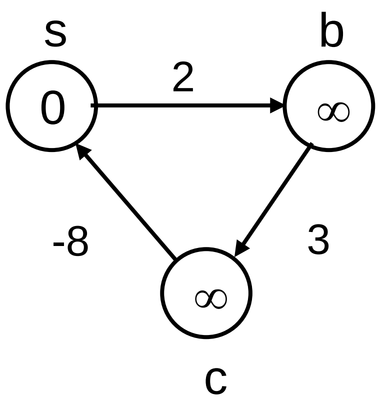
Detecting Negative Cycles: needs an extra iteration

for each edge (u, v) ∈ E do

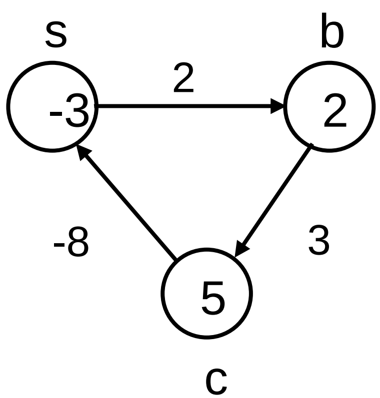
  if d[v] > d[u] + w(u, v)

    then return FALSE

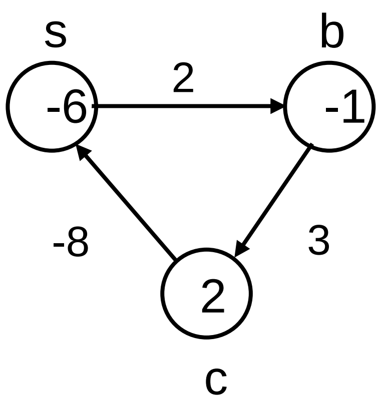
return TRUE



1st pass



2nd pass



(s,b) (b,c) (c,s)

Consider edge (s, b):

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

d[b] > d[s] + w(s, b)
  → d[b]=-4
(d[b] keeps changing!)

BELLMAN-FORD Algorithm

```
INITIALIZE-SINGLE-SOURCE(V, s)          O(V)
for i ← 1 to |V| - 1                    O(V)
    for each edge (u, v) ∈ E            O(E)    O(VE)
        RELAX(u, v, w)
for each edge (u, v) ∈ E                O(E)
    if d[v] > d[u] + w(u, v)
        return FALSE
return TRUE
```

Time: **O(V+VE+E)=O(VE)**

Dijkstra's Algorithm

- Cannot handle negative-weights!
  - $w(u, v) > 0, \ \forall (u, v) \in E$

- Each edge is relaxed **only once!**

Dijkstra's Algorithm (cont'd)

- At each iteration, it maintains two sets of vertices:

V

S

V-S

$d[v]=\delta(v)$

$d[v]\geq\delta(v)$

estimates have converged to the shortest path solution

estimates have not converged yet

Initially, S is empty

Dijkstra's Algorithm (cont.)

- Vertices in V–S reside in a **min-priority queue** **Q**
  - Priority of u determined by d[u]
  - The "highest" priority vertex will be the one having the smallest **d[u]** value.

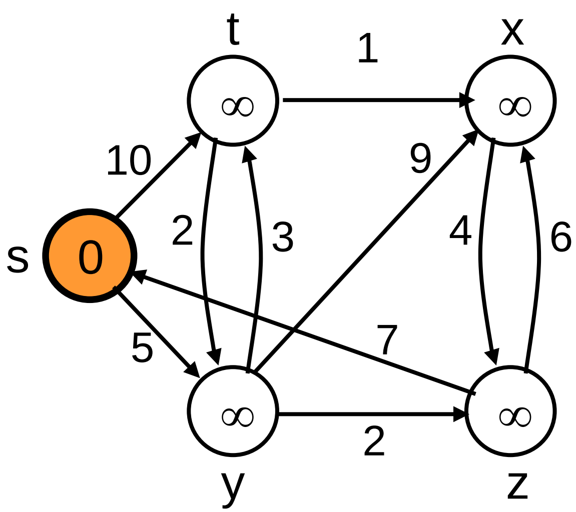
Steps

1) Extract a vertex u from Q
2) Insert u to S
3) Relax all edges leaving u
4) Update Q

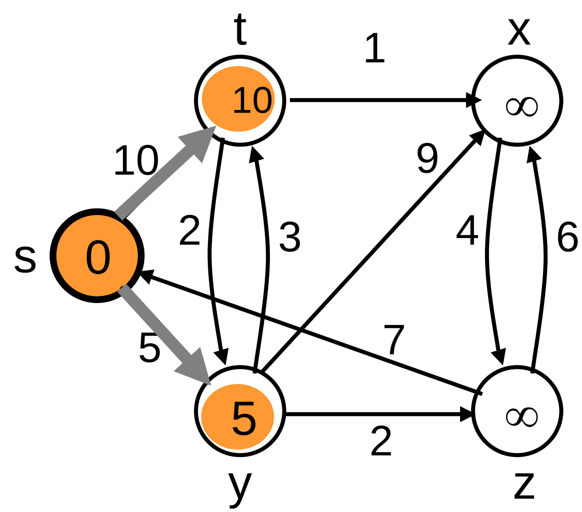
Dijkstra (G, w, s)

S=<> Q=<s,t,x,z,y>

Initialization

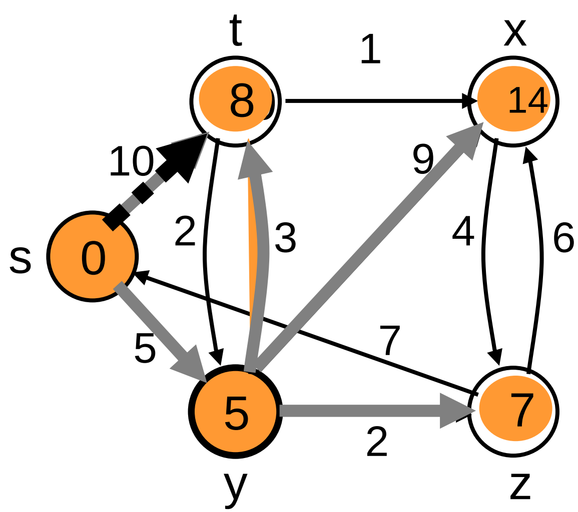


S=<s> Q=<y,t,x,z>

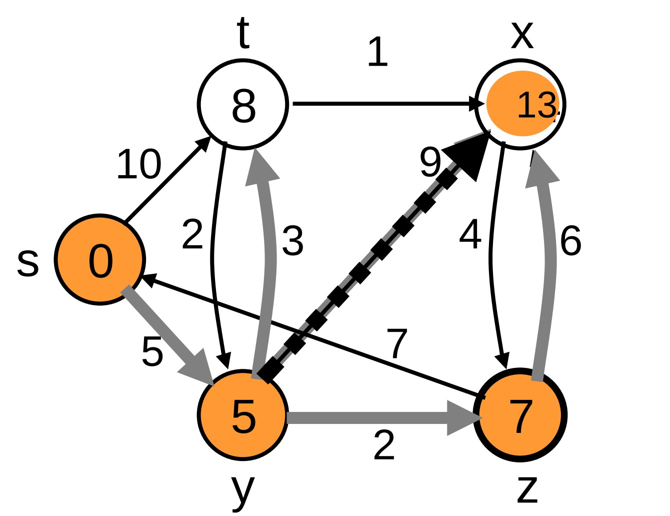


Example (cont.)

S=<s,y> Q=<z,t,x>

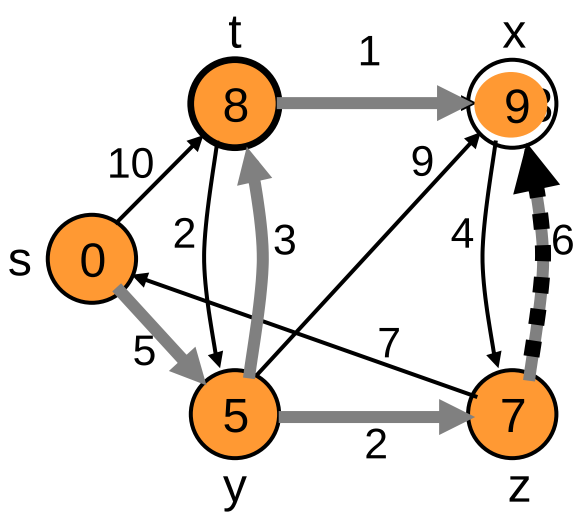


S=<s,y,z> Q=<t,x>

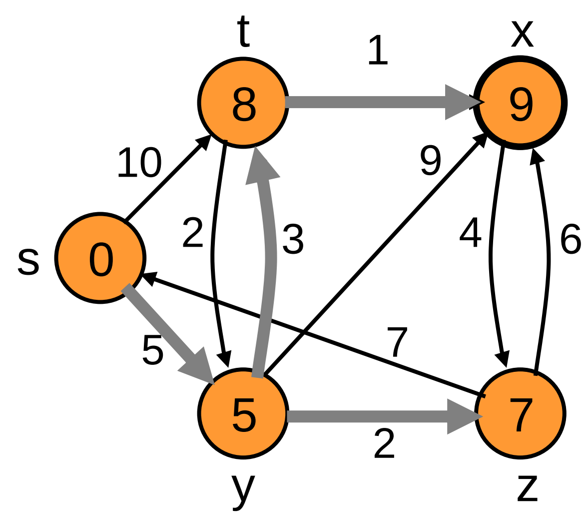


Example (cont.)

S=<s,y,z,t> Q=<x>



S=<s,y,z,t,x> Q=<>



Note: use back-pointers to recover the shortest path solutions!

Dijkstra (G, w, s)

INITIALIZE-SINGLE-SOURCE(V, s) ← O(V)

S ← ∅

build priority heap

Q ← V[G] ← O(V logV)

while Q ≠ ∅ ← O(V) times

u ← EXTRACT-MIN(Q) ← O(logV)

S ← S ∪ {u}

for each vertex v ∈ Adj[u] ← $O(E_{vi})$

RELAX(u, v, w)

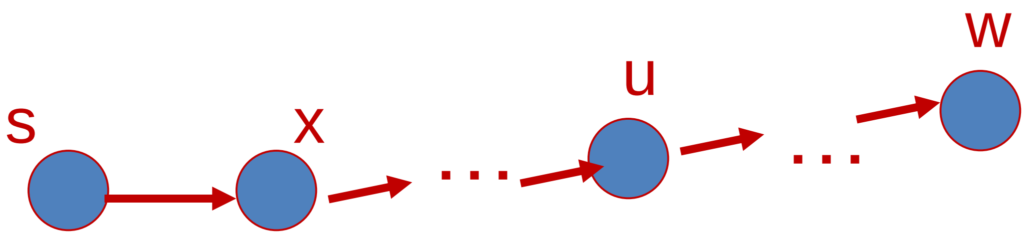
Update Q (DECREASE_KEY) ← O(logV)

$\Big\}$ $O(E_{vi} \log V)$

Overall: $O(V+2V\log V+(E_{v1}+E_{v2}+\ldots)\log V) = O(V\log V+E\log V)=O(E\log V)$

Improving Dijkstra's efficiency

- Suppose the shortest path from s to w is the following:



- If u is the i-th vertex in this path, it can be shown that d[u] → δ (u) at the i-th iteration:
  - move u from **V-S** to **S**
  - **d[u]** never changes again

Add a flag for efficiency!

INITIALIZE-SINGLE-SOURCE(V, s)

S ← $\emptyset$

Q ← V[G]

while Q $\neq \emptyset$

u ← EXTRACT-MIN(Q)

S ← S $\cup$ {u}; ← mark u

for each vertex v $\in$ Adj[u]

If v not marked

RELAX(u, v, w)

Update Q (DECREASE_KEY)

Dijkstra vs Bellman-Ford

- Bellman-Ford

$O(VE)$

V^2 if G is sparse: $E=O(V)$

V^3 if G is dense: $E=O(V^2)$

- Dijkstra

$O(E\log V)$

$V\log V$ if G is sparse: $E=O(V)$

$V^2\log V$ if G is dense: $E=O(V^2)$

Revisiting BFS

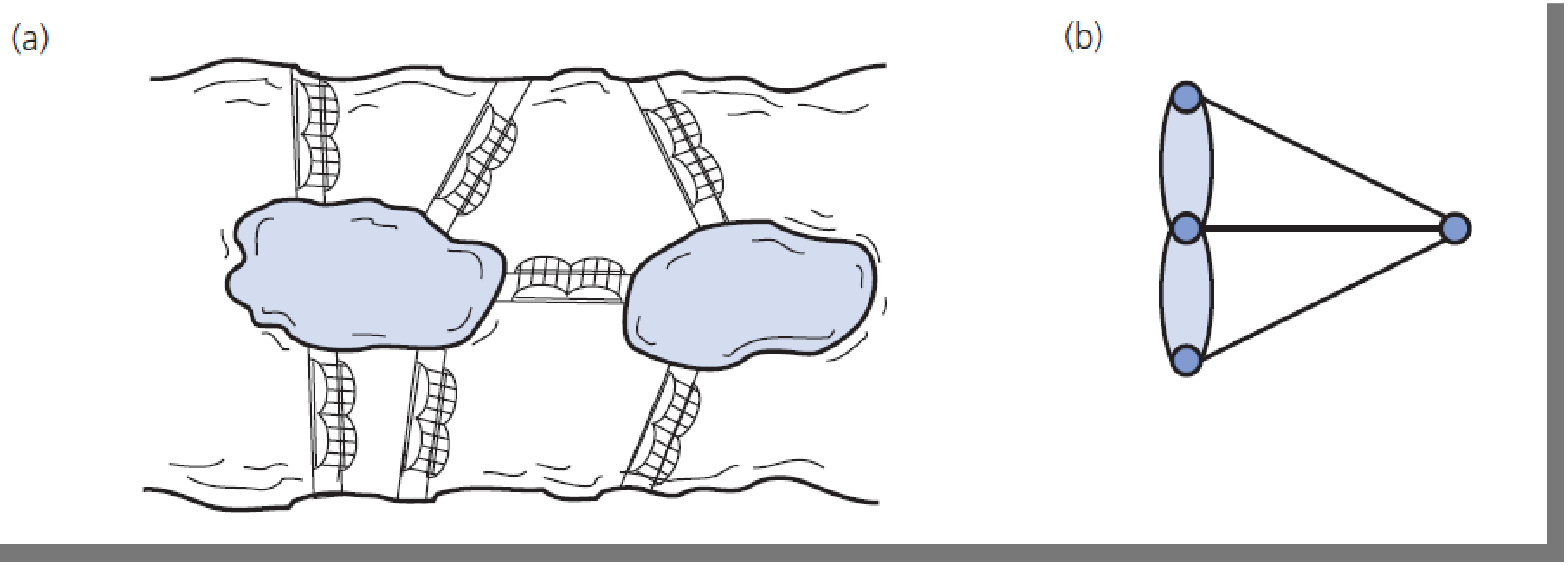
- BFS can be used to solve the shortest path problem when the graph is weightless or when all the weights are equal.
  - Path with lowest number of edges
    - i.e., connections

- Need to “mark” vertices before Enqueue!
  - i.e., do not allow duplicates

Circuits

- Circuit
  - Another name for type of cycle common in statement of certain types of problems
  - Typical circuits either visit every vertex once or every edge once
- Euler Circuit
  - Begins at vertex v
  - Passes through every edge exactly once
  - Terminates at v

Circuits



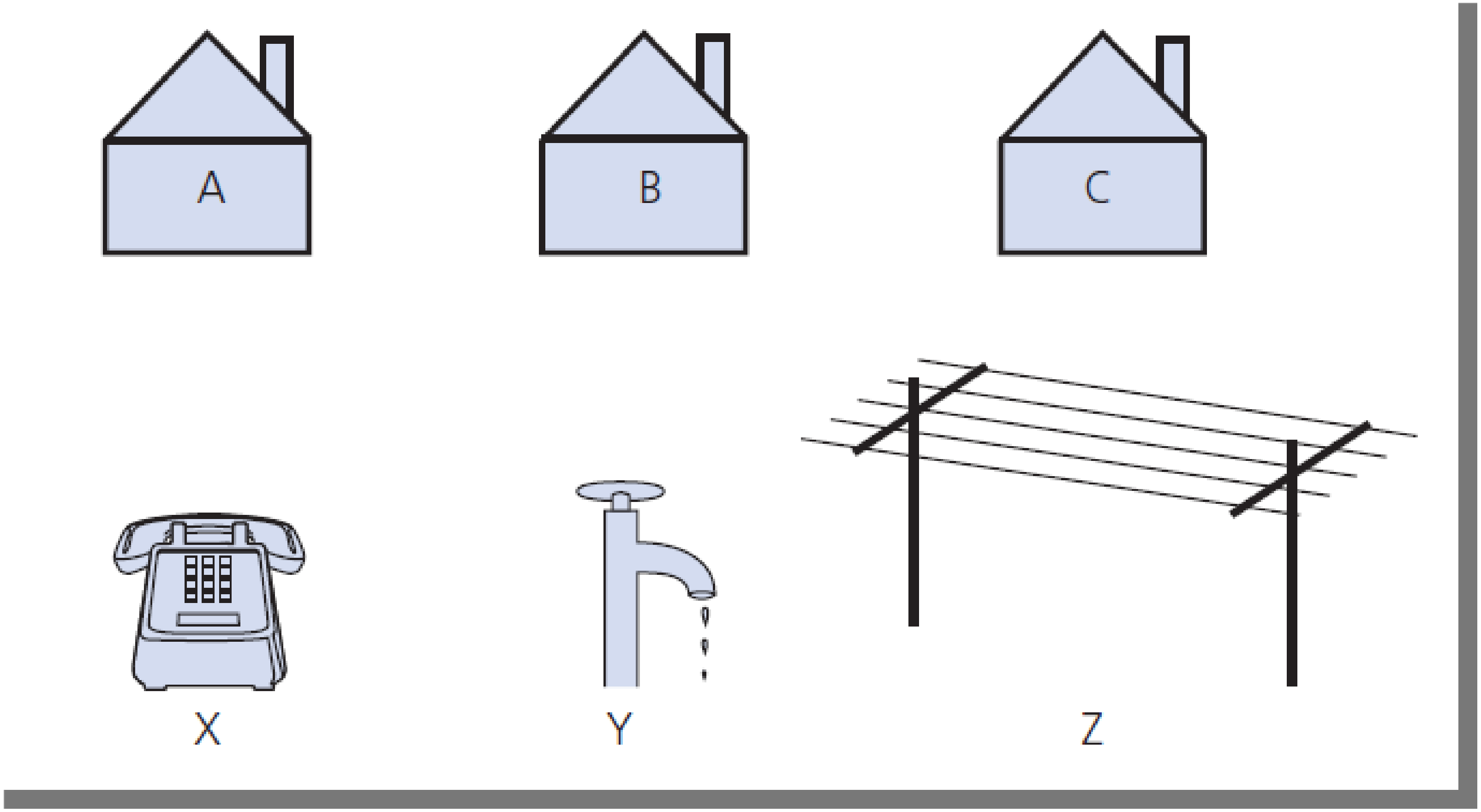
- (a) Euler’s bridge problem and
  (b) its multigraph representation

Some Difficult Problems

- Hamilton circuit
  - Begins at vertex *v*
  - Passes through every vertex exactly once
  - Terminates at *v*
- Variation is "traveling salesperson problem"
  - Visit every city on his route exactly once
  - Edge (road) has associated cost (mileage)
  - Goal is determine least expensive circuit

Some Difficult Problems

- The three utilities problem



Remember

also


SEMESTER IS ALMOST OVER?
ASIA
EVERYTHING IS DUE NEXT WEEK
quickmeme.com

and


ONLY 1 CHEAT SHEET ALLOWED
CHALLENGE ACCEPTED
memecrunch.com

finally

Its due until 11:59 PM on Wed, May 9, 2018.


YOU HAVE 7 DAYS
TO SAVE THE SEMESTER
LET THE GAMES BEGIN