Computational Problem: Finding Shortest Paths

AUBURN UNIVERSITY

Hugh Kwon

Slides adapted from Dr. Debswapna Bhattacharya's class

- Given a weighted directed graph g=(V,E), with each edge having real-valued weights, the weight of a path from node u to node v is the sum of the weights of its edges. The shortest path from u to v is the path with the minimum path weight (if no such path exists, its weight is undefined

- Given a single start node s, find shortest paths from s to all other nodes.

- The breadth-first search algorithm finds the shortest paths (i.e. paths with the least number of edges) from the start node to all other nodes when edges have no weights.
- This is the same as finding the shortest paths if all edge weights are 1.
- Thinking Assignment: Will the BFS algorithm find the shortest paths if all edges had the same constant weight c?

- An algorithm for finding SSSPs can also solve the following problems:

1. Single Destination Shortest Paths (how?)
2. Single Pair Shortest Path (how?)
3. All Pairs Shortest Paths (how?)

- **Subpaths of shortest paths are shortest paths**
  Lemma 24.1: If $p=<v_0,\ldots,v_k>$ is a shortest path from v_0 to v_k in a weighted directed graph $G=(V,E)$, then, for any i and j such that $0 \le i \le j \le k$, let p_{ij} be the subpath of p from v_i to v_j. Then p_{ij} is the shortest path from v_i to v_j.

- Proof: By contradiction.

- Edges with negative weights pose no problem.
- Shortest path weight may be negative in this case.
- But if there is a negative weight cycle that is reachable from the start node s, it is a problem! Why?
- In this case the shortest path is not defined and shortest path weight is $-\infty$

- Thus, a shortest path cannot contain a negative weight cycle.

- Can a shortest path contain any positive or zero weight cycles. Why or why not?

- Thus, we can conclude that shortest paths must be simple paths (a simple path is one that contains no cycles).

- Any simple (acyclic) path in a graph G with n nodes and m edges can only contain at most n nodes and n-1 edges.

- Therefore, we can also conclude that shortest paths can have at most n nodes and at most n-1 edges.

- Use the attribute predecessor or previous (π) attached to nodes.

- Use the attribute distance (d) attached to nodes.

INITIALIZE-SINGLE-SOURCE (G,s) Complexity $\Theta(n)$

1. for each node v in G.V
2. v.d = ∞
3. v.π = NIL
4. s.d = 0

- v.d = upper bound on the weight of a shortest path from s to v
- v. π = previous node on the shortest path from s to v
- G is the adjacency list representation; Each node in the adjacency list of a vertex contains the edge weight.
- Relaxing an edge (u,v): testing if the currently known shortest path from s to v can be improved by going through the currently known shortest path from s to u and then from u to v along the edge (u,v). This is the only way in which a current estimate of a shortest path can change.

RELAX (u,v,w) Complexity $\Theta(1)$

1. if v.d > u.d+w(u,v)
2. v.d = u.d + w(u,v)
3. v.π = u

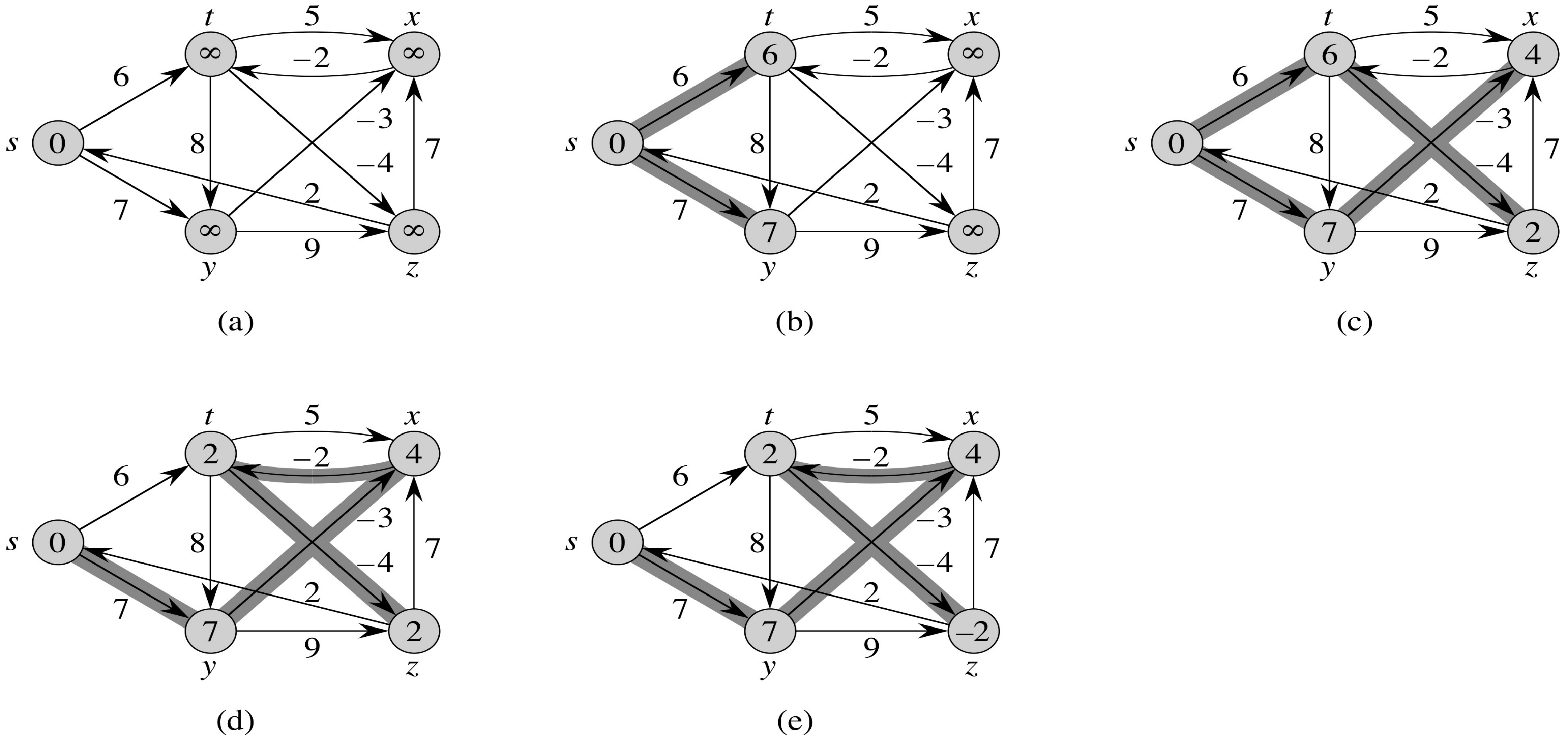
Bellman-Ford Algorithm

- Returns a Boolean indicating whether there is a negative weight cycle reachable from source node s. If not, it determines shortest paths from s to all other vertices.

```
BELLMAN-FORD(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
for i=1 to n-1
    for each edge (u,v) in G.E
        RELAX (u,v,w)
for each edge (u,v) in G.E
    if v.d > u.d+w(u,v)
        return false
return true
```

Complexity: $O(nm)$ where $n=|V|$ and $m=|E|$ [or $O(VE)$]

Bellman-Ford Algorithm



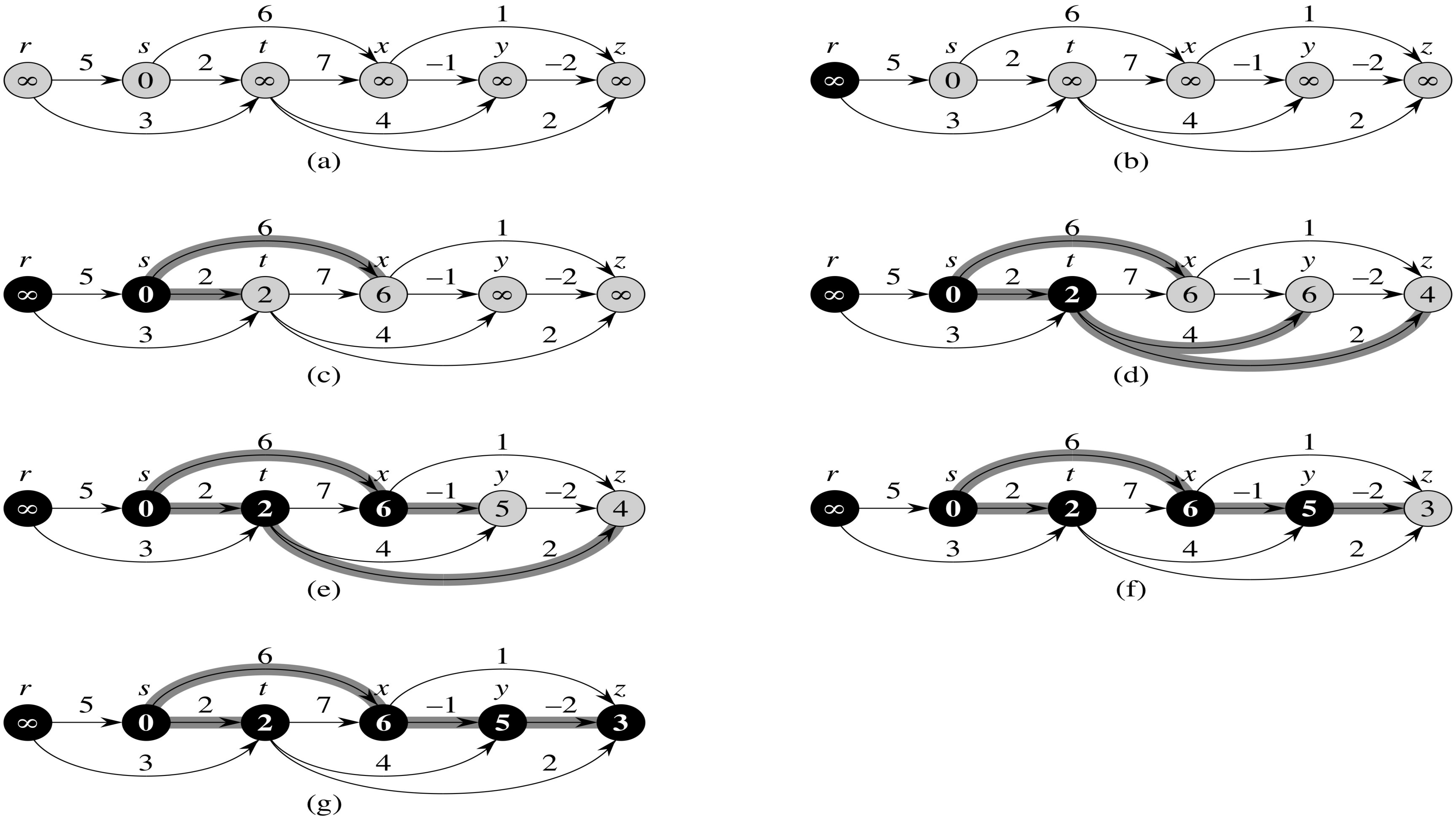
DAG-SHORTEST-PATHS(G,w,s)

1. topologically sort vertices of G
2. INITIALIZE-SINGLE-SOURCE(G,s)
3. for each vertex u taken in topological order
4.   for each vertex v in G.Adj[u]
5.   RELAX(u,v,w)

- By relaxing the edges of a weighted directed acyclic graph according to the topological order of its nodes, shortest paths can be computed much faster, in $\Theta(m+n)$ time.

- Note that s can be any vertex, not necessarily the first in the topological order

SSSP Algorithm for DAGs

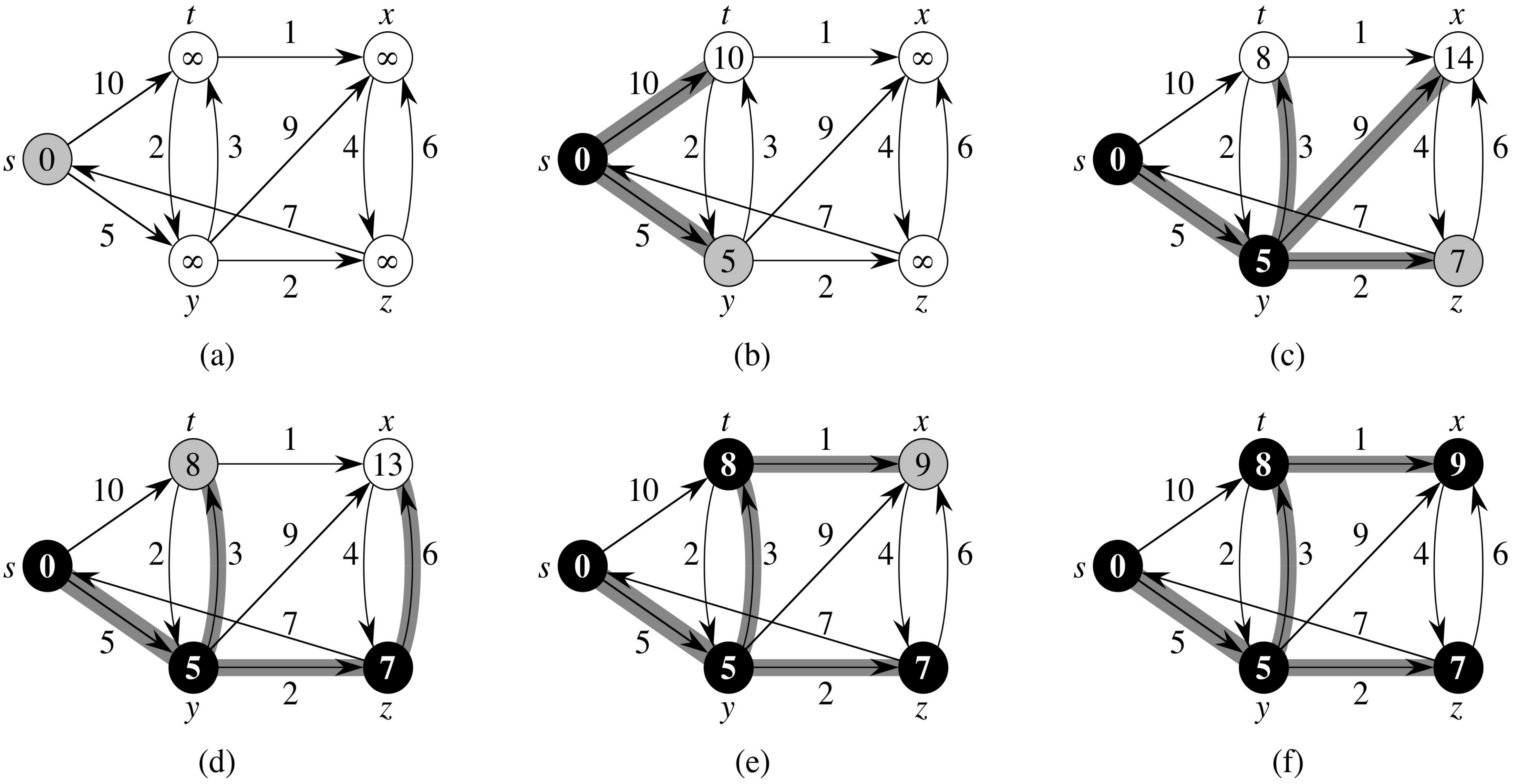


- Edge weights must not be negative. Maintains a set S of vertices whose shortest paths from s have already been determined. Repeatedly selects a vertex u from V-S with a minimum shortest-path weight estimate u.d (use a min priority queue based on the d attribute), and relaxes all edges leaving u.

DIJKSTRA(G,w,s)

```
1.  INITIALIZE-SINGLE-SOURCE(G,s)
2.  S=empty set
3.  Build min priority queue Q with nodes in G.V based on d
4.  while Q≠empty
5.      u=EXTRACT-MIN(Q)
6.      S=S U {u}
7.  for each vertex v in G.AdjacencyList[u]
8.      RELAX(u,v,w) //substitute step 2 of RELAX with a DECREASE-KEY operation
```

Dijkstra’s Algorithm



- All algorithms go through the initialization step and then relax graph edges repeatedly. They differ in the number of times and order in which edges are relaxed.

1. Bellman-Ford
   - each edge is relaxed exactly $|V|-1=n-1$ times
   - works on graphs with negative weight edges
   - is able to detect negative weight cycles
   - complexity $O(mn)$
2. Shortest Paths in Directed Acyclic Graphs
   - each edge is relaxed exactly once
   - works only on acyclic graphs
   - linear algorithm: $\Theta(m+n)$
   - works on graphs with negative weight edges
3. Dijkstra's Algorithm
   - each edge is relaxed exactly once
   - edge weights must be nonnegative
   - complexity $O(n^2)$

- Chapter 24
- Read sections 24.1 – 24.3
- Omit all theorems, corollaries, lemmas and proofs (except those mentioned in these slides)
- Omit "Properties of shortest paths and relaxation" (p. 649-650)
- Omit the complexity analysis of Dijkstra's algorithm (p. 661-662)
- Read everything else
- Understand the PERT chart application discussed on p. 657 to find the longest path – not discussed in class

- Try problems
  - 24.1-1, 24.1-4
  - 24.2-1, 24.2-2, 24.2-4
  - 24.3-1, 24.3-2, 24.3-3

AUBURN UNIVERSITY