COMTEK3,CCT3 & ESD3: ALGORITHMS
Minimum Spanning Tree & Shortest Paths

Ramoni Adeogun
Associate Professor and Head of the AI for Communication Group
Wireless Communication Networks Section (WCN)
Email: ra@es.aau.dk

October, 2023

AALBORG UNIVERSITET

Outline

- Weighted Graphs
- Minimum Spanning Trees
  - Kruskal's algorithm
  - Prim's algorithm
- Shortest Path Algorithms
  - Bellman-Ford
  - Shortest-paths finding on DAGs
  - Dijkstra
- Summary

Minimum Spanning Tree

- Spanning tree of a graph $G = (V, E)$ is a tree $T = (V', E')$ such that $V' = V$ and $E' \subset E$
  - Example:

- Note that a graph can have many spanning trees.

- A minimum spanning tree, T in a weighted graph G with weight function w is a spanning tree with minimal weight,

$$T = \arg\min_{T' \leq G} \left(w(T')\right), \; w(T) = \sum_{e \in T} w(e)$$

Minimum Spanning Tree

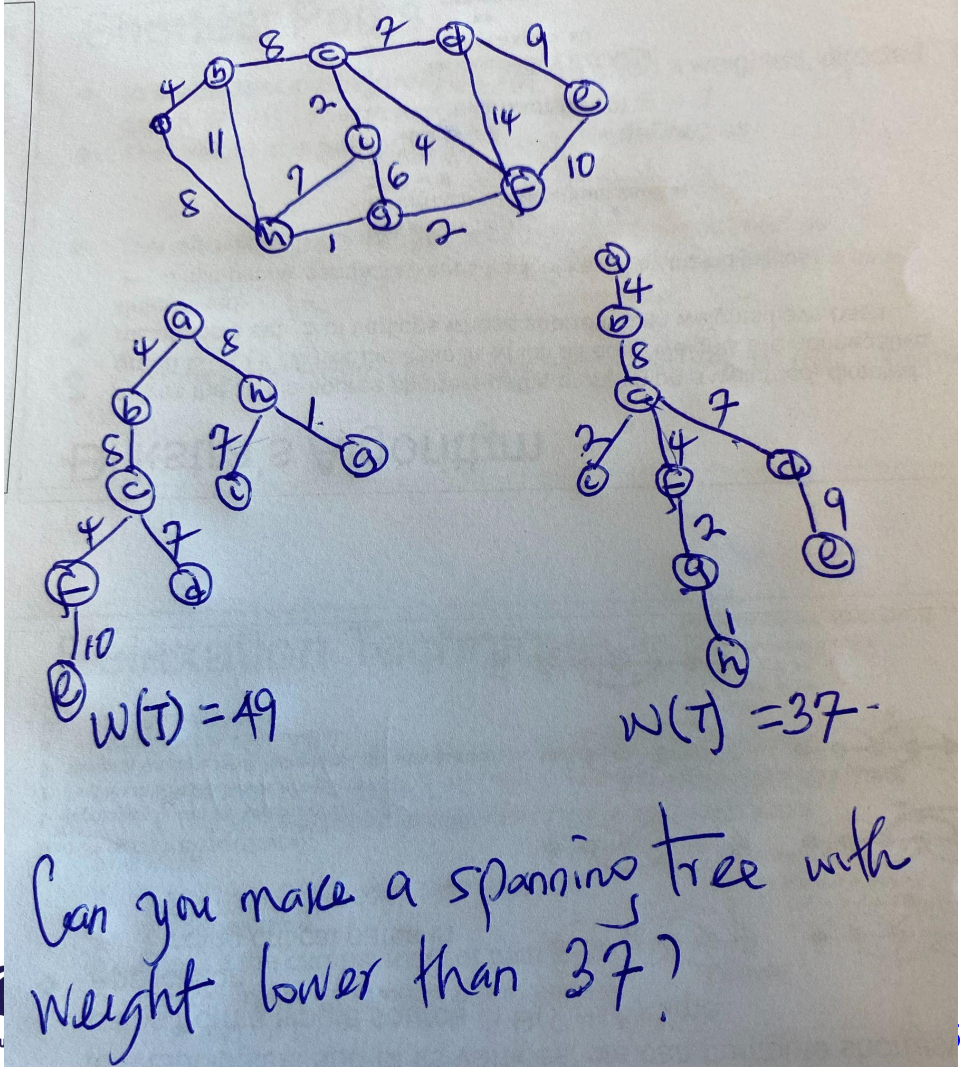
Minimum spanning trees appear in many engineering applications, e.g.,

- A network operator is planning the next generation cellular communication systems, and needs to connect the new base stations with optical cables.
  - What is the cheapest way to do this?
    - Note: Cost scales with the length of cable
- A carrier company regularly delivers packets to a number of customers.
  - What is the best route plan such that
    - an unbounded (1) number of can deliver the packets with the shortest delivery time
      - Assuming that time to drop a packet can be neglected

MST - Example

Are there other MSTs with weight equal to 37?

Is MST of a given graph, G, unique?

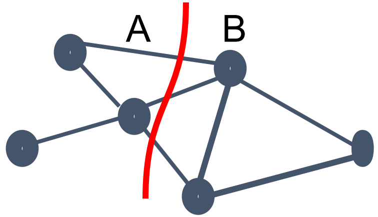


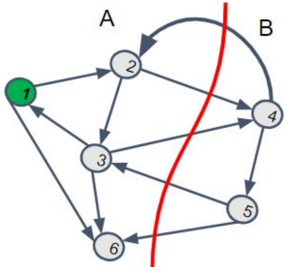
Cut, Cut Weight and Light Edge

- Split the vertex set V of a graph $G = (V, E)$ into two disjoint sets:

  $$V = A \cup B, \; A \cap B = \emptyset$$

  The set of edges from A to B is called a cut and is denoted $c(A, B)$





$C(A, B) = \{(2,3), (3,4)\}$
$C(B, A) = \{ \quad \}$

- For weighted graphs we define the weight of a cut, c, as:

  $$w(c) = \sum_{e \in c} w(e)$$

- **<u>A light weight:</u>** an edge e in a cut, c, with minimum weight.

Greedy Algorithm

- A greedy algorithm is a step-wise optimization algorithm
  - always make the choice which seems best at the moment ignoring any future steps
- Greedy algorithms do not always find optimal solutions, but often yield good solutions.
- There are many examples of greedy algorithms for problems involving graphs
  - Prim's
  - Kruskal's
  - Dijkstra's,
    - and others

Disjoint-Set Data Structure

- Kruskal's algorithms uses a disjoint-set data structure (DSDS)

- A DSDS maintains a collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint sets
  - Each set is identified by a representative which is a member of it

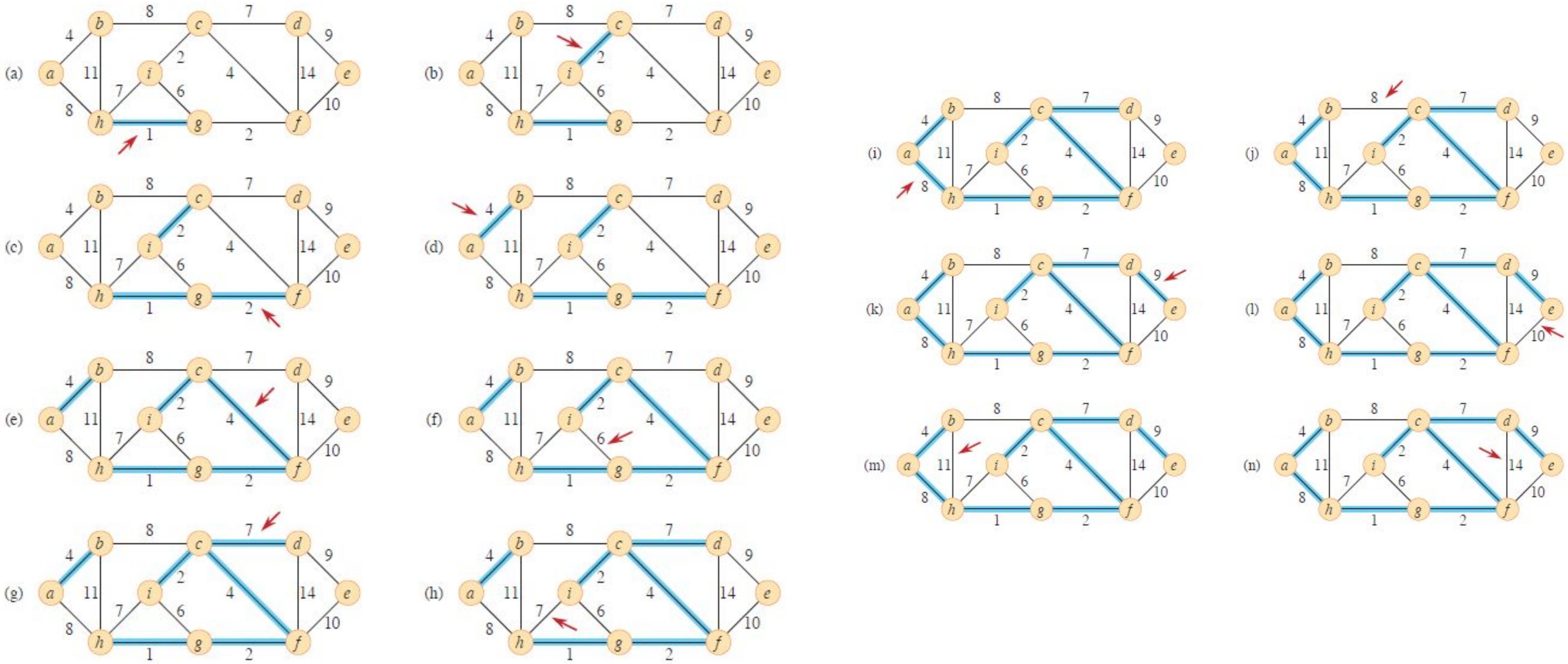
- A DSDS supports three operations:
  - Make-Set(x): creates new set $\{x\}$ with representative x
  - Union(x,y): unites 2 sets containing x and y into a new set $S_x \cup S_y$ with a new representative
  - Find(x): returns the representative of the set containing x

Kruskal's Algorithm

- used to construct a minimum spanning tree in an undirected graph
- traverses all edges - lightest edge first - and adds them to the solution set until they no longer connect new parts of the graph
- each set contains the vertices of a tree in the current forest

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
create a single list of the edges in G.E
sort the list of edges into monotonically increasing order by weight w
for each edge (u, v) taken from the sorted list in order
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Kruskal’s Algorithm



Prim's Algorithm

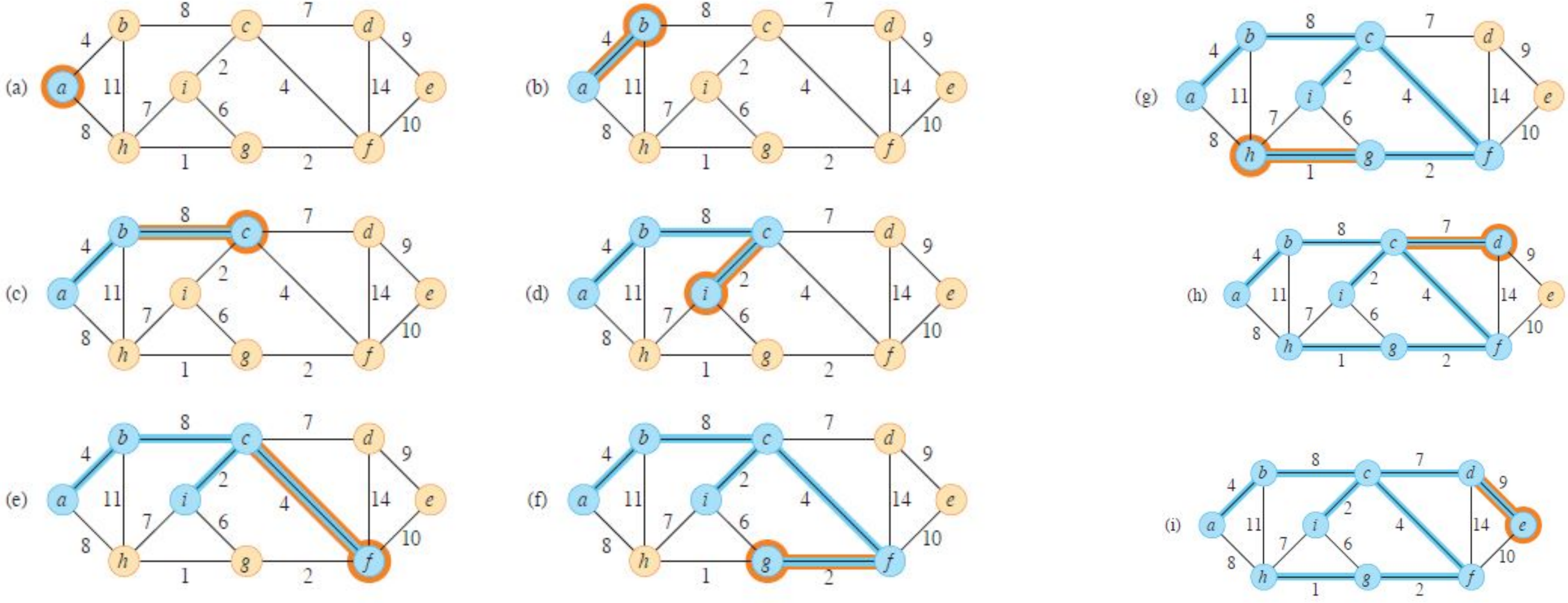
- used to construct a minimum spanning tree of a weighted graph
- the tree is grown from an arbitrary root until it spans the whole graph
- Each step adds a light edge which connects the current tree to an isolated vertex

Vertices not yet in the spanning tree are kept in a min-priority queue, Q, based on the key attribute.

- Attributes of a vertex:
  - $v.key$: minimum edge weight between v and a vertex in the tree, if no edge $v.key = \infty$
  - $v.\pi$: parent in the tree

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)          // add u to the tree
    for each vertex v in G.Adj[u]   // update keys of u's non-tree neighbors
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

Prim’s Algorithm



Shortest Path Algorithms

Shortest Path Algorithms

- Many applications require finding shortest paths in a graph
  - Routing in computer networks
  - Route planning in navigation equipment
  - Finding “critical paths” in projects

- There exist many shortest path finding algorithms
  - We will study
    - Bellman-Ford Algorithm
    - Shortest paths finding in DAGs
    - Dijkstra Algorithm
  - There are many more of such algorithms

Shortest Path Problems (SPPs)

- **Single-source SPP**: find a shortest path from a specified source to each vertex in a graph
- **Single-destination SPP**: find a shortest path from each vertex in a graph to specified destination
  - reversal of edge direction changes this to single-source SPP
- **Single-pair SPP**: find the shortest u - v path given u and v
- **All-pairs SPP**: find the shortest path from u to v for all pairs of vertices

Shortest Paths

- In a **shortest-paths problem**, we are given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$
- The weight of a path $p = \langle v_0, v_1, \ldots v_k \rangle$ is defined as

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

- The **shortest-path weight** from u to v is then defined as

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

- A shortest path from u to v is any path p with weight $w(p) = \delta(u, v)$

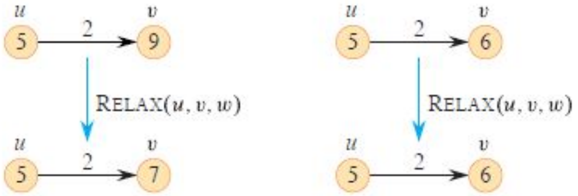
Relaxation Technique

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

- Relaxation
  a. A technique used in the shortest-path finding algorithms
  b. maintains an **attribute** $v.d$; $\forall v \in V$ **=> shortest-path estimate**
     - upper bound on the weight of shortest path from s to v
- The idea
  a. Initialize shortest path estimates and predecessors
  b. check if the current shortest path to v can be improved through u and update $v.d$ and $v.\pi$

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Example

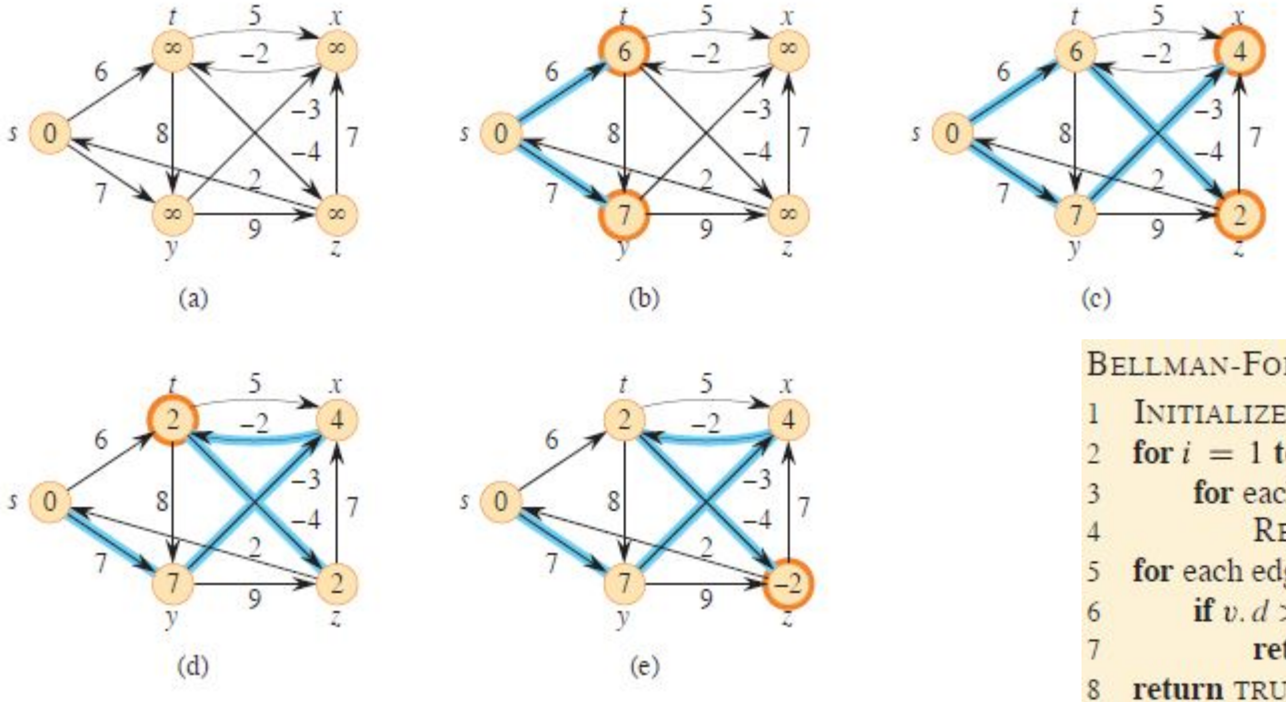


Bellman-Ford Algorithm

- The Bellman-Ford algorithm:
  - solves the single-source SPP
    - including for cases with negative edge weights $w(e)$ for some or all edges $e \in E$
  - relaxes edges progressively until $v.d = \delta(s, v)$
  - returns TRUE iff the graph contains no reachable negative cycles from s
  - Running time is $O(VE)$
    - But why?

```
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
```

The Bellman-Ford Algorithm - Ex



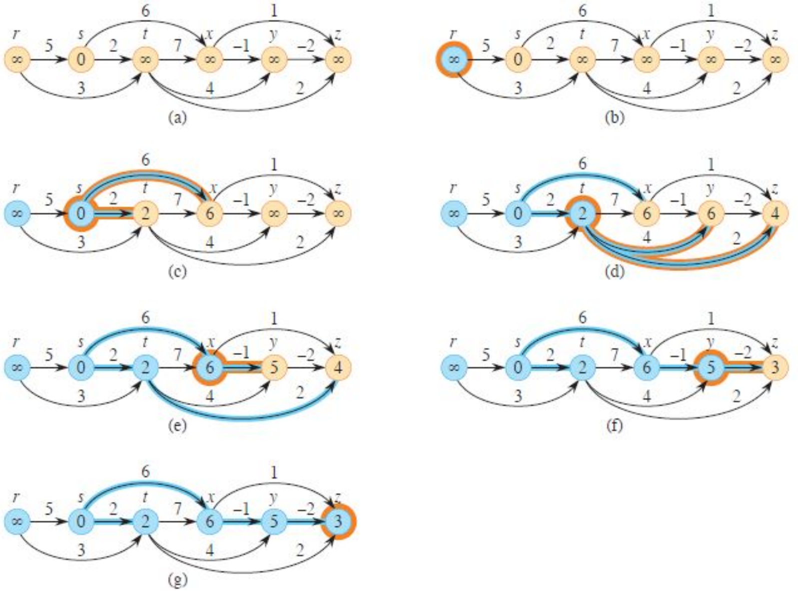

```
BELLMAN-FORD(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| - 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```


AALBORG UNIVERSITET

Single-Source SSP in DAGs

- By relaxing the edges of a weighted DAG, $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(V + E)$ time
- Application
  - Finding critical paths in PERT diagrams

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u ∈ G.V, taken in topologically sorted order
    for each vertex v in G.Adj[u]
        RELAX(u, v, w)
```

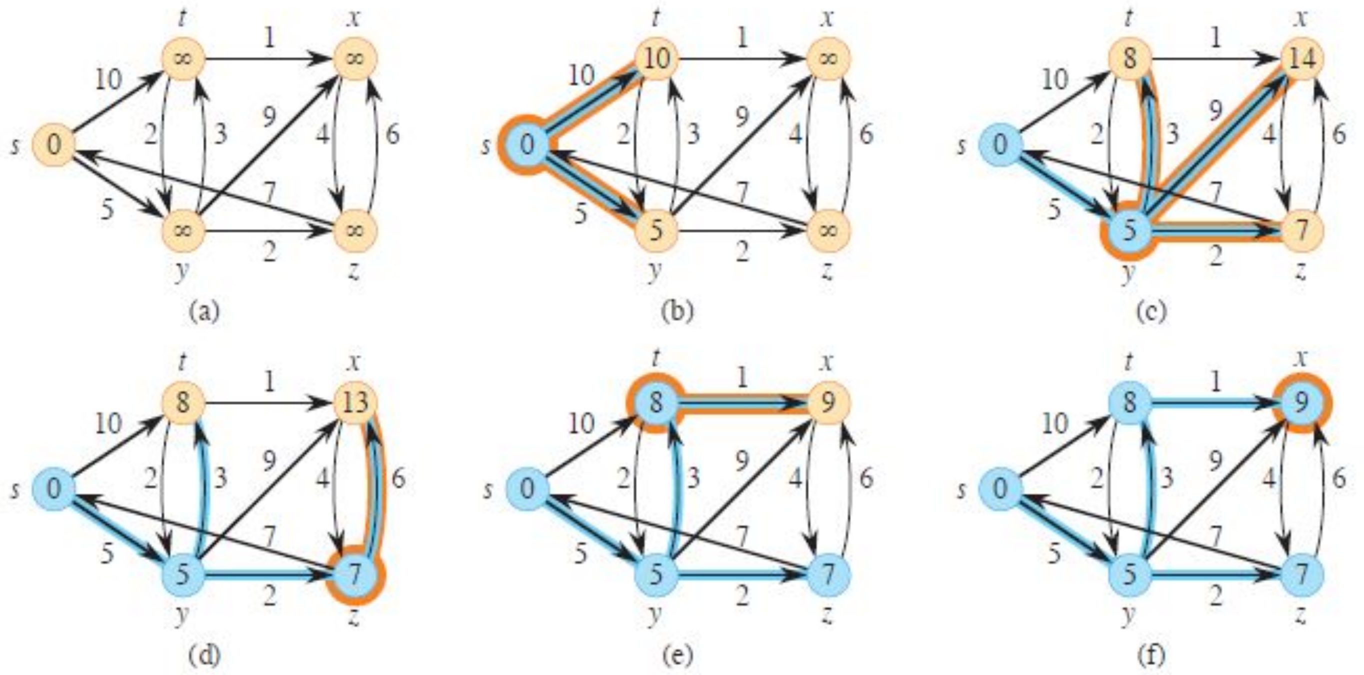


Dijkstra's Algorithm

- solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are **nonnegative**.
- maintains a set S of vertices whose shortest path weights have been determined
  - a min-priority queue, Q, keeps track of vertices, keyed by their d values

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v in G.Adj[u]
        RELAX(u, v, w)
        if the call of RELAX decreased v.d
            DECREASE-KEY(Q, v, v.d)
```

Dijkstra's algorithm



Summary

- Minimum spanning tree => spanning tree with minimal weight
- To construct minimum spanning, we can use
  - Prim's algorithm -> from …….. graph
  - Kruskal's algorithm -> from ……. graph

- Shortest path problems
  - single source, single destination, single-pair, all-pairs

- Shortest-paths finding
  - Bellman's algorithm
  - Dijkstra's algorithm