Graphs (cont.)

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

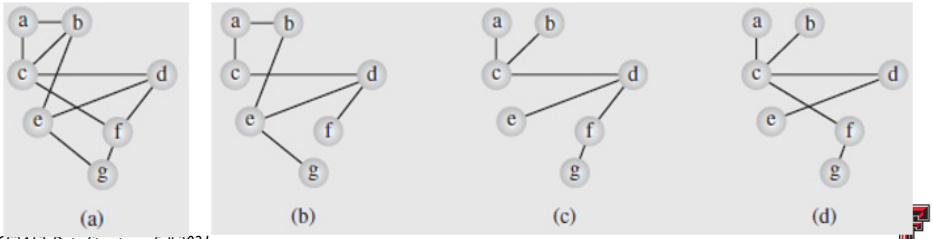
Lecture 16

sunho.lim@ttu.edu

Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

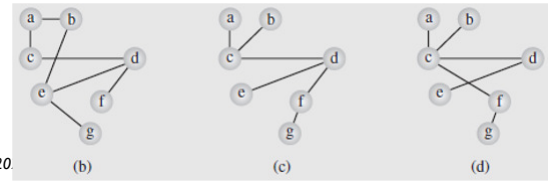
Spanning Tree

- Consider,
  - an airline that has routes between seven cities
  - If economic hardships force the airline to **cut routes**, which ones should be kept to preserve a route to each city, if only indirectly?
  - However, we want to make sure we have **the minimum connections necessary** to preserve the routes



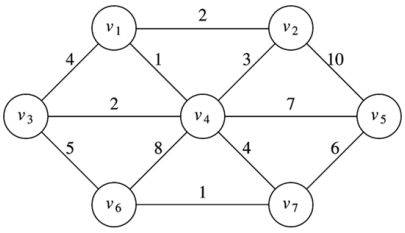
Spanning Tree (cont.)

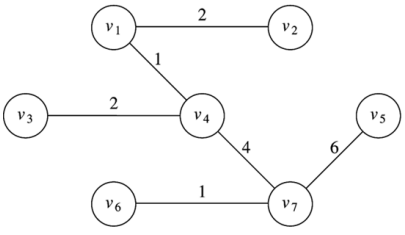
- A possibility of **multiple spanning trees**
  - the minimum number of edges
- Don't know which of these might be optimal,
  - haven't taken distances into account
- The airline,
  - wanting to **minimize costs** → want to use the **shortest distances for the connections**
- the ***minimum spanning tree***,
  - where the sum of the edge weights is minimal



Minimum Spanning Tree

- An undirected graph, G
- A tree formed from graph edges
  - connect ALL the vertices of G at **lowest total cost**
  - a minimum spanning tree exists if and only if G is connected
  - the number of edges is |V| - 1
  - an **acyclic** tree



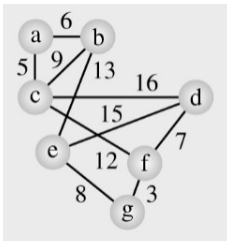


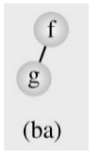
Minimum Spanning Tree (cont.): Kruskal Algorithm

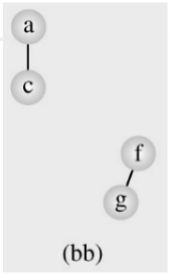
- A greedy strategy,
  - order the **edges** by weight
  - continuously select the edges in order of **smallest weight**
  - accept an edge if it does not cause a **cycle**
  - maintain a **forest** – a collection of trees
- The algorithm is as follows:

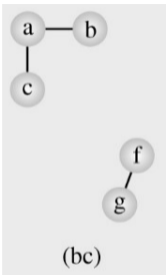
```
KruskalAlgorithm(weighted connected undirected graph)
  tree = null;
  edges = sequence of all edges of graph sorted by weight;
  for (i = 1; i <= |E| and |tree| < |V| – 1; i++)
    if ei from edges does not form a cycle with edges in tree
      add ei to tree;
```

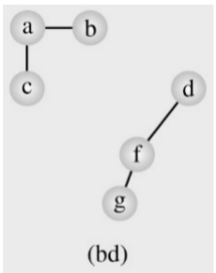

Minimum Spanning Tree (cont.): Kruskal Algorithm (cont.)



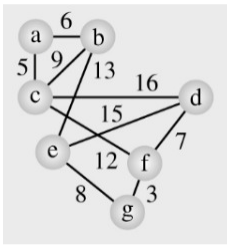


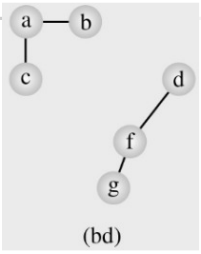




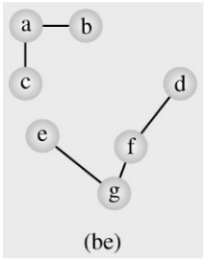


Minimum Spanning Tree (cont.): Kruskal Algorithm (cont.)

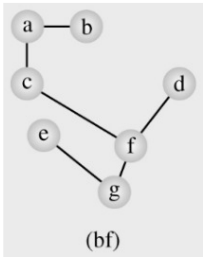




(bd)



(be)

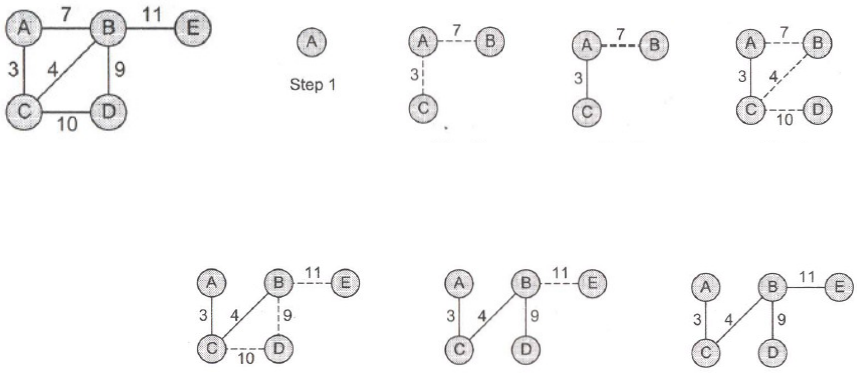


(bf)

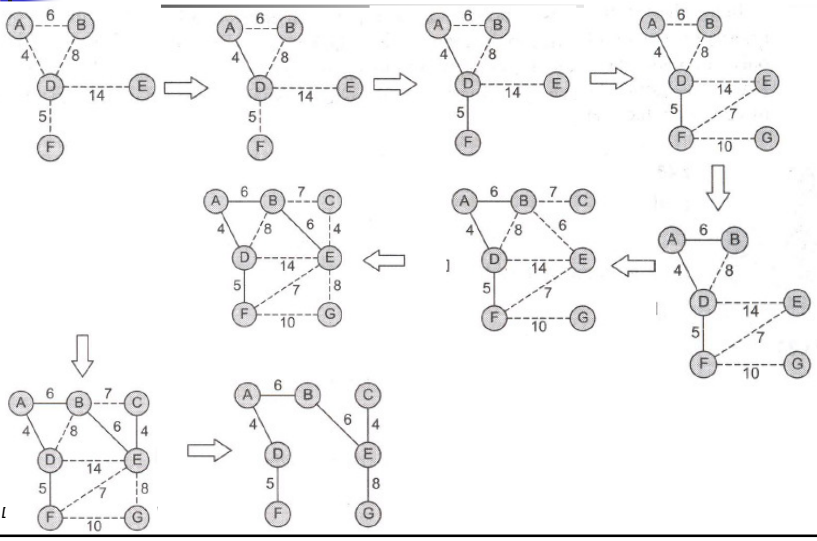
Minimum Spanning Tree (cont.): Prim Algorithm

- Run on undirected graph
- Growing the tree in successive states
  - in each stage, one node is picked as the **root**
  - add **an edge and an associated vertex** to the tree
- At each stage, the algorithm finds a new vertex to add to the tree,
  - by choosing the edge (u, v) such that the **cost of (u, v) is the smallest among all edges**, where u is in the tree and v is not
- After a vertex v is selected, for each unknown w adjacent to v,
  - **d(v) = min(d(w), c(w,v))**

Minimum Spanning Tree (cont.): Prim Algorithm (cont.)



Minimum Spanning Tree (cont.): Prim Algorithm (cont.)



Topological Sort

- Linearize the ordering vertices in a **directed acyclic graph (DAG)**
  - for every edge *uv*, *u* comes before *v* in the ordering
- For instance,
  - vertices → tasks to be performed
  - edges → constraints that one task must be performed before another
- topological ordering
  - a valid **sequence** for the tasks

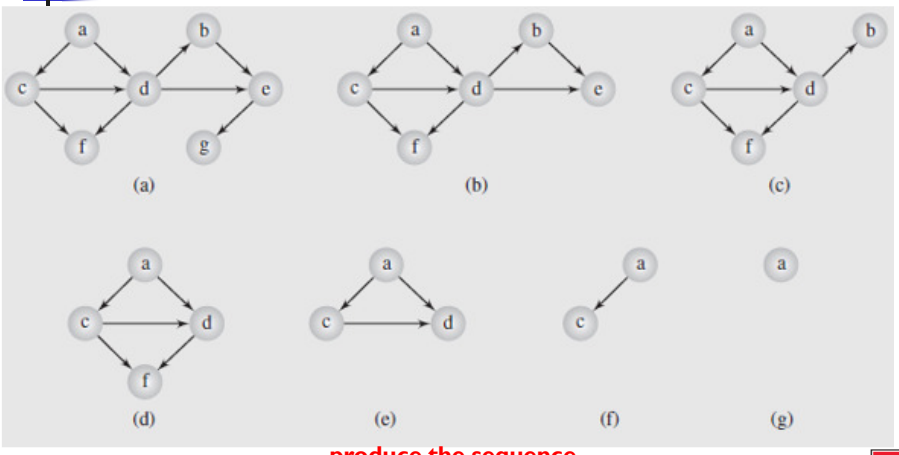
Topological Sort (cont.)

- The algorithm for the topological sort:

```
topologicalSort(digraph)
  for i = 1 to |V|
    find a minimal vertex v;
    num(v) = i;
    remove from digraph vertex v and all edges incident
      with v;
```

- locate a vertex, *v* with no outgoing edges
  - a ***minimal vertex*** or ***sink***
- remove any edges leading from a vertex to *v*

Topological Sort (cont.)



produce the sequence
g, e, b, f, d, c, a