CS302 - Data Structures

using C++

Topic: Graphs – Spanning Trees & Minimum Spanning Tree

Kostas Alexis

AUTONOMOUS ROBOTS LAB

Spanning Trees

- A tree is an undirected connected graph without cycles

Spanning Trees

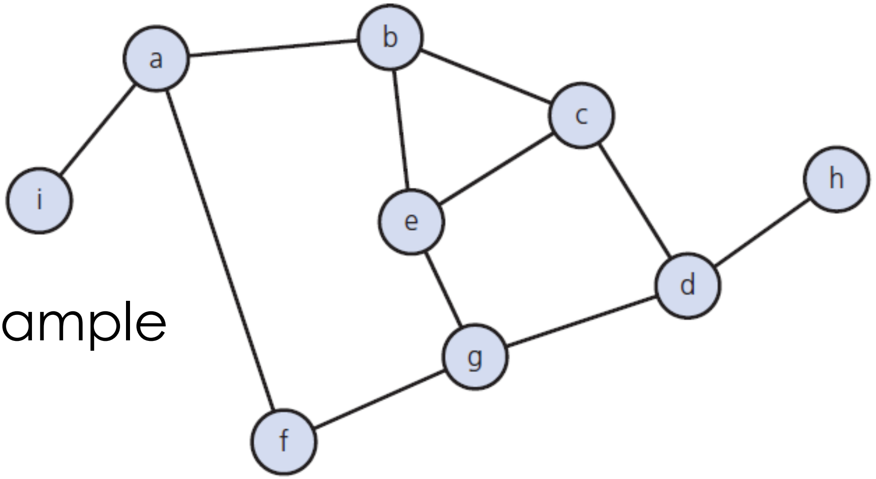
- A tree is an undirected connected graph without cycles
- **Spanning Tree:** A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree.

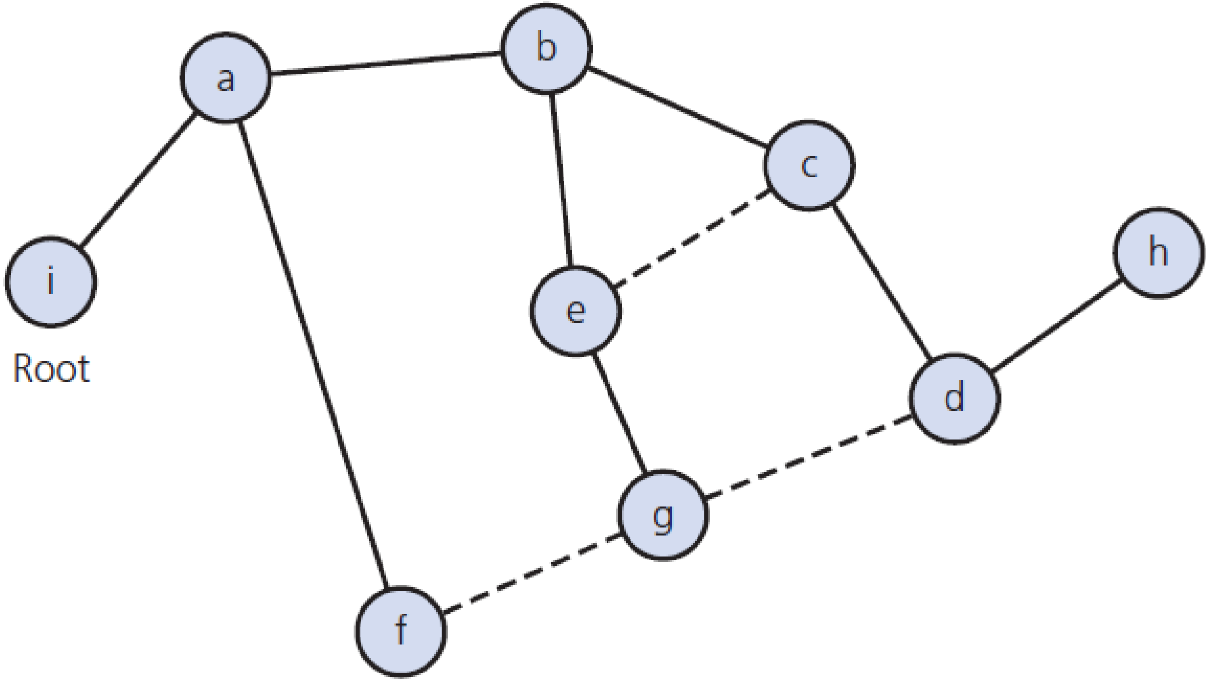
Spanning Trees

- A tree is an undirected connected graph without cycles
- **Spanning Tree:** A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree.
- **Detecting a cycle in an undirected graph**
  - **Connected undirected graph with n vertices must have at least n-1 edges**
  - **If it has exactly n-1 edges, it cannot contain a cycle**
  - **With more than n-1 edges, must contain at least one cycle**

Spanning Trees

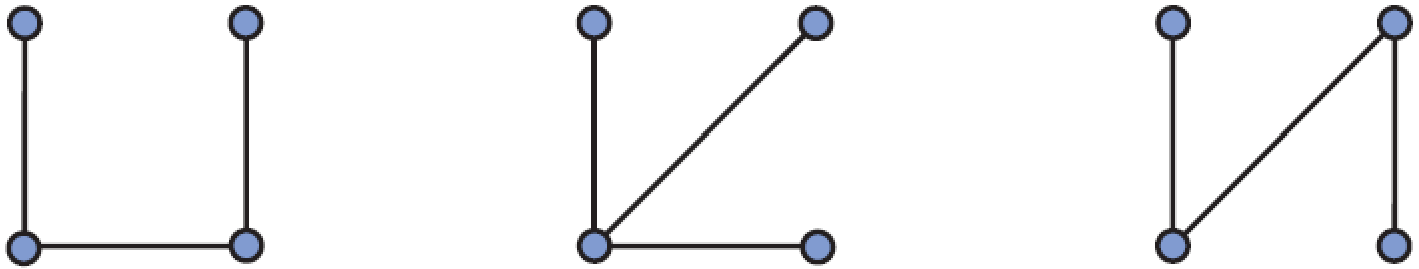
- A spanning tree for the previous connected graph example





Spanning Trees

- Connected graphs that each have four vertices and three edges



Spanning Trees

- DFS-based Spanning Tree Algorithm

```
// Forms a spanning tree for a connected undirected graph
// beginning at vertex v by using depth-first search. Recursive version
dfsTree(v: Vertex)
{
    Mark v as visited
    for (each unvisited vertex u adjacent to v)
    {
         Mark the edge from u to v
         dfsTree(u)
    }
}
```

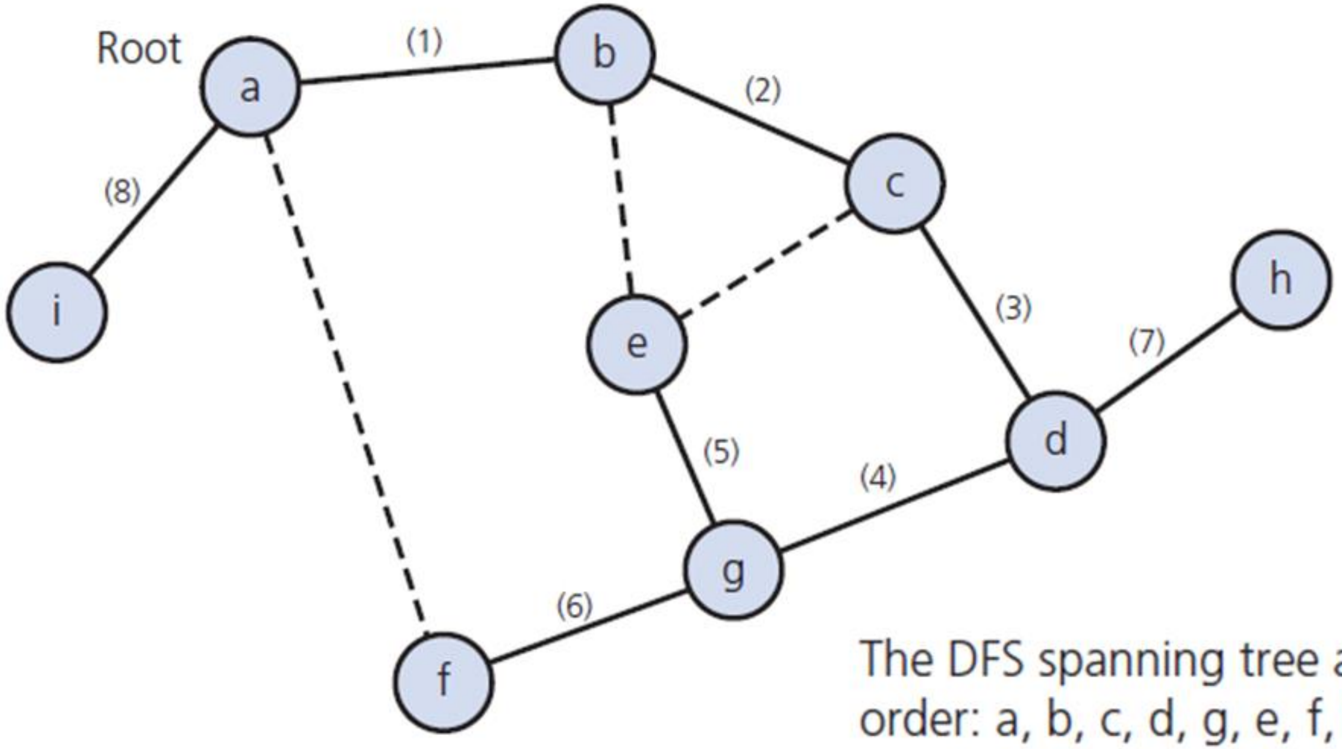
Spanning Trees

- BFS-based Spanning Tree Algorithm

```
// Forms a spanning tree for a connected undirected graph beginning at vertex v by using depth-first search. Recursive
    version
bfsTree(v: Vertex)
{
    q = a new empty queue
    // Add v to queue and mark it
    q.enqueue(v)
    Mark v as visited
    while (!q.isEmpty())
    {
         q.dequeue(w)
         // Loop invariant: there is a path from w to every vertex in the queue q
         for (each unvisited vertex u adjacent to w)
         {
              Mark us as visited
              Mark edge between w and u
              q.enqueue(u)
         }
    }
}
```

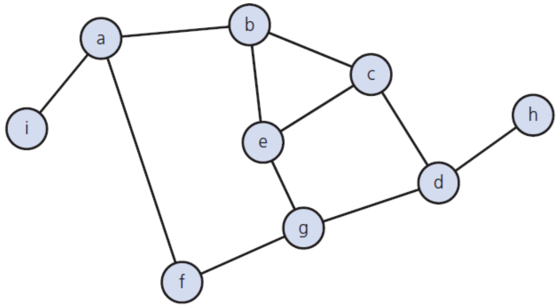
Spanning Trees

- Connected graphs that each have four vertices and three edges

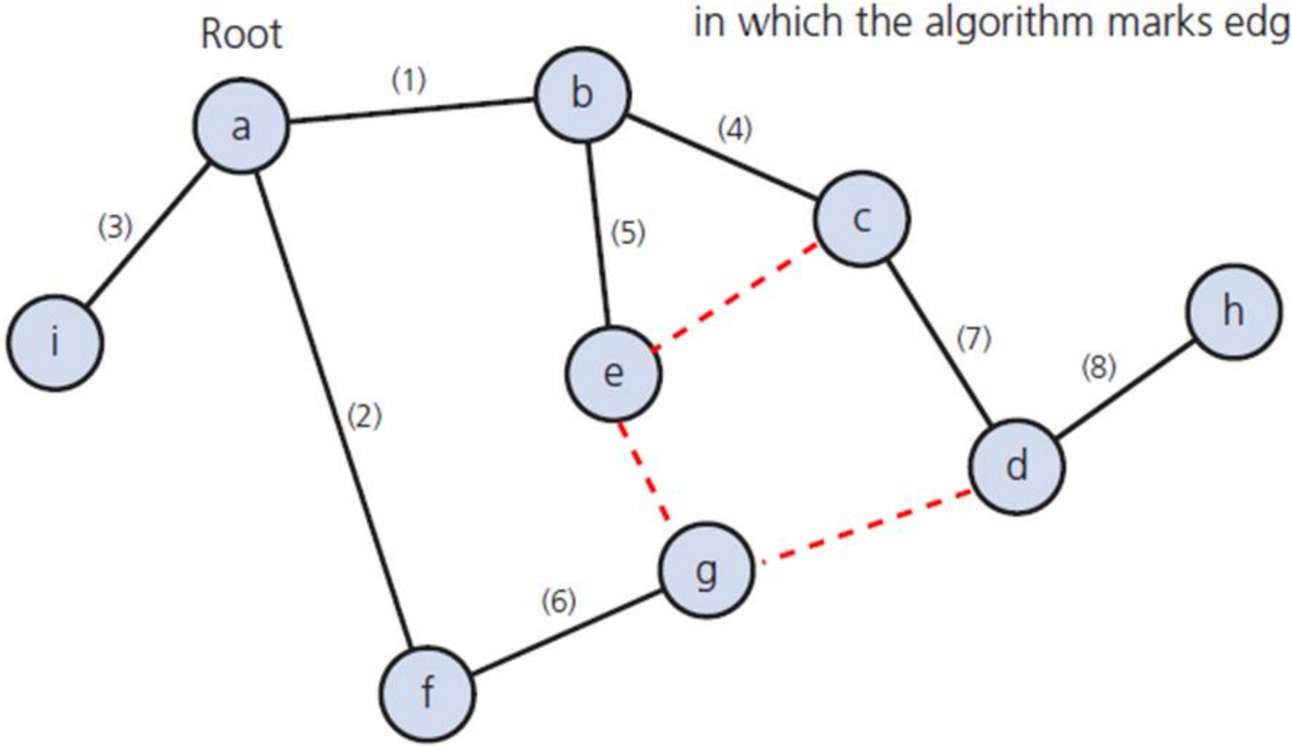


The DFS spanning tree algorithm visits vertices in this order: a, b, c, d, g, e, f, h, i. Numbers indicate the order in which the algorithm marks edges.

Spanning Trees



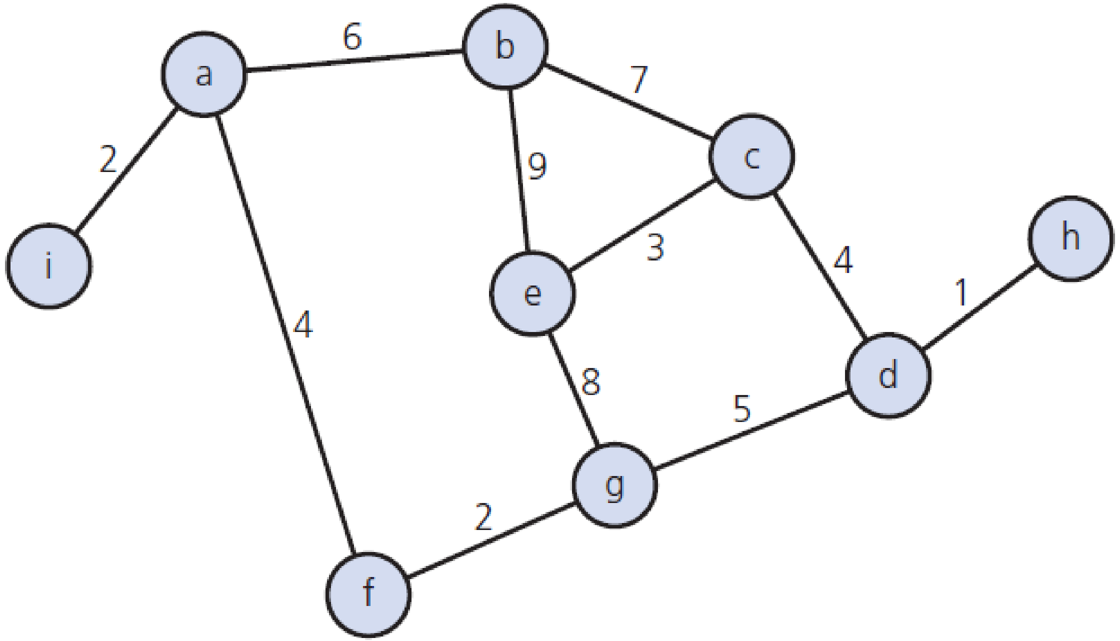
- The BFS Spanning Tree rooted at vertex a for the example directed graph



The BFS spanning tree algorithm visits vertices in this order: a, b, f, i, c, e, g, d, h. Numbers indicate the order in which the algorithm marks edges.

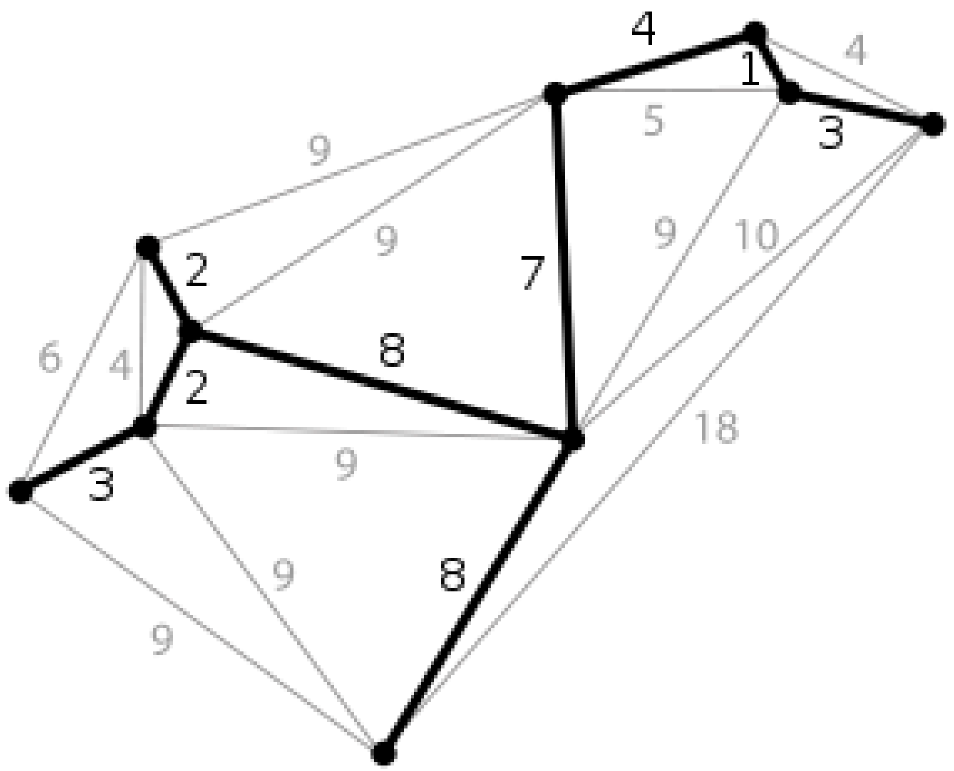
Minimum Spanning Trees

- A Weighted, Connected, Undirected graph



Minimum Spanning Trees

- **Minimum Spanning Tree:** A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted (un)directed graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

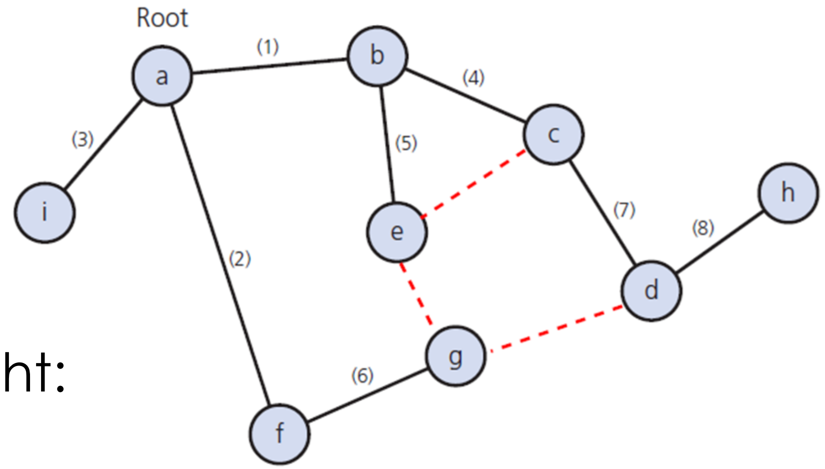


Minimum Spanning Trees

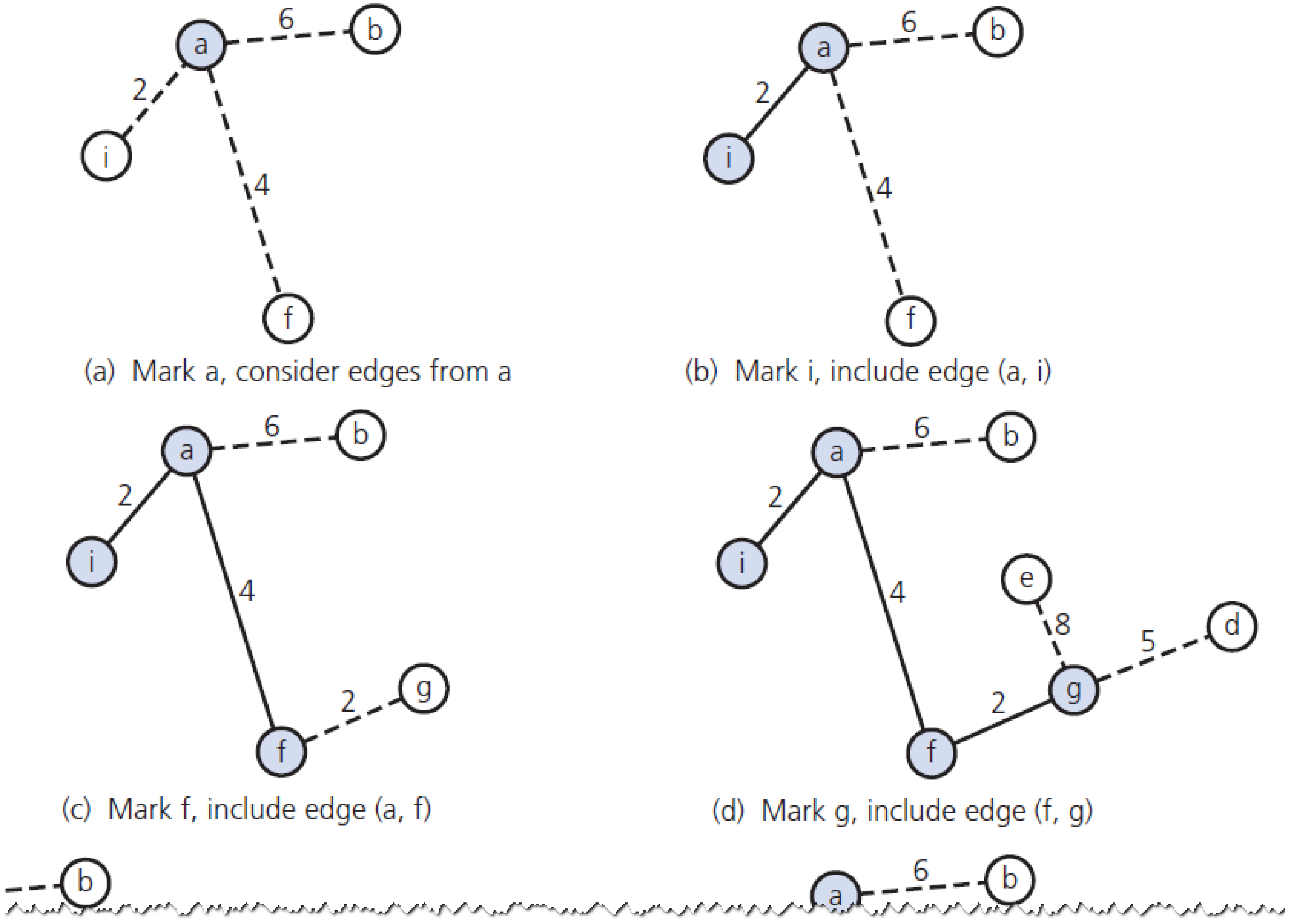
- Find Minimum Spanning Tree through “Prim’s Algorithm”

```
// Forms a minimum spanning tree for a weighted, connected, undirected graph whose weights are >=0,
// beginning at vertex r
primsAlgorithm(r: Vertex)
{
   Mark vertex r as visited and include it in the minimum spanning tree
   while (there are unvisited vertices)
   {
       Find the least-cost edge (v,u) from a visited vertex v to some unvisited vertex u
       Mark u as visited
       Add the vertex u and the edge (v,u) to the minimum spanning tree
   }
}
```

Minimum Spanning Trees



- A trace of primsAlgorithm for the graph shown on the right:



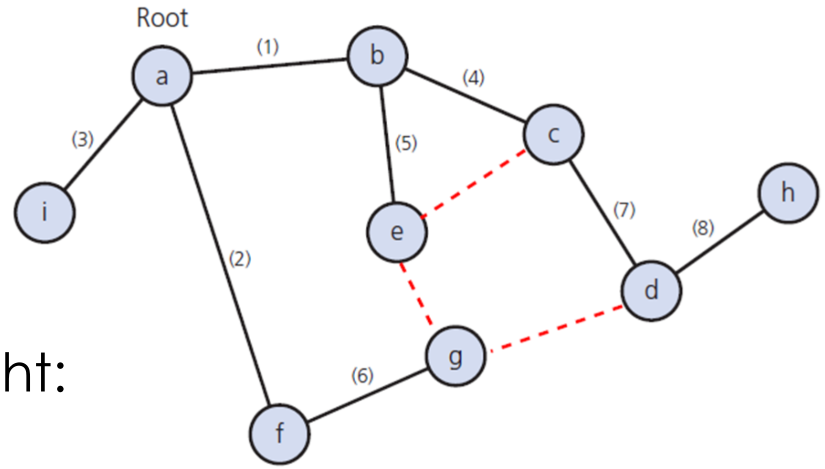
(a) Mark a, consider edges from a

(b) Mark i, include edge (a, i)

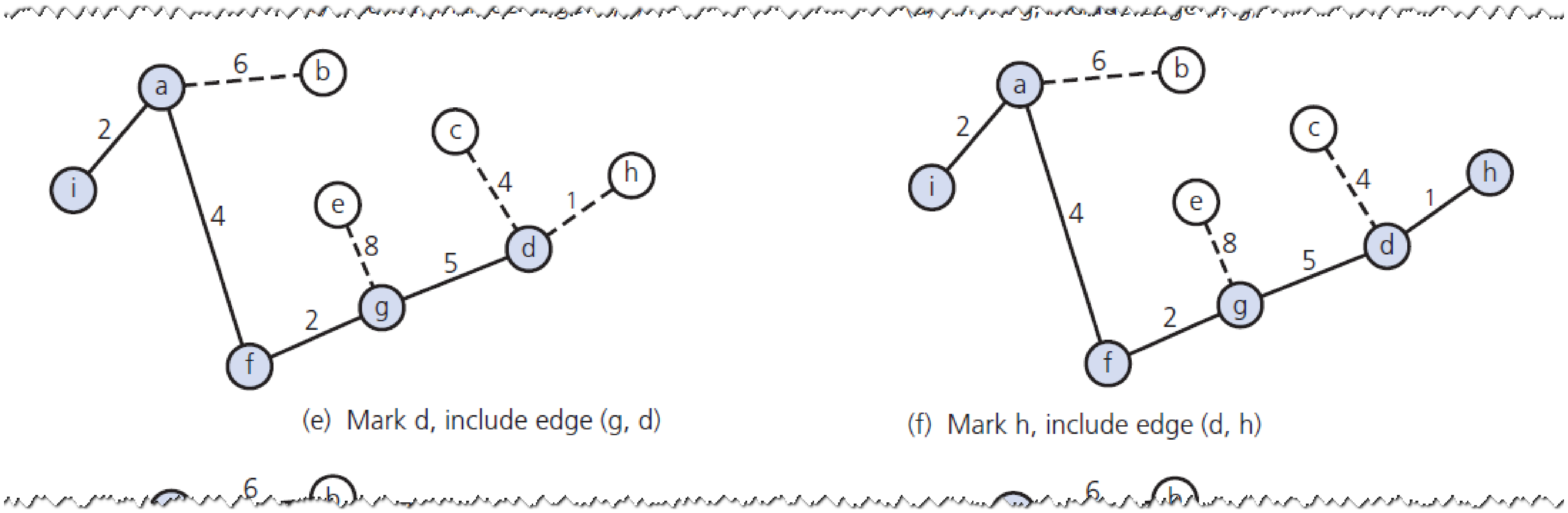
(c) Mark f, include edge (a, f)

(d) Mark g, include edge (f, g)

Minimum Spanning Trees



- A trace of primsAlgorithm for the graph shown on the right:

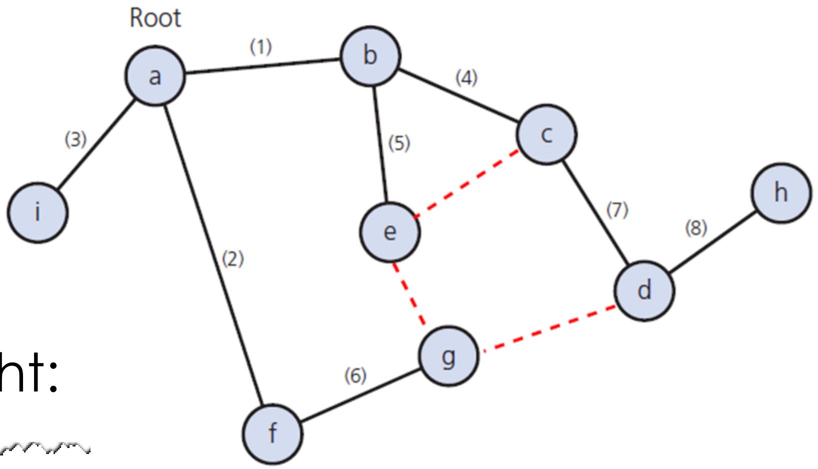


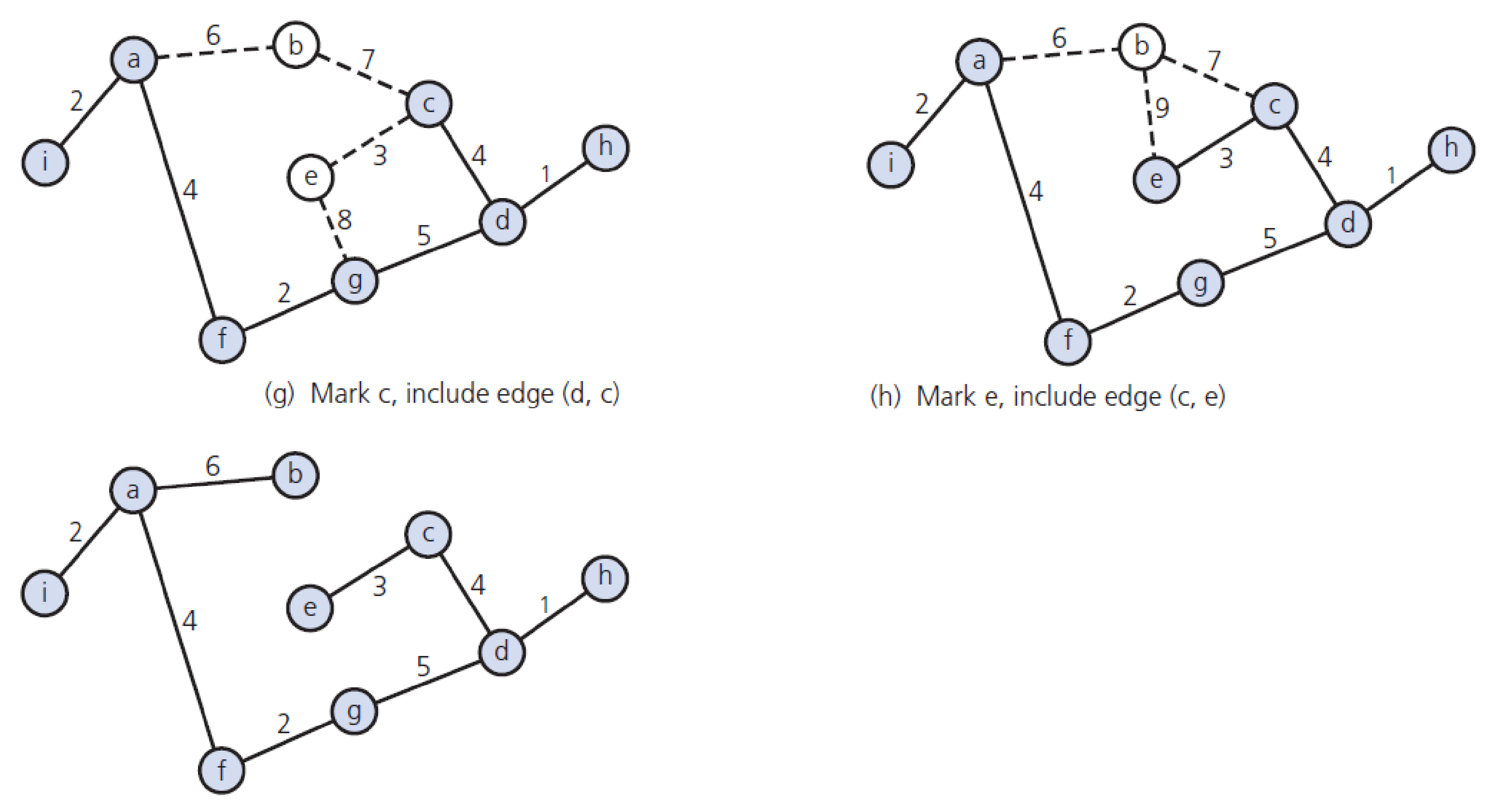
(e) Mark d, include edge (g, d)

(f) Mark h, include edge (d, h)

Minimum Spanning Trees

- A trace of primsAlgorithm for the graph shown on the right:





(e) Mark d, include edge (g, d)

(f) Mark h, include edge (d, h)

(g) Mark c, include edge (d, c)

(h) Mark e, include edge (c, e)

(i) Mark b, include edge (a, b)

Thank you