COT 5407: Introduction to Algorithms

Giri NARASIMHAN

www.cs.fiu.edu/~giri/teach/5407S19.html

Computing Connected Components

- Undirected Case
  - If graph is disconnected, then DFS/BFS will not visit all vertices and all edges when initiated from a vertex v.
  - It will only visit the component to which v belongs
  - Restart DFS/BFS from an unvisited vertex if traversal is needed
  - Number of restarts =
    - Number of connected components
- Directed case
  - Even if graph is not disconnected, DFS/BFS may not visit all vertices and all edges
  - This is because you can have a case where
    - A is reachable from B, but B is not reachable from A
    - Need to redefine connected components
  - Restart DFS/BFS from an unvisited vertex if traversal is needed

Weak/Strong Connected Components

- A directed graph is strongly connected if every vertex is reachable from every other vertex.
  - Note that this means that for a pair of vertices (A,B), we must have that A is reachable from B and B is reachable from A
- A directed graph is weakly connected if for every pair of vertices (A,B), either A is reachable from B or B is reachable from A.
- We will not study this further for this class …

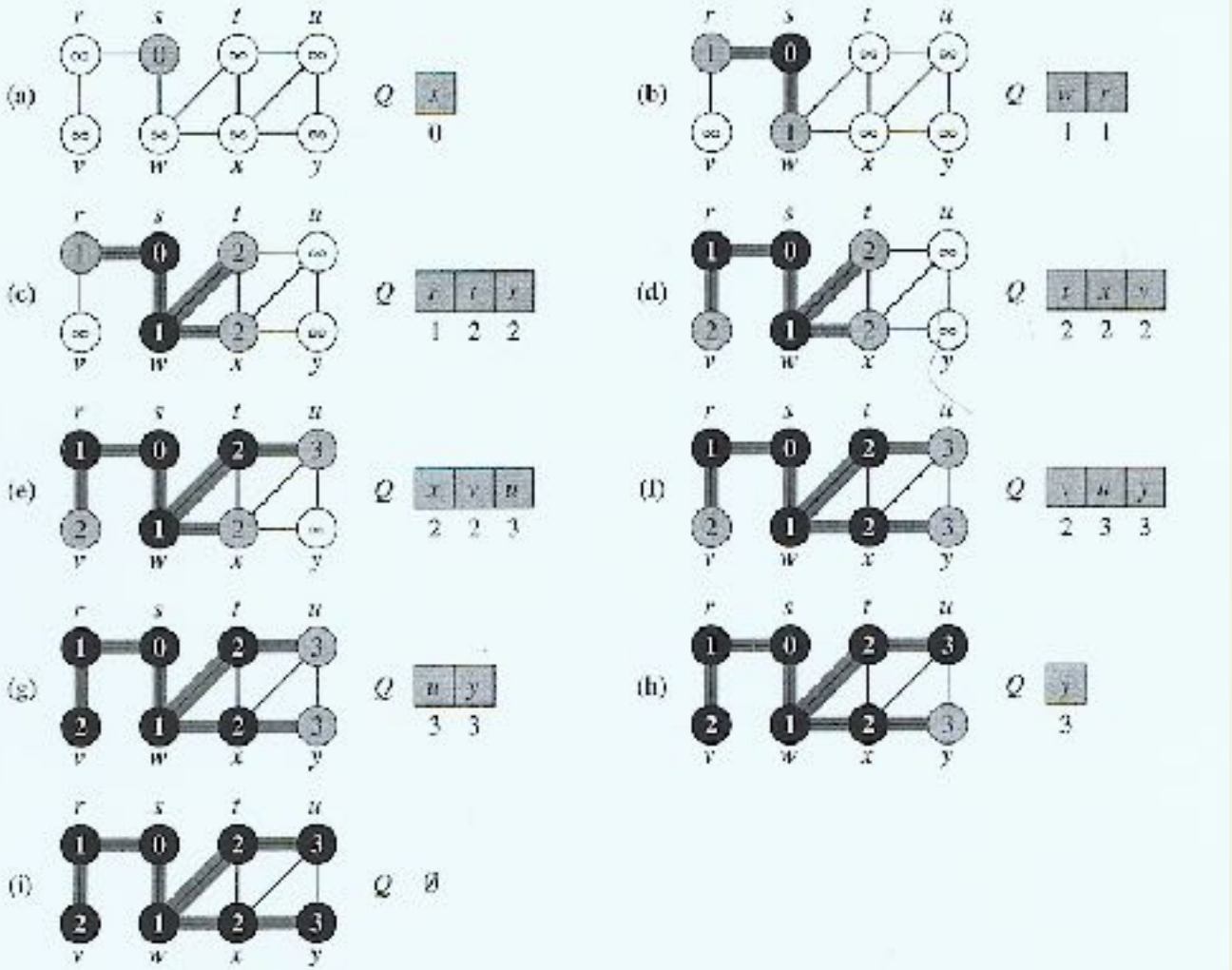


Figure 22.3 The operation of BFS on an undirected graph. Tree edges are shown shaded as they are produced by BFS. Within each vertex u is shown $d[u]$. The queue Q is shown at the beginning of each iteration of the **while** loop of lines 10–18. Vertex distances are shown next to vertices in the queue.

Breadth First Search

```
BFS(G,s)
1.    For each vertex u ∈V[G] – {s} do
2.        color[u] ← WHITE
           d[u] ← ∞
          π[u] ← NIL
      Color[u] ← GRAY
      D[s] ← 0
      π[s] ← NIL
      Q ← Φ
      ENQUEUE(Q,s)
      While Q ≠ Φ do
11.       u ← DEQUEUE(Q)
12.       VisitVertex(u)
13.       for each v ∈ Adj[u] do
14.             VisitEdge(u,v)
15.             if (color[v] = WHITE) then
16.                 color[v] ← GRAY
17.                 d[v] ← d[u] + 1
18.                π[v] ← u
19.                 ENQUEUE(Q,v)
20.        color[u] ← BLACK
```

Minimum Spanning Tree

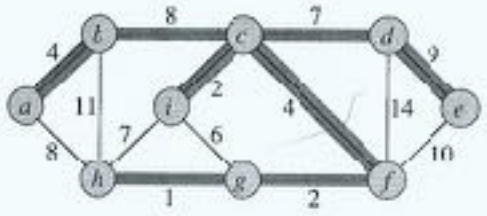


Figure 23.1 A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

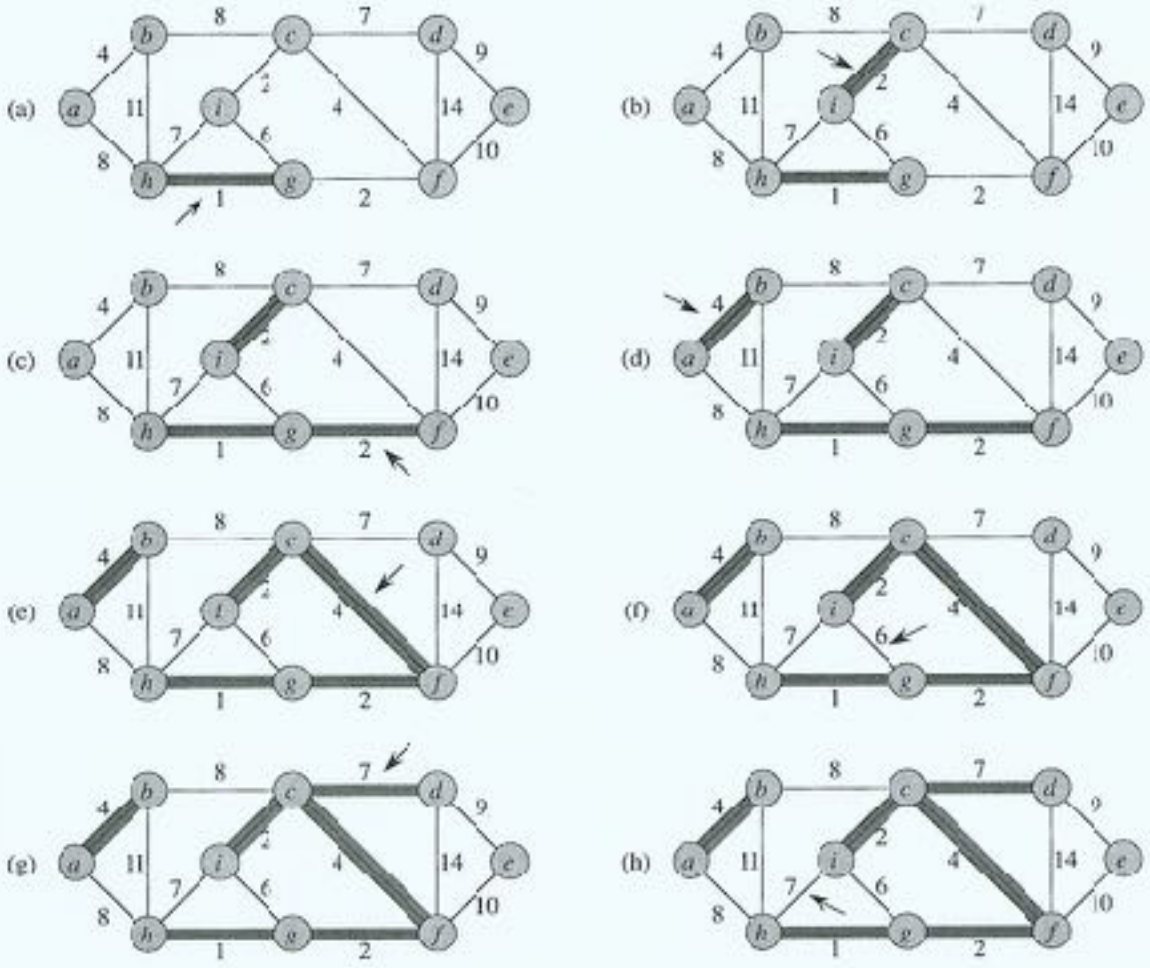
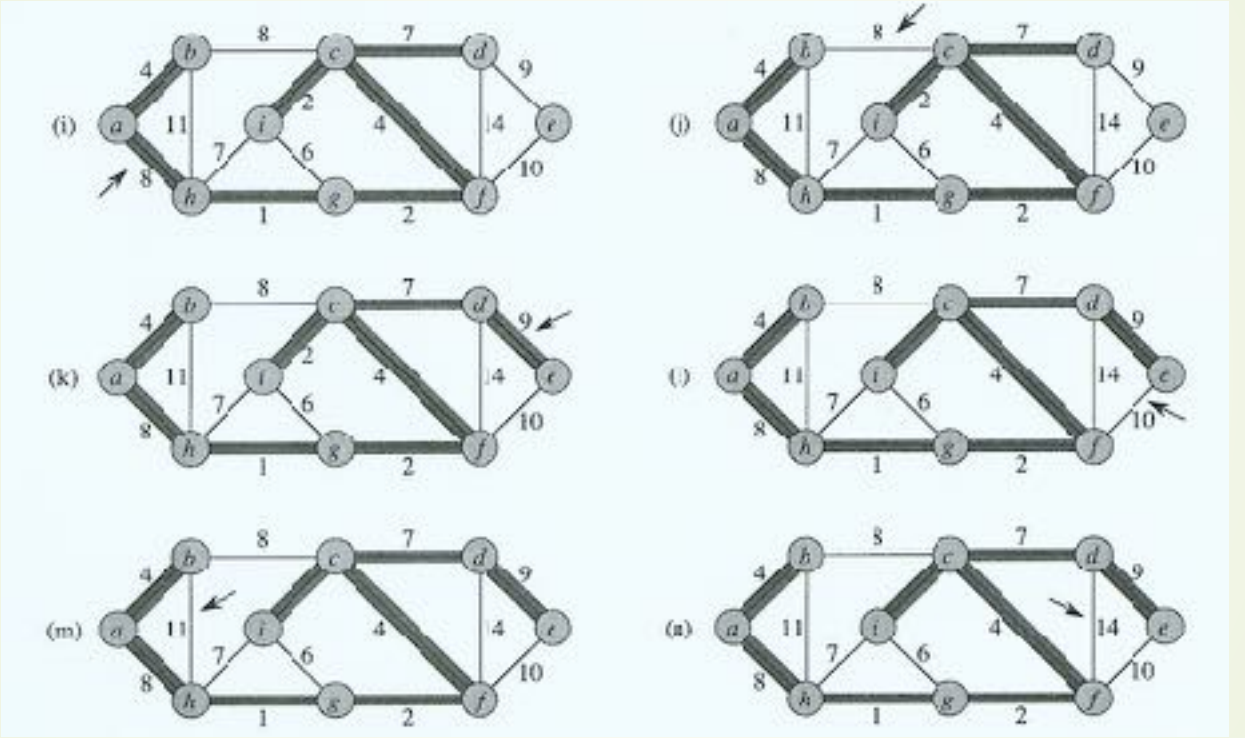


Figure 23.4 The execution of Kruscal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.


(i)
(j)
(k)
(l)
(m)
(n)

Minimum Spanning Tree

```
MST-KRUSKAL(G, w)
1.  A ← ∅
2.  for each vertex v ∈ V G|
3.      do MAKE-SET(v)
4.  sort the edges of E by nondecreasing weight w
5.  for each edge (u, v) ∈ E, in order by nondecreasing weigh
6.      do if FIND-SET(u) ≠ FIND-SET(v)
7.            then A ← A ∪ {(u, v)}
8.                 UNION(u, v)
9.  return A
```

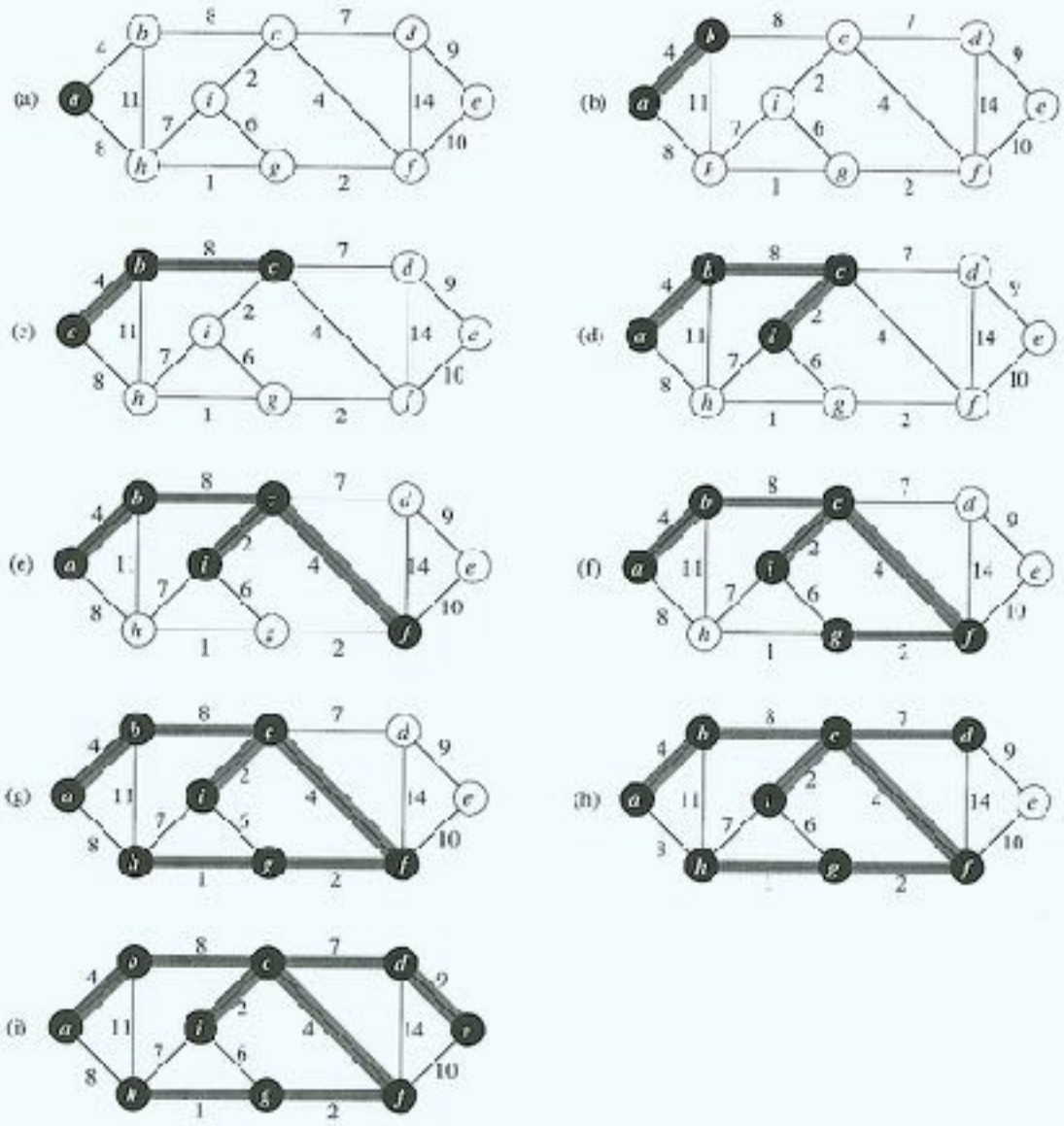


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.

```
MST-KRUSKAL(G, w)
1.  A ← ∅
2.  for each vertex v ∈ V[G]
3.      do MAKE-SET(v)
4.  sort the edges of E by nondecreasing weight w
5.  for each edge (u, v) ∈ E, in order by nondecreasing weigh
6.      do if FIND-SET(u) ≠ FIND-SET(v)
7.            then A ← A ∪ {(u, v)}
8.                 UNION(u, v)
9.  return A
```

```
MST-PRIM(G, w, r)
1.  Q ← V[G]
2.  for each u ∈ Q
3.      do key[u] ← ∞
4.  key[r] ← 0
5.  π[r] ← NIL
6.  while Q ≠ ∅
7.        do u ← EXTRACT-MIN(Q)
8.           for each v ∈ Adj[u]
9.               do if v ∈ Q and w(u, v) < key[v]
10.                 then π[v] ← u
11.                      key[v] ← w(u, v)
```

Proof of Correctness: MST Algorithms

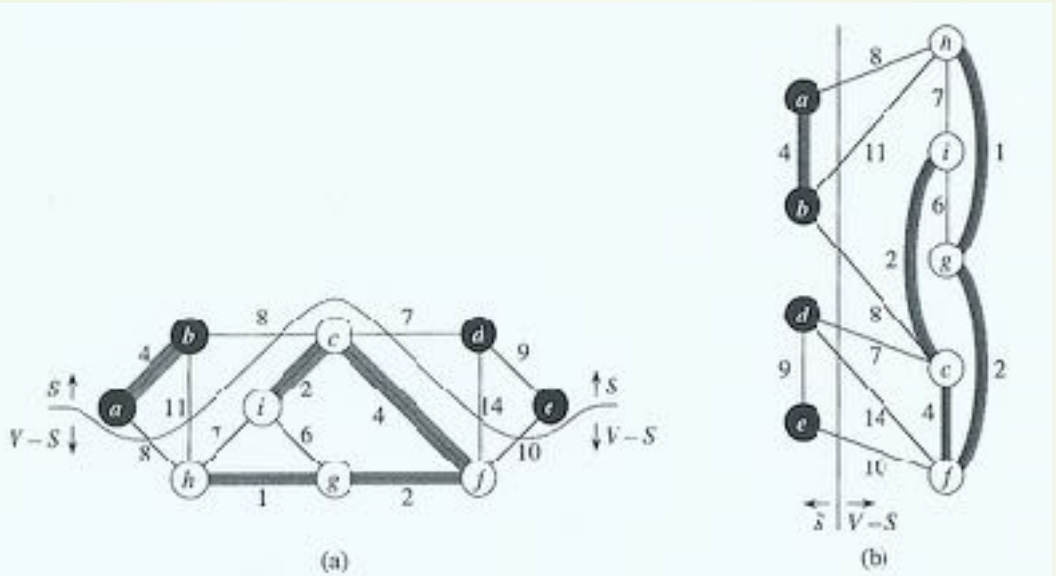


Figure 23.2 Two ways of viewing a cut $(S, V - S)$ of the graph from Figure 23.1. **(a)** The vertices in the set S are shown in black, and those in $V - S$ are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge crossing the cut. A subset A of the edges is shaded; note that the cut $(S, V - S)$ respects A, since no edge of A crosses the cut. **(b)** The same graph with the vertices in the set S on the left and the vertices in the set $V - S$ on the right. An edge crosses the cut if it connects a vertex on the left with a vertex on the right.