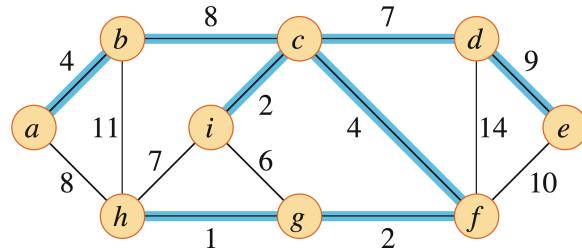


# Chapter 21: Minimum Spanning Trees

Given a connected, undirected graph  $G = \langle V, E \rangle$ , each edge  $(u, v) \in E$  has a weight  $w(u, v)$ . The minimum spanning tree  $T$  is an acyclic subset of  $E$  that connects all vertices of  $V$  and whose weight  $w(T) = \sum_{(u,v) \in T} w(u, v)$  is minimized.

Since  $T$  is acyclic and connects all vertices, it is a tree. We call it a **minimum-weight spanning tree**. It is usually just called a **minimum spanning tree**.

The two algorithms that we will study both use a **greedy approach** to the problem but they differ in how they apply this approach.



## Growing a Minimum Spanning Tree

- The generic algorithm:

**Loop Invariant:** Manage a set  $A$  of edges that is always a subset of some minimum spanning tree.

Initially  $A = \emptyset$ . At each step, we add an edge  $(u, v) \in E$  into  $A$ , where  $(u, v)$  is **safe** to  $A$ .

An edge  $(u, v)$  is **safe** to  $A$  if  $A \cup \{(u, v)\}$  is still a subset of some minimum spanning tree.

```
Generic-MST(G, w)
1. A = {} //empty set
2. while A does not form a minimum spanning tree
3.     find an edge (u,v) that is safe to A
4.     A = A U { (u, v) }    // U: union
5. return A
```

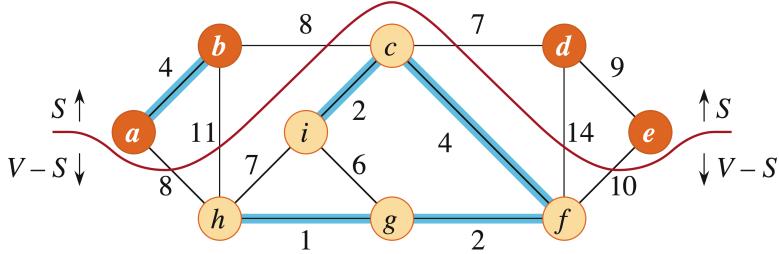
- What kind of edges are safe to  $A$ ?

- Let us define a **cut**  $(S, V - S)$  of an undirected graph  $G = \langle V, E \rangle$  as a partition of  $V$ .
- An edge  $(u, v) \in E$  **crosses the cut**  $(S, V - S)$  if one of its end points is in  $S$  and the other one is in  $V - S$ .
- A **cut respects a set A of edges** if no edge in  $A$  crosses the cut.

- An edge is a **light edge crossing a cut** if its weight is the minimum among all edges crossing the cut.

**Theorem:** If  $A$  is a subset of  $E$  that is included in some minimum spanning tree for  $G$ . Let  $(S, V - S)$  be any cut of  $G$  that respects  $A$ , and let  $(u, v)$  be a light edge crossing  $(S, V - S)$ . Then, the edge  $(u, v)$  is safe to  $A$ .

We will skip the proof of the theorem. The above theorem suggests a way to find a minimum spanning tree. For example:



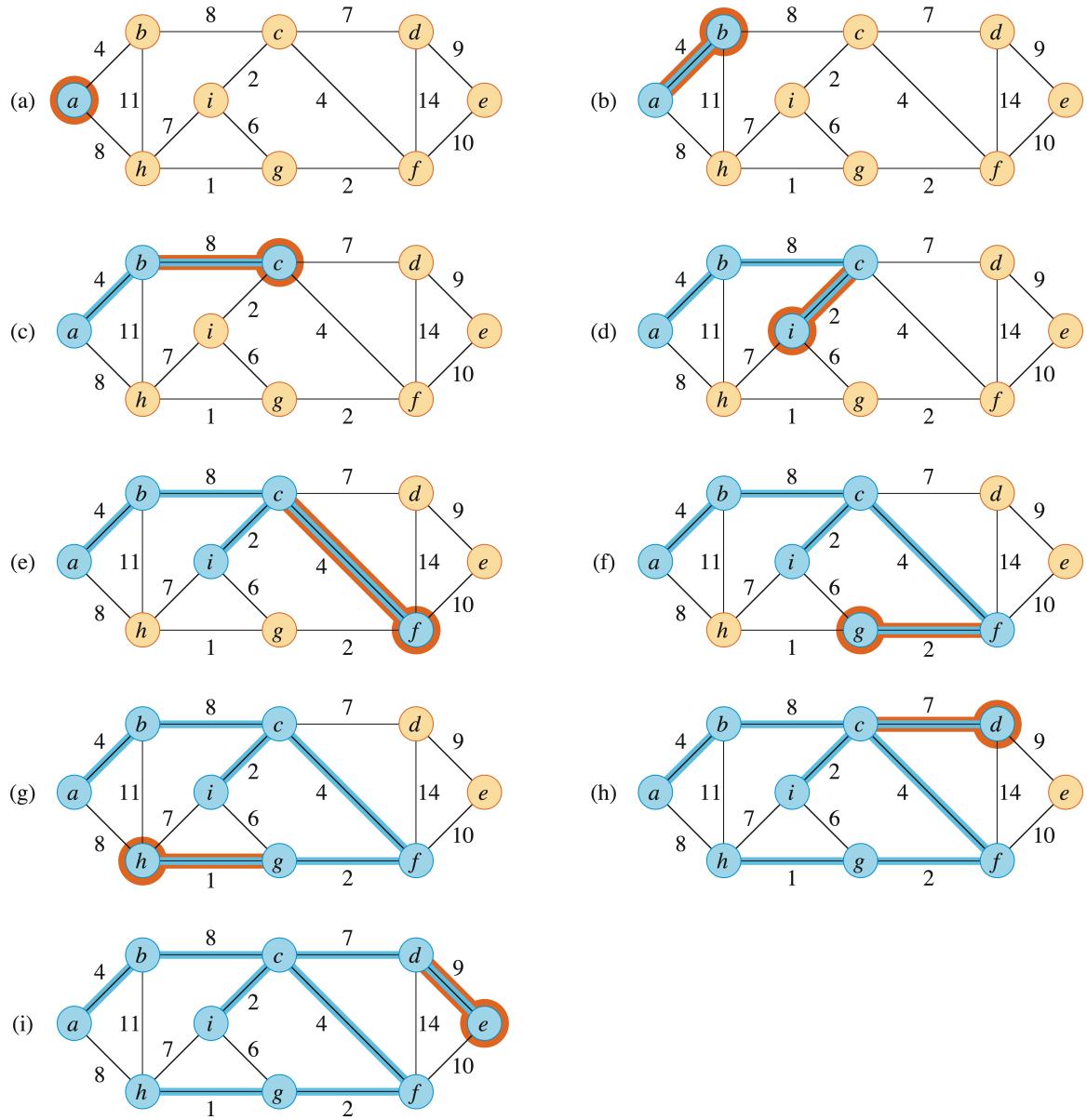
It is easy for human with eye vision to find a sequence of cuts that respects  $A$ . But for a computer algorithm, it is not easy. Any minimum spanning tree algorithm tries to suggest a sequence of cuts respects the growing  $A$ .

## Prim's Algorithm

The set  $A$  maintained is a growing tree. The safe edge added to  $A$  at each step is always the least-weight edge crossing the cut  $(B, V - B)$ , where  $B$  is the set of vertices connected by edges in  $A$ . The algorithm is greedy because the tree is augmented at each step with an edge that contributes the minimal amount of possible weight.

```
MST-PRIM(G, w, r) // r: source vertex
1. for each u in G.V
2.   u.key = infinity
3.   u.parent = NIL
4. r.key = 0
5. Q = {} // Q: priority queue
6. for each vertex u in G.V
7.   INSERT(Q, u)
8. while Q != {}
9.   u = EXTRACT-MIN(Q) // remove from PQ and add u to the tree
10.  for each v in G.Adj[u] //update keys of u's non-tree neighbors
11.    if v in Q and w(u,v) < v.key
12.      v.parent = u
13.      v.key = w(u,v)
14.      DECREASE-KEY(Q, v, w(u,v))
```

Example:



Prim's algorithm uses a priority queue  $Q$  based on a key field.

The queue  $Q$  maintains all vertices that are still not in the growing tree.

The *key* field for each vertex  $v$  in  $Q$ ,  $v.key$ , is the weight of the light edge connecting  $v$  to the tree.

Actually, the sequence of cuts suggested by Prim is the sequence of cuts that separate the growing tree from the rest of vertices at each step.

## Kruskal's Algorithm

The safe edge added to  $A$  at each step is always the least-weight edge in the graph that connects two distinct components.

The algorithm is greedy because at each step it adds to  $A$  an edge with the least possible weight.

```
MST-KRUSKAL (G, w)
1. A = {}
2. for each vertex v in G.V
3.     do MAKE-SET(v)
4. create a single list of the edges in G.E
5. sort the list of edges monotonically increasing weight w
6. for each edge (u,v) taken from the sorted list in order
7.     do if FIND-SET(u) != FIND-SET(v)
8.         then A = A U {(u,v)}
9.             UNION(u,v)
10. return A
```

Example:

