Lakehead
UNIVERSITY

COMP 4433: Algorithm Design and Analysis

Dr. Y. Gu

Feb. 27, 2023 (Lecture 11)

Midterm Help Info

- TA Khaled will be available on Friday morning (March 3), exact time TBD and will be announced on the course website.

- Instructor’s Q&A session will be on Saturday afternoon (March 4) 1- 3. Zoom link will be set up and provided on course website.

- TA Mohammad will be available on Sunday (March 5) 6-8pm. Zoom link will be available on course website announcement.

Greedy Algorithm and Minimum Spanning Tree (MST)

Overview

- Define the problem of MST
- Review of some elementary graph algorithm
  - Breadth-First Search
  - Depth-First Search
- Greedy Algorithm for MST
- The algorithms of Kruskal and Prim

Review Graph Algorithms (Cont.)

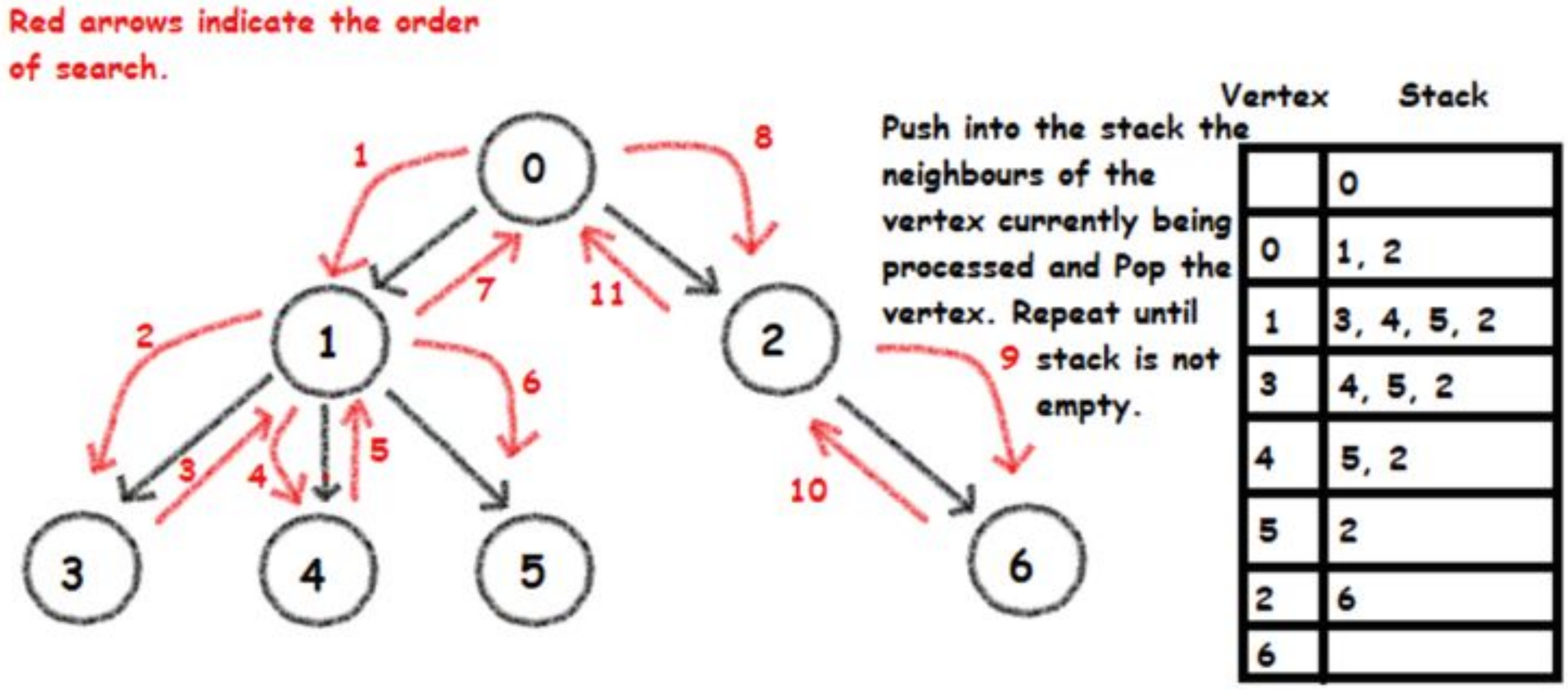
The Depth-first search

DFS may be composed of several trees that is different from the BFS. Instead define a predecessor tree, we define predecessor subgraph (may be a forest) as $G_\pi = (V, E_\pi)$, where

$$E_\pi = \{(v.\pi, v) : v \in V \text{ and } v.\pi \neq \text{NIL}\}.$$

For the DFS, we visit the vertex in depth recursively and then search backtracks (actually used stack since we use recursive procedure calling). We use two attributes to record time-stamps.

One Example



Red arrows indicate the order of search.

Push into the stack the neighbours of the vertex currently being processed and Pop the vertex. Repeat until stack is not empty.

Vertex	Stack
	0
0	1, 2
1	3, 4, 5, 2
3	4, 5, 2
4	5, 2
5	2
2	6
6	

Depth First Search

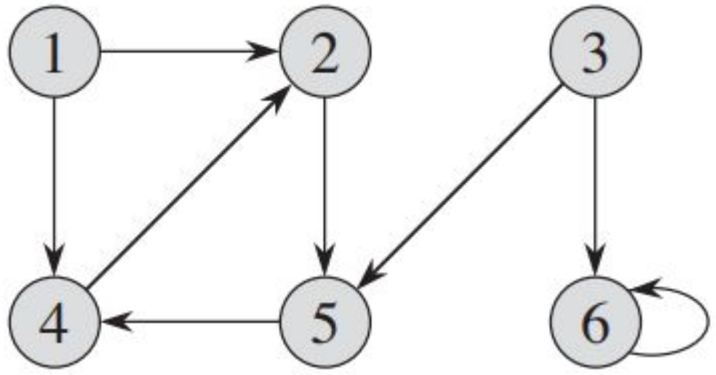
DFS Sample Code

```
DFS($G$)
for each vertex $u \in G.V$
    $u.color$ = WHITE
    $u.\pi$ = NIL
$time = 0$
for each vertex $u \in G.V$
    if $u.color$ == WHITE
        DFS-VISIT($G, u$)
```

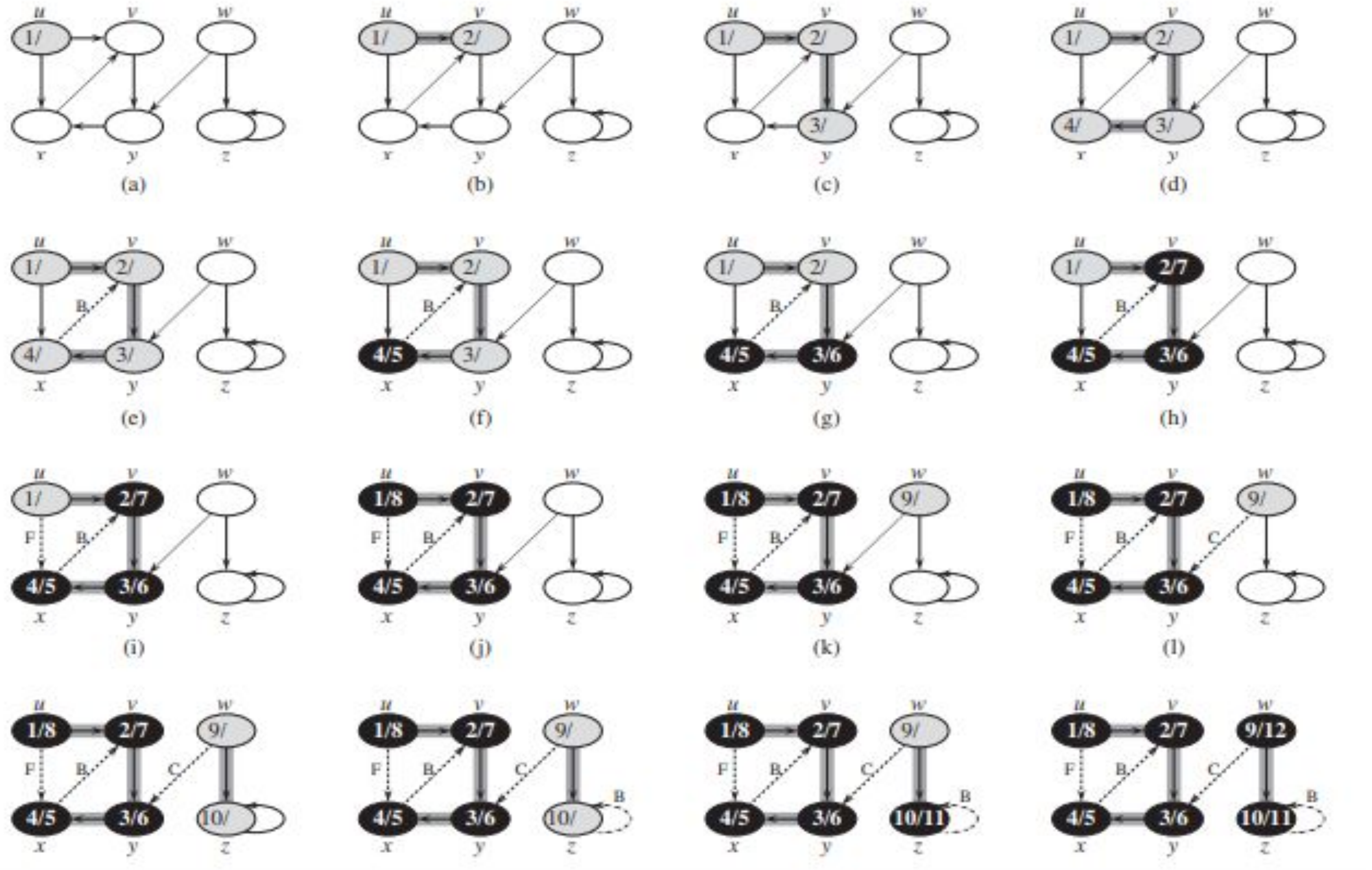
```
DFS-VISIT($G, u$)
$time = time + 1$                // white vertex $u$ has just been discovered
$u.d = time$
$u.color$ = GRAY
for each $v \in G.Adj[u]$        // explore edge $(u, v)$
    if $v.color$ == WHITE
        $v.\pi = u$
        DFS-VISIT($G, v$)
$u.color$ = BLACK                // blacken $u$; it is finished
$time = time + 1$
$u.f = time$
```

$v.d$ records when v is first discovered (grayed v), and $v.f$ records the the search finishes v's adjacency list (blackens v).

DFS Algorithm Illustration



DFS Algorithm Illustration



DFS Analysis and Application

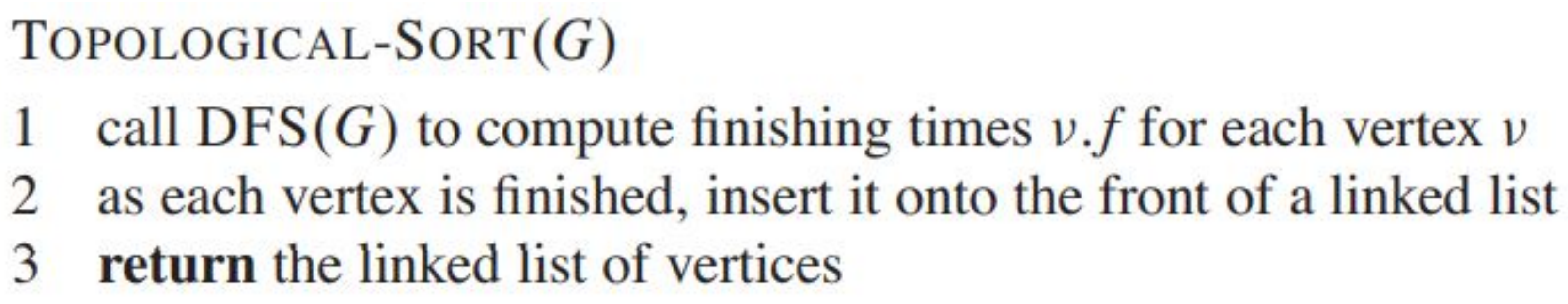
Since $\sum_{v \in V} |adj[v]| = \Theta(E)$ in DFS-Visit, and the initialization and the for loop in line 4 of DFS execute $\Theta(V)$ time, the running time for the DFS is $\Theta(V + E)$.

As an application of DFS procedure, we consider a **topological sort** of a directed acyclic graph, or a **dag**.

A topological sort of a dag G = (V, E) is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before v in the ordering.

Note that if the graph contains a cycle, then no liner ordering is possible.

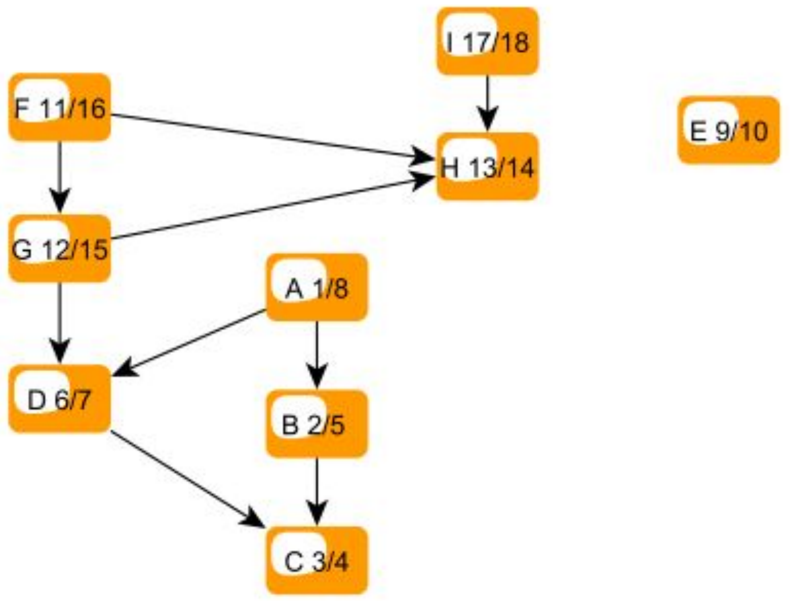
Topological Sort

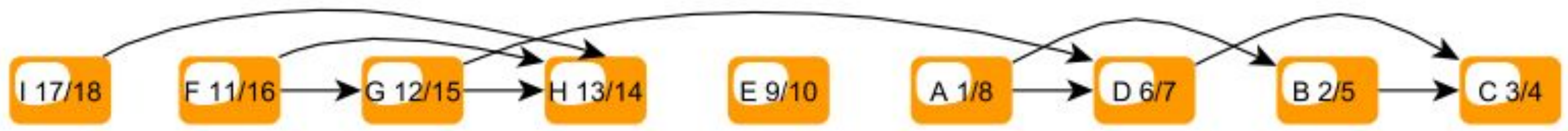


```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

We can perform a topological sort in time Θ(V + E), since DFS takes Θ(V + E) time and it takes O(1) time to insert each of the |V | vertices onto the front of the linked list.

Topological Sort Example





Greedy Algorithm for MST

Definition of MST Problem

Let G = (V, E) be an undirected connected graph with a weight function: E → R. An acyclic set T ⊆ E that connects all of the vertices of G is called a spanning tree of G. We want to find T whose total weight

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

is minimum.

Such a problem is called minimum-spanning-tree problem.

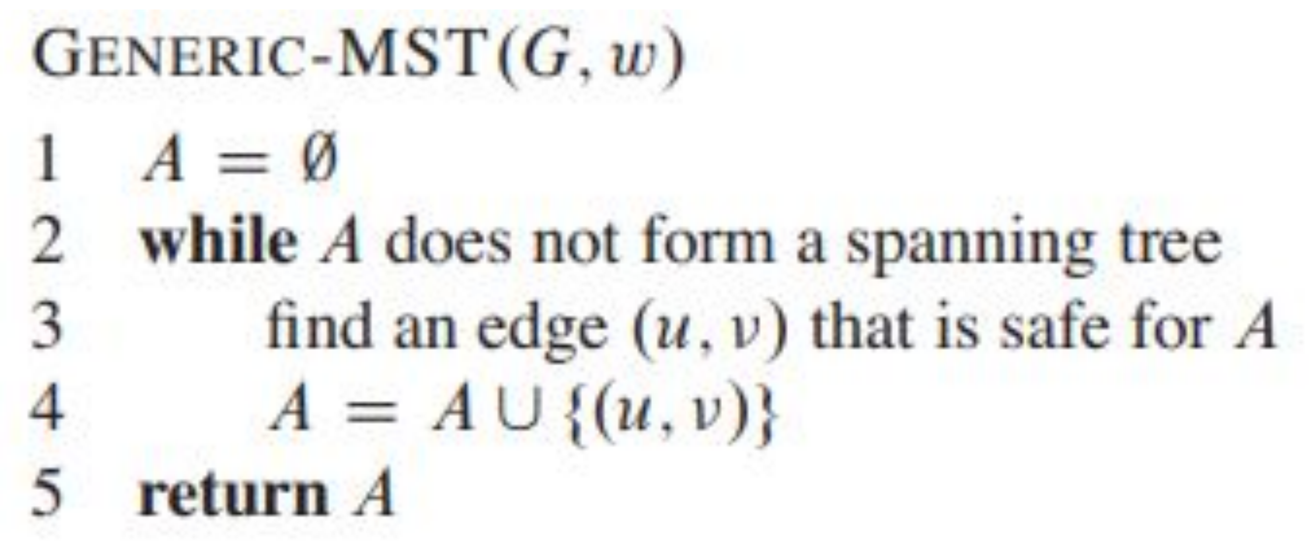
Greedy Algorithm of MST

We can use a greedy approach to the problem.

The main idea is that we can grow the minimum tree one edge at a time such that the subset chosen is a subset of some minimum spanning tree.

Suppose a subset A is chosen, we can determine an edge (u, v) that we can add to A such that A $\cup$ {(u, v)} is also a subset of a minimum spanning tree. We call such an edge a **safe edge** for A.

Greedy Algorithms of MST



```
GENERIC-MST(G, w)
1  $A = \emptyset$
2  while $A$ does not form a spanning tree
3      find an edge $(u, v)$ that is safe for $A$
4      $A = A \cup \{(u, v)\}$
5  return $A$
```

The initialization $A = \varnothing$ in the procedure satisfies the loop invariant. The maintenance is done by adding safe edge. We need to prove that safe edge exists and we have some method to find out it.

Analysis of the Algorithm

To prove that, we need some definitions.

- A cut (S, V-S) of an undirected graph G = (V, E) is a partition of V .
- We say that an edge (u, v) ∈ E **crosses** the cut (S, V − S) if one of its endpoints is in S and the other is in V − S.
  We say that 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 of any edge crossing the cut.

In general we will say that an edge is a **light edge** satisfying a given property if its weight is the minimum of any edges satisfying the property.

Analysis of the Algorithm

Theorem 23.1.

Let G = (V, E) be a connected, undirected graph with a real-values weight function w defined on E. Let A be 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 (u, v) is safe for A.

Greedy Algorithm Analysis

Proof.

Let T be a minimum spanning tree that includes A, and assume that T does not contain the light edge (u, v), since otherwise we are done. We will construct another minimum spanning tree T ′ that includes A ∪ {(u, v)}. If the edge (u, v) is added to T, then it forms a cycle with the edges on the simple path p from u to v in T. Since u and v are on opposite sides of the cut (S, v − S), at least one edge in T lies on the simple path p and also crosses the cut. Let (x, y) be any such edge. The edge (x, y) is not in A, because the cut respects A. Since (x, y) is on the unique simple path from u to v in T , removing (x, y) breaks T into two components. Adding (u, v) reconnects them to form a new spanning tree T ′ = (T − {(x, y)}) ∪ {(u, v)}

Greedy Algorithm Analysis

We next show that T ′ is a minimum spanning tree. Since (u, v) is a light edge crossing (S, V − S) and (x, y) also crosses this cut, w(u, v) ≤ w(x, y). Therefore, w(T ′) = w(T) − w(x, y) + w(u, v) ≤ w(T). But T is a minimum spanning tree , so w(T) ≤ w(T ′). Therefore w(T) = w(T ′) and T ′ must be a minimum spanning tree. Since A ⊆ T ′ and A ∪ {(u, v)} ⊆ T ′ , (u, v) is safe for A.

□

In the procedure Generic-MST and in Theorem 4.3.6, the set A is a subset of edges. A must be acyclic, but not necessary connected. So A is a forest, and each of the connected components is a tree. The while loop in Generic-MST executes |V | − 1 times because the spanning tree has |V | − 1 edges and each loop adds one edge to A

Corollary

Corollary
Let G = (V, E) be a connected, undirected graph with a weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, and let C = (V_C , E_C) be a connected component (tree) in the forest G_A = (V, A). If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

Proof. The cut (V_C , V − V_C) respects A, and (u, v) is a light edge for this cut. Therefore, (u, v) is safe for A.

The Algorithms of Kruskal and Prim

To use the Generic-MST, we need some method to find safe edge in the statement line 3 of the procedure. Two algorithms described here elaborate on that method. For the implementation of graphs, we use the adjacency lists.

- In **Kruskal**'s algorithm, the set A is a **forest** whose vertices are all those of the given graph. The safe edge added to A is always a least-weight edge in the graph **that connects two distinct components**.
- In **Prim**'s algorithm, the set A forms a **single** tree. The safe edge added to A is always a least-weight edge **connecting the tree to a vertex not in the tree**.

The algorithms of Kruskal and Prim

To use the Generic-MST, we need some method to find safe edge in the statement line 3 of the procedure. Two algorithms described here elaborate on that method. For the implementation of graphs, we use the adjacency lists.

- In **Kruskal**’s algorithm, the set A is a forest whose vertices are all those of the given graph. The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.
- In **Prim**’s algorithm, the set A forms a single tree. The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

Kruskal Algorithm Intuition

- Create a forest *F* (a set of trees), where each vertex in the graph is a separate tree
- Create a sorted set *S* containing all the edges in the graph
- While *S* is nonempty and *F* is not yet spanning
  - remove an edge with minimum weight from *S*,
  - if the removed edge connects two different trees then add it to the forest *F*, combining two trees into a single tree.

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Kruskal Algorithm

To implement Kruskal Algorithm, we need some simple procedures to maintain the "forest". For a vertex x, we assign a parent x.p (some vertex which represent the subset that contains x) and a rank p.rank (an integer which can be viewed as the level in a tree that x sits) to it. To initialize the setting, the following procedure is called.

```
Make-Set(x)
1    x.p = x
2    x.rank = 0
```

Kruskal Algorithm

Then we need to merge some subsets of the vertices into one subset. Suppose x and y are two vertices in two disjoint subsets. We want to merge them to one subset. Then basically we just need to change the parent for one of the vertices. The following procedure decides how to change one parent.

```
Link(x, y)
if (x.rank > y.rank )
    y.p = x
else
    x.p = y
    if (x.rank == y.rank)
        y.rank = y.rank + 1
```

Kruskal Algorithm

The procedure uses the vertex with larger rank as the parent. In this way, we can keep the height of the tree lower. Now if you have changed the parent of x, then all the vertices in the same subset need to be changed. The procedure Find-Set is used to find the parent of a vertex in general.

```
  Find-Set(x)
1    if $x \neq x.p$
2         $x.p$ = Find-Set ($x.p$)
3    return $x.p$
```

Kruskal Algorithm

The procedure uses the vertex with larger rank as the parent. In this way, we can keep the height of the tree lower. Now if you have changed the parent of x, then all the vertices in the same subset need to be changed. The procedure Find-Set is used to find the parent of a vertex in general.

```
  procedure Find-Set(x)
1    if x ≠ x.p
2         x.p = Find-Set (x.p)
3    return x.p
```

Kruskal Algorithm

UNION(x, y) unites the dynamic sets that contain x and y, say S_x and S_y, into a new set that is the union of these two sets.

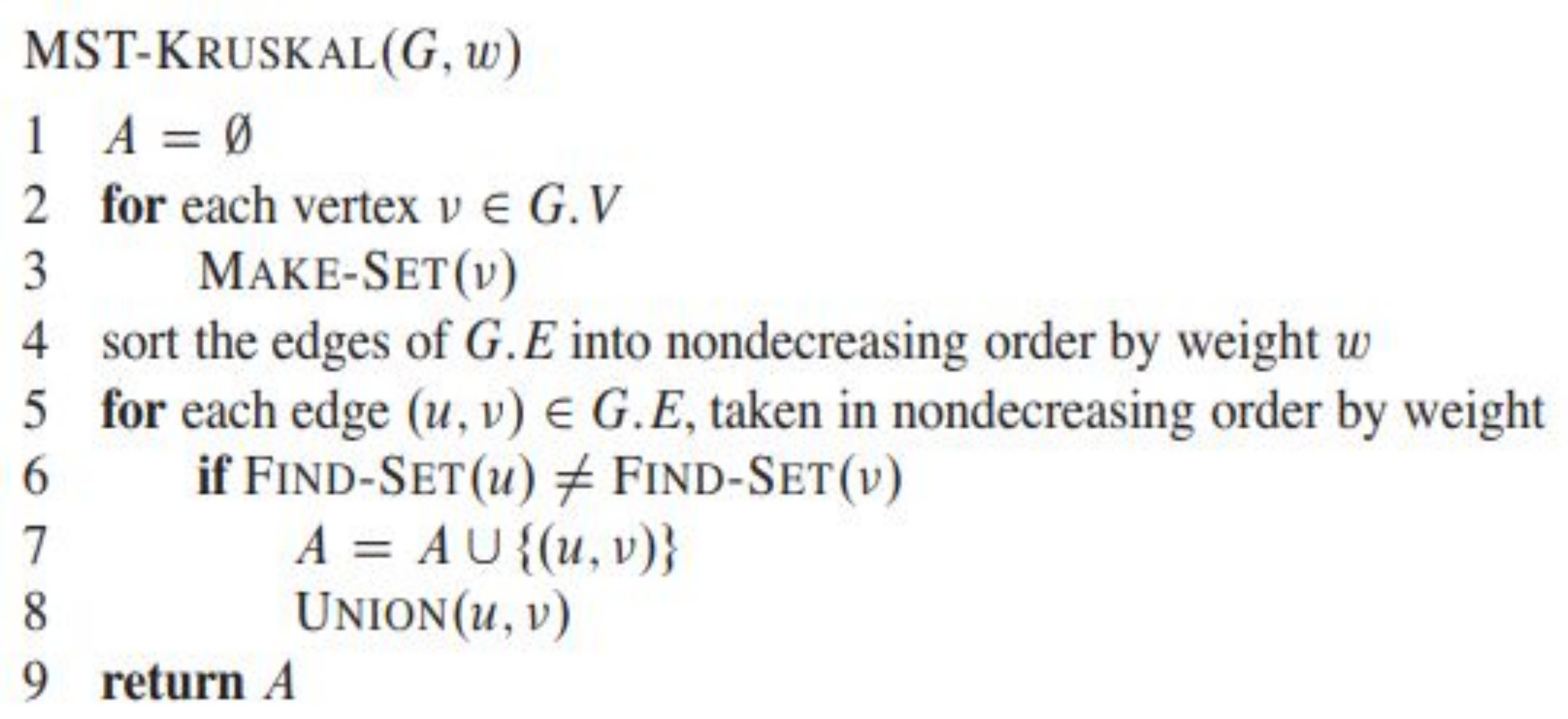
```
  UNION(x, y)
1         LINK(FIND-SET(x), FIND-SET(y))
```

Kruskal Algorithm

UNION(x, y) unites the dynamic sets that contain x and y, say S_x and S_y, into a new set that is the union of these two sets.

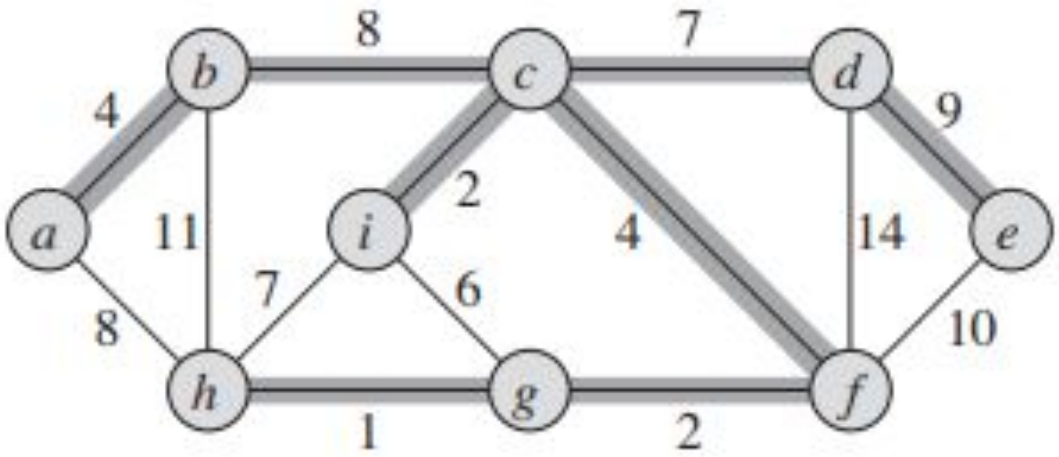
```
  UNION(x, y)
1         LINK(FIND-SET(x), FIND-SET(y))
```

Kruskal Algorithm

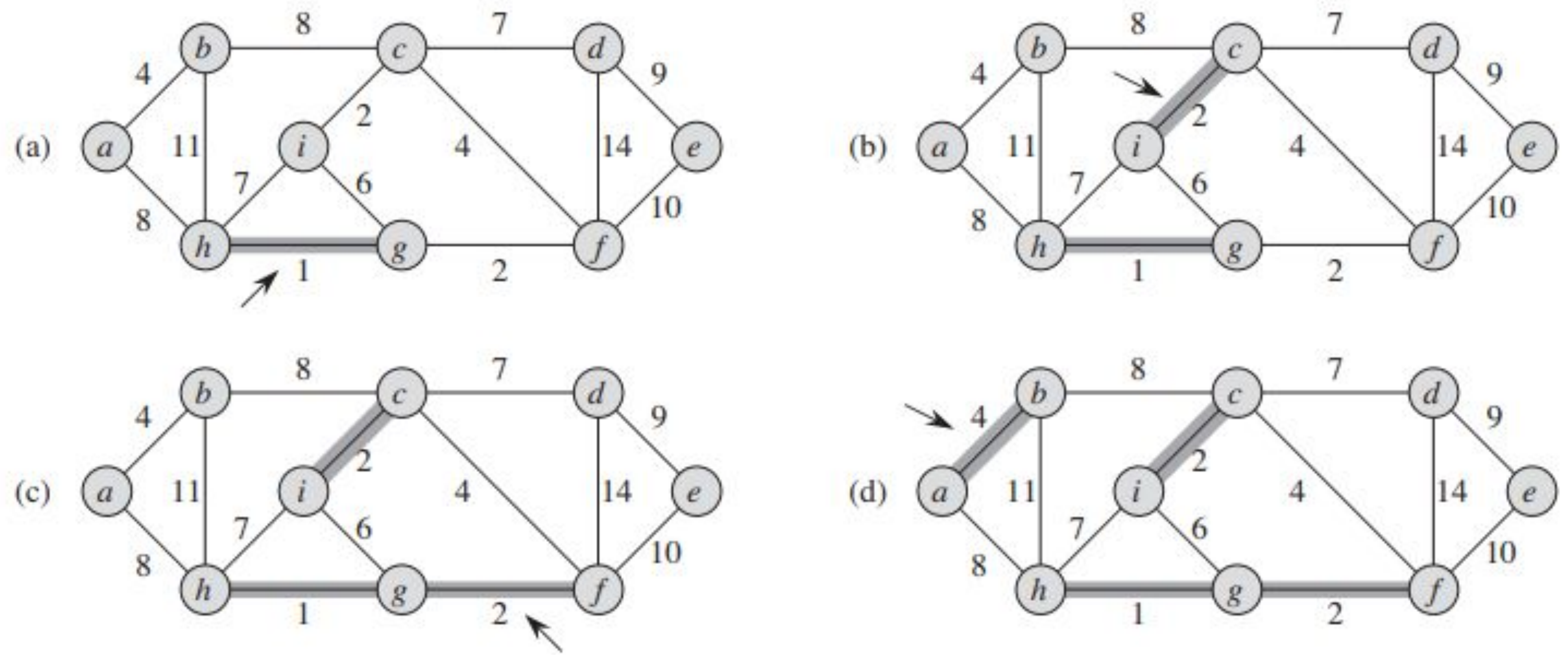


```
MST-KRUSKAL($G, w$)
$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET($v$)
sort the edges of $G.E$ into nondecreasing order by weight $w$
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
    if FIND-SET($u$) $\neq$ FIND-SET($v$)
        $A = A \cup \{(u, v)\}$
        UNION($u, v$)
return $A$
```

Example

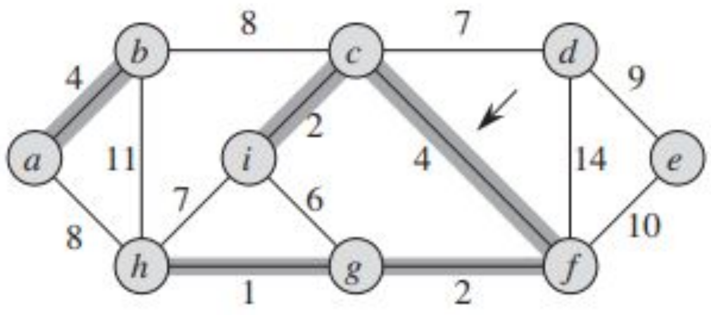


Example

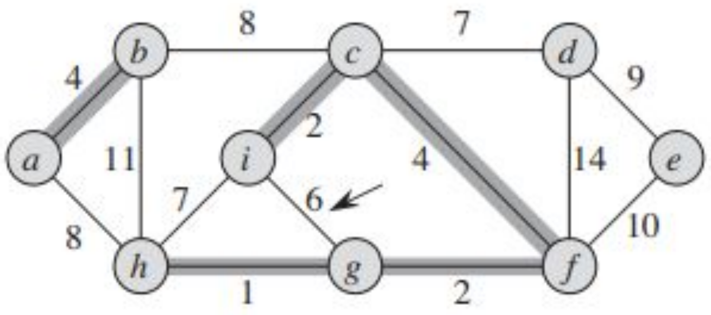


Example

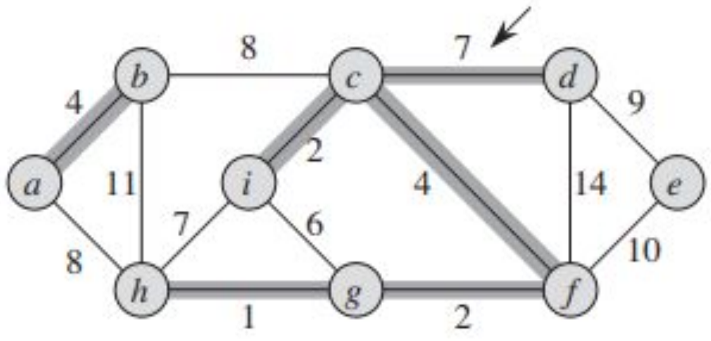
(e)



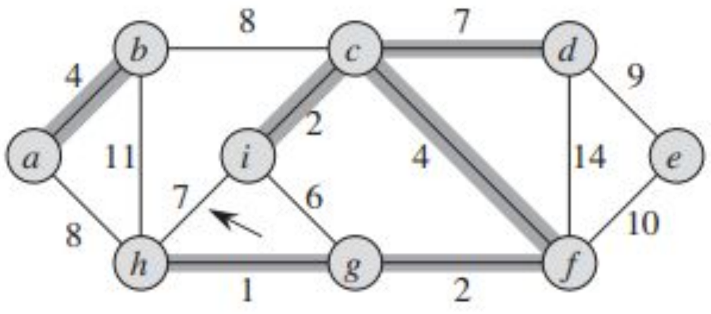
(f)



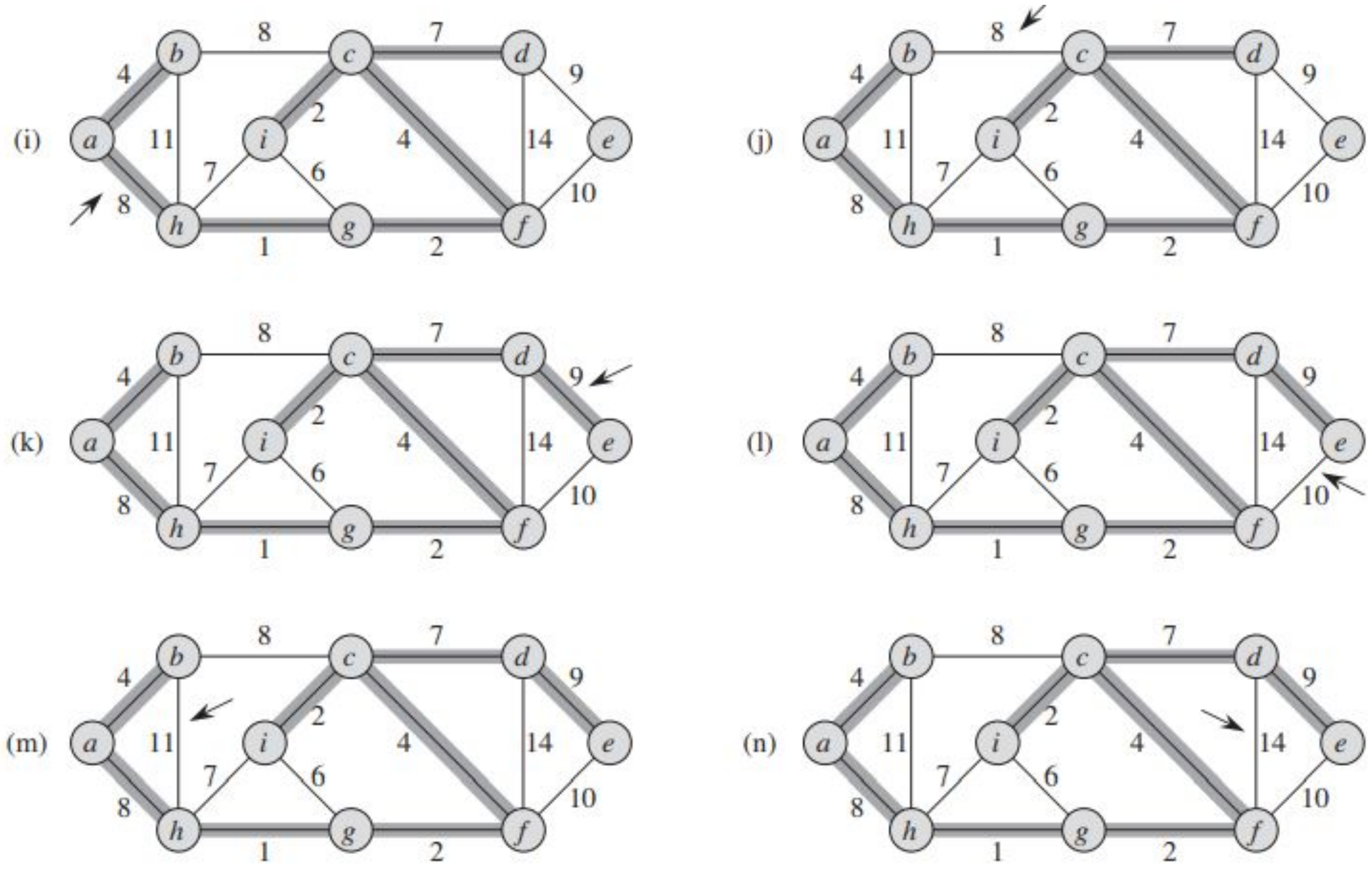
(g)



(h)



Example



Kruskal Algorithm Analysis

The for loop in line 5 examines edges in order of weight, from lowest to highest. The loop checks, for each edge (u, v), whether u and v belong to the same subtree. If they do, then they cannot be added to the forest, and so edge is discarded. Otherwise, the edge (u, v) is added to A and two subtrees are merged to one subtree.

Now we consider the running time of MST-Kruskal. The sort in line 4 is O(E lg E). When we use the Link to merge the subtrees, the height of the tree is lg V . The for loop in line 5 takes O(E) Find-Set and Union operations on the disjoint forest. Along with the |V | Make-Set operations, these take a total of O((V + E) lg V) time. Since G is connected, we have $|V| - 1 \le |E| \le |V|^2$. So we have that the running time of Kruskal's algorithm is O(E lg V).

Prim’s Algorithm Intuition

The Prim’s algorithm is also based on the generic greedy algorithm.

In Prim’s algorithm, the set A forms a single tree. The safe edge added to A is a least-weight edge connecting the tree to a vertex not in tree.

The algorithm may informally be described as performing the following steps:

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

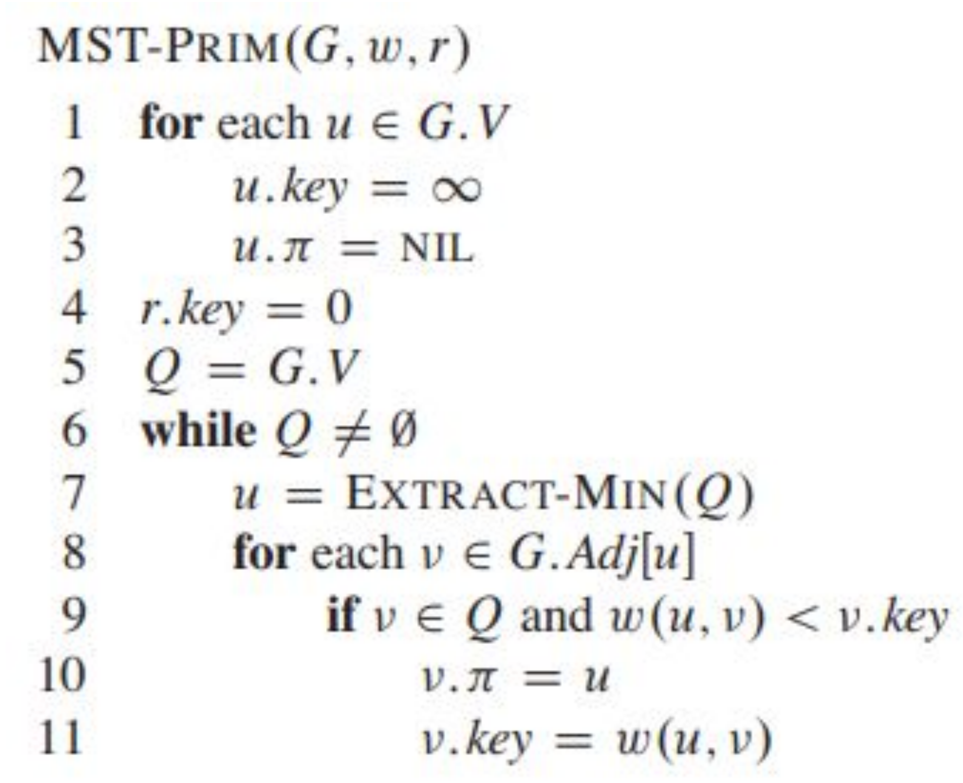
Prim’s Algorithm

In the Prim’s algorithm, each vertex v is assigned an attribute key which is the minimum weight of any edge connecting v to a vertex in the tree. If no such an edge exist, v.key =∞. Another attribute v.π names the parent of v in the tree.

We use a min-priority queue Q based on the key attributes to house all the vertices not in the tree yet. The Extract-Min (Q) will return the minimum element and then delete it from Q. The algorithm implicitly maintains the set A as

$$A = \{(v, v.\pi) : v \in V - \{r\} - Q\}.$$

Prim’s Algorithm

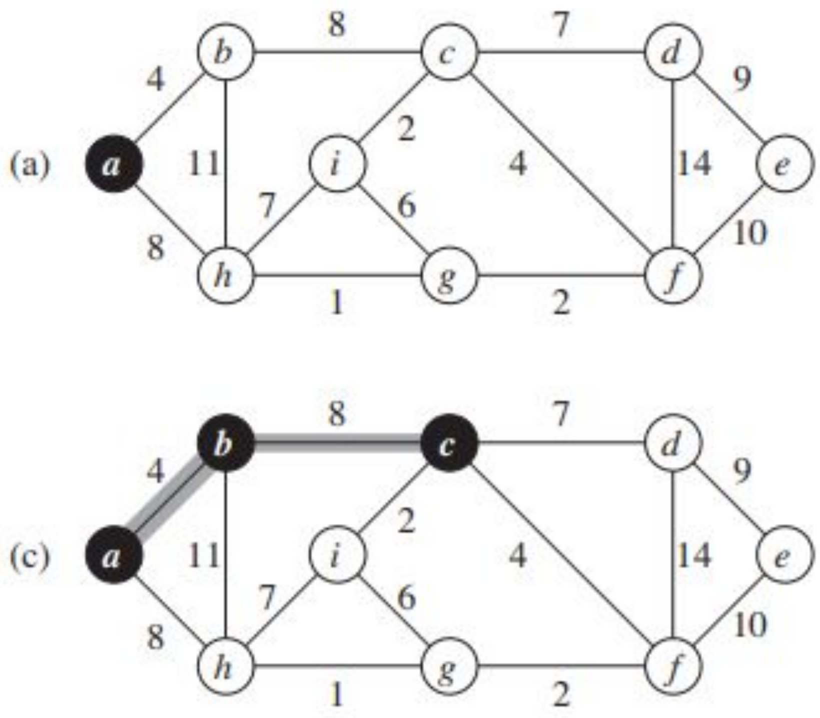


```
MST-PRIM$(G, w, r)$
for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$
```

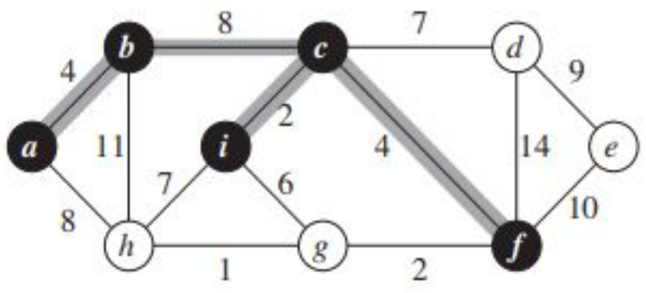
When the algorithm terminates, the min-priority queue Q is empty; the minimum spanning tree A for G is thus

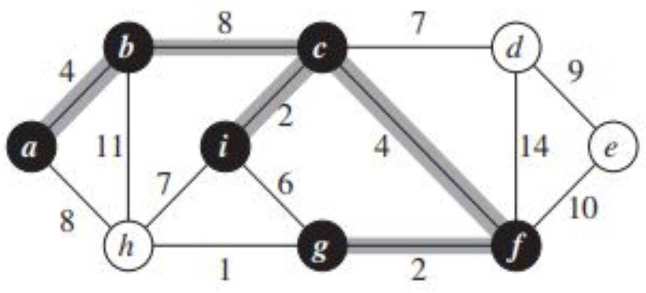
$$A = \{(v, v.\pi) : v \in V - \{r\} \}.$$

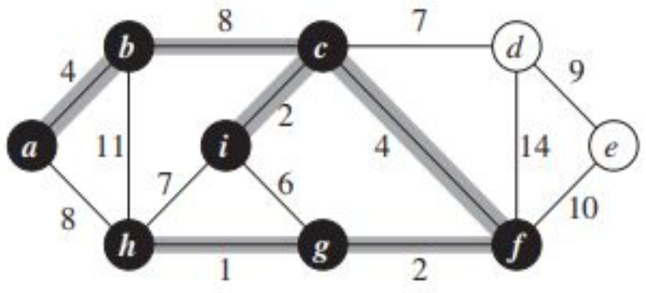
Prim’s Algorithm Example

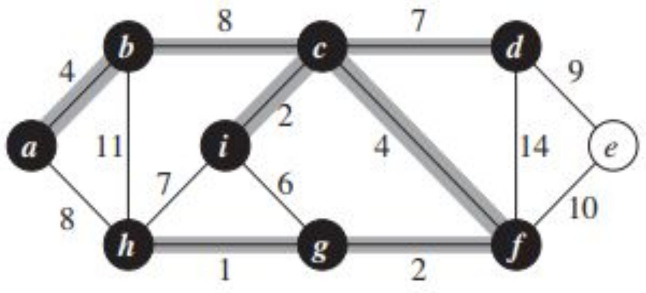


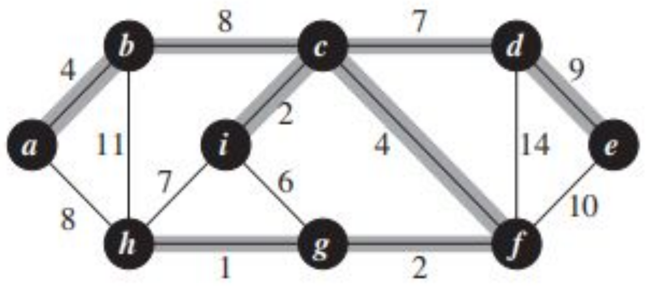
Prim’s Algorithm Example

(e)


(f)


(g)


(h)


(i)


Prim’s Algorithm Analysis

The initial Q uses O(V) time and we can arrange Q as min-heap.

The while in line 6 executes |V| times, and since each Extract-Min operation takes O(lg V) time, the total time for all calls to Extract-Min is O(V lg V).

The for loop in line 8 executes O(E) times altogether, since the sum of the lengths of all adjacency lists is 2|E|. Since the Q is a min-heap, the operations are in O(lg V) time.

The total time for Prim’s algorithm is O(V lgV + E lgV) = O(E lgV).

If we use a Fibonacci heap, the running time of Prim’s algorithm improves to O(E + V lg V).

After Class

- After class: Part V 21.1-21.3, Part VI 22.3, 23.1