Minimum Spanning Tree

Let G = (V, E, W) be a weighted connected undirected graph. A tree is an acyclic subgraph of G. A spanning tree is a tree that spans all vertices in G; it has |V| vertices and |V| - 1 edges. Let the cost of a spanning tree be the sum of the weights attached to the |V| - 1 edges of the tree. A minimum spanning tree (MST) is a spanning tree whose cost is a minimum.

An MST shows the most economical way of connecting all vertices of a weighted undirected graph together using the edges of the graph. It has many applications: connecting all logic circuits of a VLSI chip to the power source using a minimum amount of silicon area or running telephone wires to a set of houses in a new development, etc.

Sometimes a minimum spanning tree is called a minimum cost spanning tree or MCST.

We will study two Algorithms: Kruskal's and Prim's. G is an undirected weighted connected graph.

MST-Kruskal is an example of a **greedy** algorithm; to find the globally-optimum solution to a problem a greedy algorithm makes a series of locally-optimum choices. The locally-optimum choice of MST-Kruskal is to connect any two trees together with lowest added cost. A greedy algorithm may not produce the globally-optimum solution for a problem.

How can we implement Kruskal's algorithm efficiently? Sort all edges in E by weight so we can then select edges in order by non decreasing weight; Heapsort does this in $O(|E|\lg|E|)$ time. As we select edges we must discard any edge between two vertices in the same tree. The algorithm connects trees together as it proceeds so we need an efficient method of determining whether two vertices are in the same tree or not.

Disjoint-set Forests: All the vertices in a graph tree, T, are put in a tree, S, that's not related to the graph. S-trees are manipulated to keep them short so it is easy to find the root from any vertex in a tree. Two vertices are in the same tree if and only if they have the same S root. When two graph trees are combined, the corresponding S trees are combined by making the root of the shorter tree a child to the root of the longer tree. If *v* is a vertex then *v.p* is a pointer to its parent in the S-tree and *v.rank* is an upper bound on the path length between v and its farthest descendant in the S-tree. If *v* is the root of an S-tree then *v.p* = *v*.

MAKE-SET(v) puts vertex v into its own S-tree. FIND-SET(v) returns the root of the S-tree containing *v* and compresses the path from *v* to its root so later searches go much faster. UNION(u, v) unites the S-tree containing *u* with the S-tree containing *v*.

```
MAKE-SET(v)

v.p = v
v.rank = 0
```

```
UNION(u, v)

LINK(FIND-SET(u), FIND-SET(v))
```

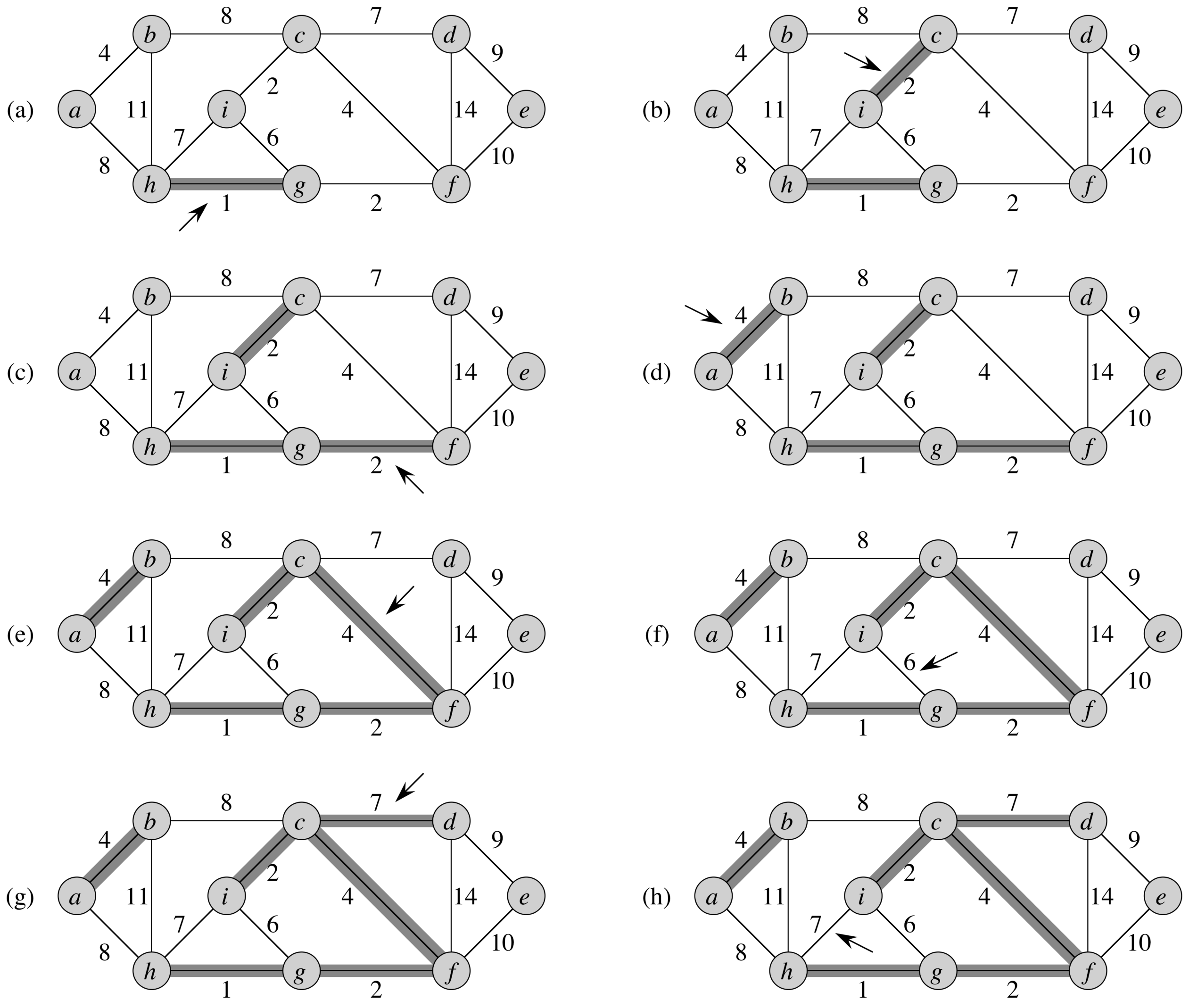
```
LINK(u, v)

if u.rank > v.rank
    v.p = u
else u.p = v
    if u.rank == v.rank
        v.rank = v.rank + 1
```

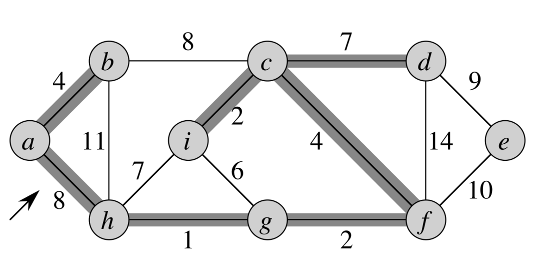
```
FIND-SET(u)

if $u \neq u.p$
    $u.p$ = FIND-SET(u.p)
return u.p
```

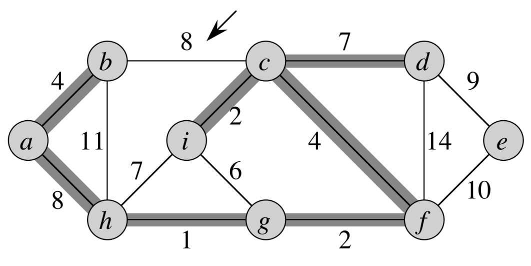
MST-Kruskal(G) returns a set, A, containing the edges in a minimum spanning tree for weighted undirected connected graph G.

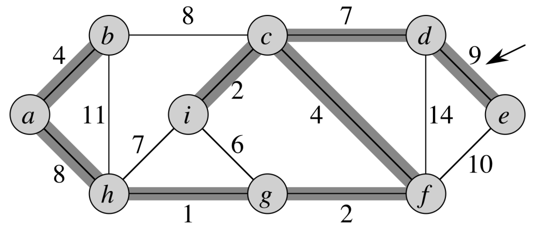
MST-Kruskal(G)

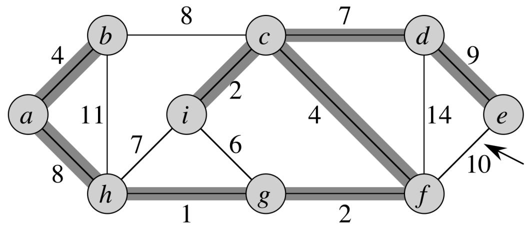
```
A = Φ
for each vertex v ϵ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ϵ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A U {(u, v)}
        UNION(u, v)
return A
```

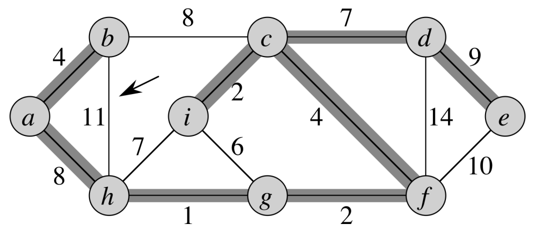

(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

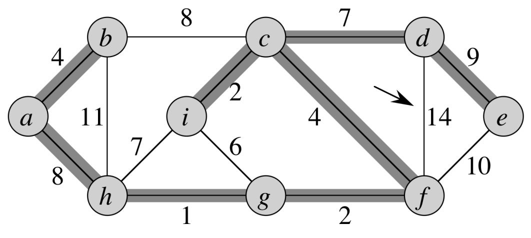
(i)


(j)


(k)


(l)


(m)


(n)


What is the running time for MST-Kruskal? Initialization is $O(|V|)$. Sorting the edges is $O(|E|\ lg|E|)$. The for-loop is executed 2|E| times. FIND-SET is called a total of 4|E| times and UNION is called |V| - 1 times. UNION also calls FIND-SET a total of 2|V| -2 times. It can be shown that all the calls to FIND-SET and UNION take $O(|E|\ lg|E|)$ time. Since the graph is connected $|E| \geq |V| - 1$ so MST-Kruskal runs in $O(|E|\ lg|E|)$ time.

Prim's Algorithm

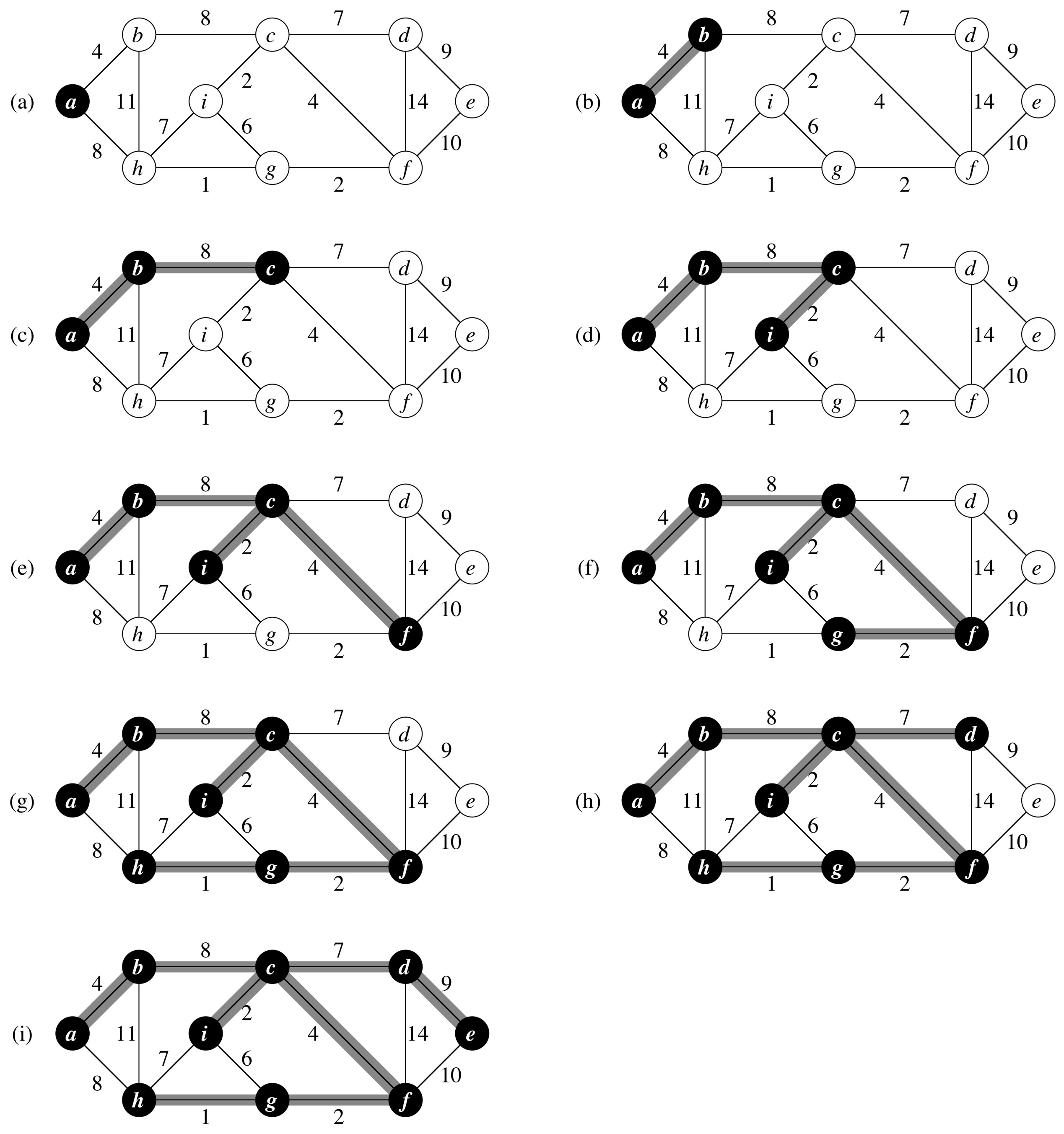
MST-PRIM is a greedy algorithm. How can we implement it efficiently? At any time in MST-PRIM, there are three classes of vertices: T-vertices that are in T; A-vertices not in T but adjacent to T-vertices; and X-vertices that are not adjacent to T-vertices. An A-vertex, v, may be adjacent to several T-vertices so we use variable, $v.\pi$, to store the index of the T-vertex closest to *v* and variable *v.key,* to store w($v.\pi$, v).

For the while-loop in MST-PRIM select an A-vertex, v, with the minimum key. Add vertex v and edge $(v.\pi, v)$ to T and then search Adj[v] for A-vertices and X-vertices adjacent to v. If u is an A-vertex adjacent to v and if w(v, u) < u.key then u is closer to v than to $u.\pi$ so change u.$\pi = v$ and *u.key = w(v, u)*. If u is an X-vertex adjacent to v then change u to an A-vertex with $u.\pi = v$ and *u.key = w(v, u)*

To easily find the A-vertex with the minimum key, we use a priority queue, Q, where maximum priority is given to the vertex with minim key.

MST-PRIM(G, w, r)

```
1   for each $u \in G.V$
2       $u.key = \infty$
3       $u.\pi = NIL$
4   $r.key = 0$
5   $Q = G.V$
6   while $Q \neq \emptyset$
7       u = EXTRACT-MIN(Q)
8       for each $v \in G.Adj[u]$
9           if $v \in Q$ and $w(u, v) < v.key$
10              $v.\pi = u$
11              $v.key = w(u, v)$
```



Prior to each iteration of the while loop of lines 6-11,

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the minimum spanning tree are those in $V - Q$.

3. For all vertices $v \in Q$, if $v.\pi \neq NIL$, then $v.key < \infty$ and *v.key* is the weight of a light edge $(v, v.\pi)$ connecting *v* to some vertex already placed into the minimum spanning tree.

What is the running time for MST-PRIM? The running time of Prim's algorithm depends on how we implement the min-priority queue *Q*. If we implement *Q* as a binary min-heap, we can use the BUILD-MIN-HEAP procedure to perform lines 1-5 in O(|V|) time. The body of the while-loop is executed |V| times and 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 lines 8-11 executes *O*(|*E*|*)* times altogether, since the sum of the lengths of all adjacency lists is 2|E|. Within the for loop, we can implement the test for membership in *Q* in line 9 in constant time by keeping a bit for each vertex that tells whether or not it is in Q, and updating the bit when the vertex that tells whether or not it is in *Q*, and updating the bit when the vertex is removed from *Q*. The assignment in line 11 involves as implicit DECREASE-KEY operation on the min-heap, which a binary min-heap supports in *O*(*lg* |*V*|) time. Thus , the total time for Prim's algorithm is
O(|*V*| *lg*|*V*| + |*E*| *lg* |*V*|) = *O* (|*E*| *lg*|*V*|).