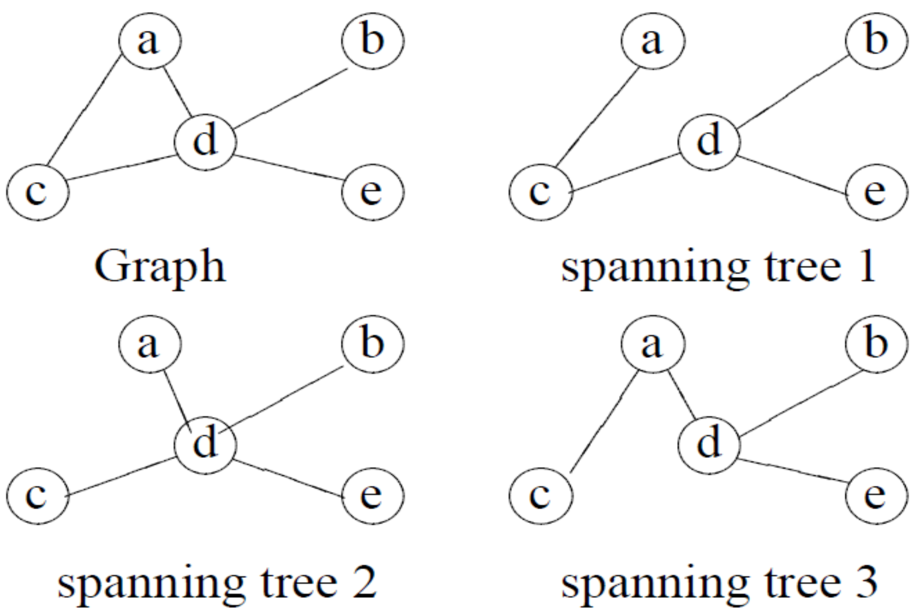
Chapter23. Minimum Spanning Tree(MST)

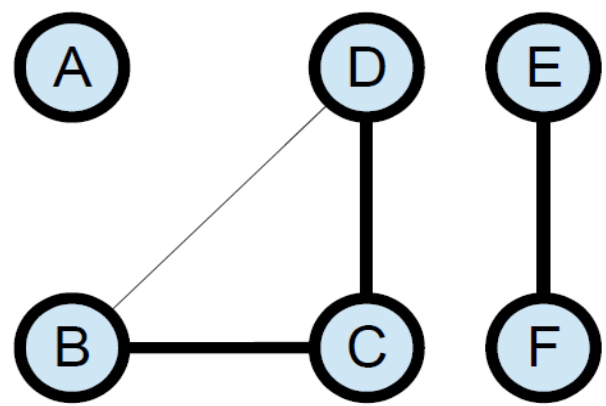
Spanning Trees, Spanning Forest

- Spanning Tree
  - A tree (i.e., connected, acyclic graph) which contains all the vertices of the graph

- Spanning forest
  - If a graph is not connected, then it is not possible to build a spanning tree. Instead, you can build a spanning forest: each connected component gets its own little spanning tree that spans just that component

Spanning Tree, Spanning Forest





Spanning Trees

Spanning Forest

Minimum Spanning Tree(MST)

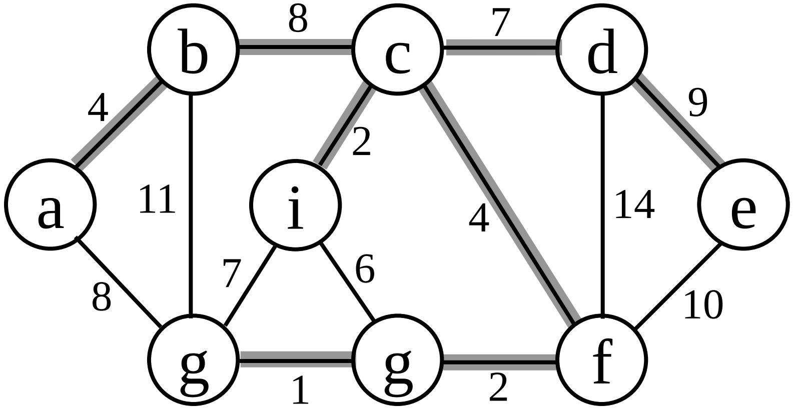
- Given a connected, undirected graph $G = (V, E)$,
  a spanning tree is an acyclic subset $T \subseteq E$ that connects all vertices in V
- If G is weighted, then T's weight is

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

- A minimum weight spanning tree (or minimum spanning tree, or MST) is a spanning tree of minimum weight
  - Not necessarily unique

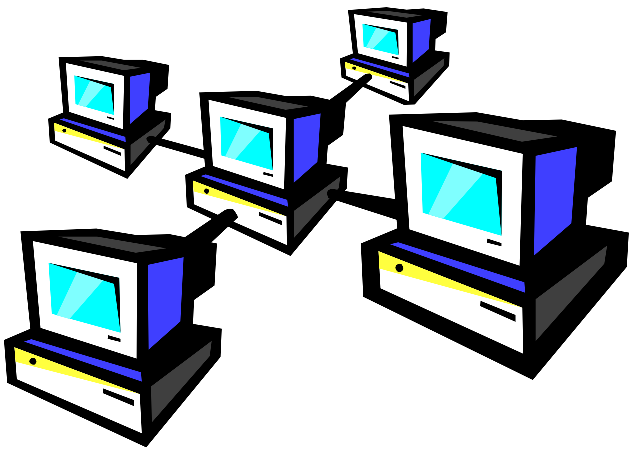
Minimum Spanning Tree(MST)

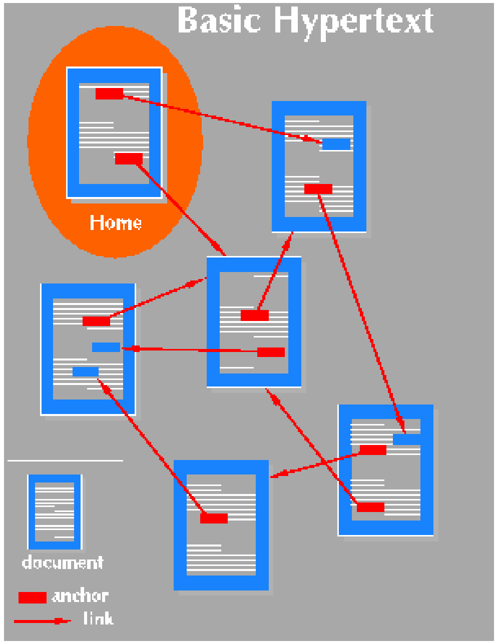
- Minimum Spanning Tree Example



Applications of MST

– Find the least expensive way to connect a set of cities, terminals, computers, etc.



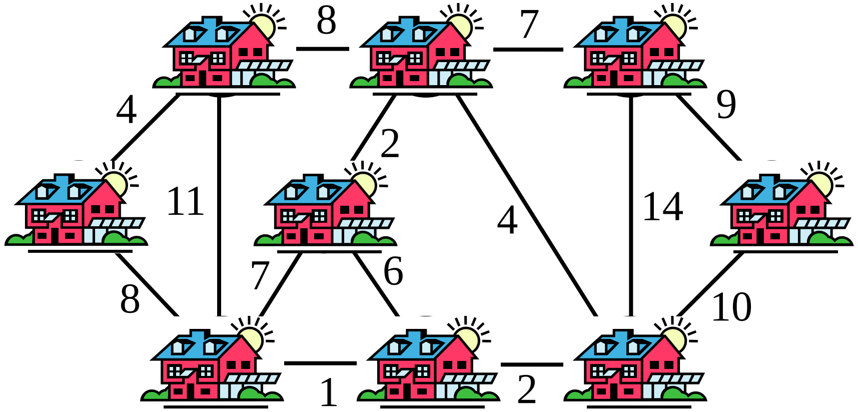


Minimum Spanning Trees

- A connected, undirected graph:
  - Vertices = houses, Edges = roads
- A weight w(u, v) on each edge (u, v) $\in$ E

Find T $\subseteq$ E such that:

1. T connects all vertices
2. $w(T) = \Sigma_{(u,v)\in T}\ w(u, v)$ is minimized



Properties of Minimum Spanning Trees

- Minimum spanning tree is **<u>not</u>** <u>unique</u>

- MST has no cycles
  - We can take out an edge of a cycle, and still have the vertices connected while reducing the cost
- # of edges in a MST:
  - |V| - 1

Examples of MST

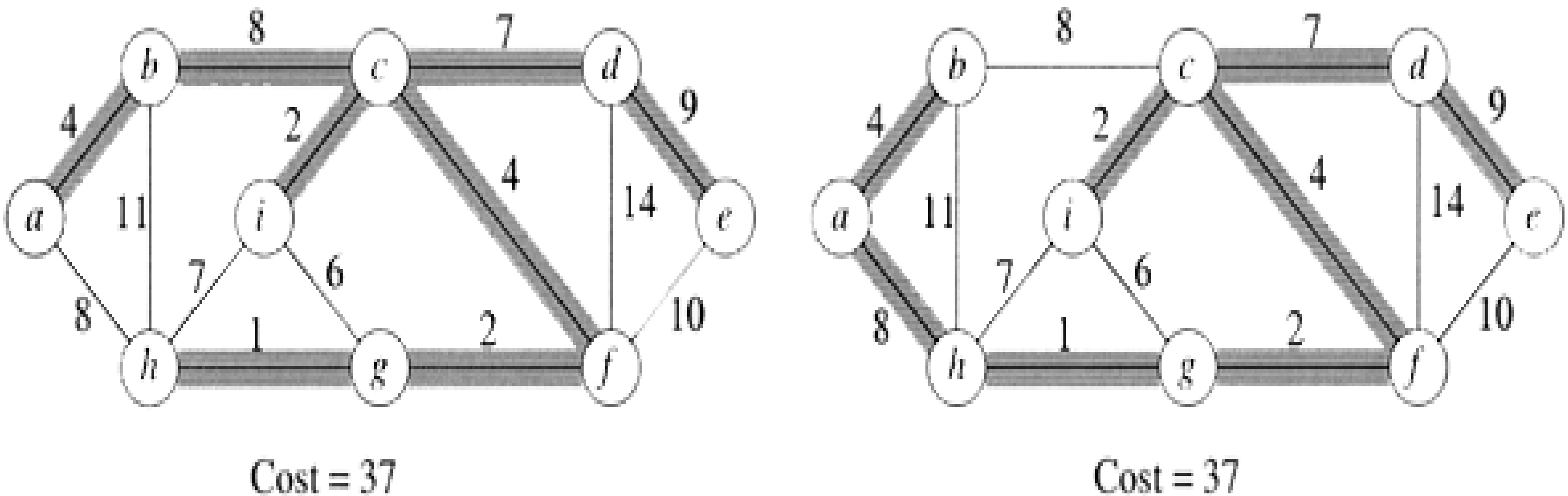


Figure 1: Minimum spanning tree.

- Not only do the edges sum to the same value, but the same set o f edge weights appear in the two MSTs.
  NOTE: An MST may not be unique.

MST

- Basic idea of computing (“growing”) an MST:
  - construct the MST by successively select edges to include in the tree.
  - guarantee that after the inclusion of each new selected edge, it forms a subset of some MST.
- One of the most famous greedy algorithms, along with Huffman coding

MST

- Two basic properties:

  1. Optimal substructure: optimal tree contains optimal subtrees.

  Let T be an MST of G = (V, E). Removing (u, v) of T partitions T into two trees T1 and T2. Then T1 is an MST of G1 = (V1, E1) and T2 is an MST of G2 = (V2, E2).

  Proof. Note that w(T) = w(T1) + w(u, v) + w(T2). There cannot be a better subtree than T1 or T2, otherwise T would be suboptimal.

MST

2. Greedy-choice property:

Let T be an MST of $G = (V, E)$, $A \subseteq T$ be a subtree of T, and (u, v) be min-weight edge in G connecting A and $V - A$. Then $(u, v) \in T$.

Proof. If $(u, v) \notin T$,

- $(u, v) \cup T$ forms a cycle,
- replace one of edges of T by (u, v) form a new tree T (this is contradiction to T is MST)

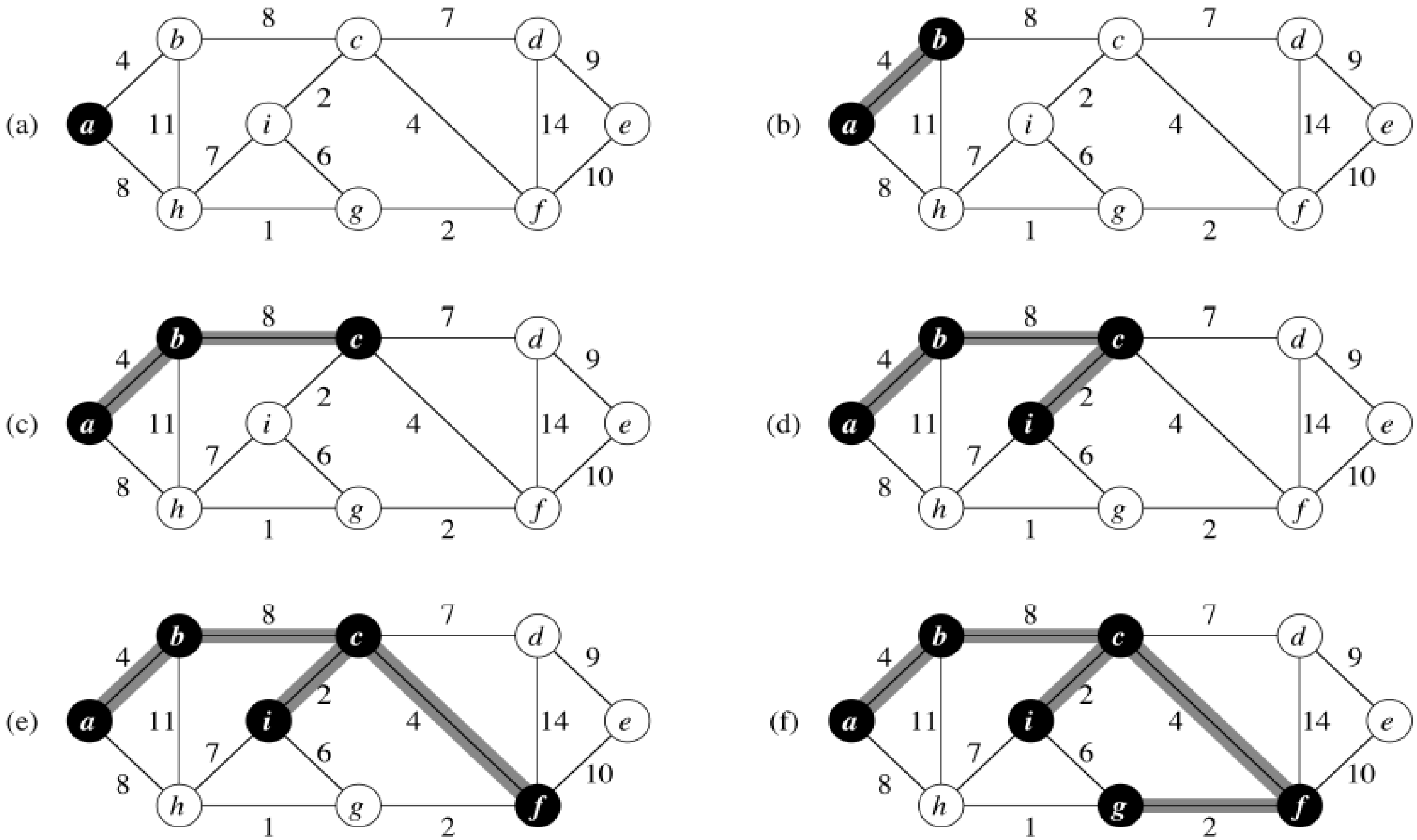
Prim's MST Algorithm

- Prim's algorithm
  - builds one tree, so that A is always a tree
  - starts from a root r
  - at each step, find the next lightest edge crossing cut (A, V − A) and add this edge to A ("greedy choice")
- *How to find the next lightest edge quickly?*
  Answer: use a priority queue

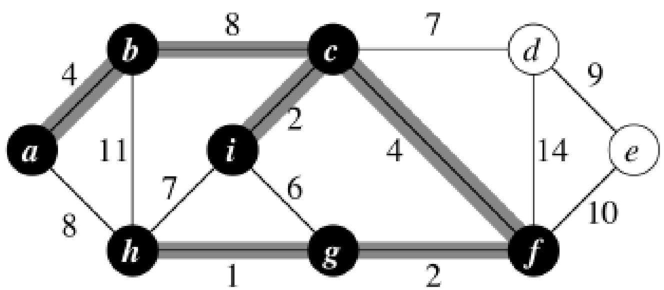
MST_Prim(G,w,r)

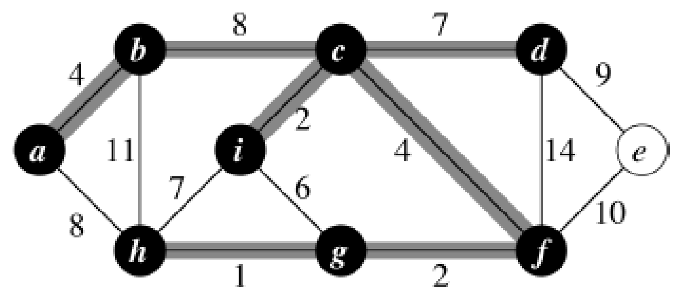
```
A = ∅
for each vertex v ∈ V do
    key[v] = ∞
    π[v] = NIL
end
key[r] = 0
Q = V
while Q ≠ ∅ do
    u = EXTRACT-MIN(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u, v) < key[v] then
            π[v] = u
            key[v] = w(u, v)

    end
end
```

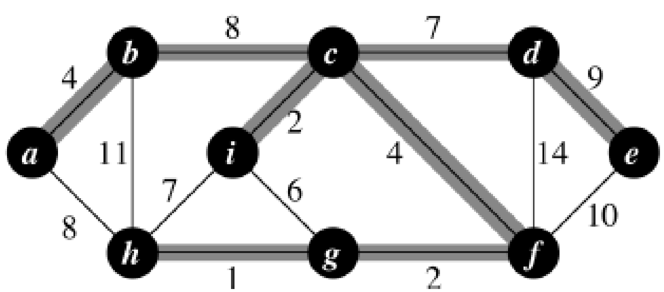
Example



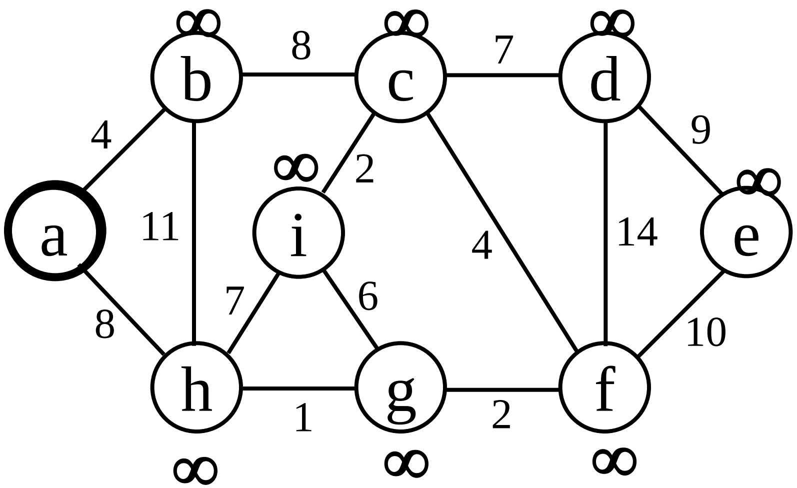
Example

(g)

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

(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)

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

Example

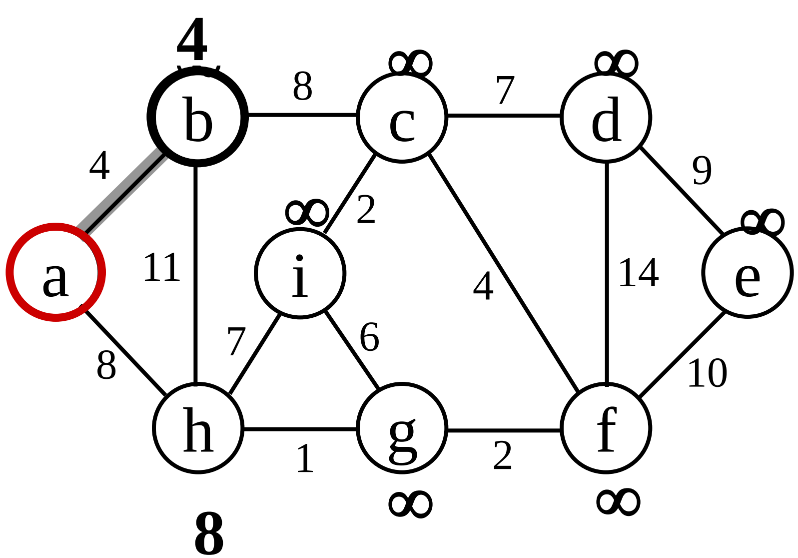


$0 \;\; \infty\ \infty\ \ \infty\ \infty\ \infty\ \infty\ \infty\ \infty$

$Q = \{a, b, c, d, e, f, g, h, i\}$

$V_A = \emptyset$

Extract-MIN(Q) $\Rightarrow$ a



key [b] = 4 $\quad$ π [b] = a

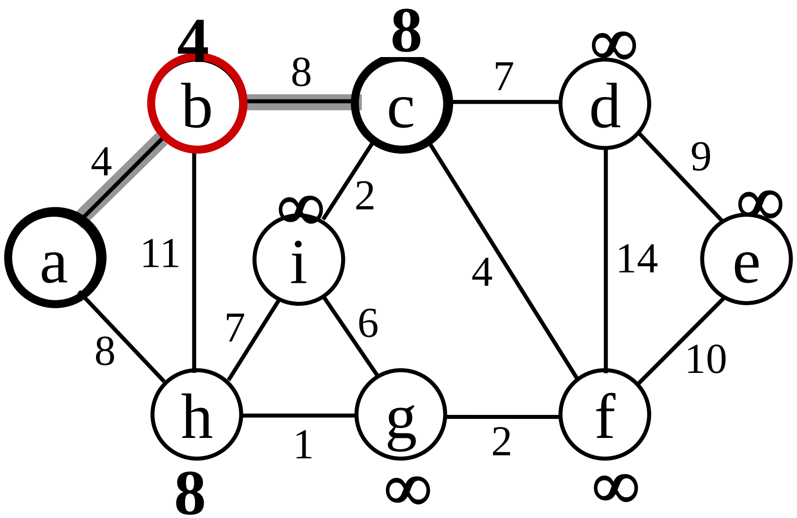
key [h] = 8 $\quad$ π [h] = a

$\mathbf{4}\ \infty\ \infty\ \infty\ \infty\ \infty\ \mathbf{8}\ \infty$

$Q = \{b, c, d, e, f, g, h, i\}$ $\quad V_A = \{a\}$

Extract-MIN(Q) $\Rightarrow$ b

Example



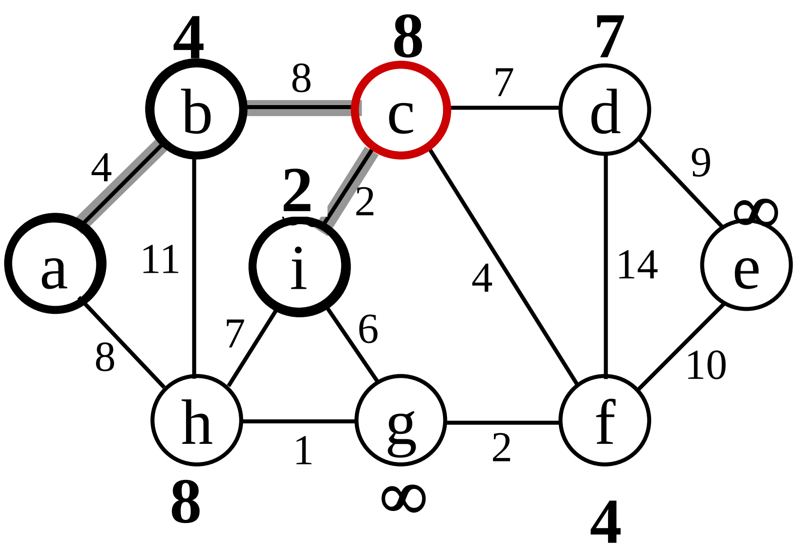
key [c] = 8 π [c] = b

key [h] = 8 π [h] = a (unchanged)

8 ∞ ∞ ∞ ∞ 8 ∞

Q = {c, d, e, f, g, h, i} V_A = {a, b}

Extract-MIN(Q) ⇒ c



key [d] = 7 π [d] = c

key [f] = 4 π [f] = c

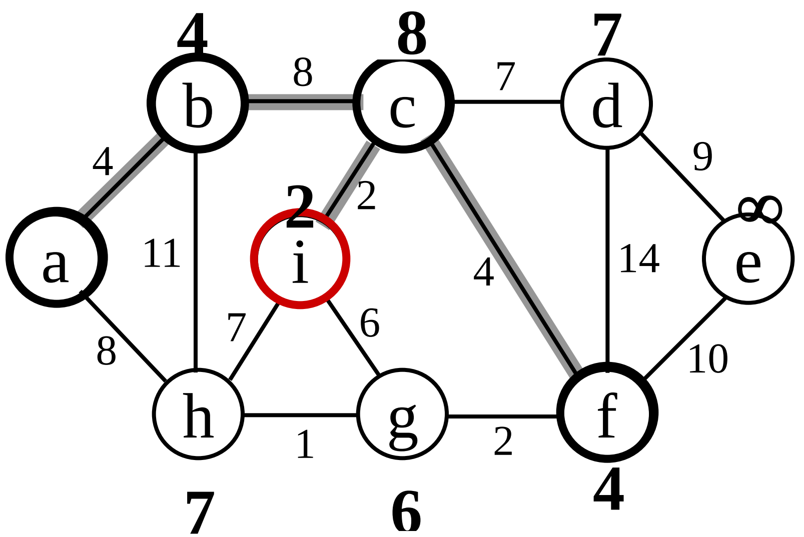
key [i] = 2 π [i] = c

7 ∞ 4 ∞ 8 2

Q = {d, e, f, g, h, i} V_A = {a, b, c}

Extract-MIN(Q) ⇒ i

Example



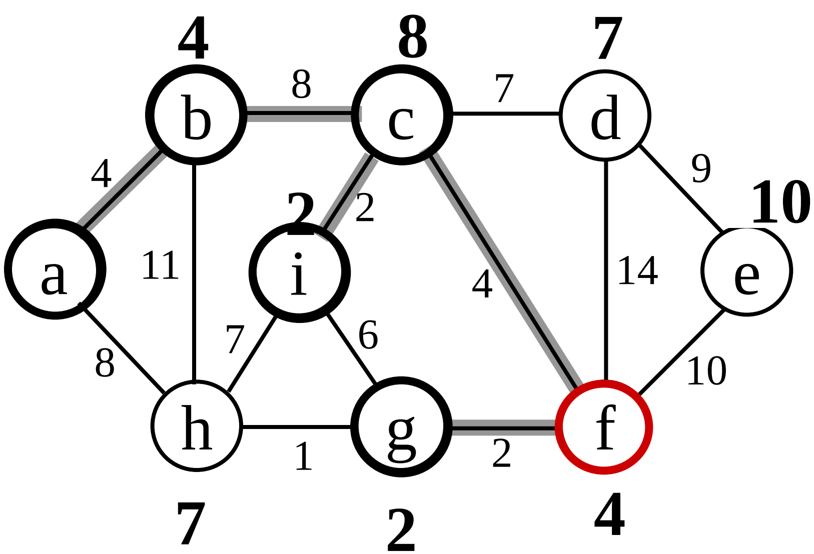
key [h] = 7 π [h] = i

key [g] = 6 π [g] = i

7 ∞ 4 6 8

Q = {d, e, f, g, h} V_A = {a, b, c, i}

Extract-MIN(Q) ⇒ f



key [g] = 2 π [g] = f

key [d] = 7 π [d] = c (unchanged)

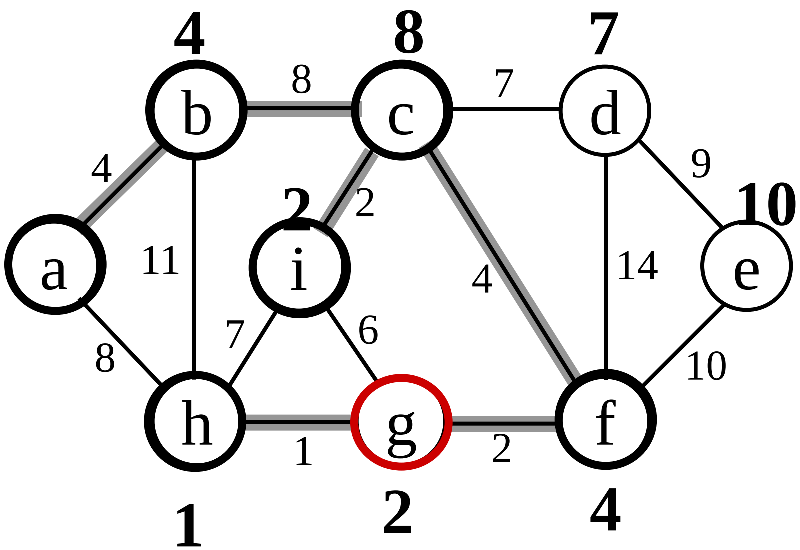
key [e] = 10 π [e] = f

7 10 2 8

Q = {d, e, g, h} V_A = {a, b, c, i, f}

Extract-MIN(Q) ⇒ g

Example

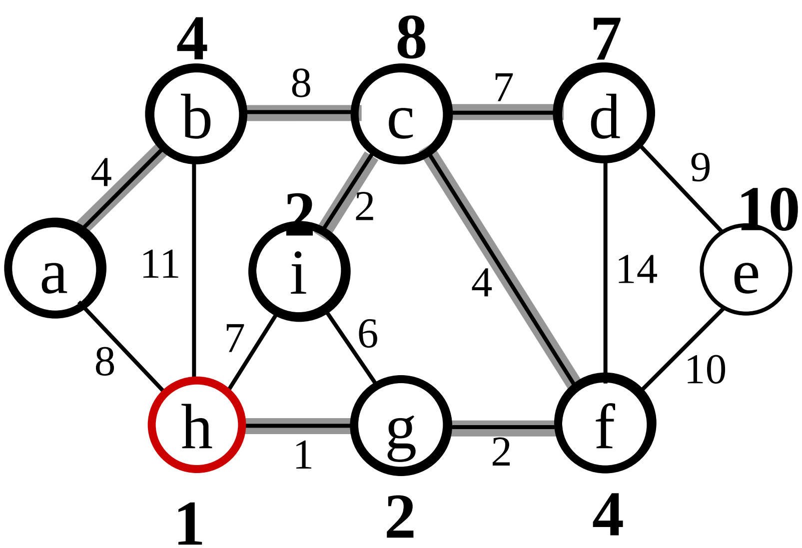


key [h] = 1 π [h] = g

7 10 1

Q = {d, e, h} V_A = {a, b, c, i, f, g}

Extract-MIN(Q) $\Rightarrow$ h

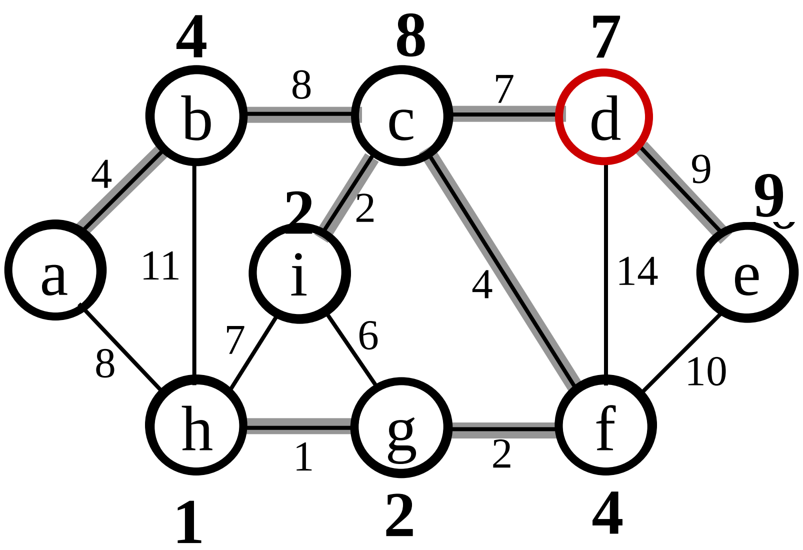


7 10

Q = {d, e} V_A = {a, b, c, i, f, g, h}

Extract-MIN(Q) $\Rightarrow$ d

Example



key [e] = 9   π [e] = f

9

Q = {e} V_A = {a, b, c, i, f, g, h, d}

Extract-MIN(Q) ⇒ e

Q = ∅ V_A = {a, b, c, i, f, g, h, d, e}

Analysis of Prim's Algorithm

- Building heap takes time O(|V|)
- Make |V| calls to Extract-Min, each taking time O(log|V|)
- For loop iterates O(|E|) times
  - In for loop, need constant time to check for queue membership and O(log|V|) time for decreasing v's key and updating heap
- yields total time of O(|V|log|V| + |E|log|V|) = O(|E|log|V|)

Kruskal’s MST Algorithm

- Kruskal Algorithm
  - scan edges in increasing of weight
  - put edge in if no loop created
- *Why does this result in MST?*
  Answer: min-weight edge is always in MST (the greedy-choice property).
- Implementation data structure: disjoint-set

Disjoint-Set

- Disjoint-Set maintains a collection of S = {S1, S2, ...Sk} of disjoint dynamic sets. Each set is identified by a representative, which is some member of the set.
- A disjoint-set data structure supports the following operations:
  - Make-set(x): creates a new set whose only member (and thus representative) is x.
  - Union(x, y): unites the sets that contain x and y, say Sx and Sy, into a new set that is the union of these two sets: Sx ∪ Sy. The representative is any member of Sx ∪ Sy.
  - Find-set(x): returns (a pointer to) the representative of the (unique) set containing x.

Operations on Disjoint Data Sets

- MAKE-SET(u) – creates a new set whose only member is u
- FIND-SET(u) – returns a representative element from the set that contains u
  - Any of the elements of the set that has a particular property
  - E.g.: S_u = {r, s, t, u}, the property is that the element be the first one alphabetically

    FIND-SET(u) = r   FIND-SET(s) = r
  - FIND-SET has to return the same value for a given set

Operations on Disjoint Data Sets

- UNION(u, v) – unites the dynamic sets that contain u and v, say S_u and S_v
  - E.g.: $S_u = \{r, s, t, u\}$, $S_v = \{v, x, y\}$

    UNION (u, v) = {r, s, t, u, v, x, y}
- Running time for FIND-SET and UNION depends on implementation.
- Can be shown to be **$\alpha(n)=O(\log n)$** where **$\alpha(\ )$** is a very slowly growing function

MST-Kruskal(G, w)

$A = \emptyset$

for *each vertex* $v \in V$ **do**

  MAKE-SET(v)

end

sort edges in E into nondecreasing order by weight w

for *each edge* $(u, v) \in E$, *taken in nondecreasing order* **do**

  if FIND-SET(u) $\neq$ FIND-SET(v) **then**

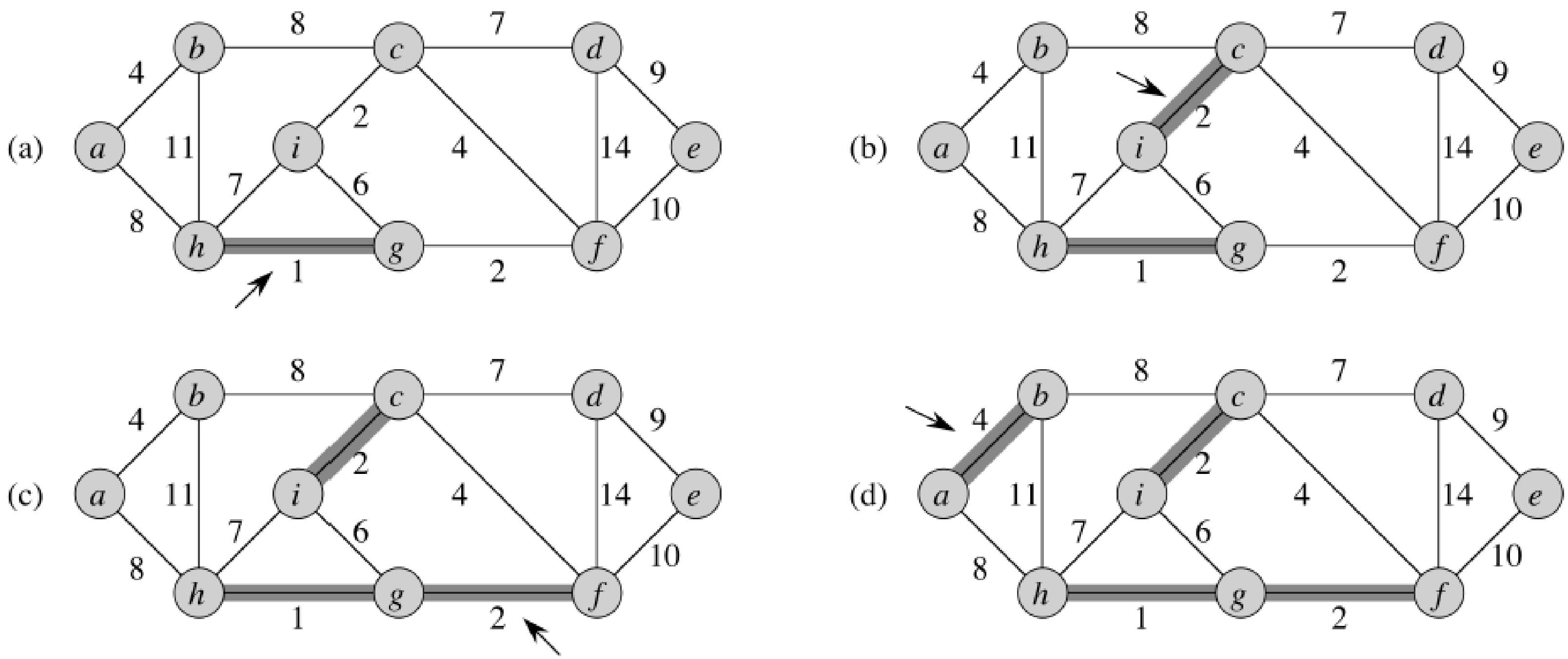
    $A = A \cup \{(u, v)\}$

    UNION(u, v)

end

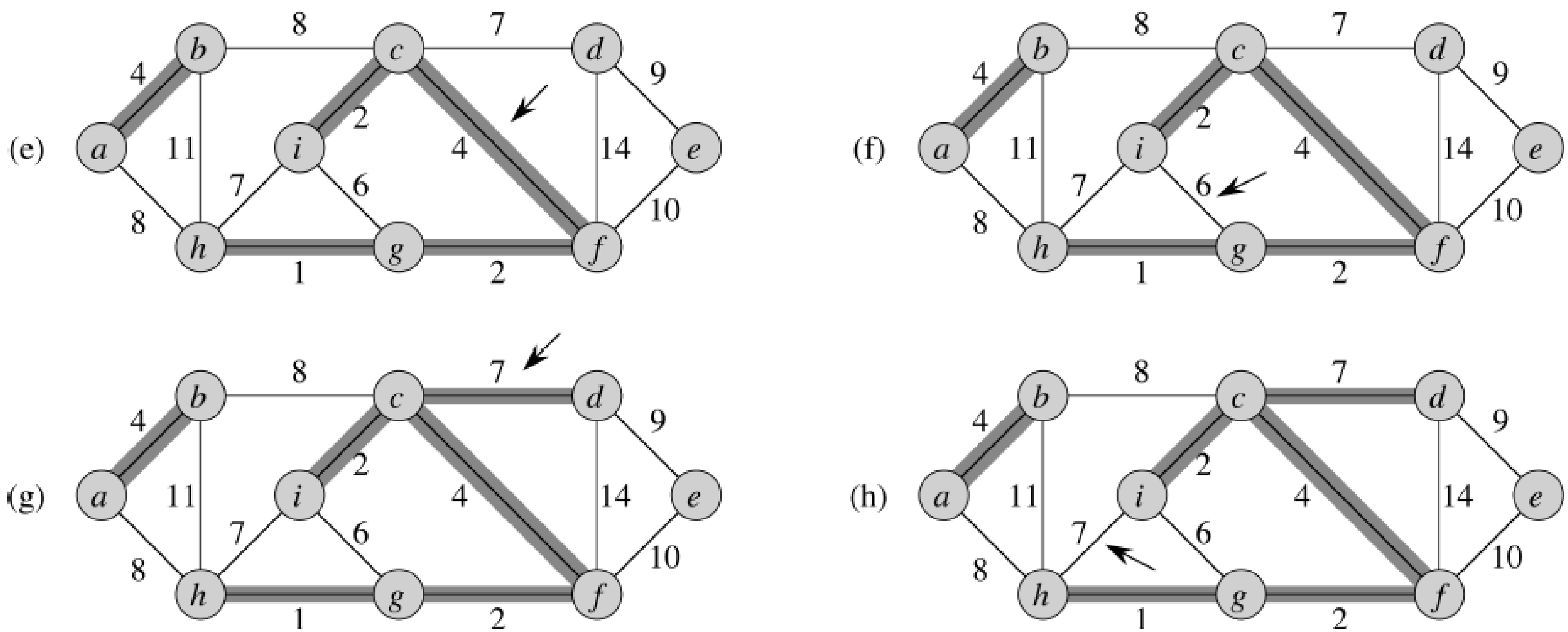
return A

Example

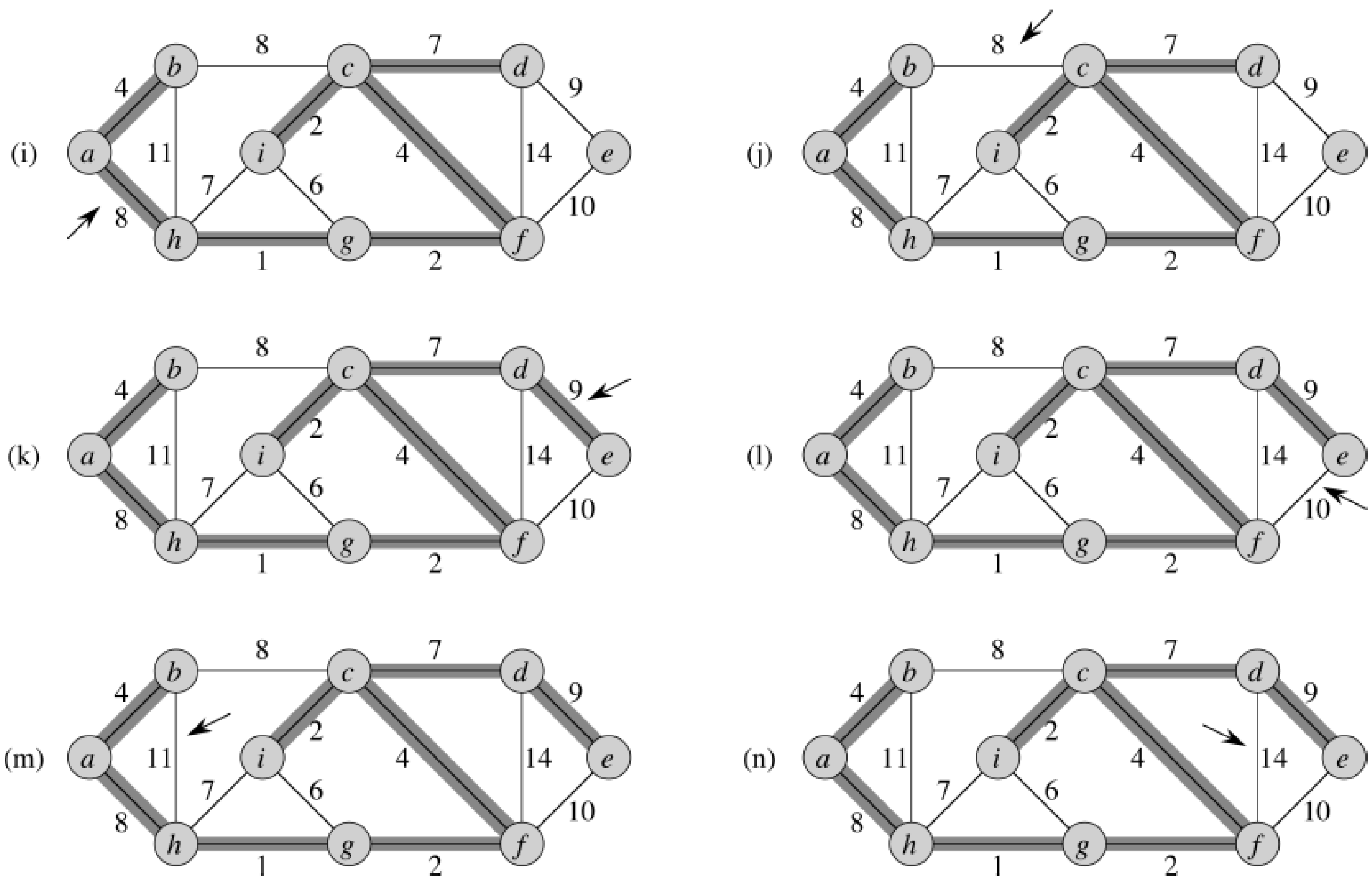


(a)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2
(b)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2
(c)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2
(d)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2

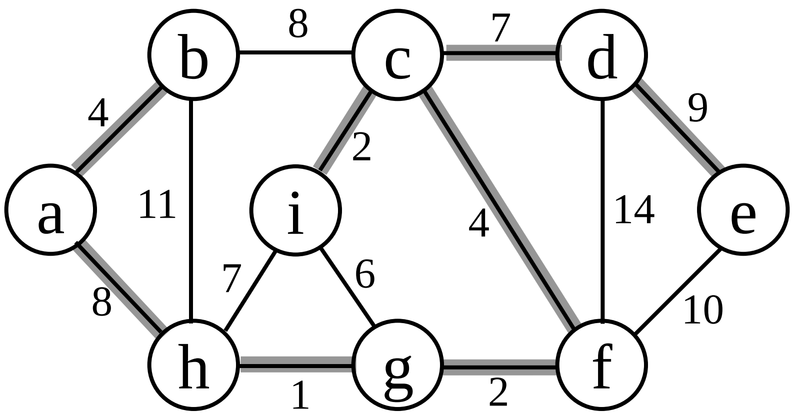
Example



Example



Example



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

{a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}

1. Add (h, g) {g, h}, {a}, {b}, {c}, {d}, {e}, {f}, {i}
2. Add (c, i) {g, h}, {c, i}, {a}, {b}, {d}, {e}, {f}
3. Add (g, f) {g, h, f}, {c, i}, {a}, {b}, {d}, {e}
4. Add (a, b) {g, h, f}, {c, i}, {a, b}, {d}, {e}
5. Add (c, f) {g, h, f, c, i}, {a, b}, {d}, {e}
6. Ignore (i, g) {g, h, f, c, i}, {a, b}, {d}, {e}
7. Add (c, d) {g, h, f, c, i, d}, {a, b}, {e}
8. Ignore (i, h) {g, h, f, c, i, d}, {a, b}, {e}
9. Add (a, h) {g, h, f, c, i, d, a, b}, {e}
10. Ignore (b, c) {g, h, f, c, i, d, a, b}, {e}
11. Add (d, e) {g, h, f, c, i, d, a, b, e}
12. Ignore (e, f) {g, h, f, c, i, d, a, b, e}
13. Ignore (b, h) {g, h, f, c, i, d, a, b, e}
14. Ignore (d, f) {g, h, f, c, i, d, a, b, e}

Analysis of Kruskal's Algorithm

```
A = ∅
for each vertex v ∈ V do                                          } O(V)
    MAKE-SET(v)
end
sort edges in E into nondecreasing order by weight w   ←——  O(ElogE)
for each edge (u, v) ∈ E, taken in nondecreasing order
do
    if FIND-SET(u) ≠ FIND-SET(v) then                             } O(ElogV)
        A = A ∪ {(u, v)}
        UNION(u, v)

end
return A
```

Running time: O(V+ElogE+ElogV)=O(ElogE)

Since E=O(V^2), we have logE=O(2logV)=O(logV)

<u>O(ElogV)</u>