UNIVERSIDAD NACIONAL DE COLOMBIA

Algoritmia Avanzada

http://dis.unal.edu.co/profesores/ypinzon/2013308

Sesión 5

Minimum Spanning Tree Graph Algorithms

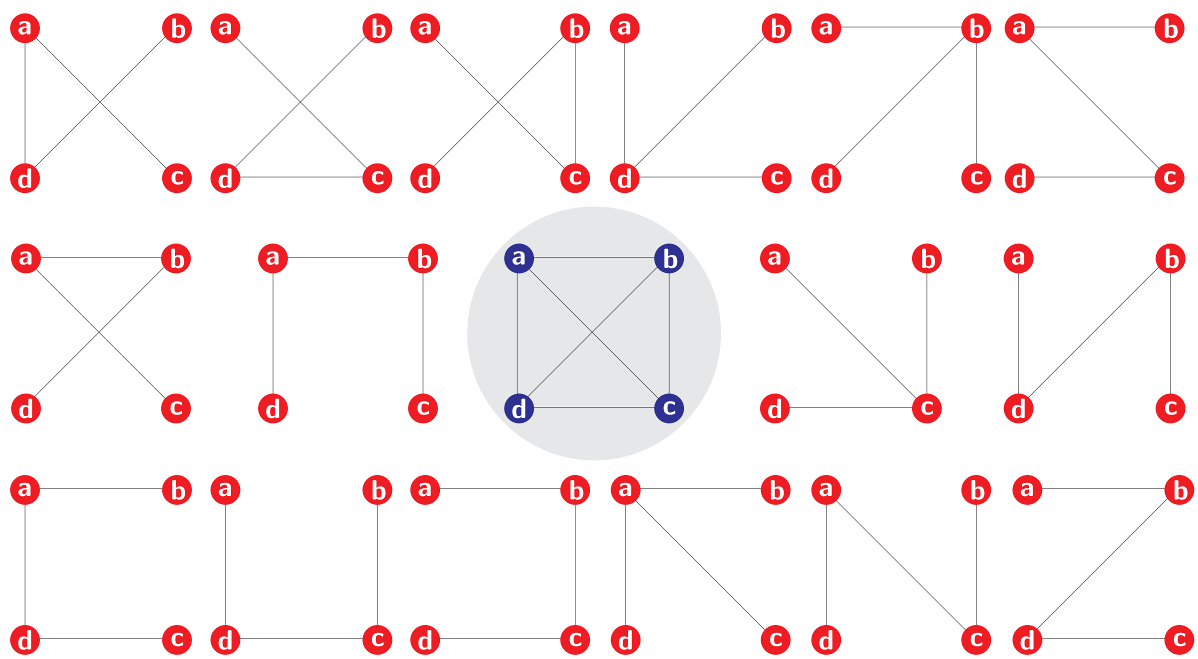
Yoan Pinzón, PhD
Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
`ypinzon@unal.edu.co`
`http://dis.unal.edu.co/profesores/ypinzon`

Session 5

- **MST (Minimum Spanning Trees)**
  - ▷ The Kruskal's Algorithm
  - ▷ The Prim's Algorithm

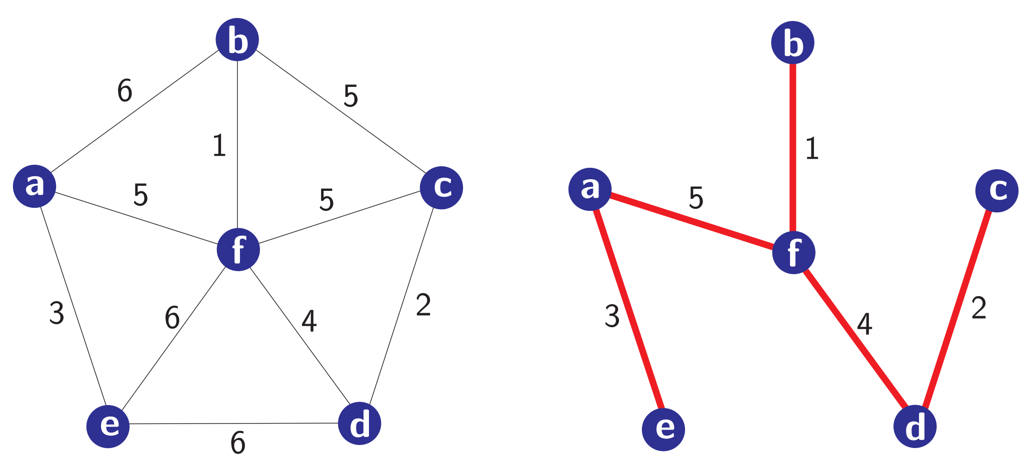
MST: Minimum Spanning Trees

A *spanning tree* of a graph is just a subgraph that contains all the vertices and is a tree. A graph may have many spanning trees; for instance the following complete graph has sixteen spanning trees:



Now suppose the edges of the graph have weights or lengths. The weight of a tree is just the sum of weights of its edges. Obviously, different trees have different lengths.

▶ **Example:**



$w = 43$ $w = 15$

The problem: how to find the minimum length/weight spanning tree?

The Kruskal's Algorithm
by Joseph Bernard Kruskal, 1956

Kruskal's algorithm is conceptually quite simple. The edges are selected and added to the spanning tree in increasing order of their weights. An edge is added to the tree only if it does not create a cycle.

▶ **Pseudo-code:**

```
MST-KRUSKAL(G, w)
  A ← ∅
  for each vertex v ∈ V[G]
      do MAKE-SET(v)
  sort the edges of E into nondecreasing order by weight w
  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
      do if FIND-SET(u) ≠ FIND-SET(v)
            then A ← A ∪ {(u, v)}
                 UNION(u, v)
  return A
```


The Kruskal's Algorithm
Time Complexity Analysis

▶ **Time Complexity:** $O(m \log n)$

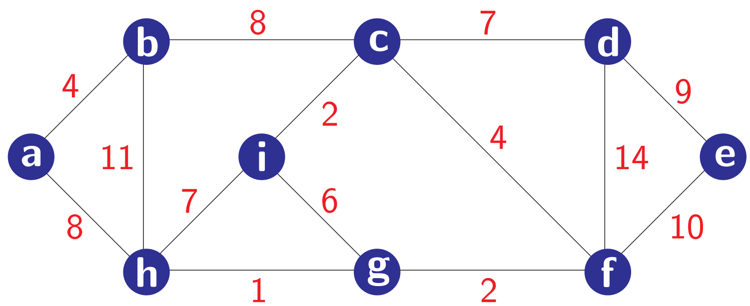
The line testing whether two endpoints are disconnected looks like it should be slow (linear time per iteration, or $O(mn)$ total). But there are some data structures that perform each test in close to constant time; this is known as the union-find problem. The slowest part turns out to be the sorting step, which takes $O(m \log n)$ time.

Kruskal's Algorithm is a standard example of a *Greedy Algorithm*.

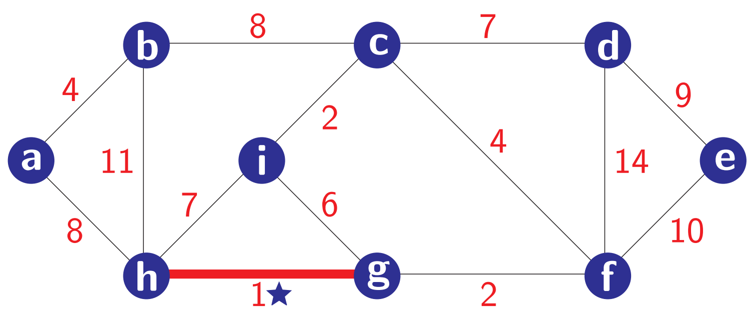
Recall: A greedy algorithm is considered greedy because it selects the best choice immediately available at each step.

The Kruskal's Algorithm
Example

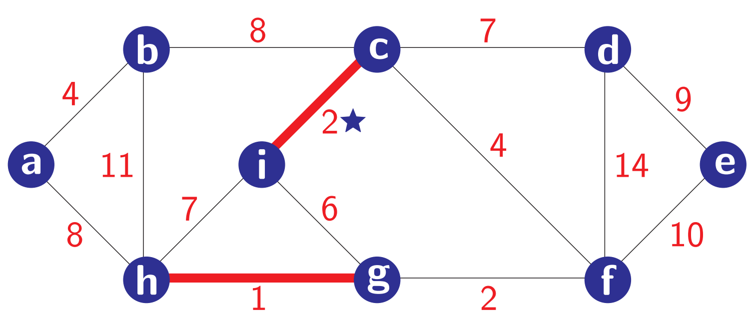
Find the MST for the following graph.



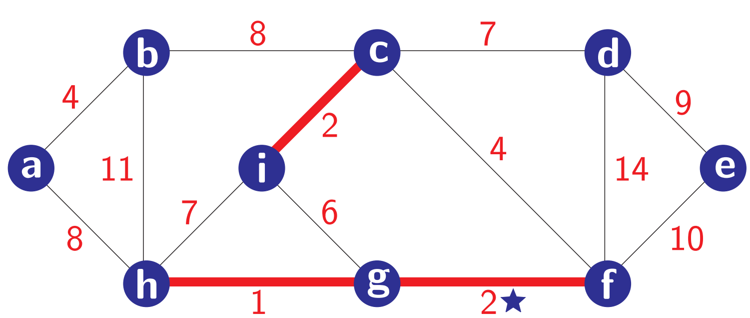
▶ **Solution:**



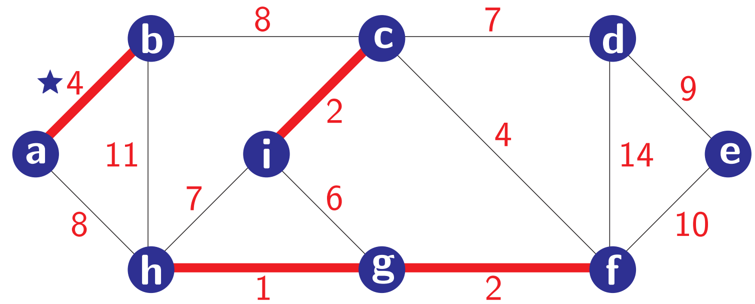
(a)



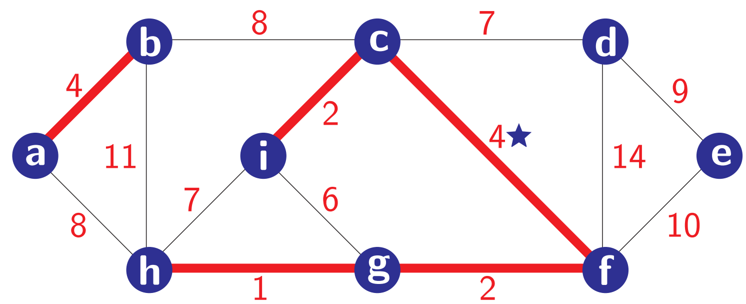
(b)



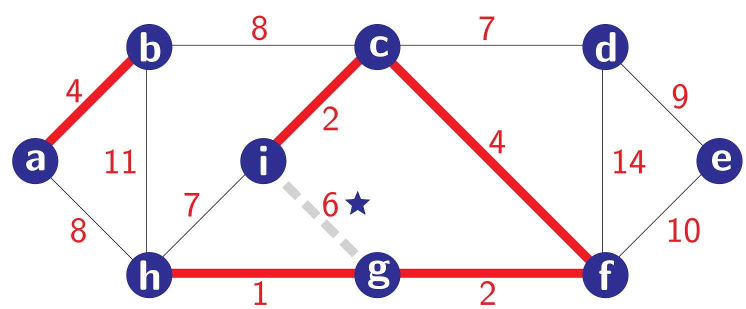
(c)


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

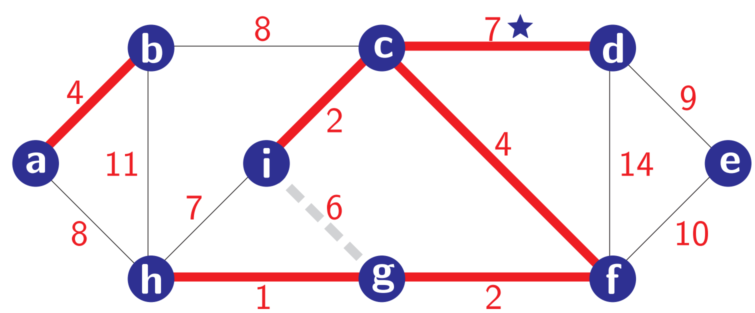
(d)


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

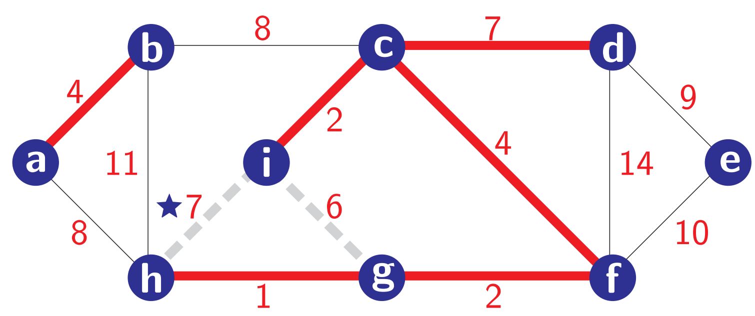
(e)


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

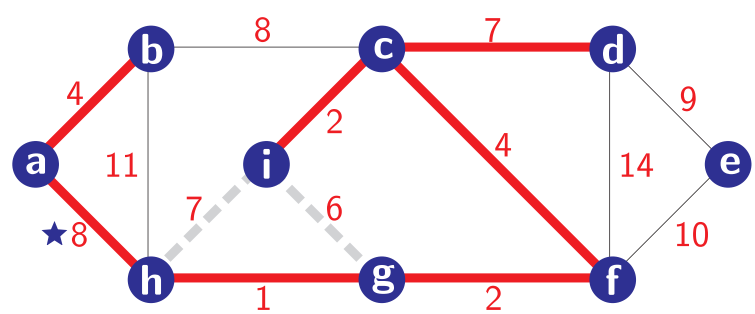
(f)


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

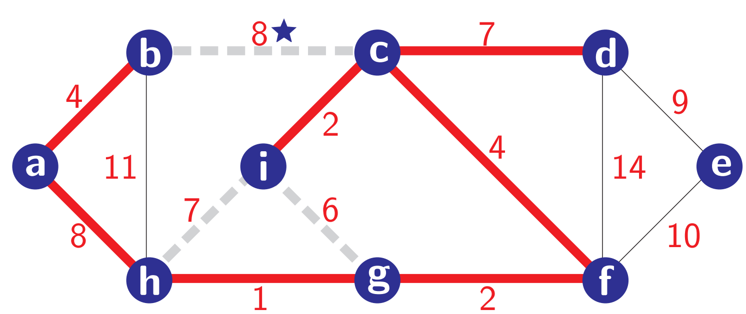
(g)


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

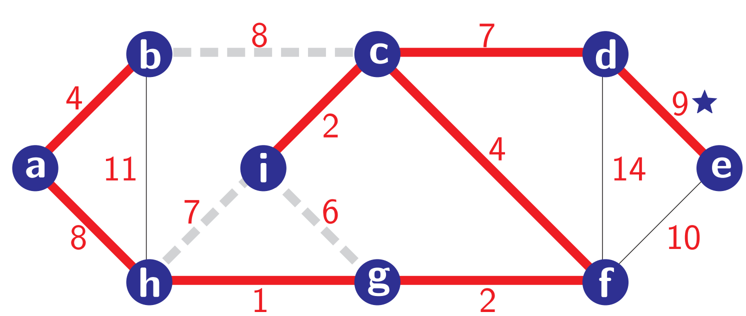
(h)


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

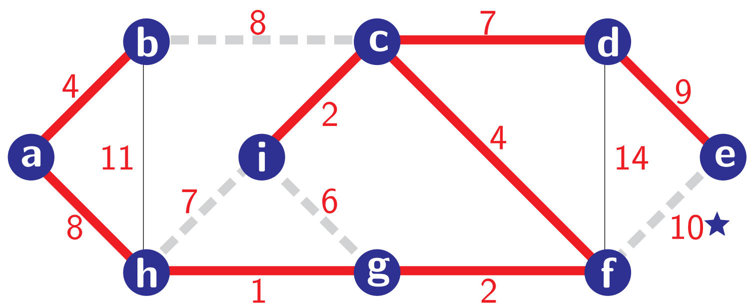
(i)


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

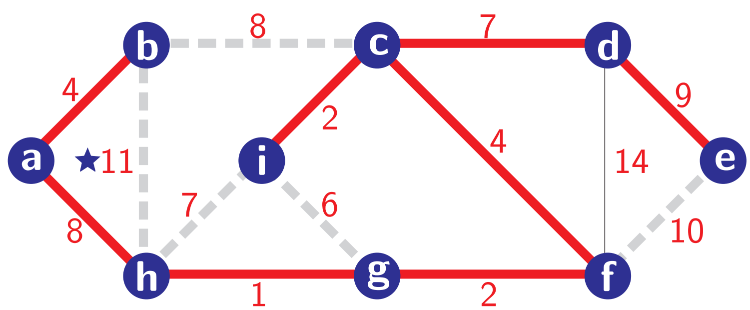
(j)


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

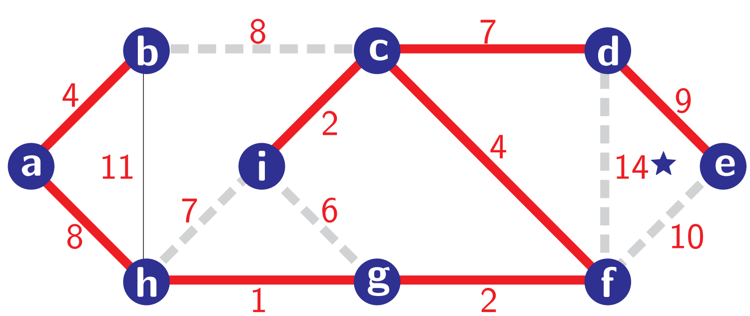
(k)


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

(l)


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

(m)


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

(n)

Edge Processed	Collection of disjoint sets								
initial state	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}
1 (h,g)✓	{a}	{b}	{c}	{d}	{e}	{f}	{g,h}		{i}
2 (i,c)✓	{a}	{b}	{c,i}	{d}	{e}	{f}	{g,h}		
2 (g,f)✓	{a}	{b}	{c,i}	{d}	{e}		{g,h,f}		
4 (a,b)✓	{a,b}		{c,i}	{d}	{e}		{g,h,f}		
4 (c,f)✓	{a,b}			{d}	{e}		{g,h,f,c,i}		
6 (i,g)	{a,b}			{d}	{e}		{g,h,f,c,i}		
7 (c,d)✓	{a,b}				{e}		{g,h,f,c,i,d}		
7 (h,i)	{a,b}				{e}		{g,h,f,c,i,d}		
8 (a,h)✓					{e}		{g,h,f,c,i,d,a,b}		
8 (b,c)					{e}		{g,h,f,c,i,d,a,b}		
9 (d,e)✓							{g,h,f,c,i,d,a,b,e}		
10 (f,e)							{g,h,f,c,i,d,a,b,e}		
11 (b,h)							{g,h,f,c,i,d,a,b,e}		
14 (d,f)							{g,h,f,c,i,d,a,b,e}		

$$A = \{(h,g),(i,c),(g,f),(a,b),(c,f),(c,d),(a,h),(d,e)\}, w = 37$$

The Prim's Algorithm

by Robert Clay Prim, 1957

Rather than build a subgraph one edge at a time, Prim's algorithm builds a tree one vertex at a time.

- **Pseudo-code:**

```
MST-PRIM(G, w, r)
    for each u ∈ V[G]
        do key[u] ← ∞
           π[u] ← NIL
    key[r] ← 0
    Q ← V[G]
    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)
```

The algorithm was first discovered in 1930 by Vojtěch Jarník and later independently by Prim in 1957 and Dijkstra in 1959.

The Prim's Algorithm
Time Complexity Analysis

▶ **Time Complexity:** $O(m \log n)$

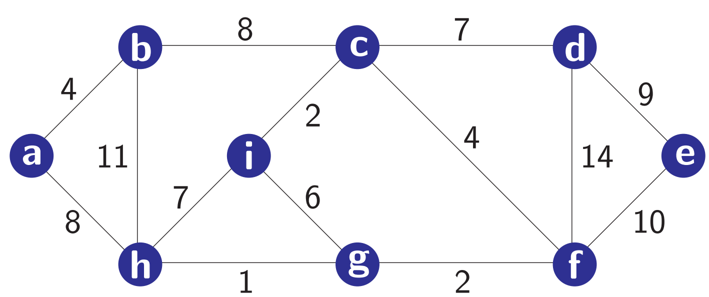
The time required by Prim's algorithm is $O(n^2)$. It will be reduced to $O(m \log n)$ if a heap is used.

Analysis: We perform n steps in which we remove the smallest element in the heap, and at most $2m$ steps in which we examine an edge $f = (u, v)$. For each of those steps, we might replace a value on the heap, reducing it's weight. (You also have to find the right value on the heap, but that can be done easily enough by keeping a pointer from the vertices to the corresponding values.) To reduce the weight of an element of a binary heap can be done in $O(\log n)$ time. Alternately by using a more complicated data structure known as a Fibonacci heap, you can reduce the weight of an element in constant time. The result is a total time bound of $O(m + n \log n)$.

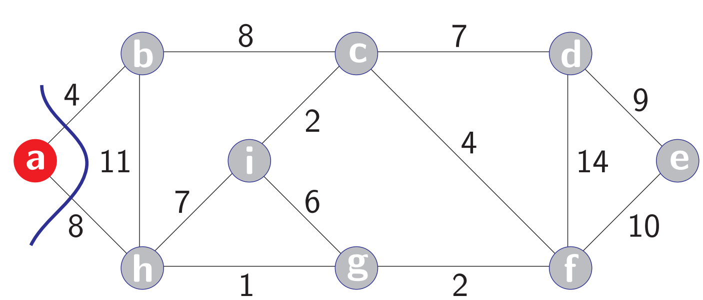
Prim's algorithm is also a greedy algorithm, in the sense that it repeatedly makes a best choice in a sequence of stages.

The Prim's Algorithm
Example

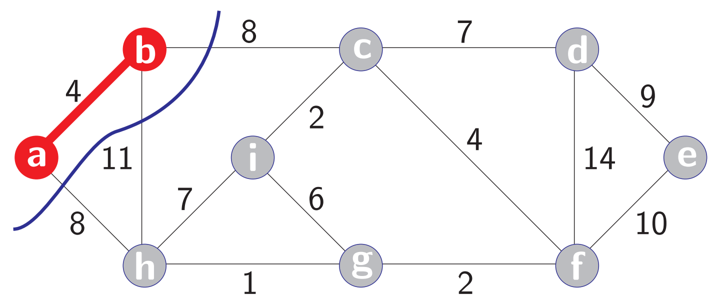
Find the MST for the following graph.



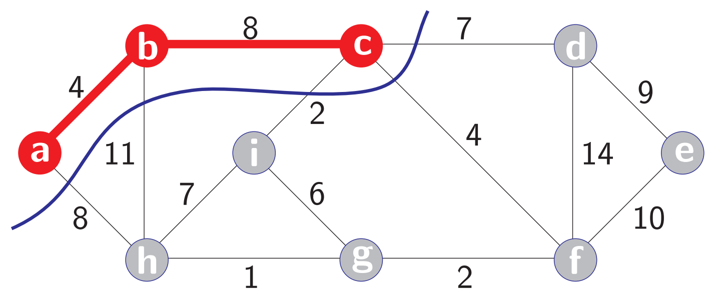
▶ **Solution:**



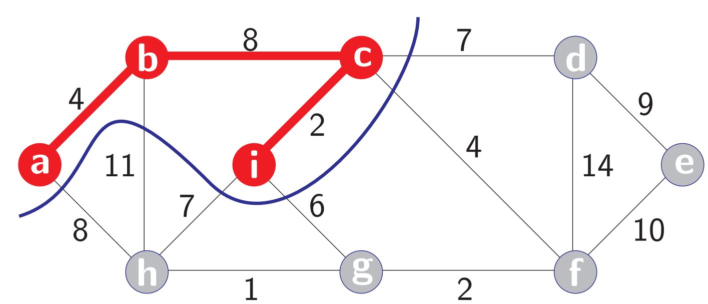
(a)


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

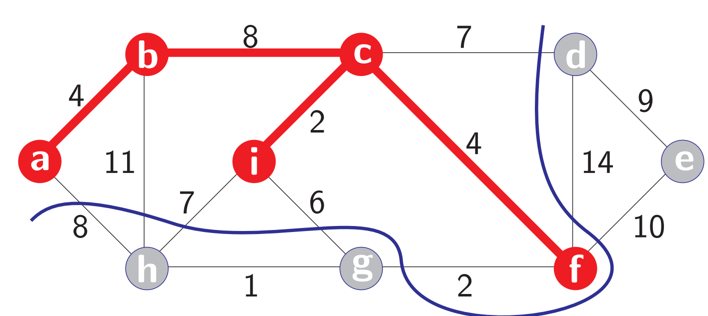
(b)


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

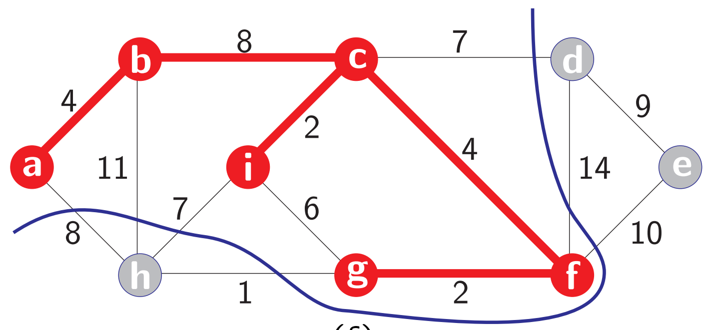
(c)

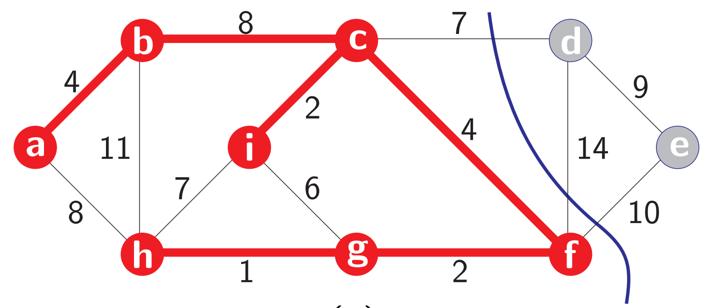

a
b
c
d
e
f
g
h
i
4
8
11
7
2
6
1
2
4
7
14
9
10
8

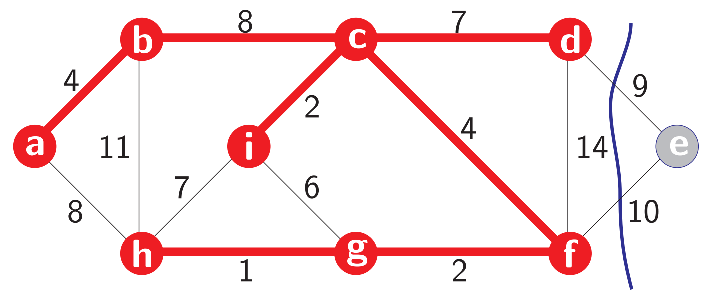
(d)

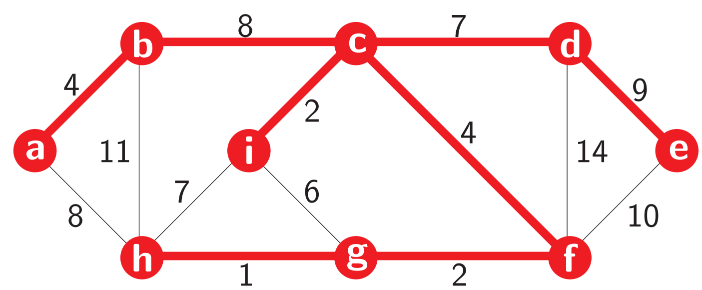

a
b
c
d
e
f
g
h
i
4
8
11
7
2
6
1
2
4
7
14
9
10
8

(e)


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


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


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


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

Prim/Kruskal – Exercise