Data Structures & Algorithms

14: MST using Kruskal's and Prim's Algorithms

Part - II

Dr Ram Prasad Krishnamoorthy

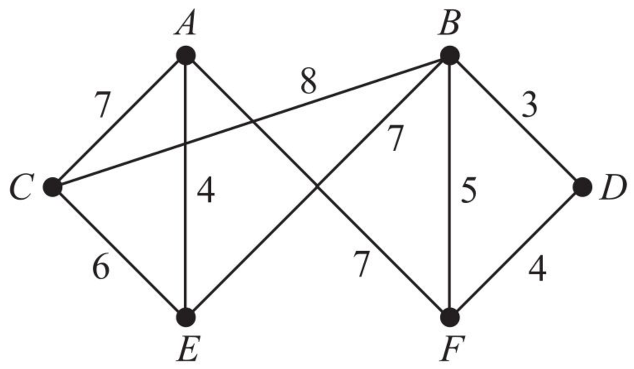
Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

SAI UNIVERSITY

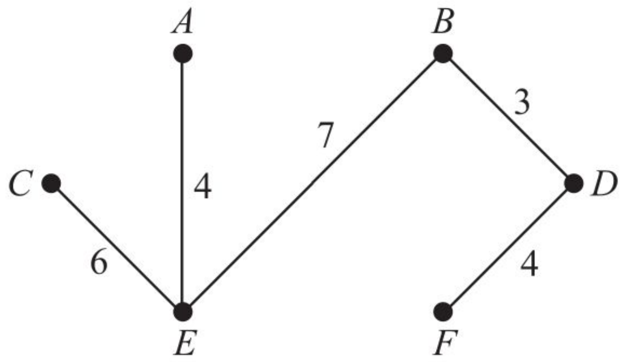
Minimum Spanning Tree (MST)

Minimum Spanning Tree



Thus the minimal spanning tree of Q which is obtained contains the edges

$$BE, \quad CE, \quad AE, \quad DF, \quad BD$$



Minimum Spanning Tree

Generic MST algorithm

GENERIC-MST(G, w)

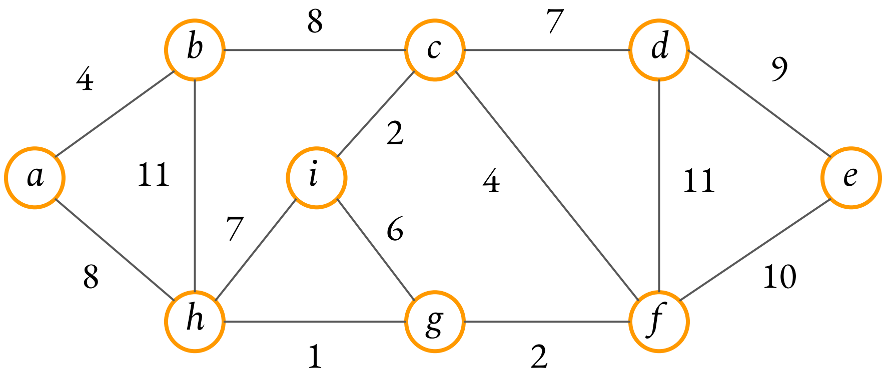
$A = \emptyset$

while A does not form a spanning tree

find an edge (u, v) that is safe for A

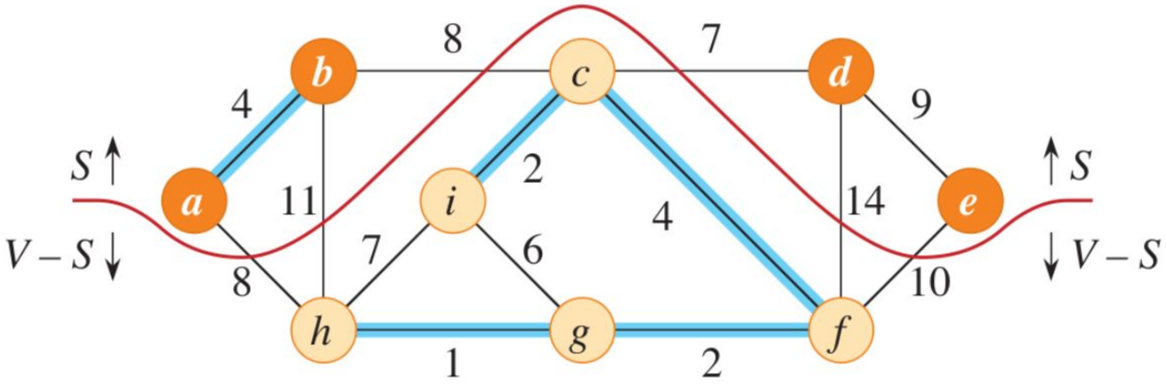
$A = A \cup \{(u, v)\}$

return A



Minimum Spanning Tree

- A ***cut*** $(S, V - S)$ is a partition of vertices into disjoint sets S and $S - V$.
- Edge $(u, v) \in E$ ***crosses*** cut $(S, V - S)$ if one endpoint is in S and the other is in $V - S$.
- A cut ***respects*** A if and only if no edge in A crosses the cut.
- An edge is a ***light edge*** crossing a cut if and only if its weight is minimum over all edges crossing the cut. For a given cut, there can be > 1 light edge crossing it.



Kruskal's Algorithm

Minimum Spanning Tree

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
create a single list of the edges in G.E
sort the list of edges into nondecreasing order by weight w
for each edge (u, v) taken from the sorted list in order
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

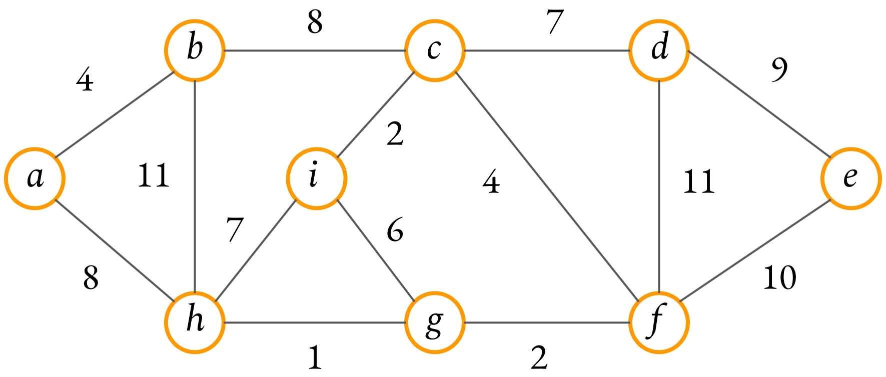
Minimum Spanning Tree

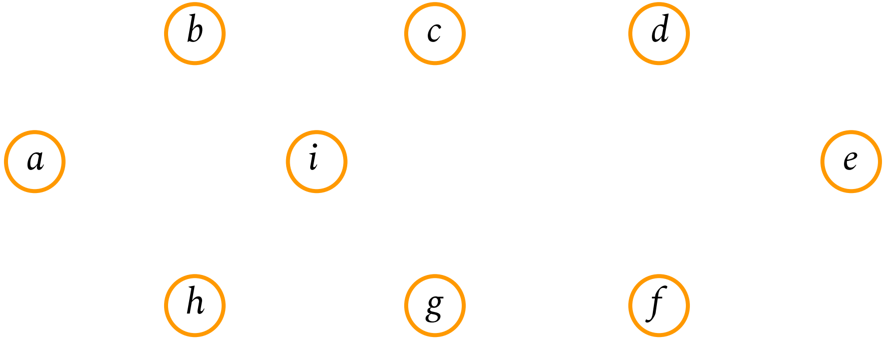
Kruskal's Algorithm

Step 1: Arrange the edges of G in order of increasing weights.

Step 2: Starting only with the vertices of G and proceeding sequentially, add each edge which *does not result in a cycle* until **n-1** edges are added.

Step 3: Exit

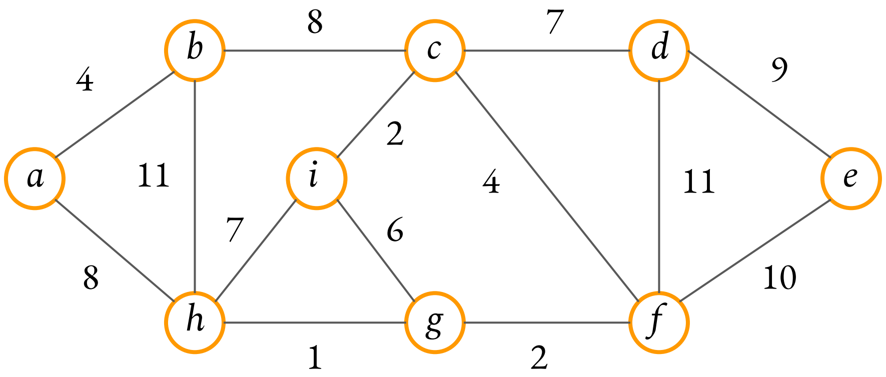


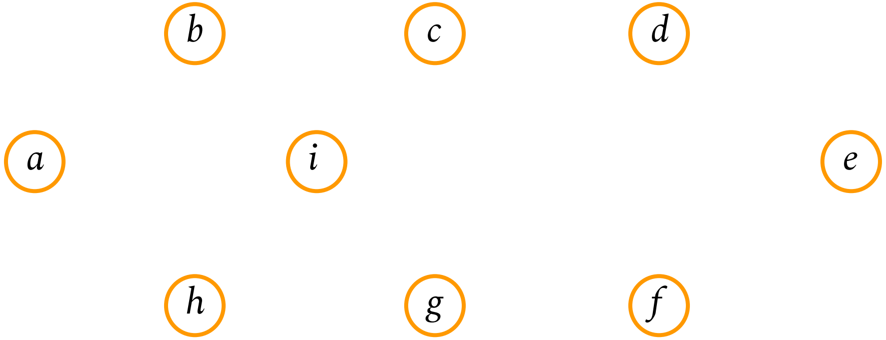


Minimum Spanning Tree

Kruskal's Algorithm

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

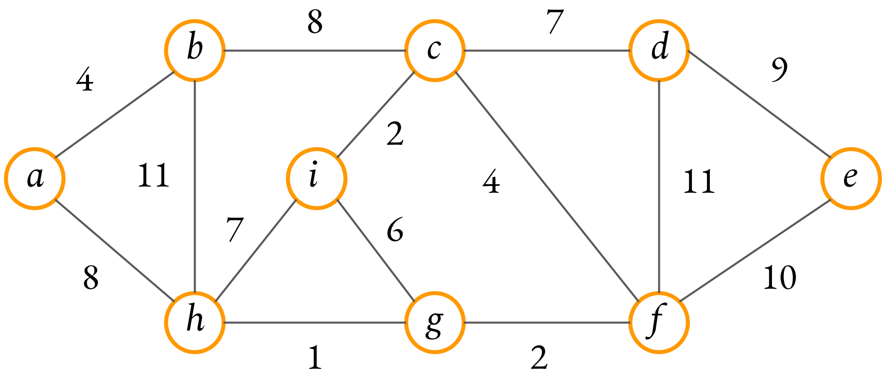


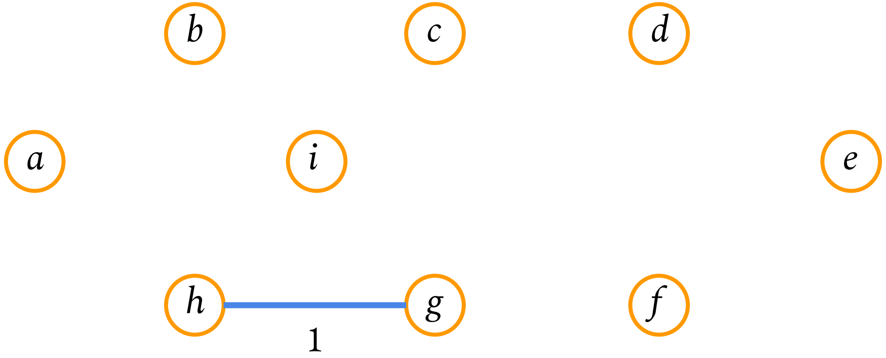


Minimum Spanning Tree

Kruskal's Algorithm

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

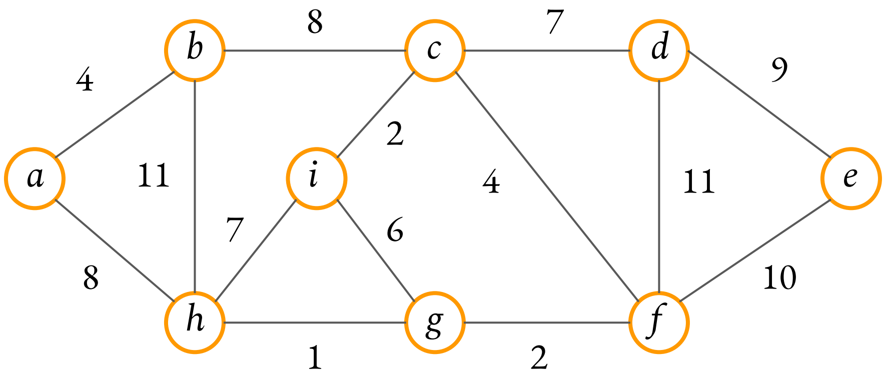


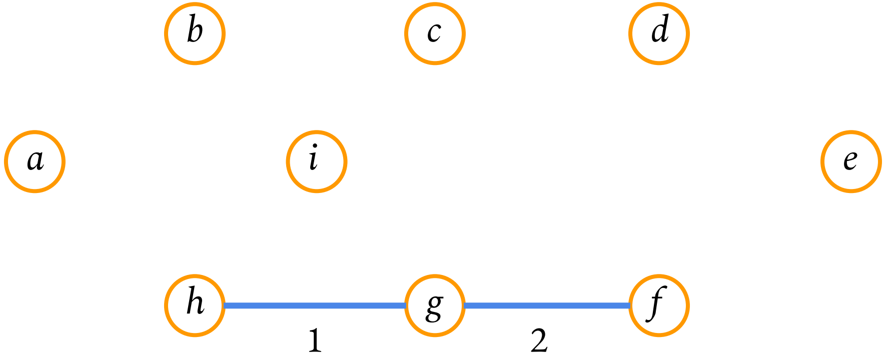


Minimum Spanning Tree

Kruskal's Algorithm

h*–*g → **1** → **safe**
g*–*f → **2** → **safe**
c*–*i → **2**
a*–*b → **4**
c*–*f → **4**
g*–*i → **6**
c*–*d → **7**
h*–*i → **7**
a*–*h → **8**
b*–*c → **8**
d*–*e → **9**
e*–*f → **10**
b*–*h → **11**
d*–*f → **11**

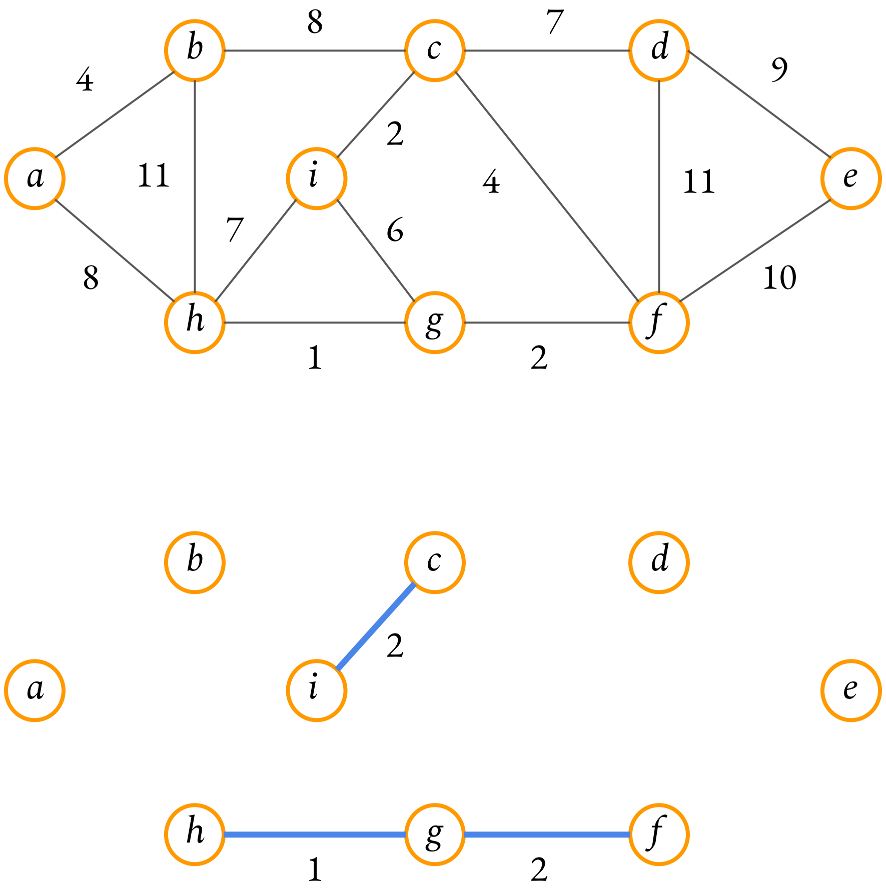




MINIMUM SPANNING TREE

Kruskal's Algorithm

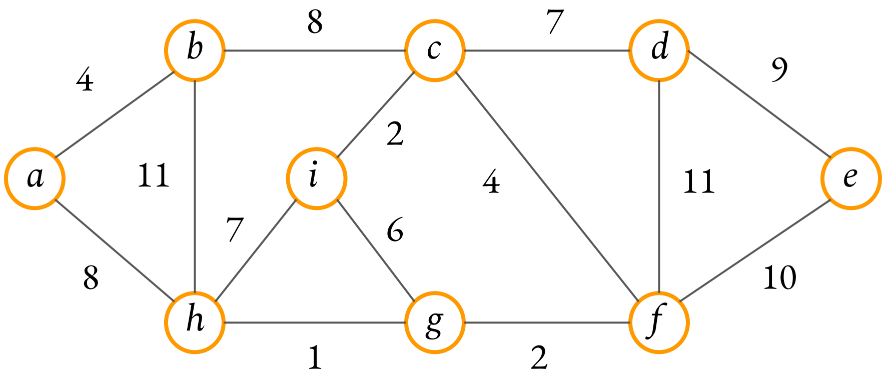
h–g → 1 → safe
g–f → 2 → safe
c–i → 2 → safe
a–b → 4
c–f → 4
g–i → 6
c–d → 7
h–i → 7
a–h → 8
b–c → 8
d–e → 9
e–f → 10
b–h → 11
d–f → 11

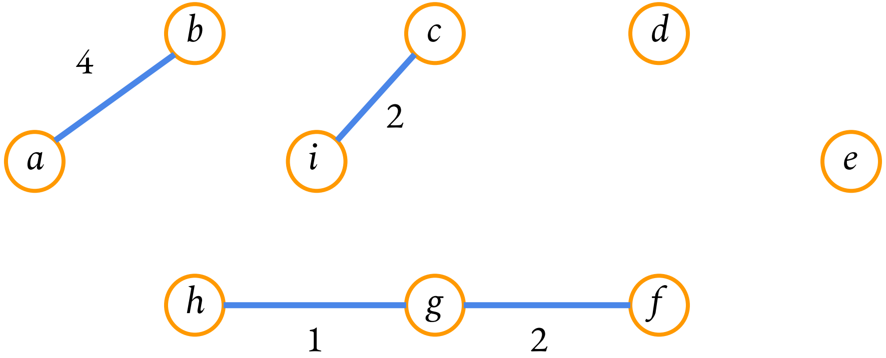


Minimum Spanning Tree

Kruskal's Algorithm

h–*g* → 1 → safe
g–*f* → 2 → safe
c–*i* → 2 → safe
a–*b* → 4 → safe
c–*f* → 4
g–*i* → 6
c–*d* → 7
h–*i* → 7
a–*h* → 8
b–*c* → 8
d–*e* → 9
e–*f* → 10
b–*h* → 11
d–*f* → 11

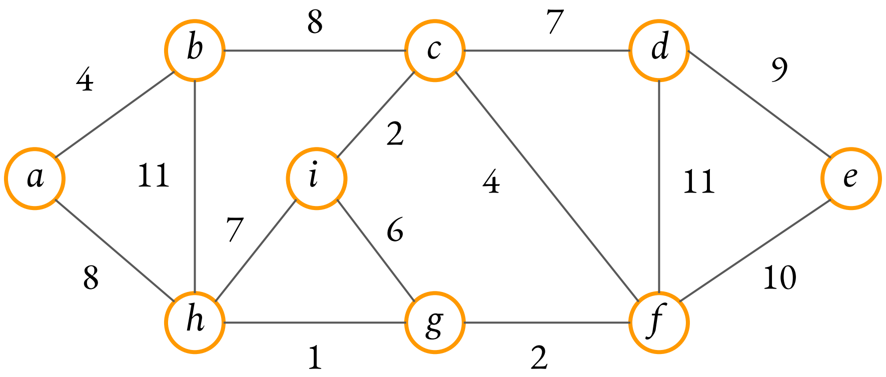


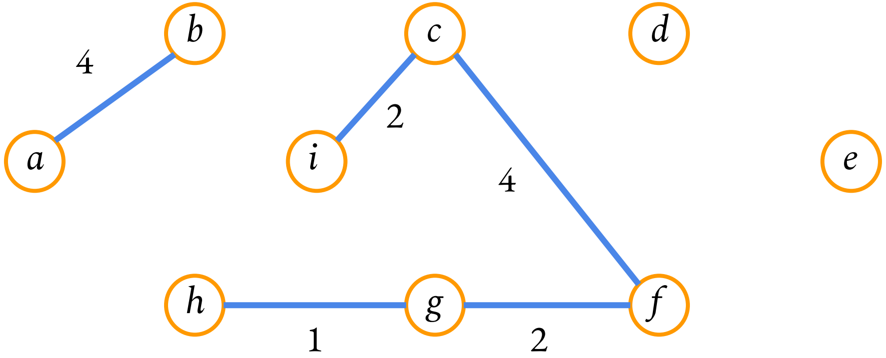


Minimum Spanning Tree

Kruskal's Algorithm

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

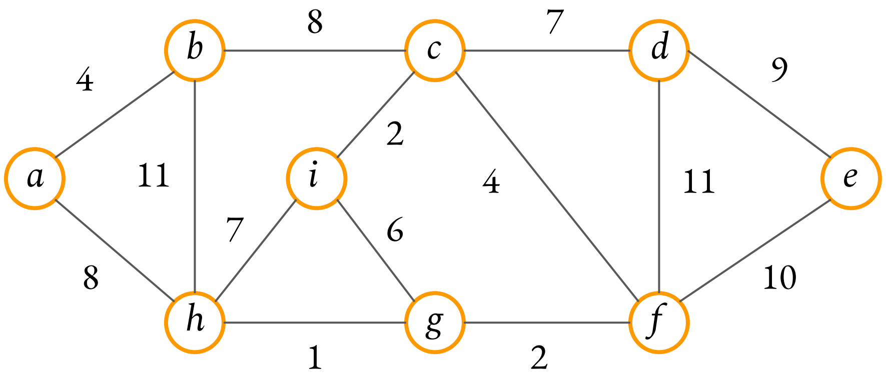


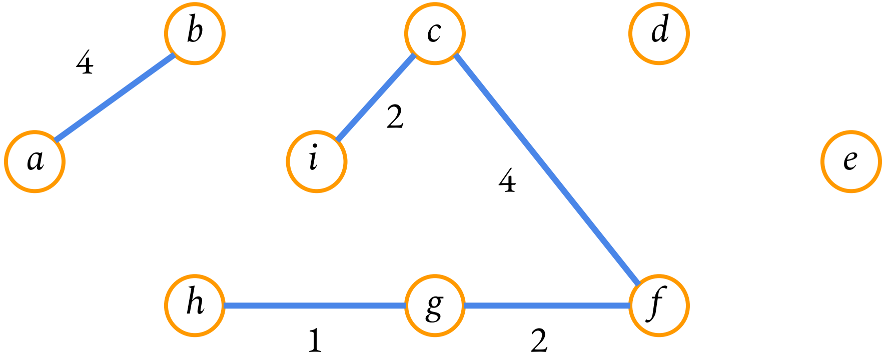


Minimum Spanning Tree

Kruskal's Algorithm

h*–*g → **1** → **safe**
g*–*f → **2** → **safe**
c*–*i → **2** → **safe**
a*–*b → **4** → **safe**
c*–*f → **4** → **safe**
g*–*i → **6** → **reject (forms cycle)**
c*–*d → **7**
h*–*i → **7**
a*–*h → **8**
b*–*c → **8**
d*–*e → **9**
e*–*f → **10**
b*–*h → **11**
d*–*f → **11**

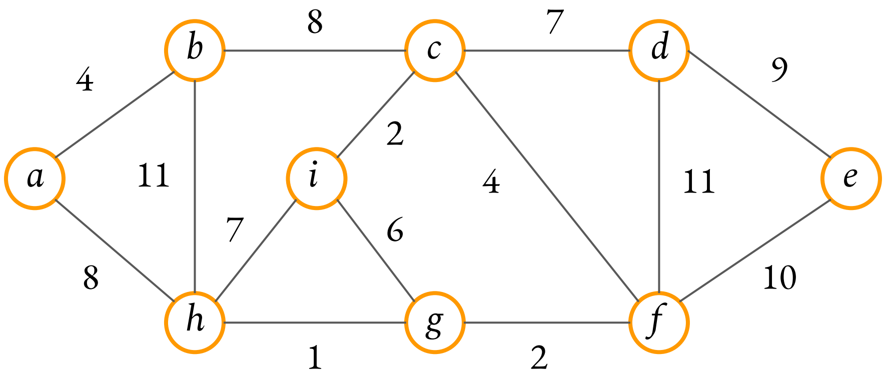


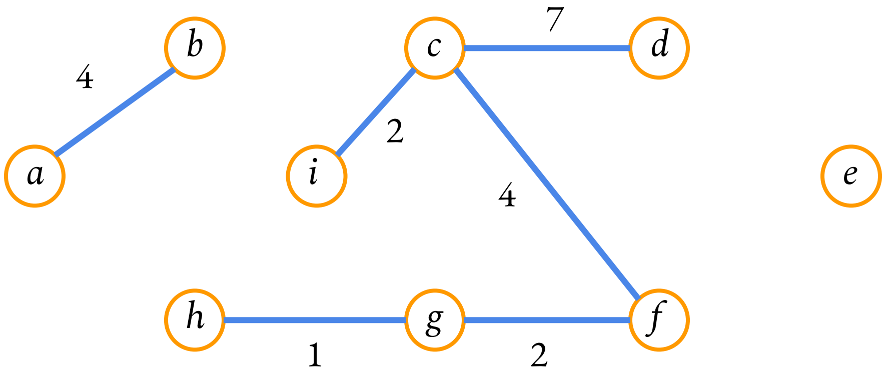


Minimum Spanning Tree

Kruskal's Algorithm

h–*g* → 1 → safe
g–*f* → 2 → safe
c–*i* → 2 → safe
a–*b* → 4 → safe
c–*f* → 4 → safe
g–*i* → 6 → reject (forms cycle)
c–*d* → 7 → safe
h–*i* → 7
a–*h* → 8
b–*c* → 8
d–*e* → 9
e–*f* → 10
b–*h* → 11
d–*f* → 11

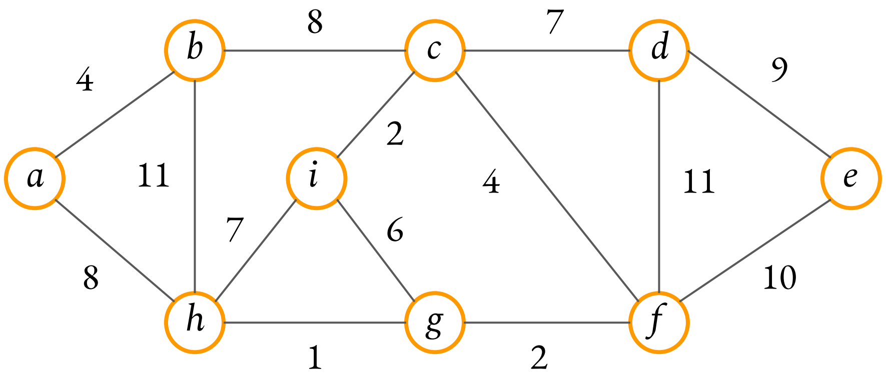


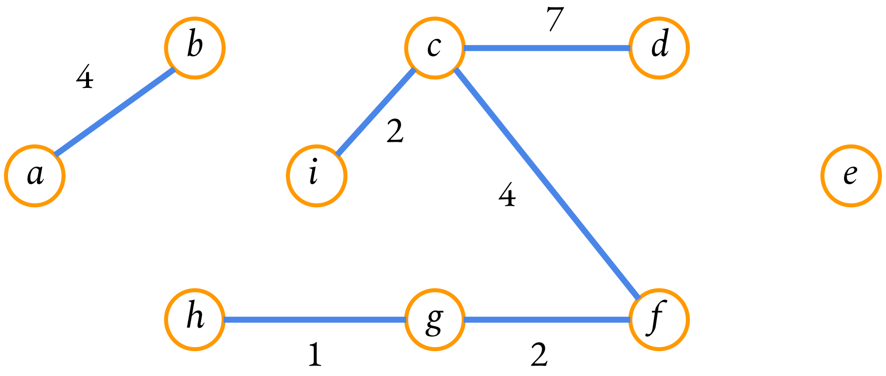


Minimum Spanning Tree

Kruskal's Algorithm

h–*g* → 1 → safe
g–*f* → 2 → safe
c–*i* → 2 → safe
a–*b* → 4 → safe
c–*f* → 4 → safe
g–*i* → 6 → reject (forms cycle)
c–*d* → 7 → safe
h–*i* → 7 → reject (forms cycle)
a–*h* → 8
b–*c* → 8
d–*e* → 9
e–*f* → 10
b–*h* → 11
d–*f* → 11

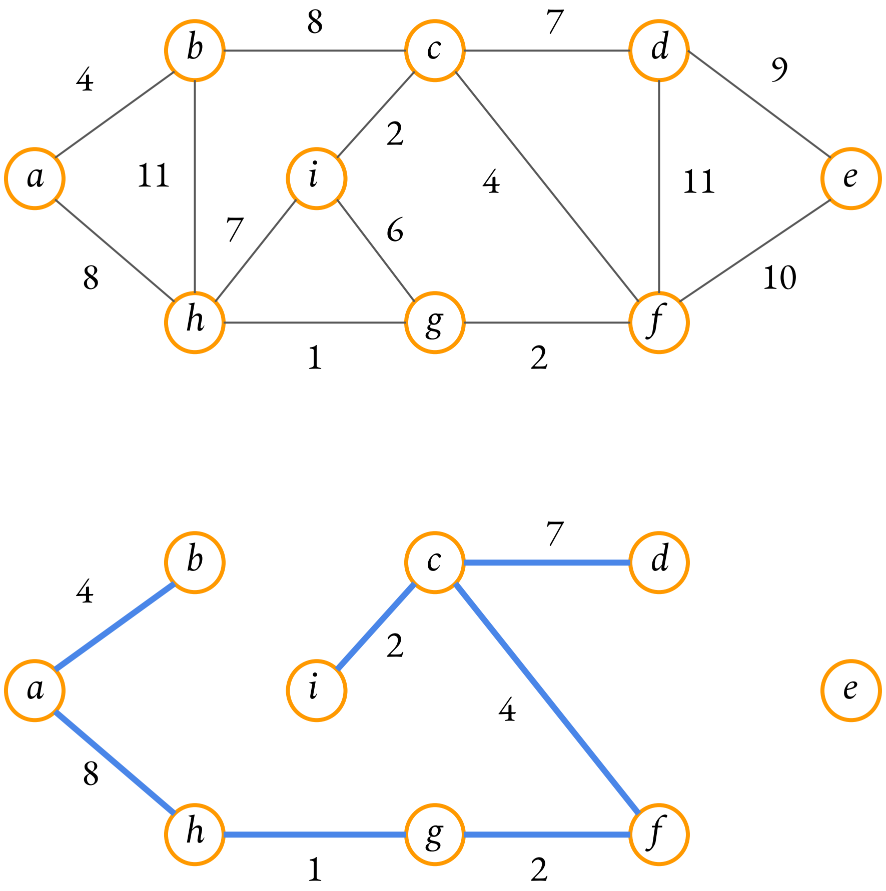




Minimum Spanning Tree

Kruskal's Algorithm

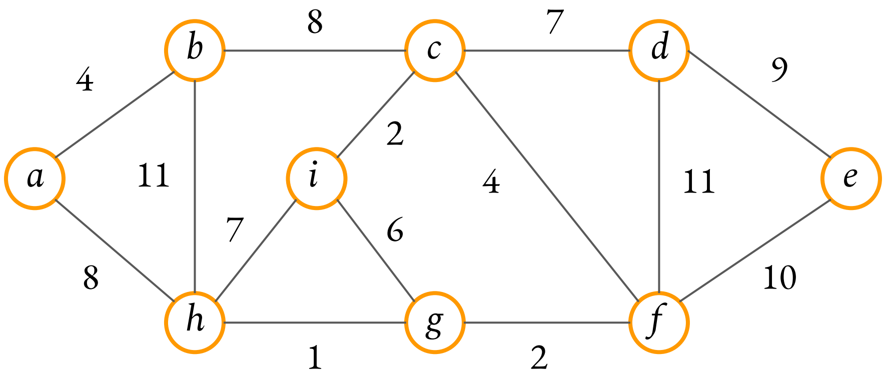
h–g **→ 1 →** **safe**
g–f **→ 2 →** **safe**
c–i **→ 2 →** **safe**
a–b **→ 4 →** **safe**
c–f **→ 4 →** **safe**
g–i **→ 6 →** **reject (forms cycle)**
c–d **→ 7 →** **safe**
h–i **→ 7 →** **reject (forms cycle)**
a–h **→ 8 →** **safe**
b–c **→ 8**
d–e **→ 9**
e–f **→ 10**
b–h **→ 11**
d–f **→ 11**

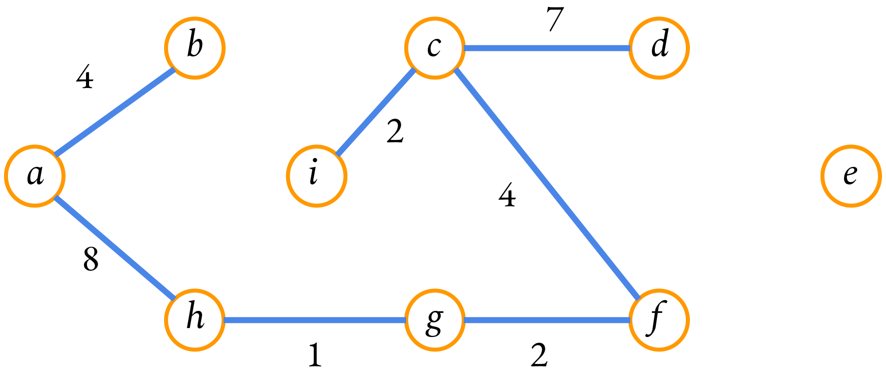


Minimum Spanning Tree

Kruskal's Algorithm

h–g → 1 → safe
g–f → 2 → safe
c–i → 2 → safe
a–b → 4 → safe
c–f → 4 → safe
g–i → 6 → reject (forms cycle)
c–d → 7 → safe
h–i → 7 → reject (forms cycle)
a–h → 8 → safe
b–c → 8 → reject (forms cycle)
d–e → 9
e–f → 10
b–h → 11
d–f → 11

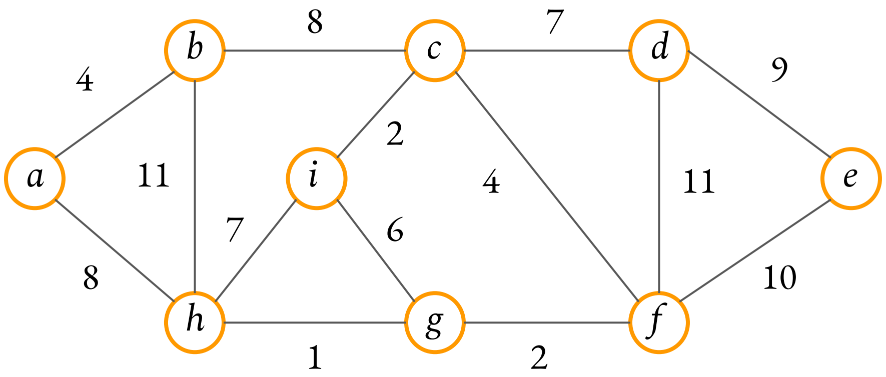


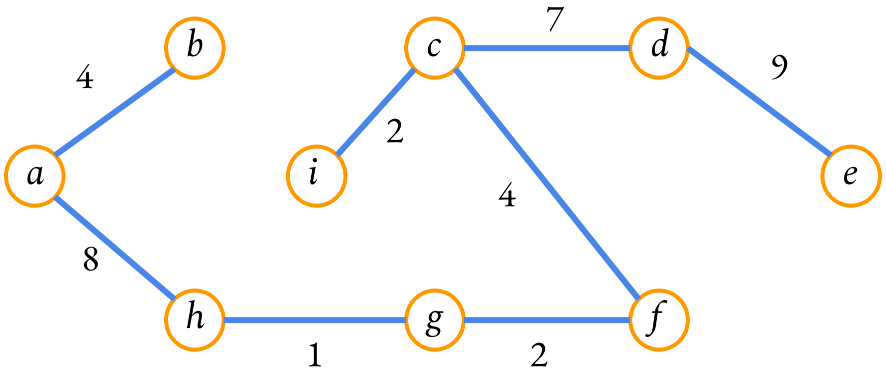


Minimum Spanning Tree

Kruskal's Algorithm

h–g → 1 → safe
g–f → 2 → safe
c–i → 2 → safe
a–b → 4 → safe
c–f → 4 → safe
g–i → 6 → reject (forms cycle)
c–d → 7 → safe
h–i → 7 → reject (forms cycle)
a–h → 8 → safe
b–c → 8 → reject (forms cycle)
d–e → 9 → safe
e–f → 10
b–h → 11
d–f → 11

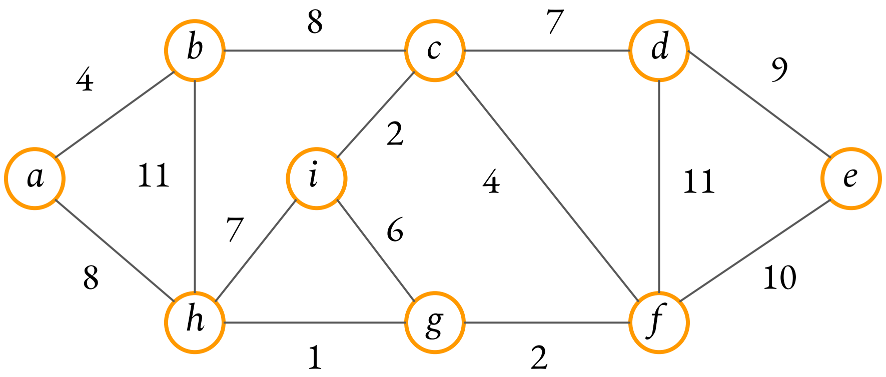


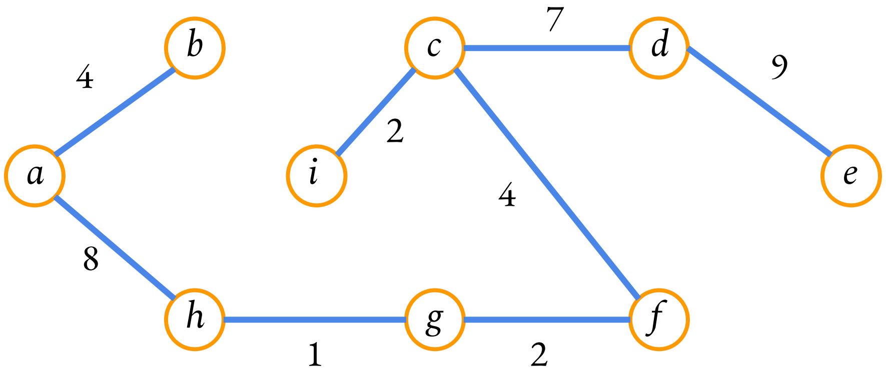


Minimum Spanning Tree

Kruskal's Algorithm

h–g → 1 → safe
g–f → 2 → safe
c–i → 2 → safe
a–b → 4 → safe
c–f → 4 → safe
g–i → 6 → reject (forms cycle)
c–d → 7 → safe
h–i → 7 → reject (forms cycle)
a–h → 8 → safe
b–c → 8 → reject (forms cycle)
d–e → 9 → safe
e–f → 10 → reject (forms cycle)
b–h → 11 → reject (forms cycle)
d–f → 11 → reject (forms cycle)





Prim's Algorithm

Minimum Spanning Tree

```
MST-PRIM(G, w, r)
  for each vertex u ∈ G.V
        u.key = ∞
        u.π = NIL
  r.key = 0
  Q = ∅
  for each vertex u ∈ G.V
        INSERT(Q, u)
  while Q ≠ ∅
        u = EXTRACT-MIN(Q)          // add u to the tree
        for each vertex v in G.Adj[u]     // update keys of u's non-tree neighbors
              if v ∈ Q and w(u, v) < v.key
                    v.π = u
                    v.key = w(u, v)
                    DECREASE-KEY(Q, v, w(u, v))
```

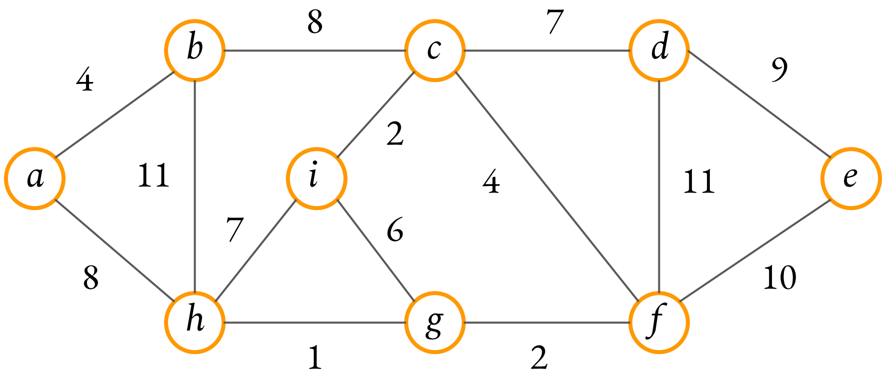
Minimum Spanning Tree

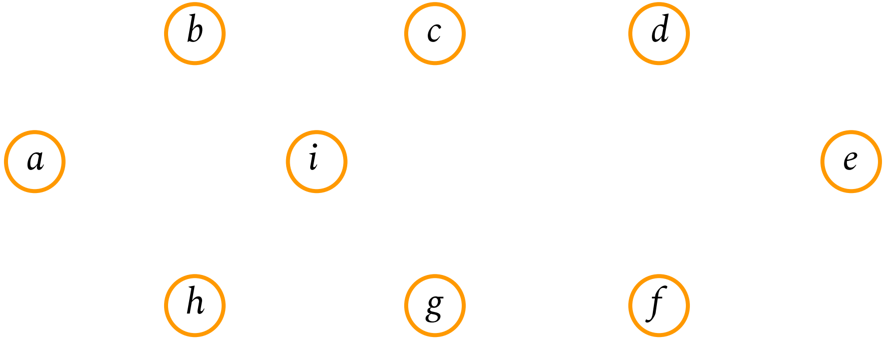
Prim's Algorithm

Step 1: Initialize a tree with a single vertex, chosen arbitrarily from the graph.

Step 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 (*no cycles*).

Step 3: Repeat step 2 (until all vertices are in the tree).





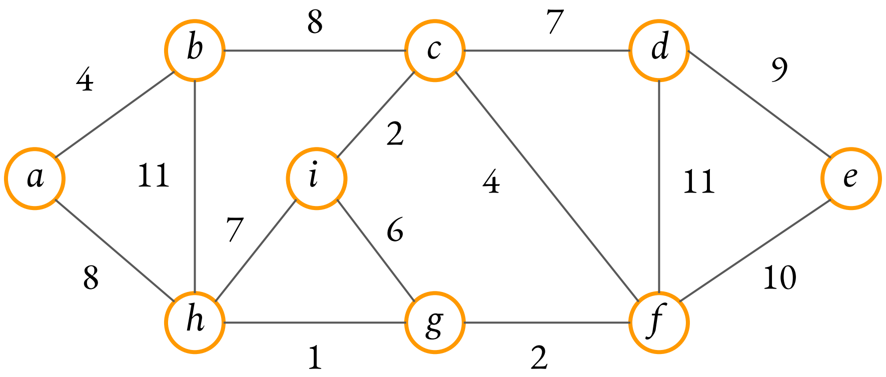
Minimum Spanning Tree

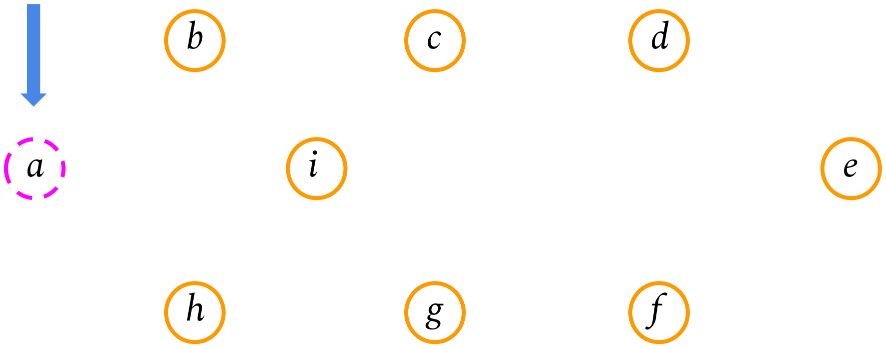
Prim's Algorithm

Step 1: Initialize a tree with a single vertex, chosen arbitrarily from the graph.

Step 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 (*no cycles*).

Step 3: Repeat step 2 (until all vertices are in the tree).



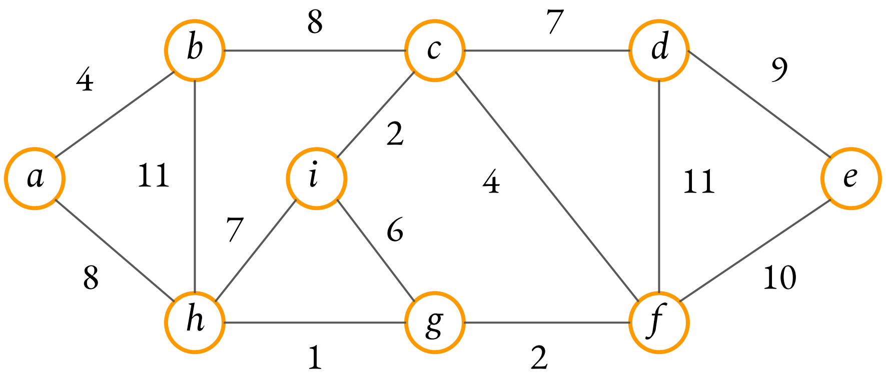


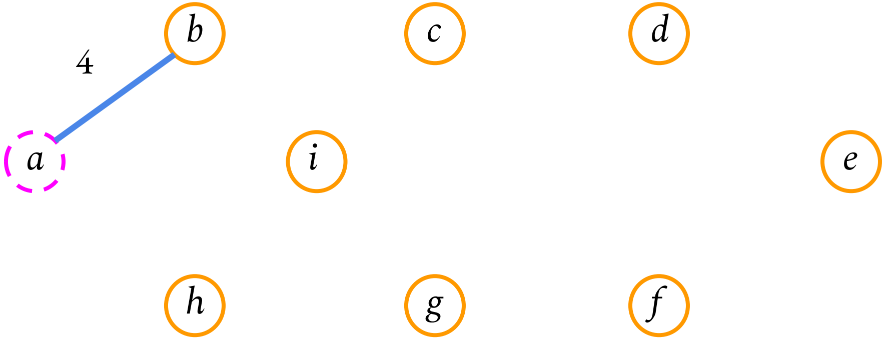
Minimum Spanning Tree

Prim's Algorithm

$a - b \;\rightarrow\; 4 \;\rightarrow$ safe

$a - h \;\rightarrow\; 8$





Minimum Spanning Tree

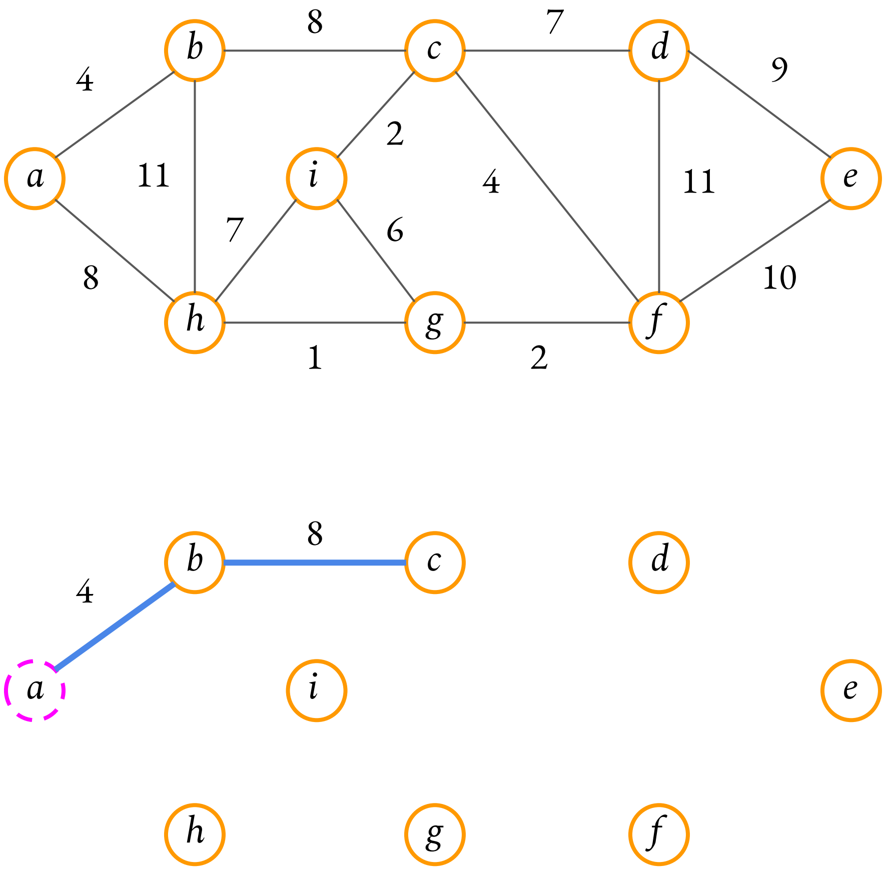
Prim's Algorithm

$a - b \rightarrow 4 \rightarrow$ safe

$a - h \rightarrow 8$

$b - c \rightarrow 8 \rightarrow$ safe

$b - h \rightarrow 11$



Minimum Spanning Tree

Prim's Algorithm

a* – *b → **4** → **safe**

a* – *h → **8**

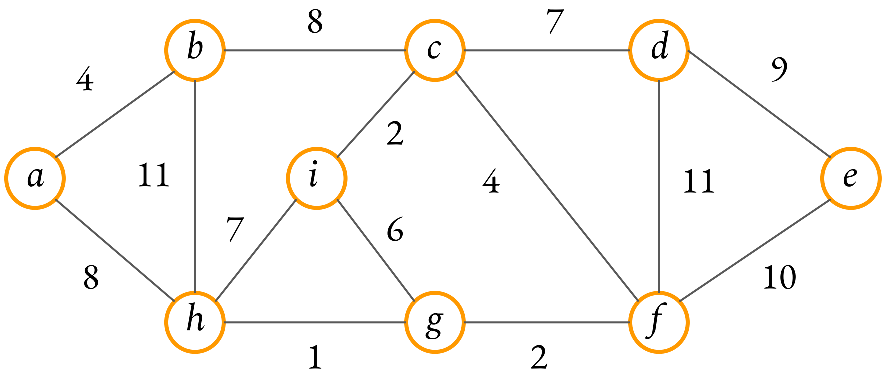
b* – *c → **8** → **safe**

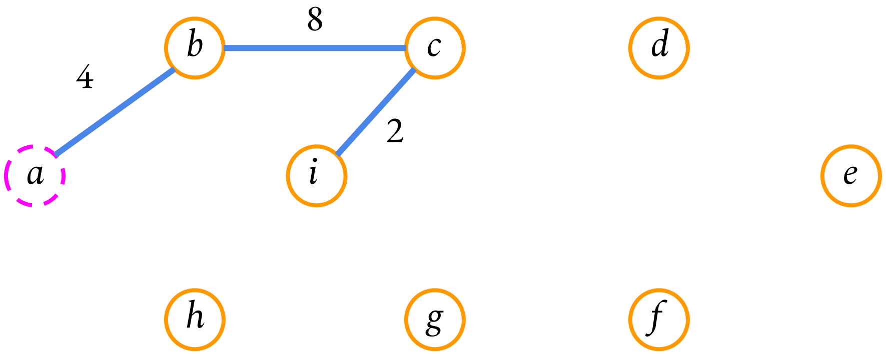
b* – *h → **11**

c* – *d → **7**

c* – *f → **4**

c* – *i → **2** → **safe**





Minimum Spanning Tree

Prim's Algorithm

a – b **→ 4 →** **safe**

a – h **→ 8**

b – c **→ 8 →** **safe**

b – h **→ 11**

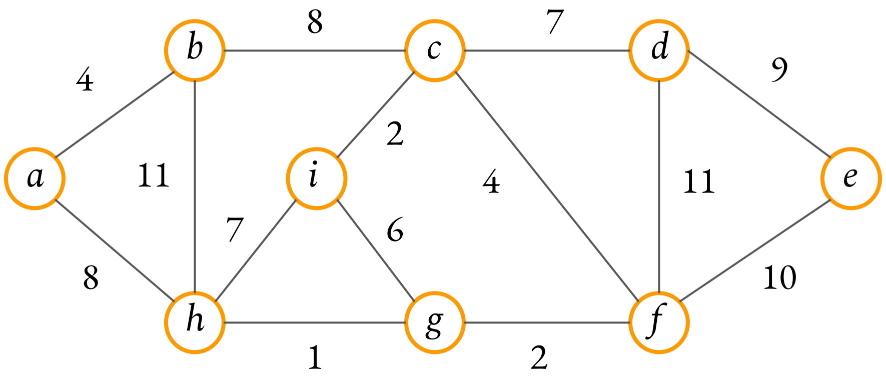
c – d **→ 7**

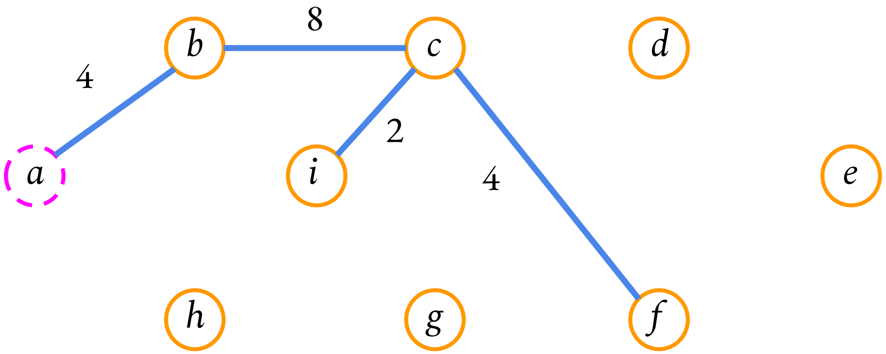
c – f **→ 4 →** **safe**

c – i **→ 2 →** **safe**

i – g **→ 6**

i – h **→ 7**





Minimum Spanning Tree

Prim's Algorithm

$a - b \rightarrow 4 \rightarrow$ safe

$a - h \rightarrow 8$

$b - c \rightarrow 8 \rightarrow$ safe

$b - h \rightarrow 11$

$c - d \rightarrow 7$

$c - f \rightarrow 4 \rightarrow$ safe

$c - i \rightarrow 2 \rightarrow$ safe

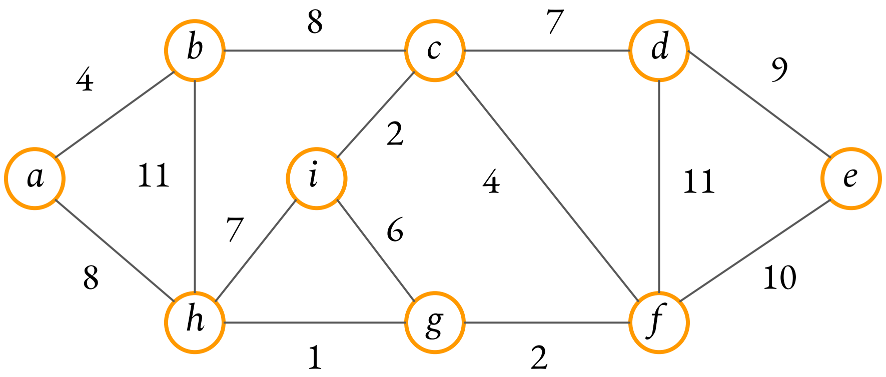
$i - g \rightarrow 6$

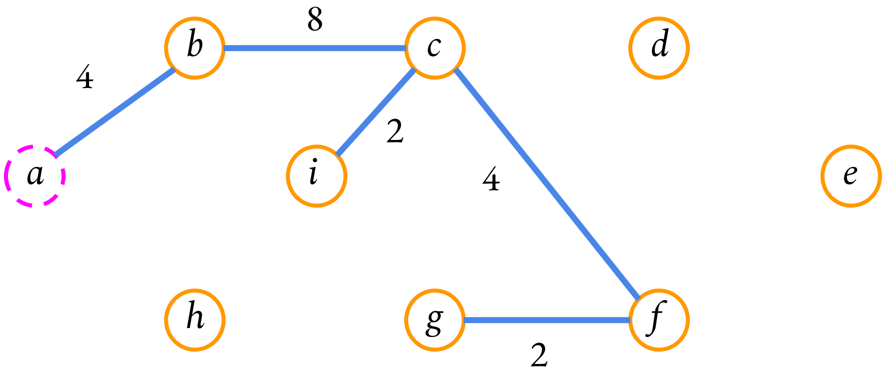
$i - h \rightarrow 7$

$f - d \rightarrow 11$

$f - e \rightarrow 10$

$f - g \rightarrow 2 \rightarrow$ safe





Minimum Spanning Tree

Prim's Algorithm

$a - h \rightarrow 8$

$b - h \rightarrow 11$

$c - d \rightarrow 7$

$i - g \rightarrow 6$

$i - h \rightarrow 7$

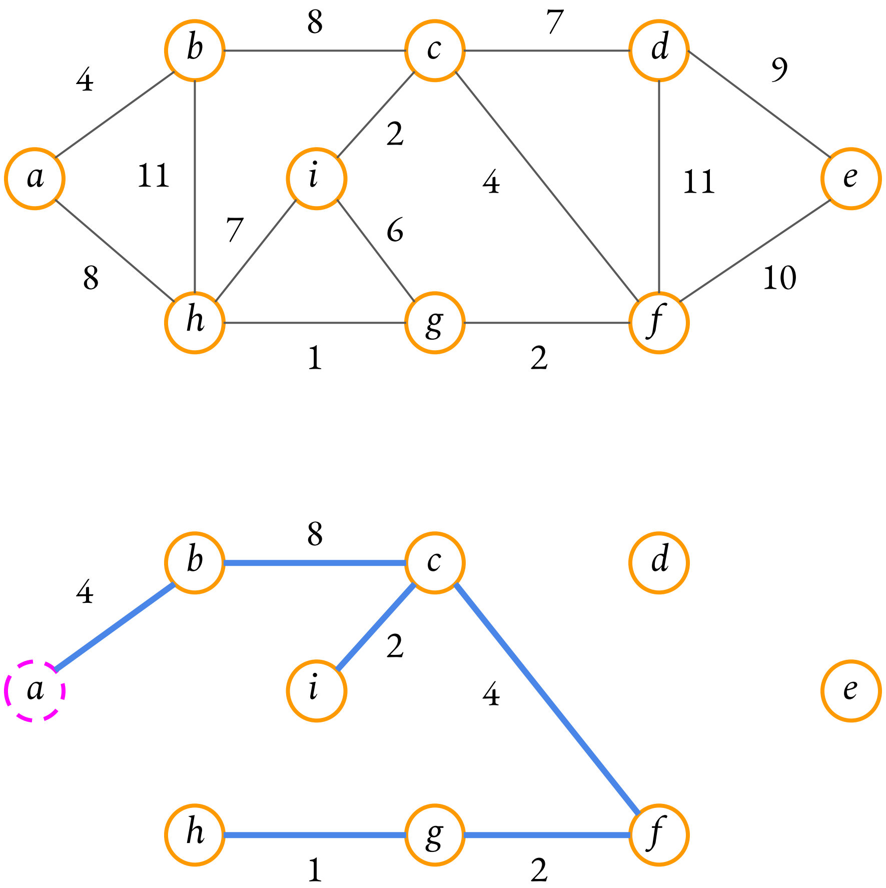
$f - d \rightarrow 11$

$f - e \rightarrow 10$

$f - g \rightarrow 2 \rightarrow$ safe

$g - h \rightarrow 1 \rightarrow$ safe

$g - i \rightarrow 7$



Minimum Spanning Tree

Prim's Algorithm

$a - h \rightarrow 8$

$b - h \rightarrow 11$

$c - d \rightarrow 7$

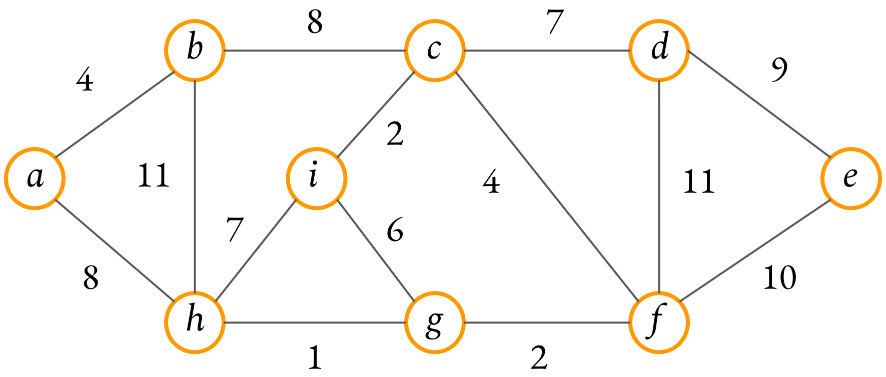
$i - g \rightarrow 6 \rightarrow$ reject (cycle)

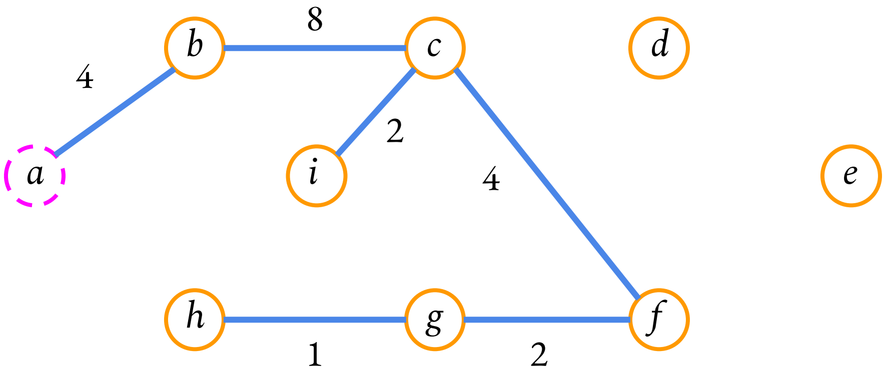
$i - h \rightarrow 7 \rightarrow$ reject (cycle)

$f - d \rightarrow 11$

$f - e \rightarrow 10$

$g - i \rightarrow 7 \rightarrow$ reject (cycle)





MINIMUM SPANNING TREE

Prim's Algorithm

$a - h \rightarrow 8$

$b - h \rightarrow 11$

$c - d \rightarrow 7 \rightarrow$ safe

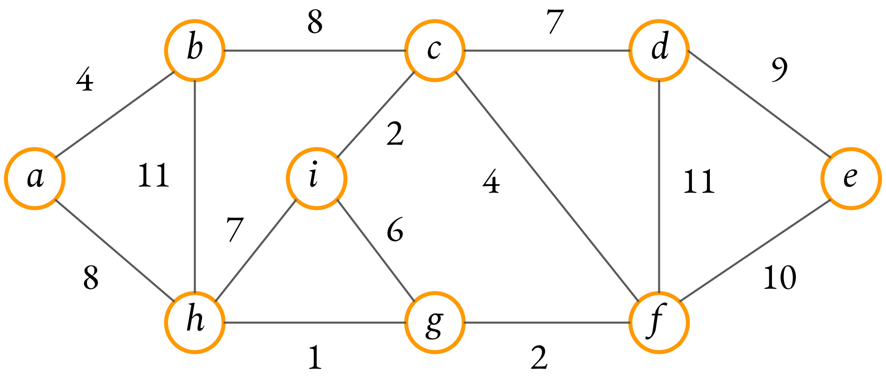
$i - g \rightarrow 6 \rightarrow$ reject (cycle)

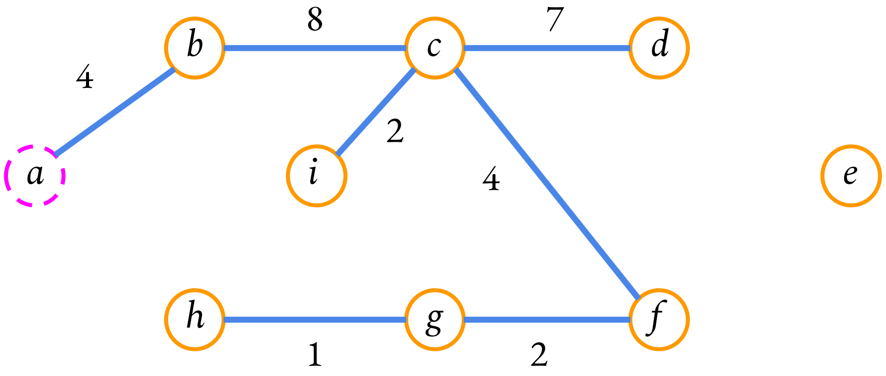
$i - h \rightarrow 7 \rightarrow$ reject (cycle)

$f - d \rightarrow 11$

$f - e \rightarrow 10$

$g - i \rightarrow 7 \rightarrow$ reject (cycle)

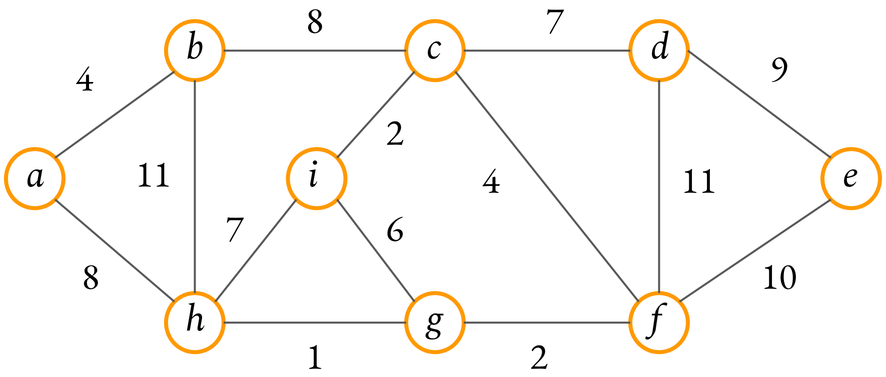


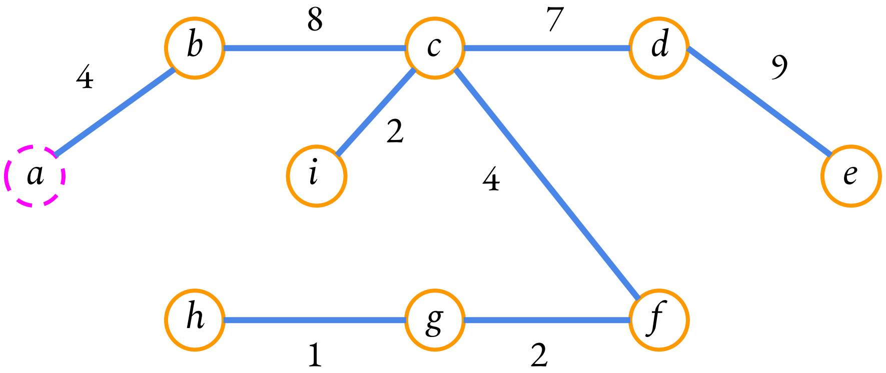


Minimum Spanning Tree

Prim's Algorithm

a* – *h → **8** → **reject (cycle)**
b* – *h → **11** → **reject (cycle)**
c* – *d → **7** → **safe**
i* – *g → **6** → **reject (cycle)**
i* – *h → **7** → **reject (cycle)**
f* – *d → **11** → **reject (cycle)**
f* – *e → **10**
g* – *i → **7** → **reject (cycle)**
d* – *e → **9** → **safe**





Minimum Spanning Tree

Prim's Algorithm

$a - h$ → 8 → reject (cycle)

$b - h$ → 11 → reject (cycle)

$c - d$ → 7 → safe

$i - g$ → 6 → reject (cycle)

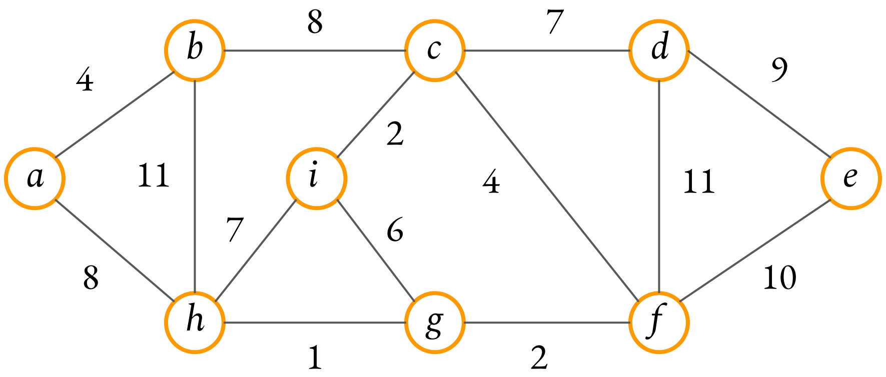
$i - h$ → 7 → reject (cycle)

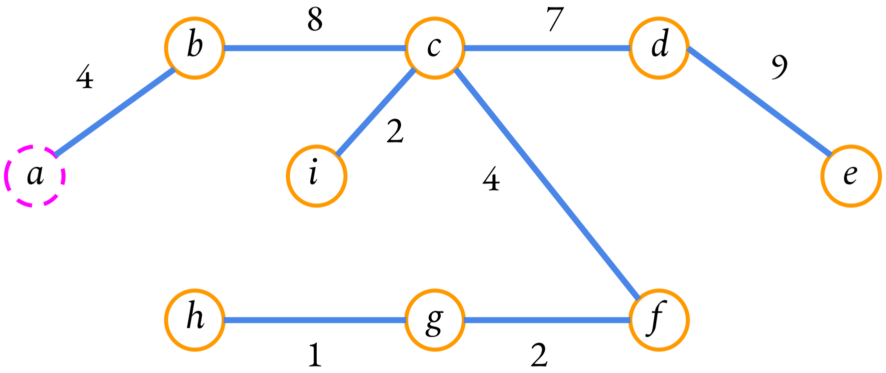
$f - d$ → 11 → reject (cycle)

$f - e$ → 10 → reject (cycle)

$g - i$ → 7 → reject (cycle)

$d - e$ → 9 → safe





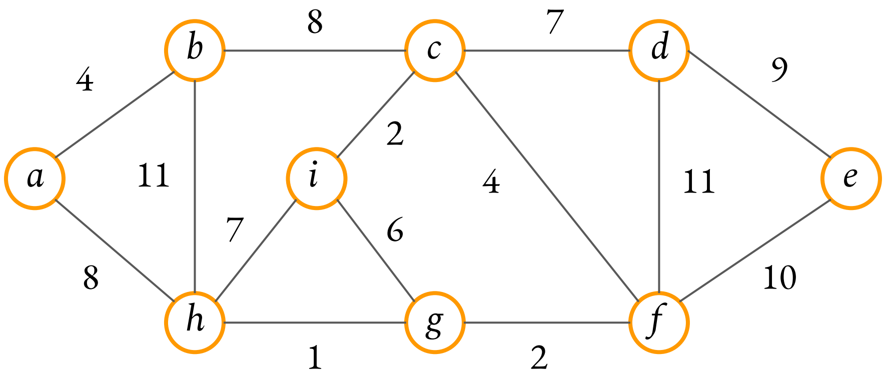
Minimum Spanning Tree

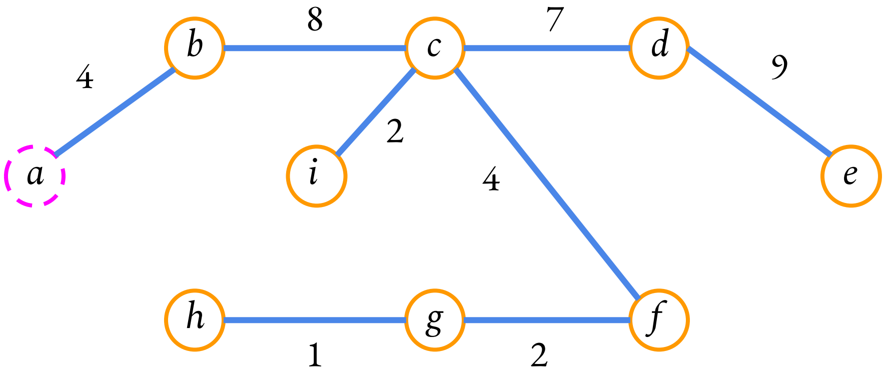
Prim's Algorithm

Step 1: Initialize a tree with a single vertex, chosen arbitrarily from the graph.

Step 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 (*no cycles*).

Step 3: Repeat step 2 (until all vertices are in the tree).





Minimum Spanning Tree

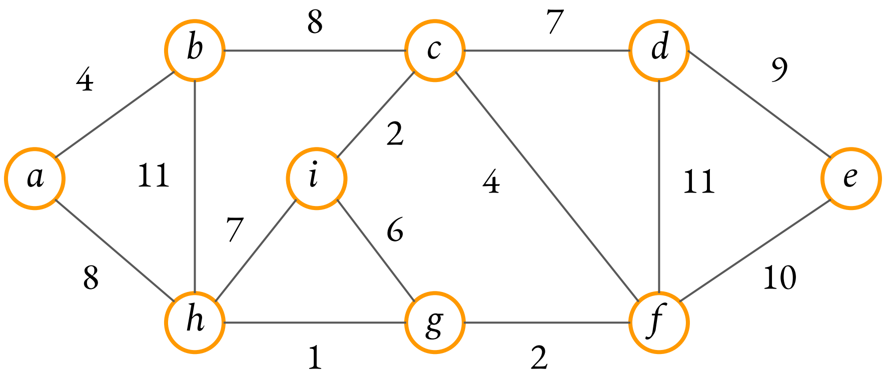
Kruskal's Algorithm

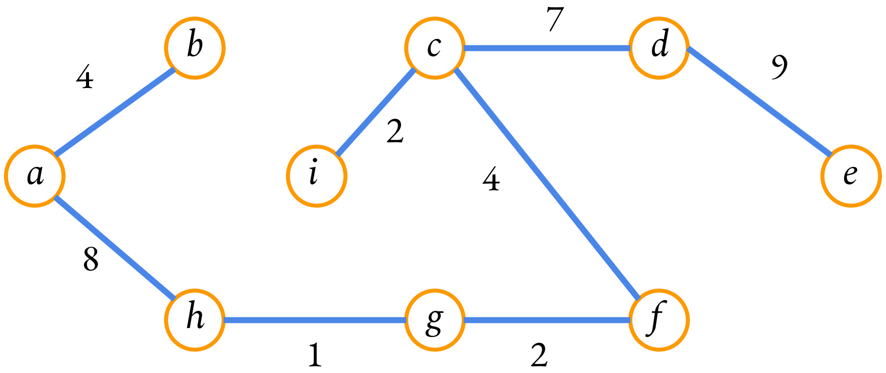
Step 1: Arrange the edges of G in order of increasing weights.

Step 2: Starting only with the vertices of G and proceeding sequentially, add each edge which *does not result in a cycle* until **n-1** edges are added.

Step 3: Exit

Note: Forest gets combined to a tree





Greedy Algorithmic Approach

Both **Kruskal's** and **Prim's** algorithms consider adding edges with lowest possible weights (**local best solution**) with the hope that it will lead to optimal MST (**global best solution**).

Kruskal's algorithm **combines forest** but Prim's algorithm **grows a tree**. Both are using the greedy strategy.