Ontario Tech Science

Kourosh Davoudi
heidar.davoudi@ontariotechu.ca

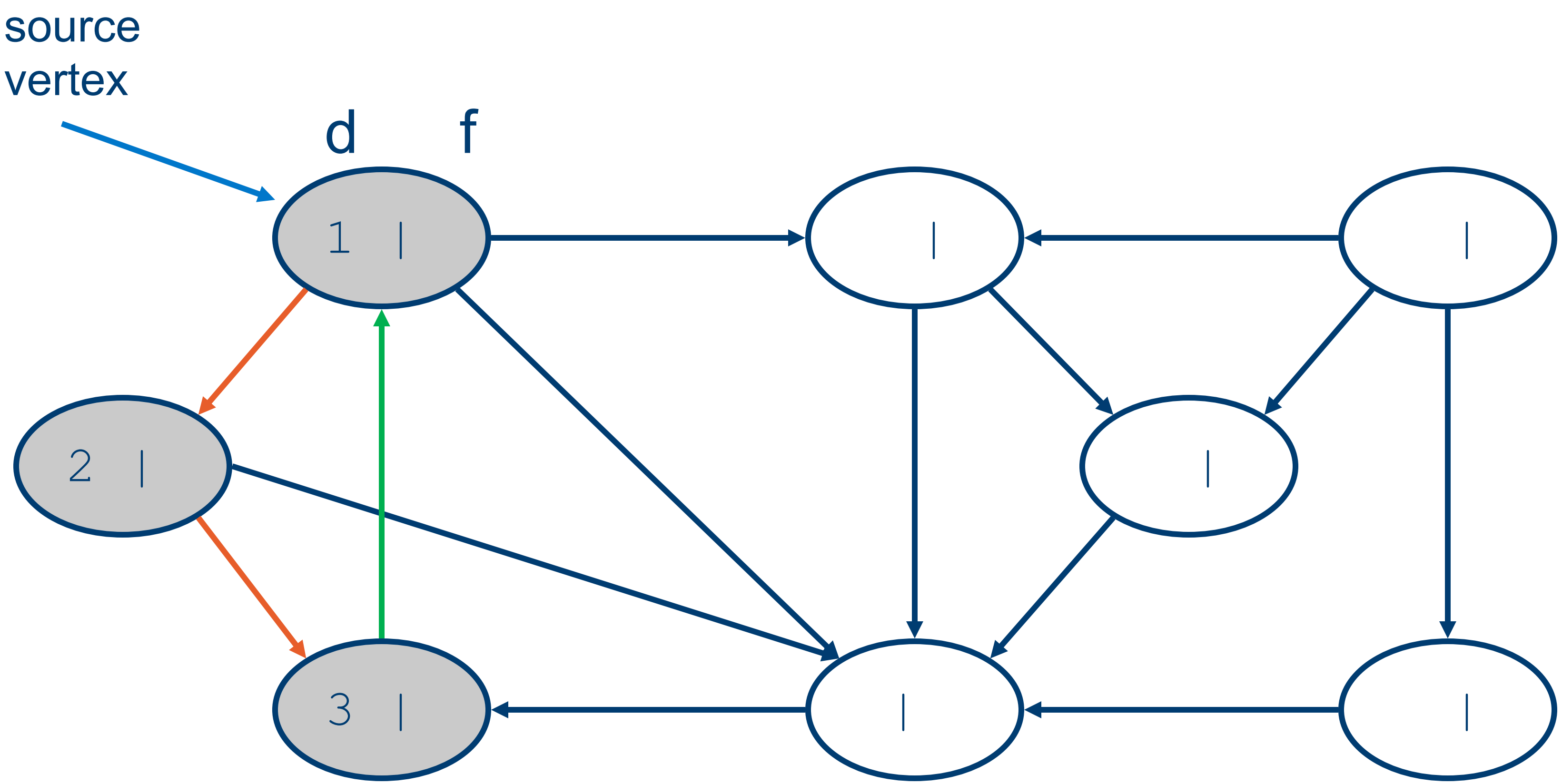
Lecture 10: Topological Sort
Strongly Connected Components
Minimum Spanning Tree

CSCI 3070U: Design and Analysis of Algorithms

Learning Outcomes

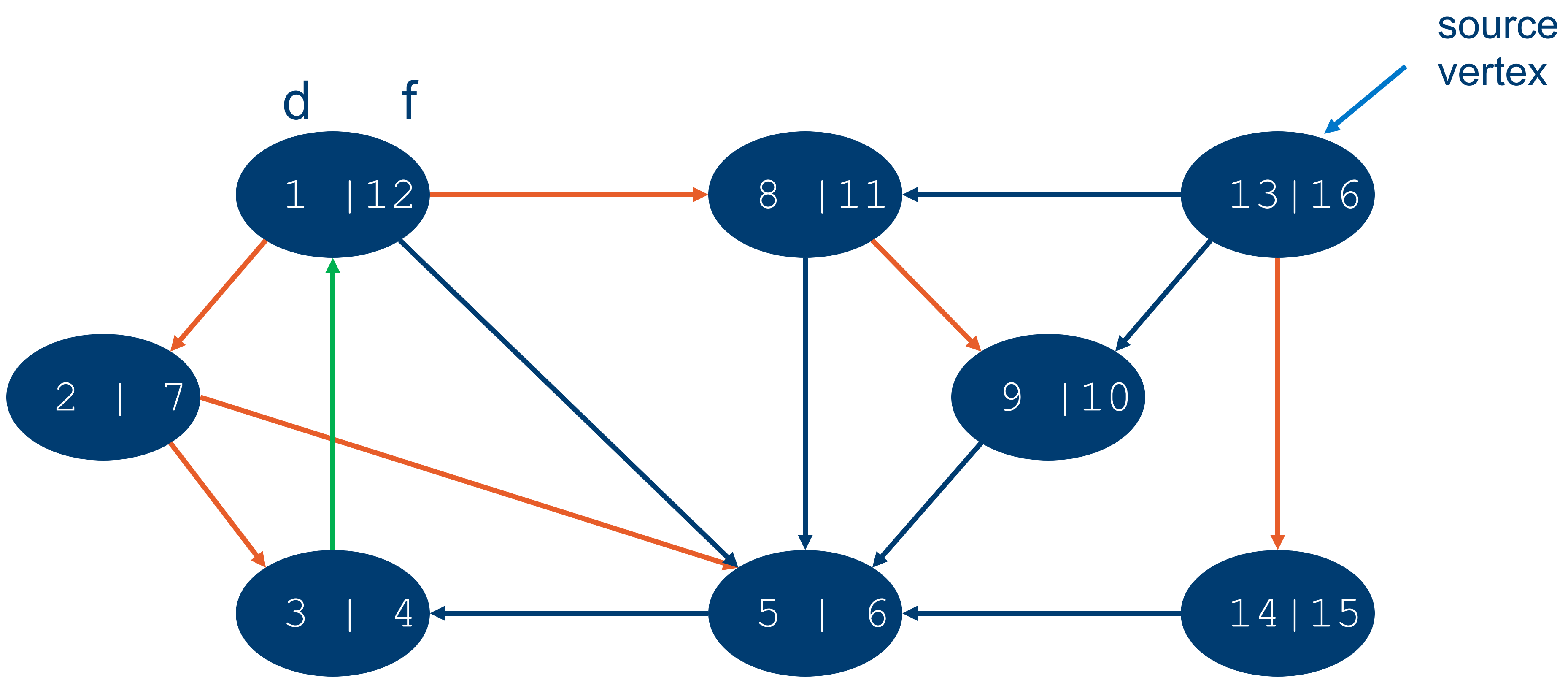
- Applications of DFS
  - Topological Sort
  - Strongly Connected Component
- Minimum Spanning Tree

DFS Example (Recap)



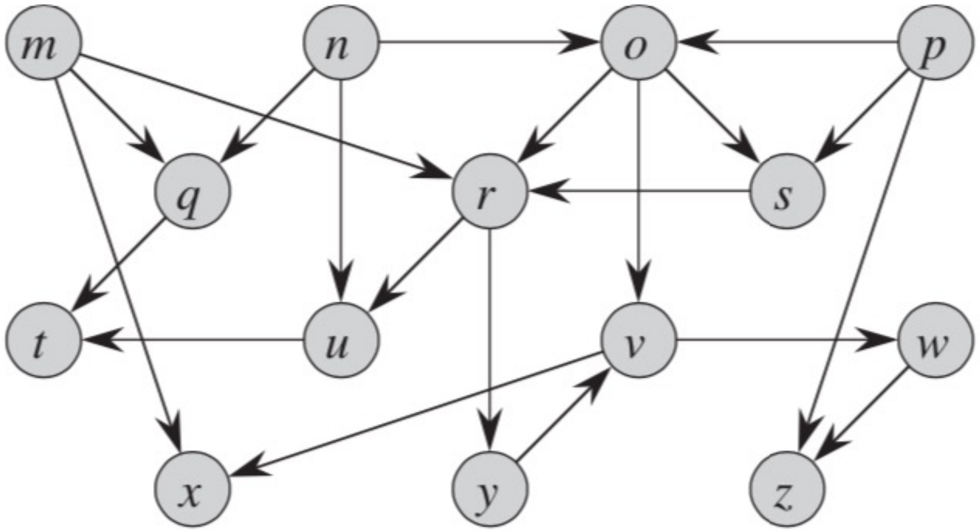
Back Edge: from descendent to ancestor (gray → gray)

DFS Example (Recap)



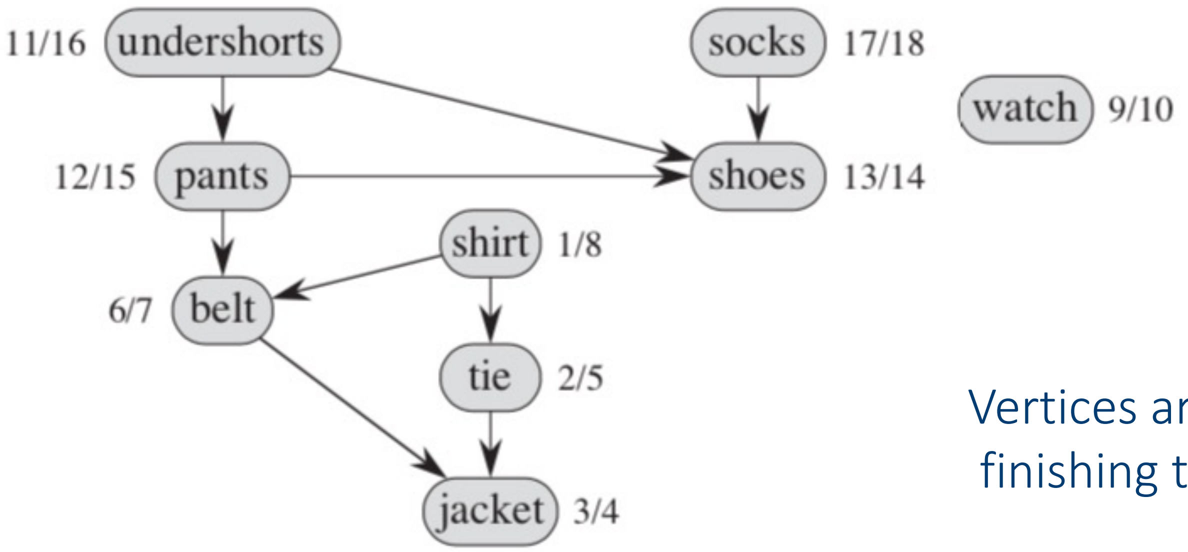
Topological Sort

- Background
  - Directed Acyclic Graph (DAG): A directed graph with no cycles

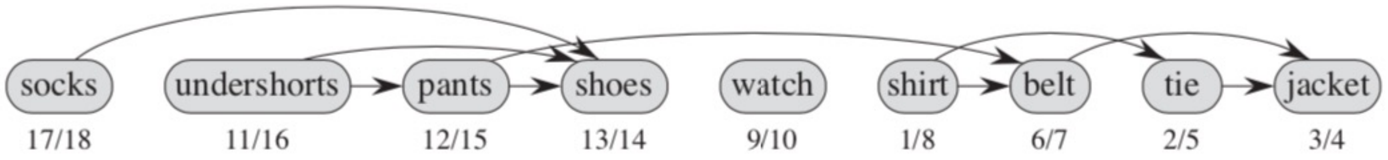


Topological Sort

- Topological sort:
    - Given a DAG, a linear ordering of vertices such that if $(u, v) \in E$ then u appears somewhere before v.



Vertices are in order of decreasing finishing times !



Topological Sort

- Topological sort:
  - Given a DAG, a linear ordering of vertices such that if $(u, v) \in E$ then u appears somewhere before v.

```
TOPOLOGICAL-SORT(G)
  call DFS(G) to compute finishing times v.f for all v ∈ G.V
  output vertices in order of decreasing finishing times
```

  - You can just record vertices as they are finished and print them in reverse order !

$$\Theta(V + E)$$

Topological Sort

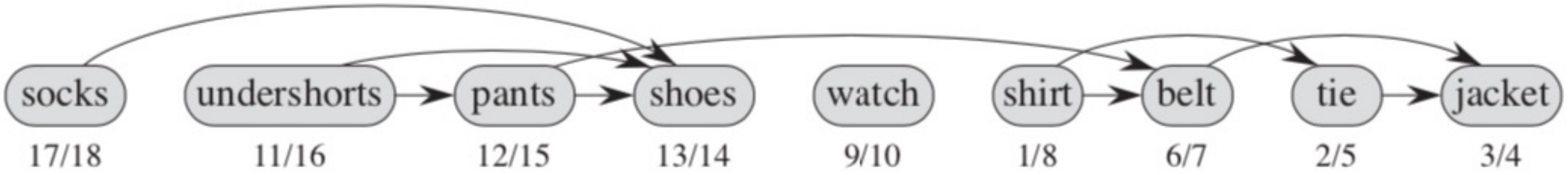
- Why it is correct?
  - Claim: If $(u, v) \in \mathrm{E} \Rightarrow u.f > v.f$ OR $u.f < v.f \Rightarrow (u, v) \notin \mathrm{E}$

    Proof: When (u, v) is explored, u is gray

    $v = gray$: contradiction ! Why?

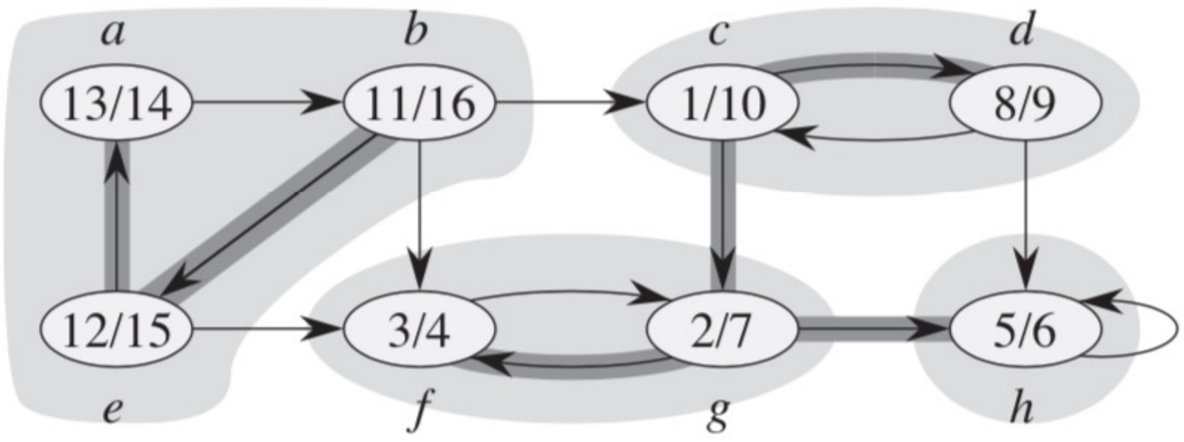
    $v = white$: v is descendent of u so $u.f > v.f$

    $v = black$: v already finished $u.f > v.f$



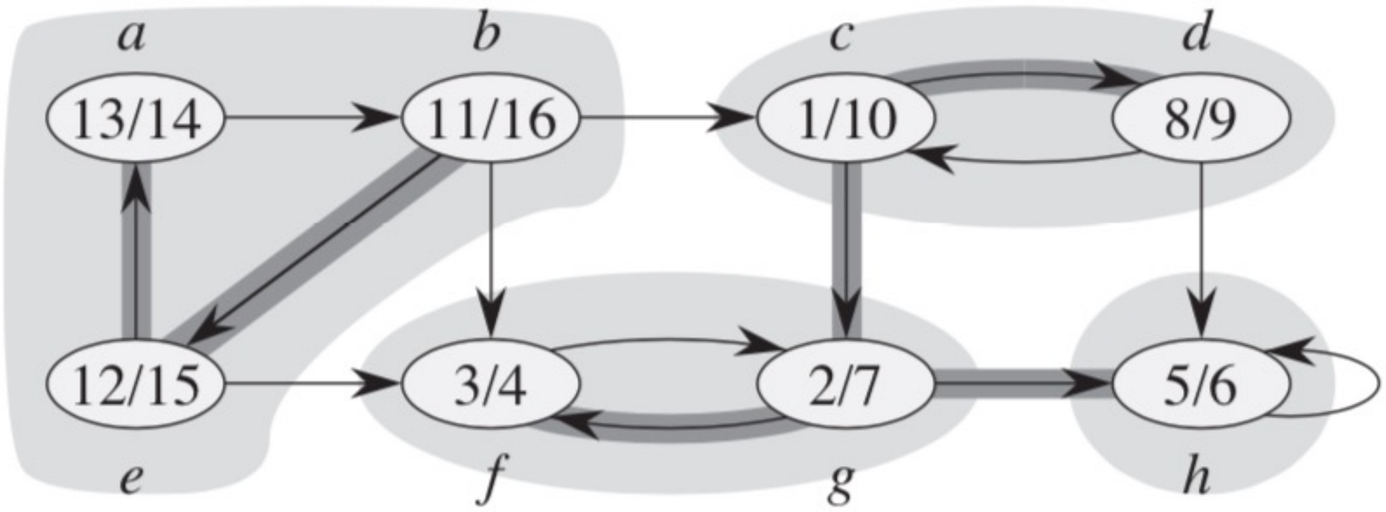
Strongly Connected Components

- Recap:
  - Given directed graph $G = (V, E)$
    - Strongly Connected Component (SCC) of G is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both u and v are reachable from each other.



Strongly Connected Components

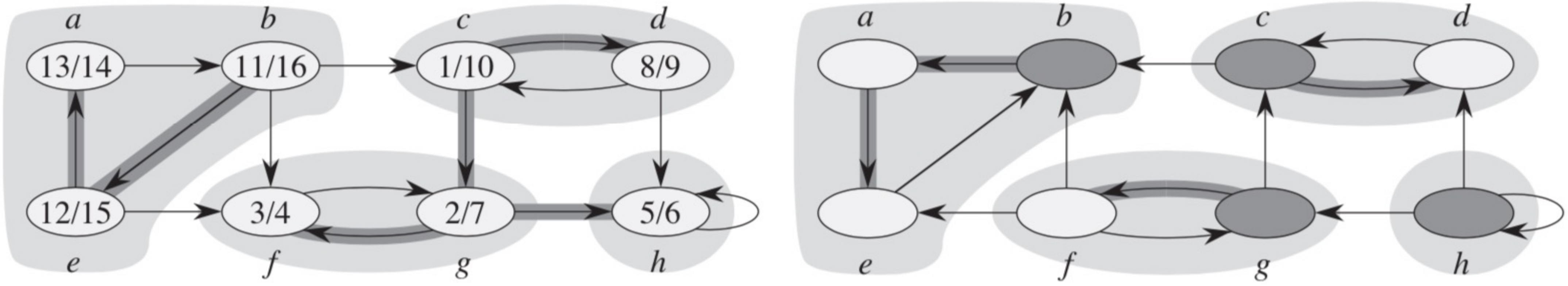
- **Problem:** How can we identify the Strongly Connected Components (SCC)?



We can use DFS to identify SCC

Strongly Connected Components

- The algorithm uses transpose graph $G^T = (V, E^T)$



$G = (V, E)$ $\qquad$ $G^T = (V, E^T)$

Graph G and G^T have exactly the same strongly connected components

Given an adjacency-list representation of G, the time to create G^T is $O(V + E)$

Strongly Connected Components

- Algorithm:

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

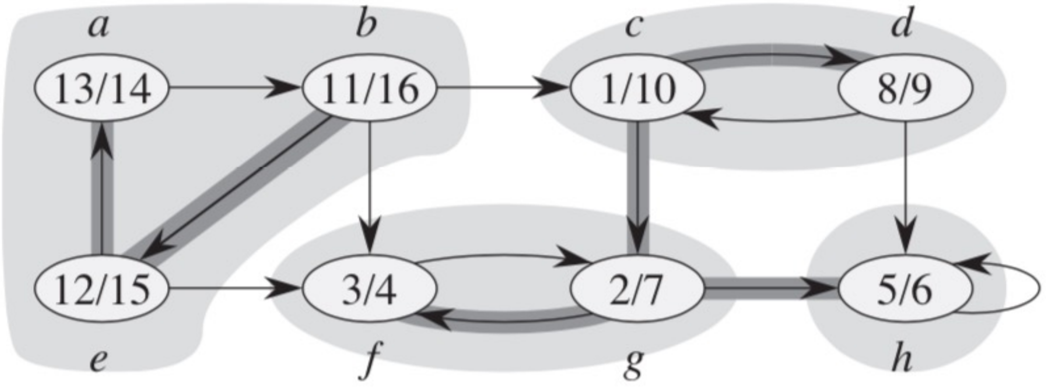
DFS(G)

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)

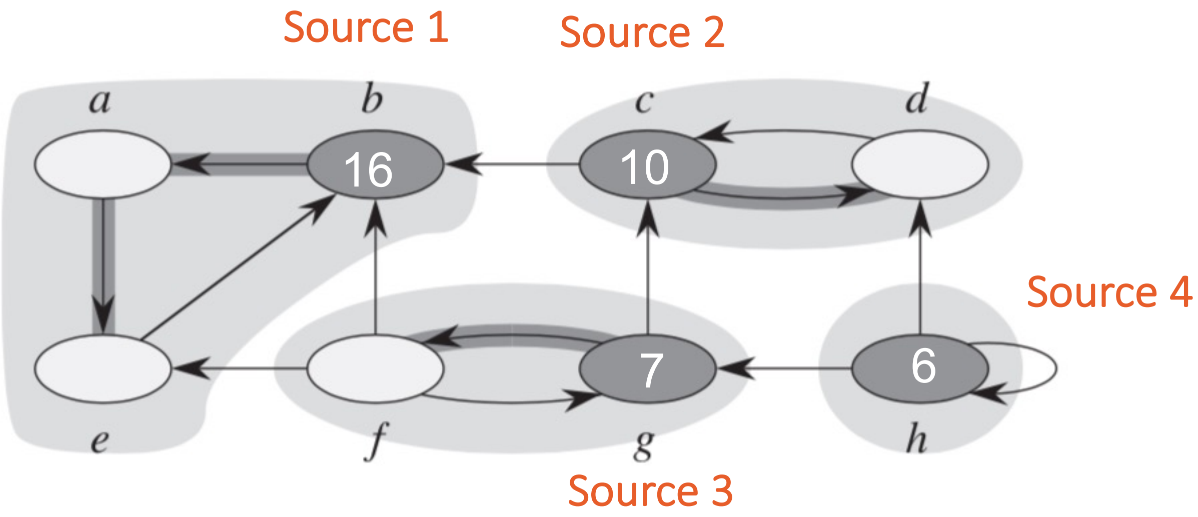
Strongly Connected Components

Example:

$G = (V, E)$

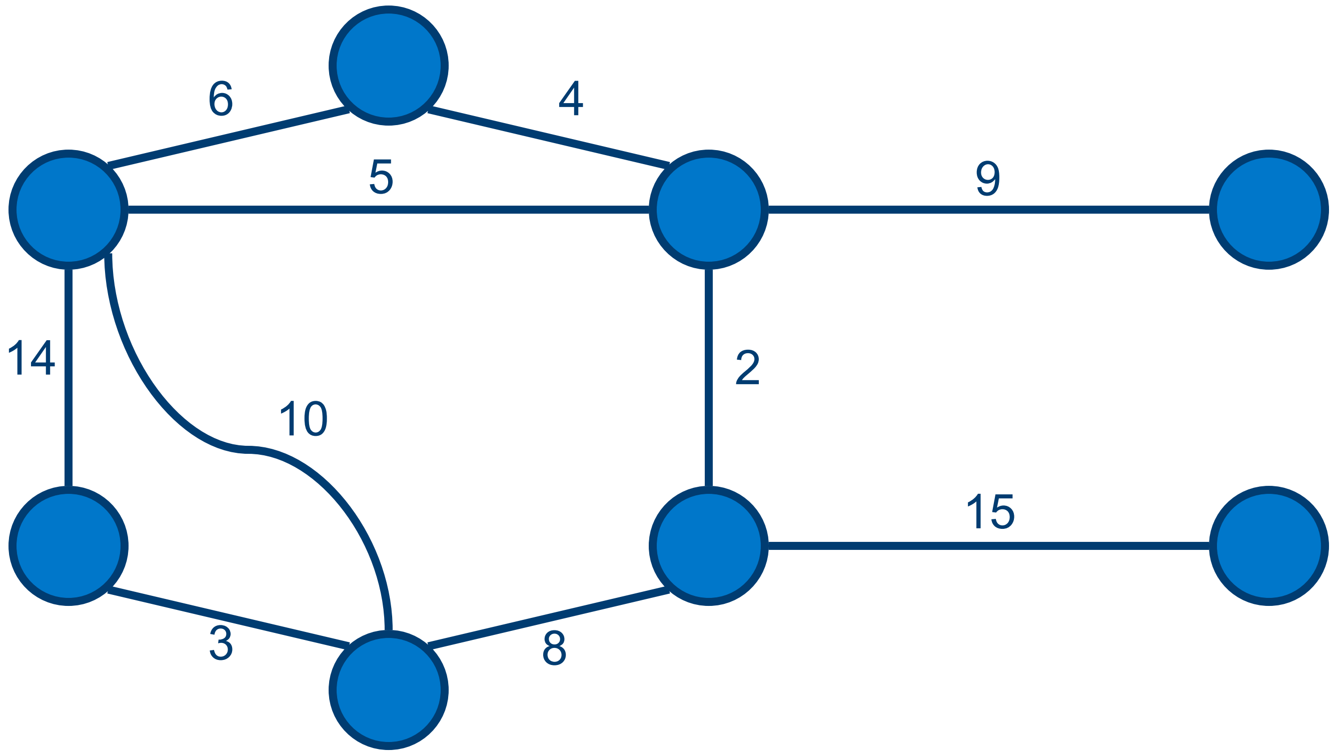


$G^T = (V, E^T)$



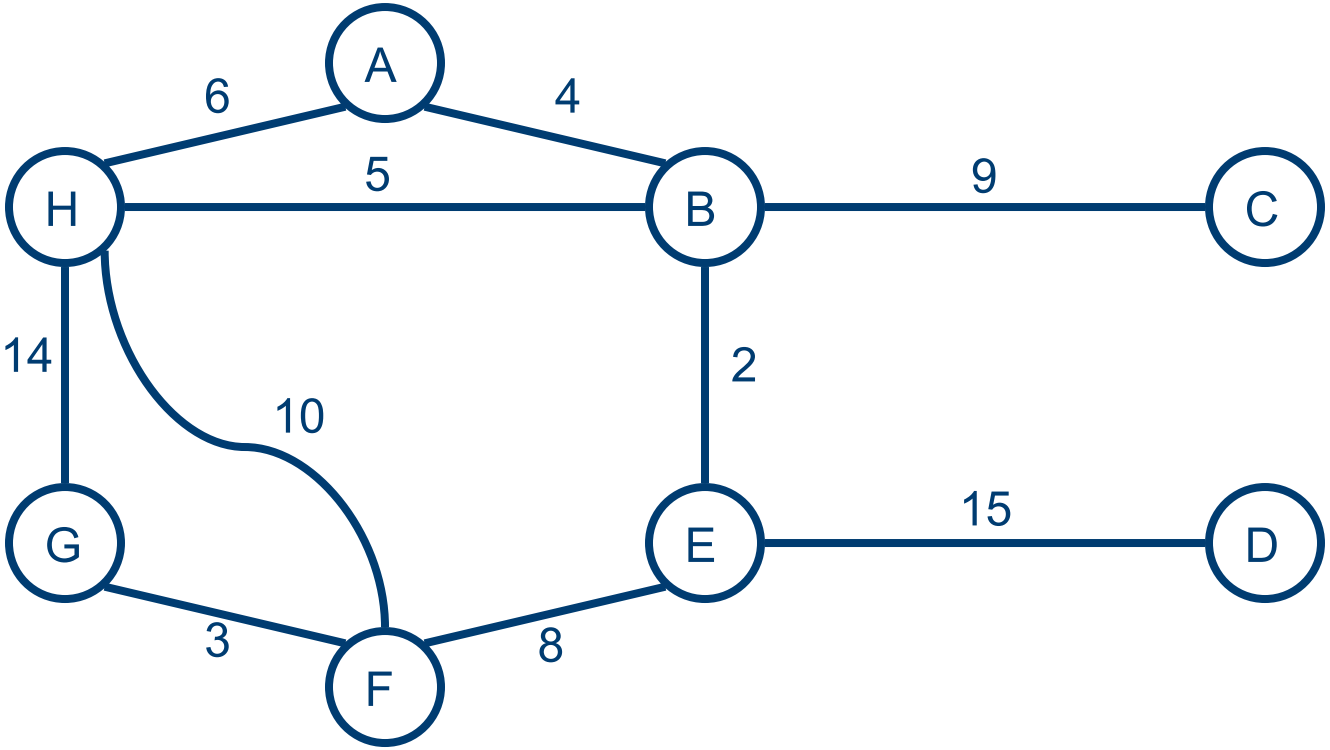
Minimum Spanning Tree

- Problem: given a connected, undirected, weighted graph, find a spanning tree using edges that minimize the total weight



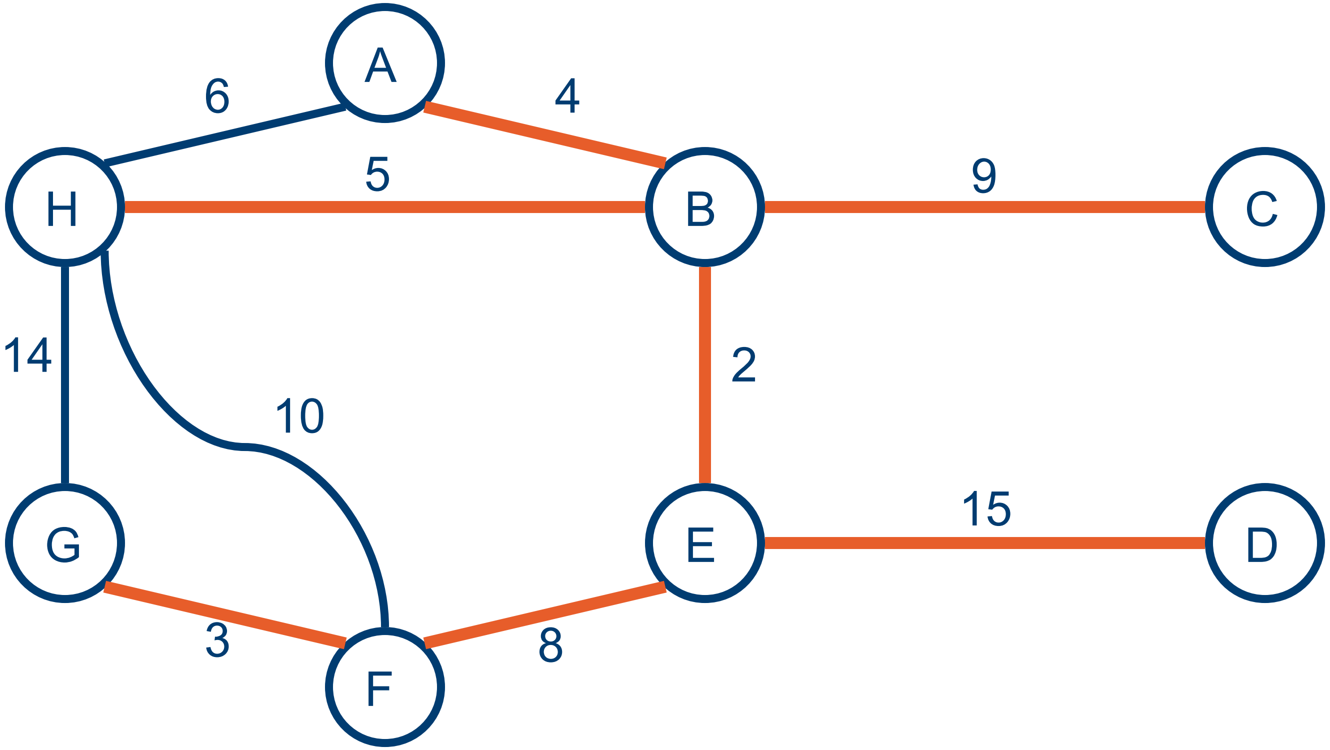
Minimum Spanning Tree

- Which edges form the minimum spanning tree (MST) of the below graph?



Minimum Spanning Tree

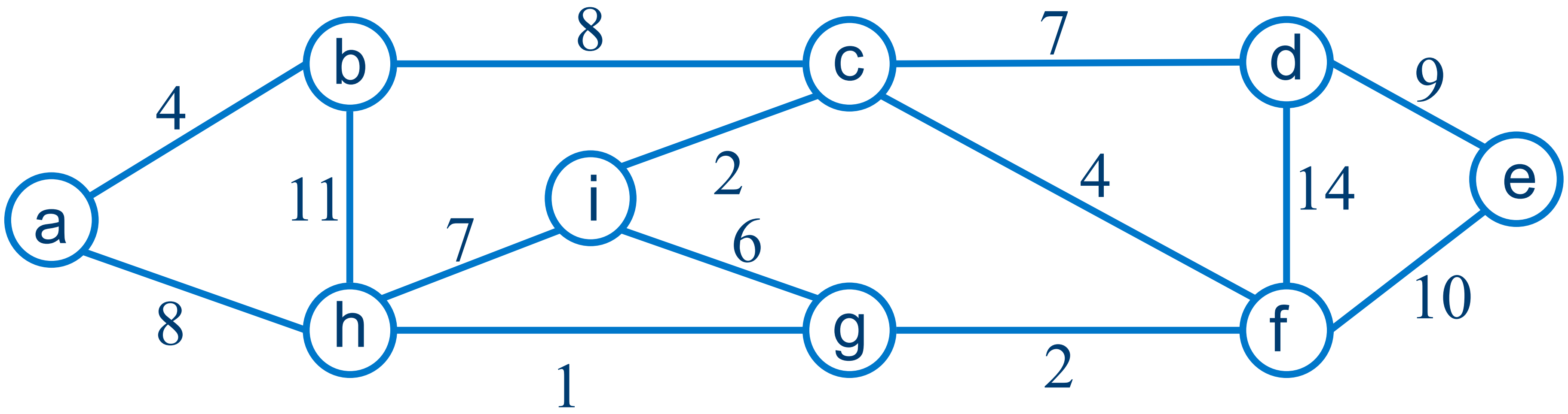
- Answer:



Kruskal's algorithm

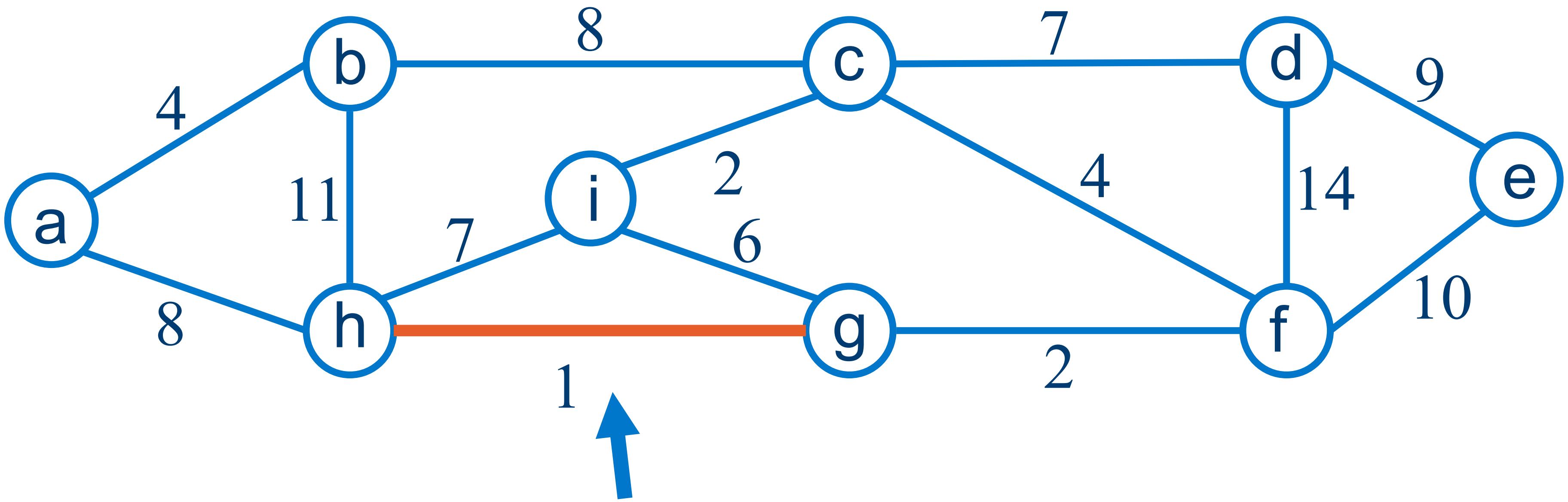
```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Kruskal’s Algorithm



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

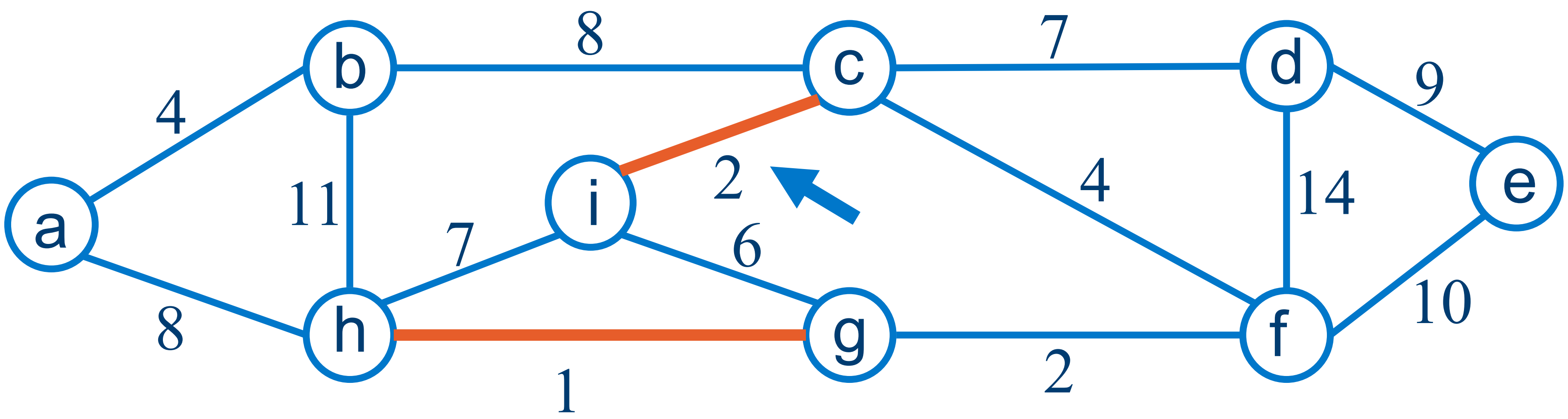
Kruskal’s Algorithm



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

{g} ∪ {h}

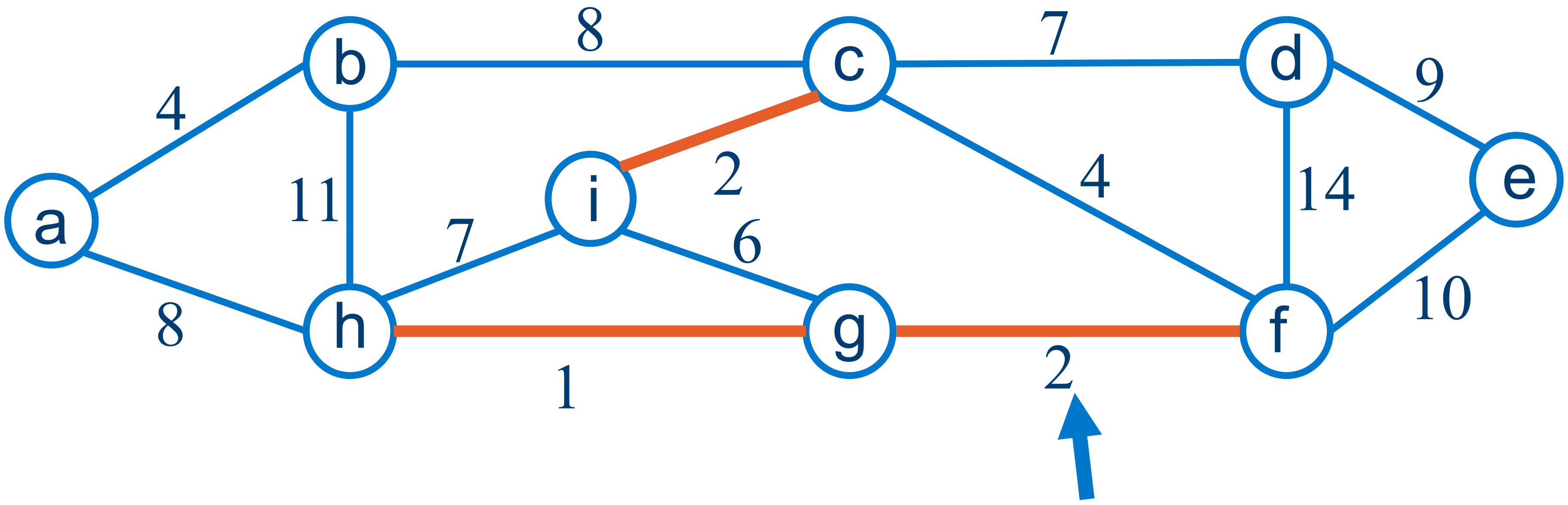
Kruskal's Algorithm



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

{c} ∪ {i}

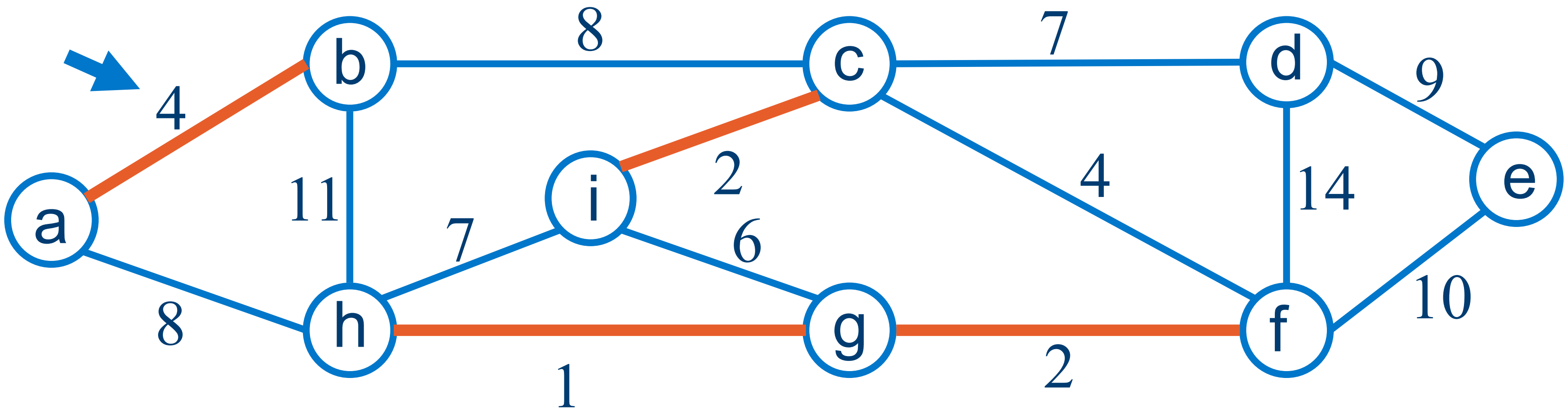
Kruskal's Algorithm



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

{g, h} ∪ {f}

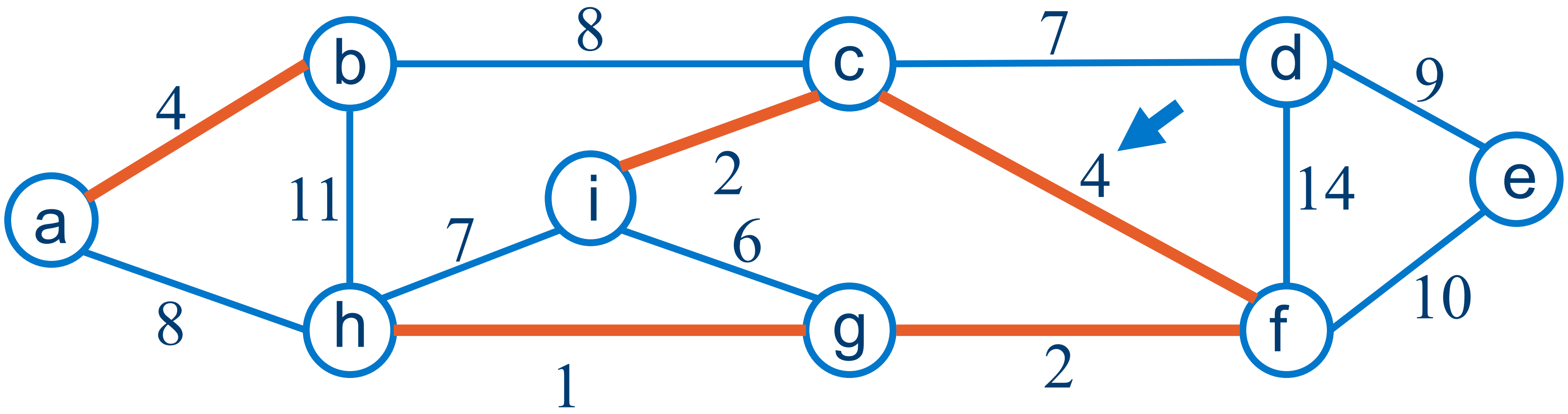
Kruskal's Algorithm



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

{a} ∪ {b}

Kruskal's Algorithm

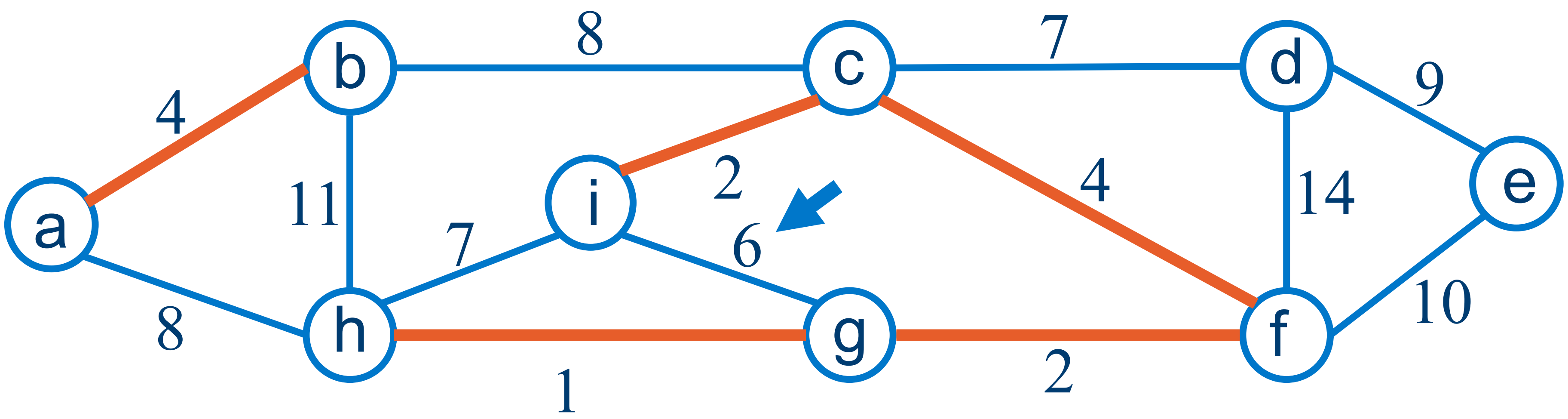


{a, b}, {d}, {e}, {c, f, g, h, i }

⬆

{f, g, h} ∪ {c, i}

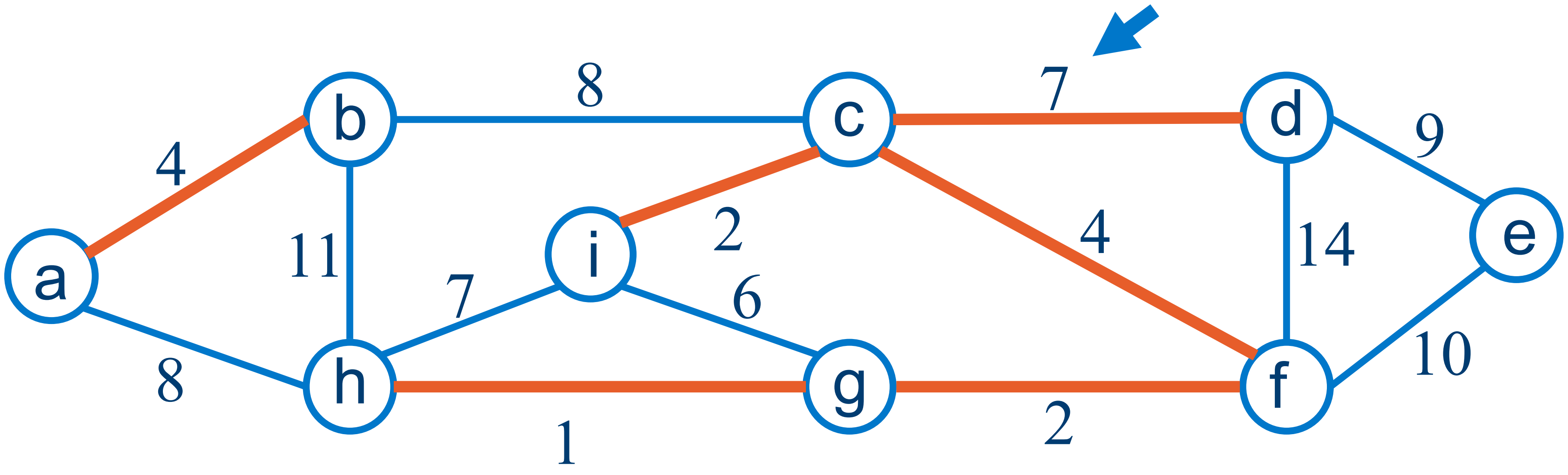
Kruskal’s Algorithm



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

$\text{FIND-SET}(u) = \text{FIND-SET}(v) \Rightarrow$ Reject !

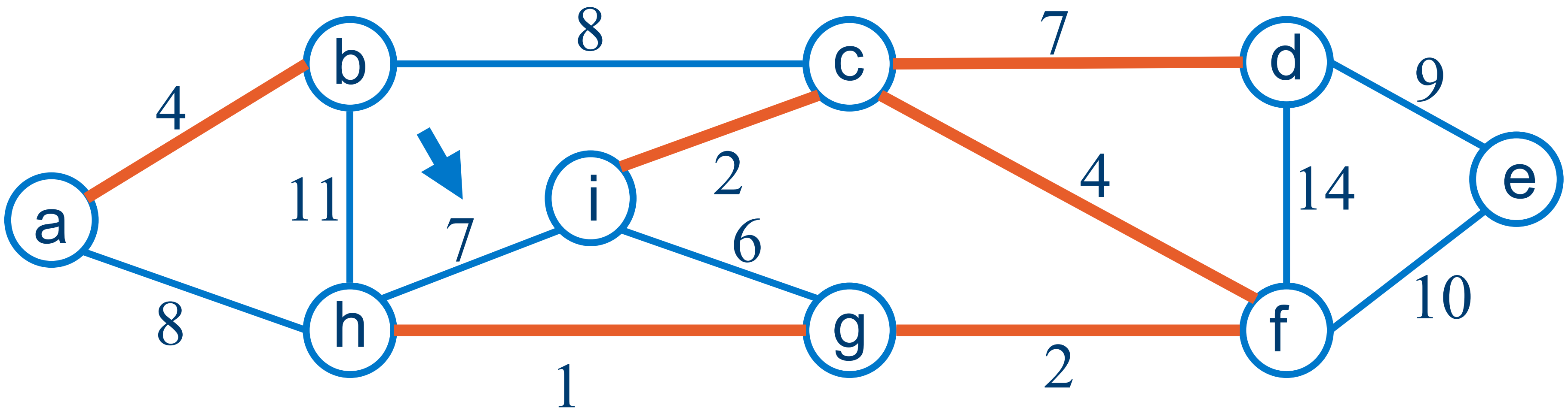
Kruskal's Algorithm



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

{c, f, g, h, i} ∪ {d}

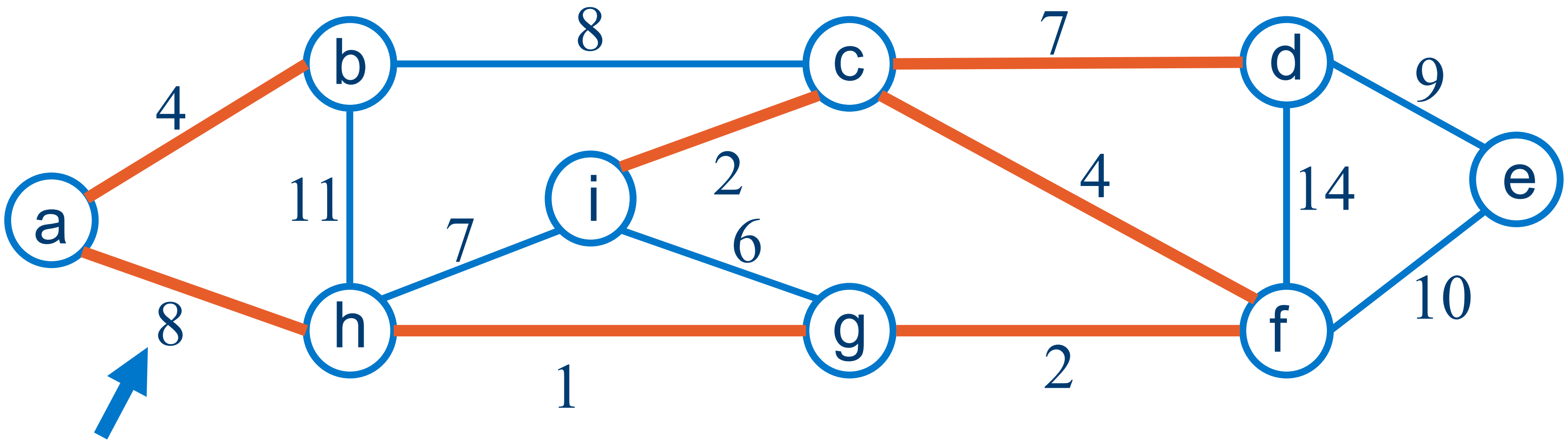
Kruskal's Algorithm



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

$\text{FIND-SET}(u) = \text{FIND-SET}(v) \Rightarrow$ Reject !

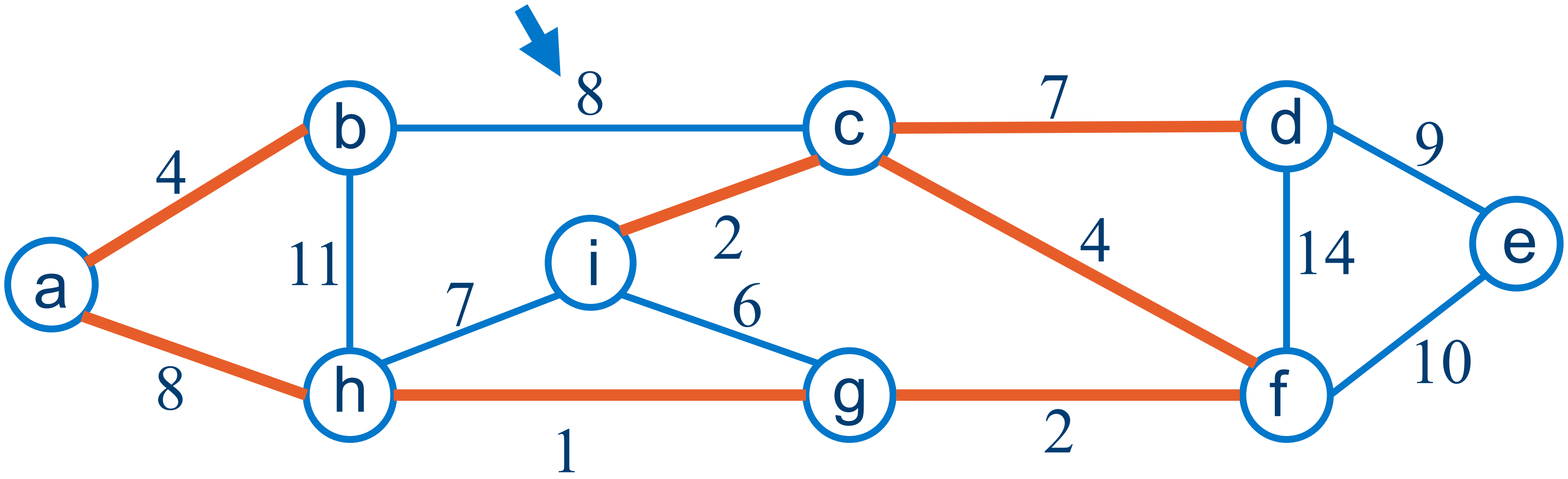
Kruskal’s Algorithm



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

{a, b} ∪ {c, d, f, g, h, i }

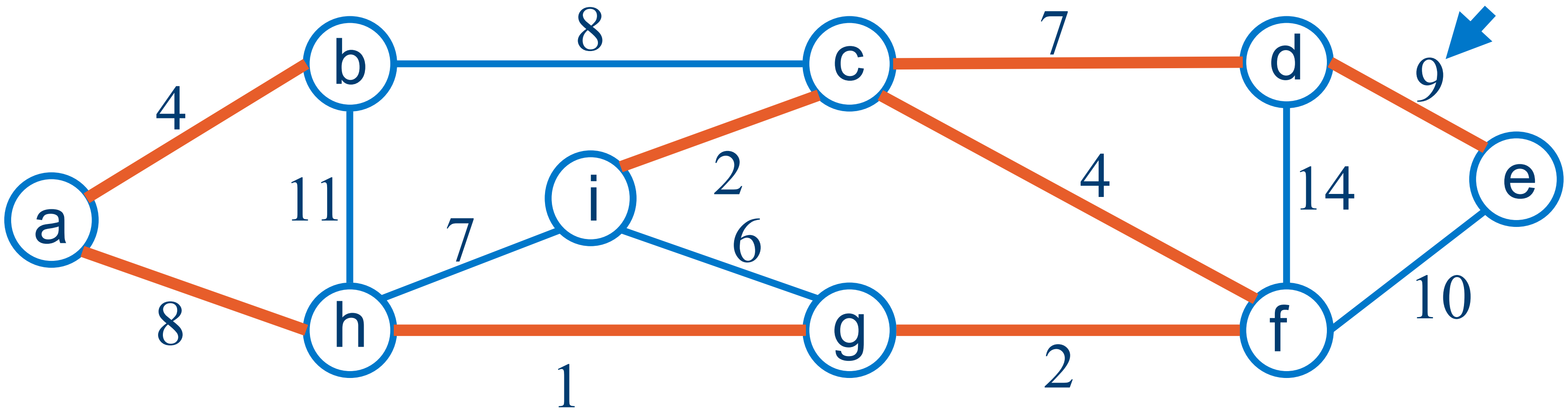
Kruskal’s Algorithm



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

$$\text{FIND-SET}(u) = \text{FIND-SET}(v) \Rightarrow$$ Reject !

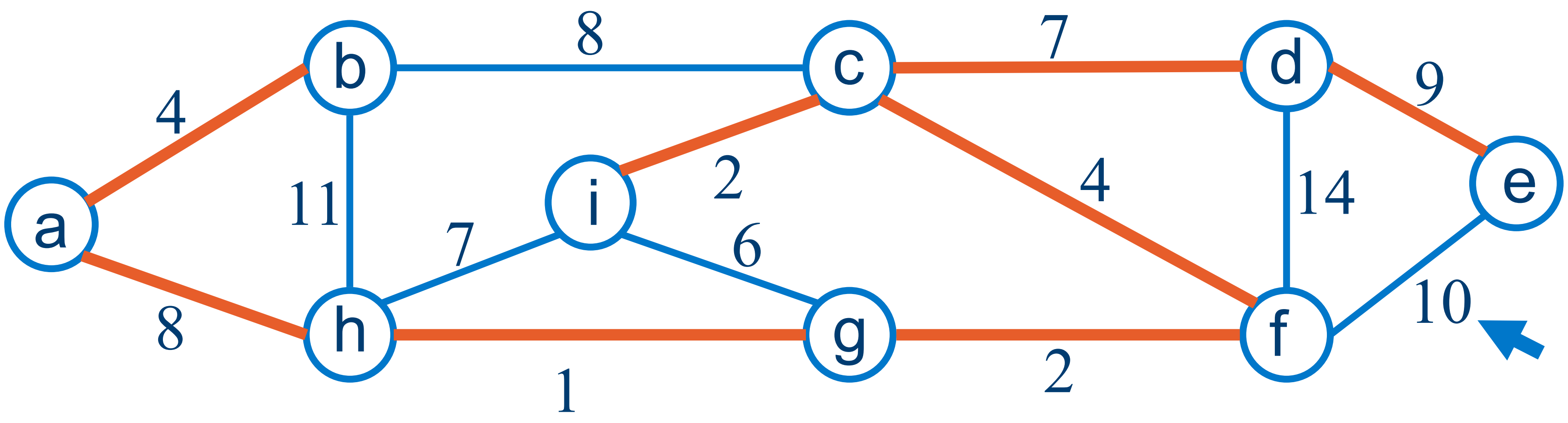
Kruskal’s Algorithm



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

{a, b, c, d, f, g, h, i } ∪ {e}

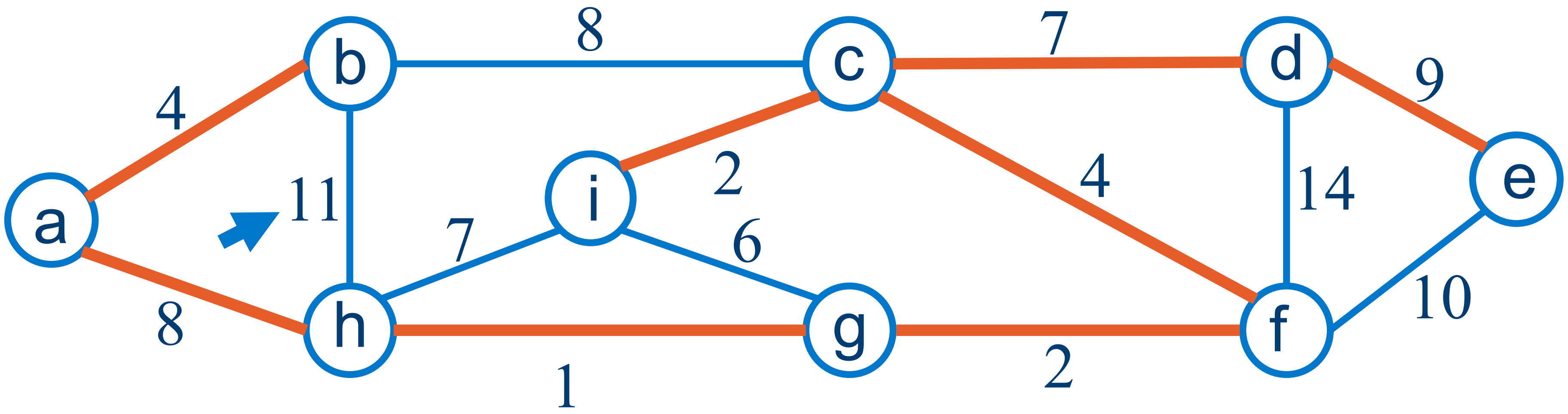
Kruskal's Algorithm



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

$\text{FIND-SET}(u) = \text{FIND-SET}(v) \Rightarrow$ Reject !

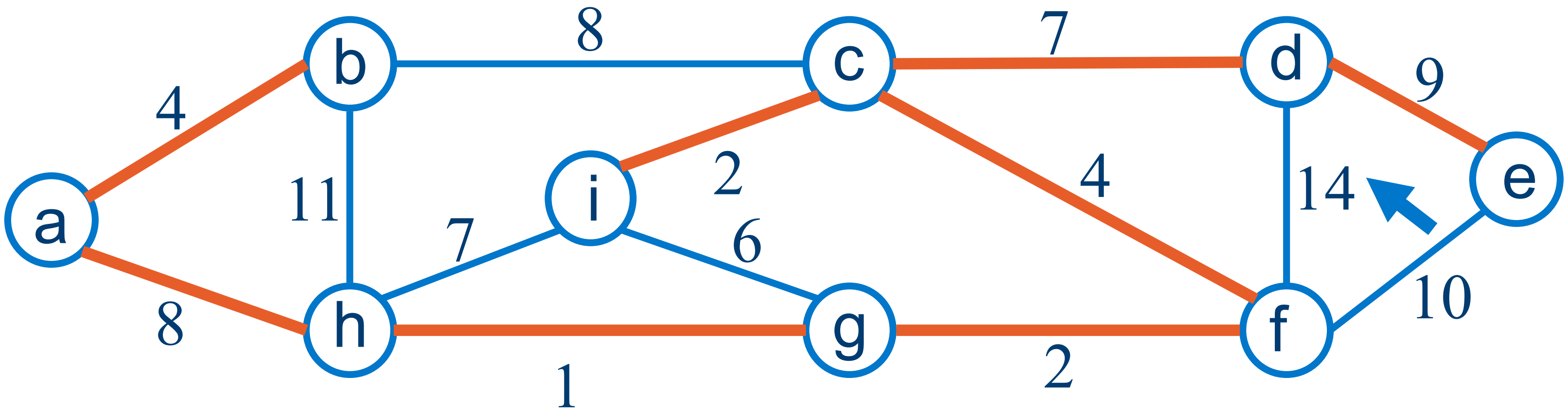
Kruskal's Algorithm



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

$\text{FIND-SET}(u) = \text{FIND-SET}(v) \Rightarrow$ Reject !

Kruskal's Algorithm



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

$\text{FIND-SET}(u) = \text{FIND-SET}(v) \Rightarrow$ Reject !

Kruskal’s algorithm

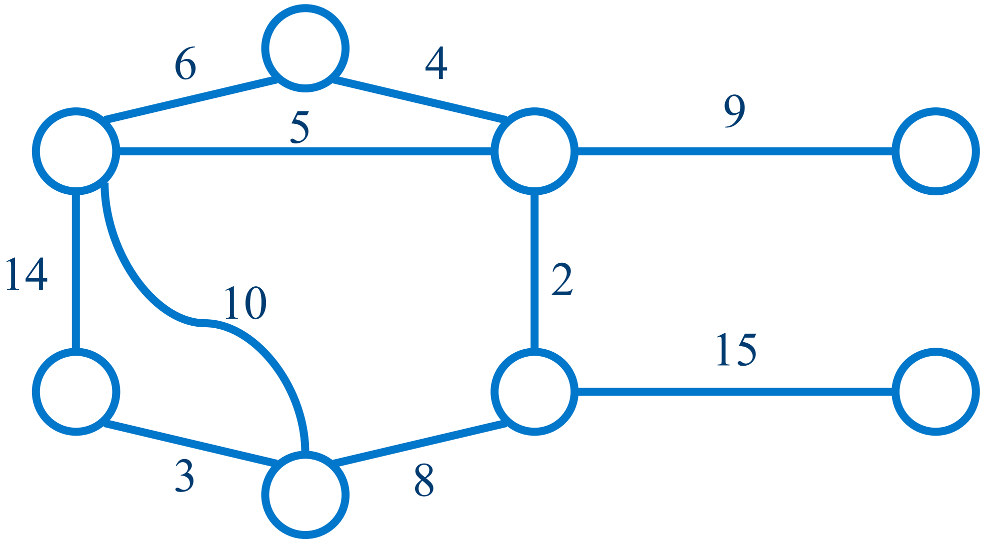
```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Time Complexity

$$O(E \log V)$$

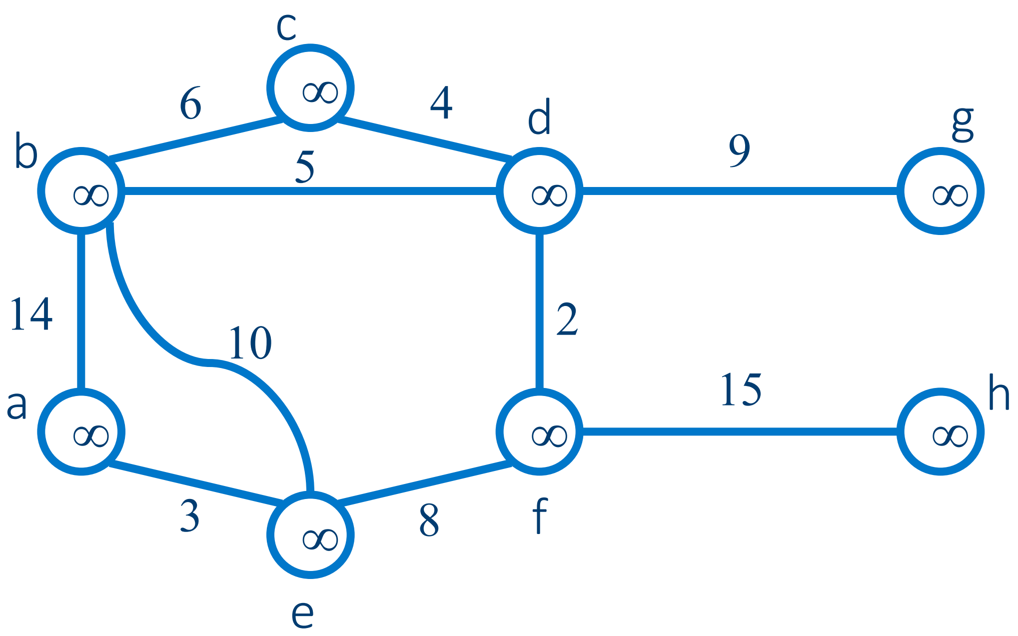
Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Prim's Algorithm

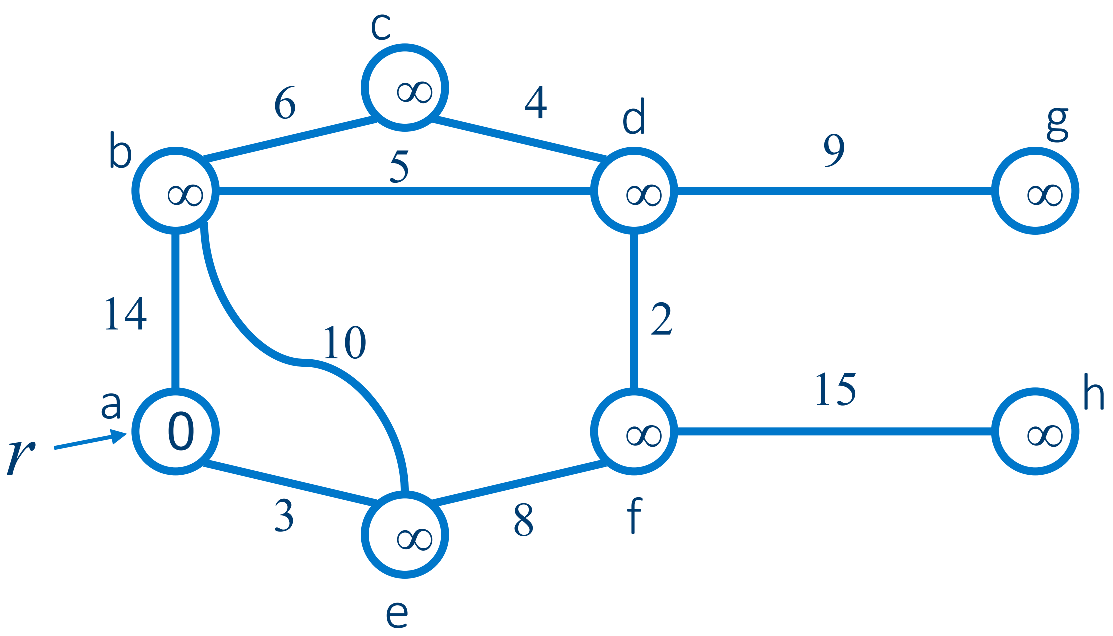
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Q: a(∞) b(∞) c(∞) d(∞) e(∞) f(∞) g(∞) h(∞)

Prim’s Algorithm

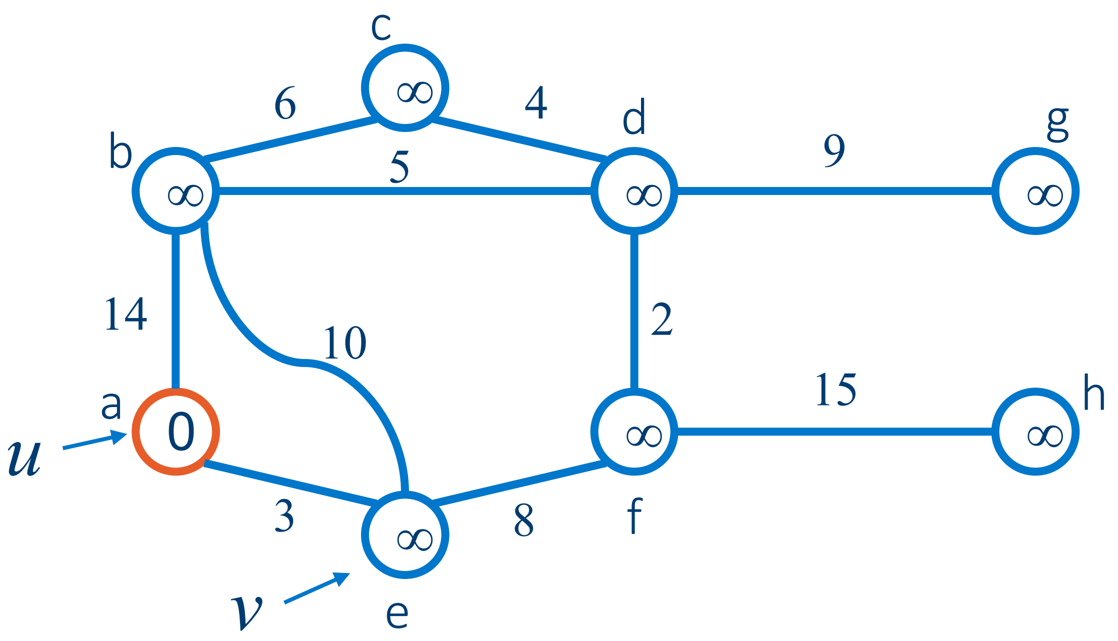
```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



Q: a(0) b(∞) c(∞) d(∞) e(∞) f(∞) g(∞) h(∞)

Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

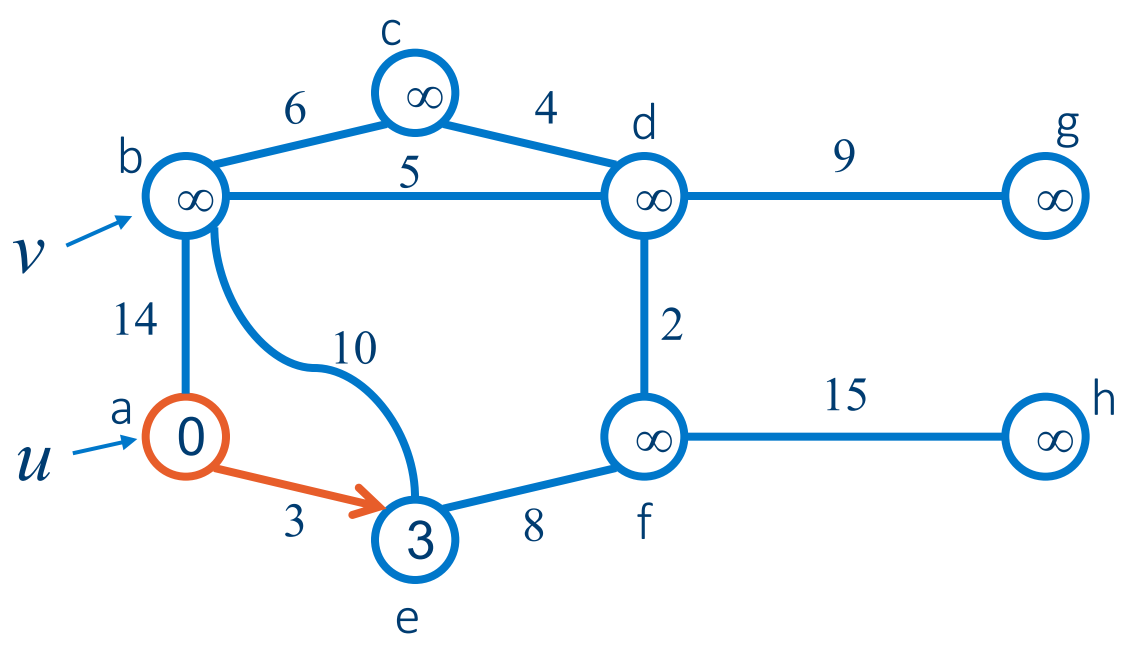


Q: ~~a(0)~~ b(∞) c(∞) d(∞) e(3) f(∞) g(∞) h(∞)

After update

Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

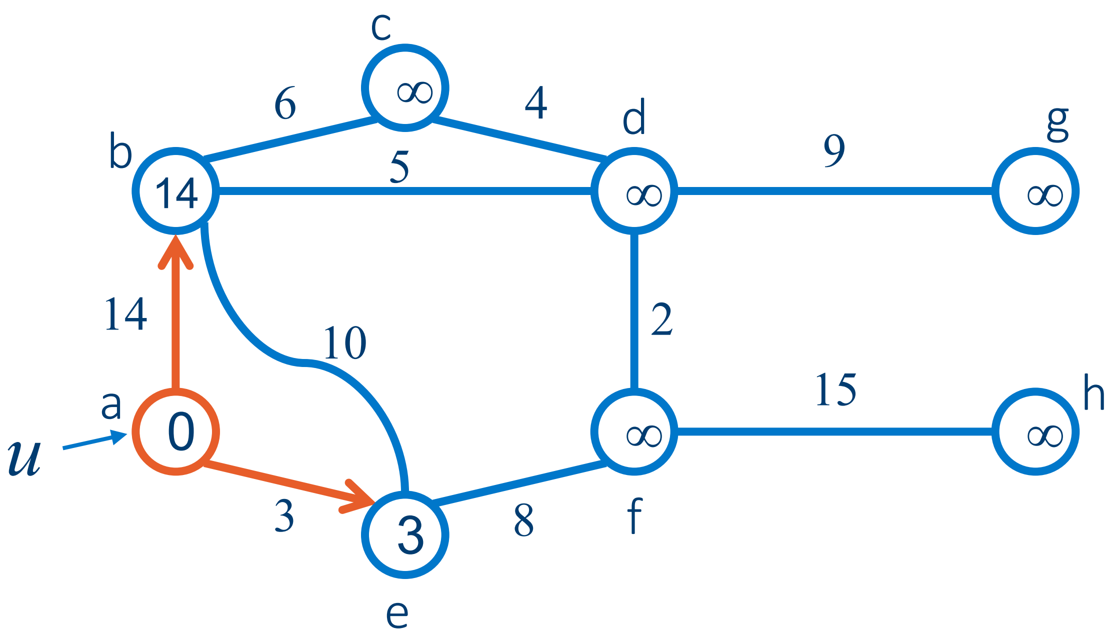


Q: ~~a(0)~~ b(14) c(∞) d(∞) e(3) f(∞) g(∞) h(∞)

After update

Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

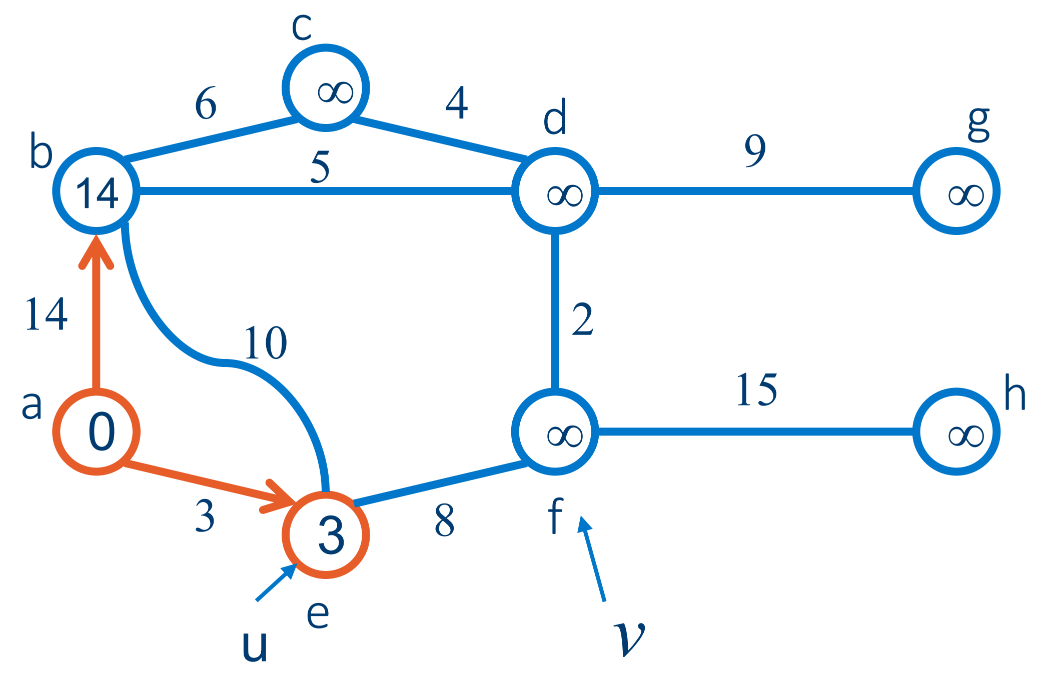


Q: ~~a(0)~~ b(14) c(∞) d(∞) e(3) f(∞) g(∞) h(∞)

Prim’s Algorithm

MST-PRIM(G, w, r)

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

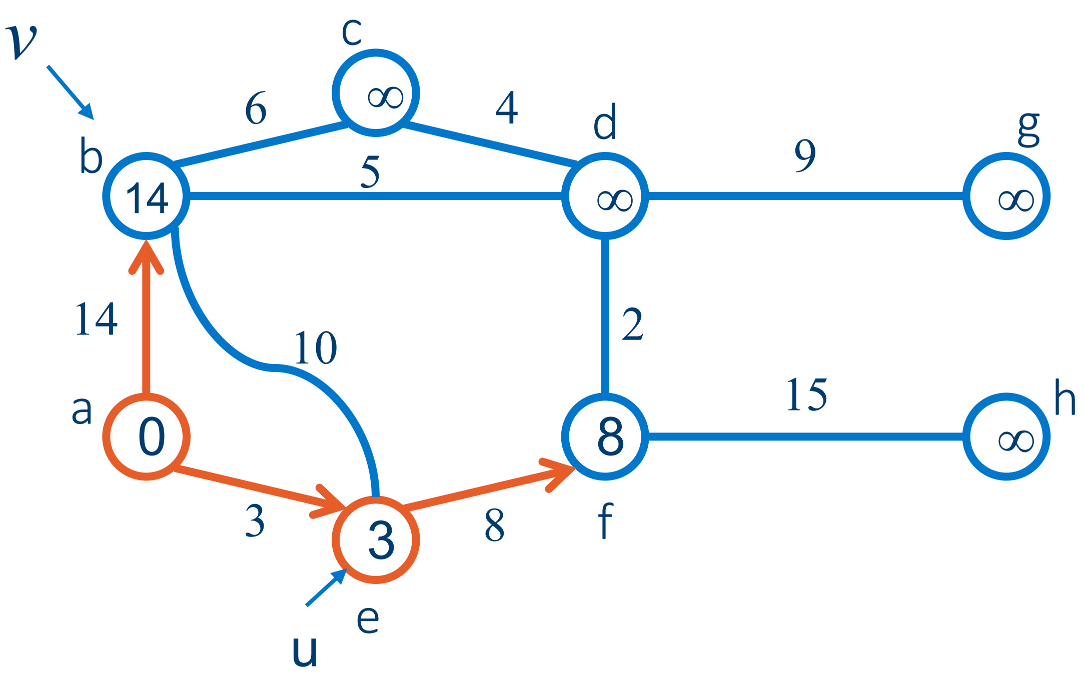


Q: ~~a(0)~~ b(14) c(∞) d(∞) ~~e(3)~~ f(8) g(∞) h(∞)

After update

Prim’s Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```

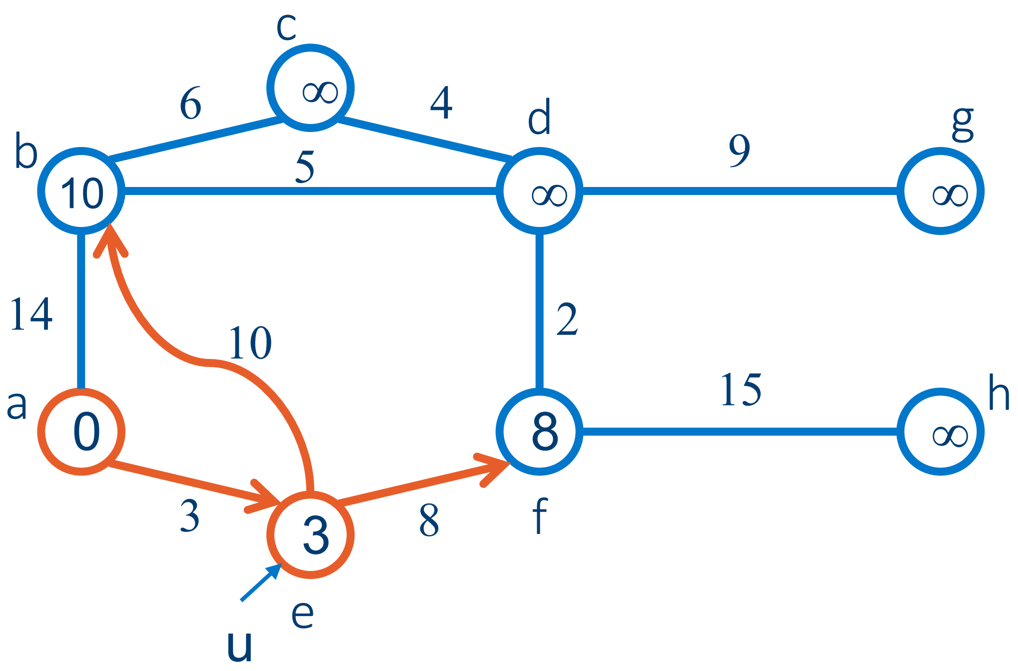


Q: ~~a(0)~~ b(10) c(∞) d(∞) ~~e(3)~~ f(8) g(∞) h(∞)

After update

Prim’s Algorithm

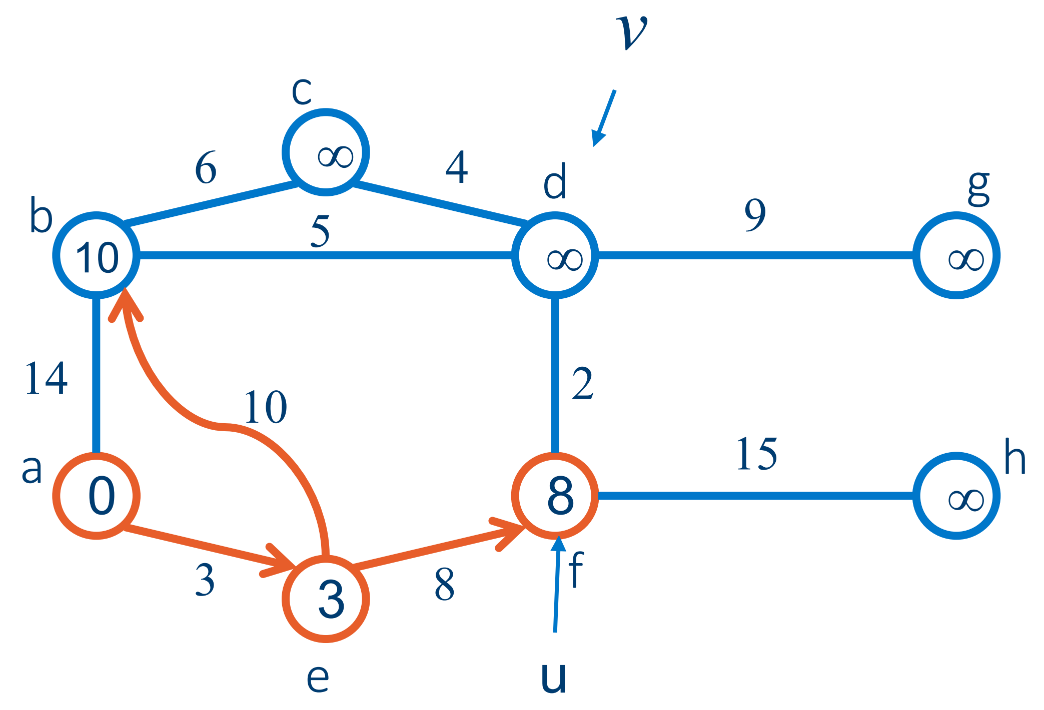
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Q: ~~a(0)~~ b(10) c(∞) d(∞) ~~e(3)~~ f(8) g(∞) h(∞)

Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



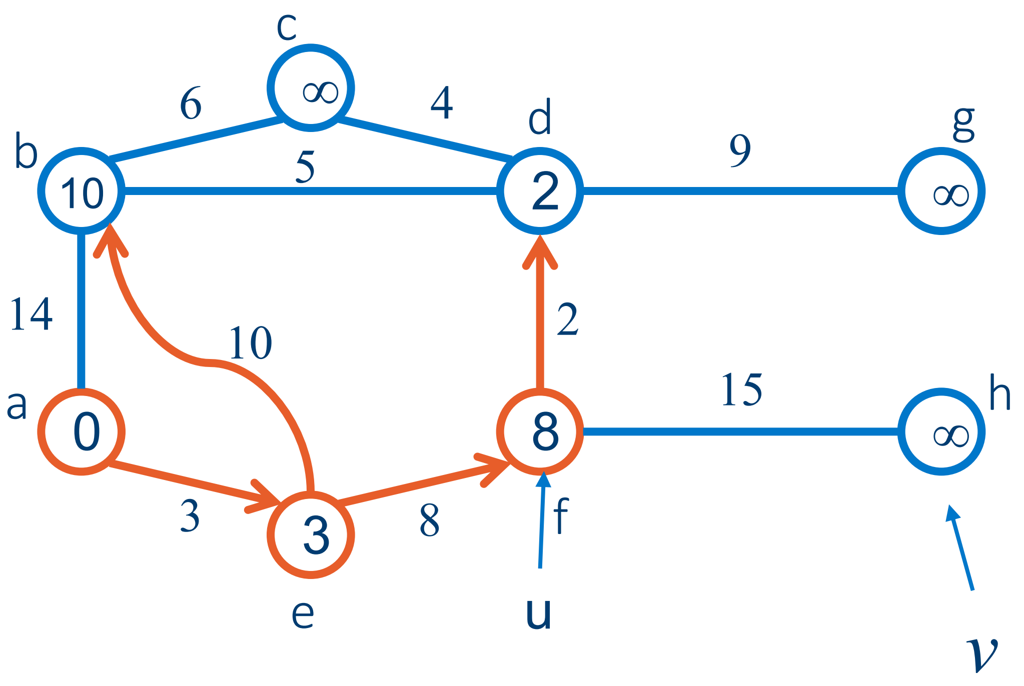
Q: ~~a(0)~~ b(10) c(∞) d(2) ~~e(3)~~ ~~f(8)~~ g(∞) h(∞)

After update

Prim's Algorithm

MST-PRIM(G, w, r)

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



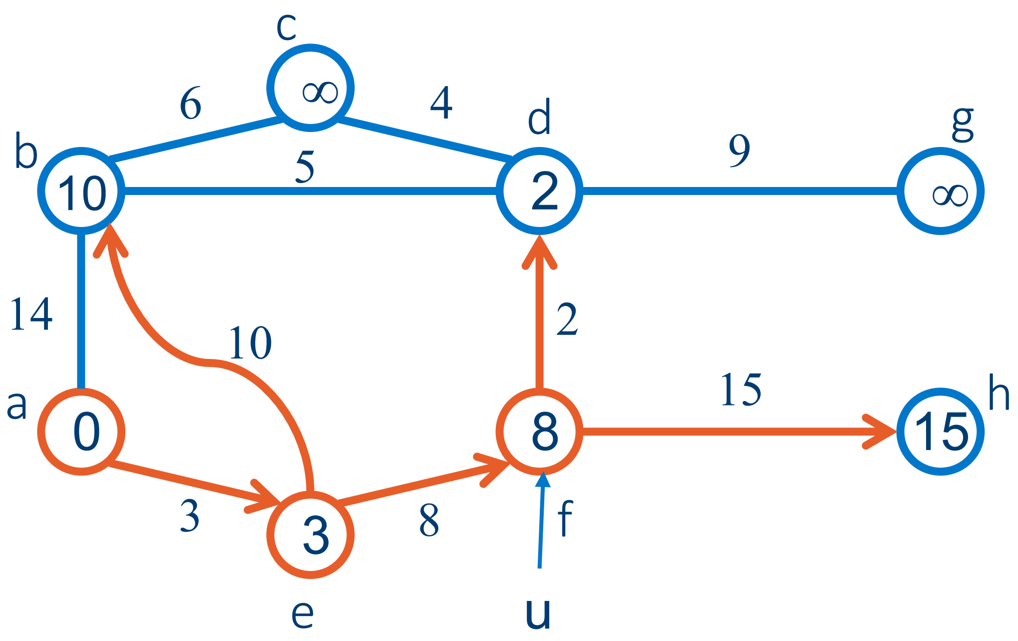
Q: ~~a(0)~~ b(10) c(∞) d(2) ~~e(3)~~ ~~f(8)~~ g(∞) h(15)

After update

Prim's Algorithm

MST-PRIM(G, w, r)

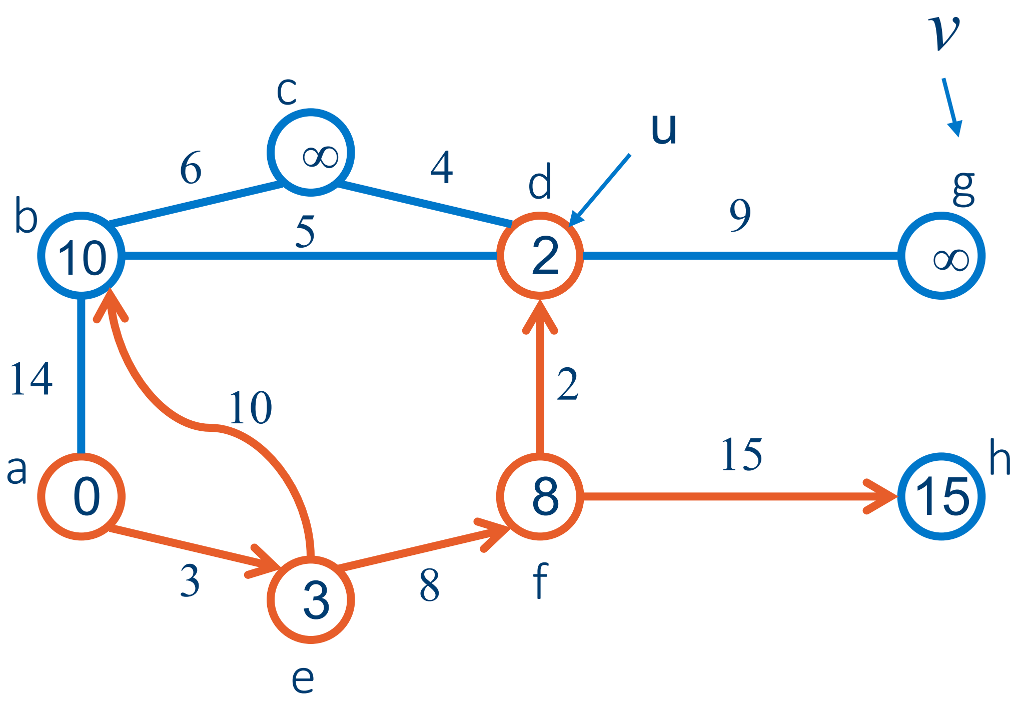
```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Q: ~~a(0)~~ b(10) c(∞) d(2) ~~e(3)~~ ~~f(8)~~ g(∞) h(15)

Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



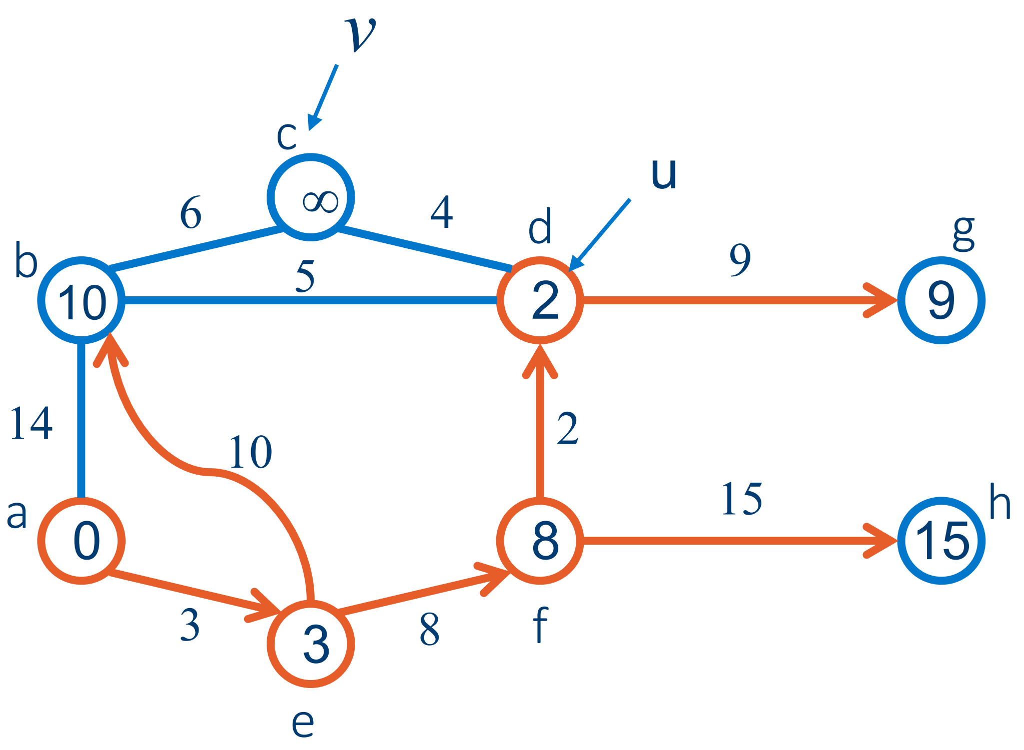
Q: ~~a(0)~~ b(10) c(∞) ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ g(9) h(15)

After update

Prim's Algorithm

MST-PRIM(G, w, r)

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

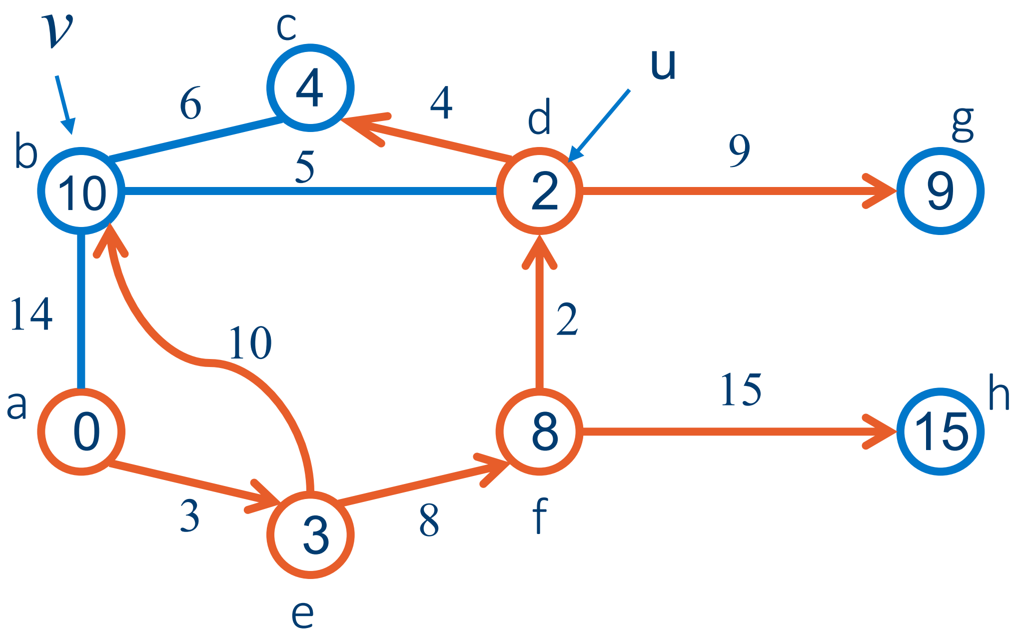


Q: ~~a(0)~~ b(10) c(4) ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ g(9) h(15)

After update

Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

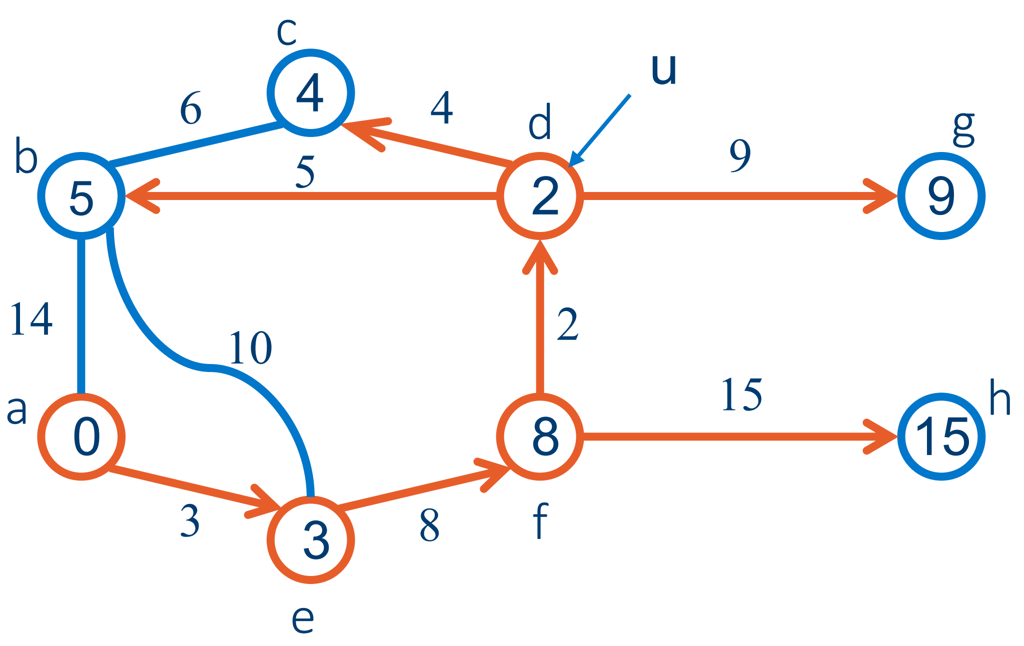


Q: ~~a(0)~~ b(5) c(4) ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ g(9) h(15)

After update

Prim’s Algorithm

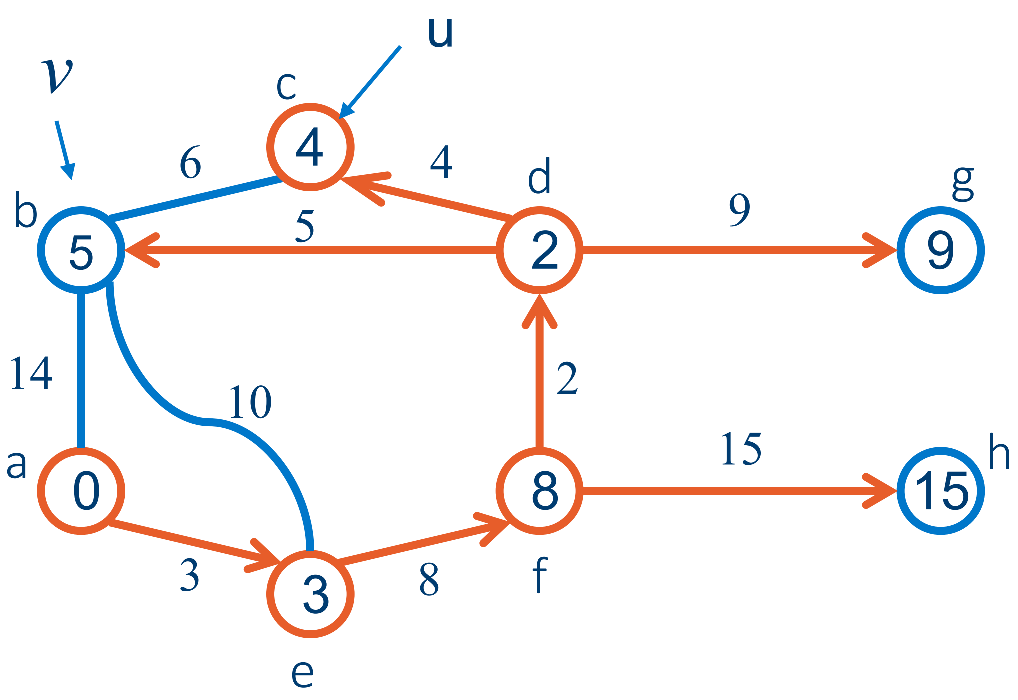
```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```



Q: ~~a(0)~~ b(5) c(4) ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ g(9) h(15)

Prim's Algorithm

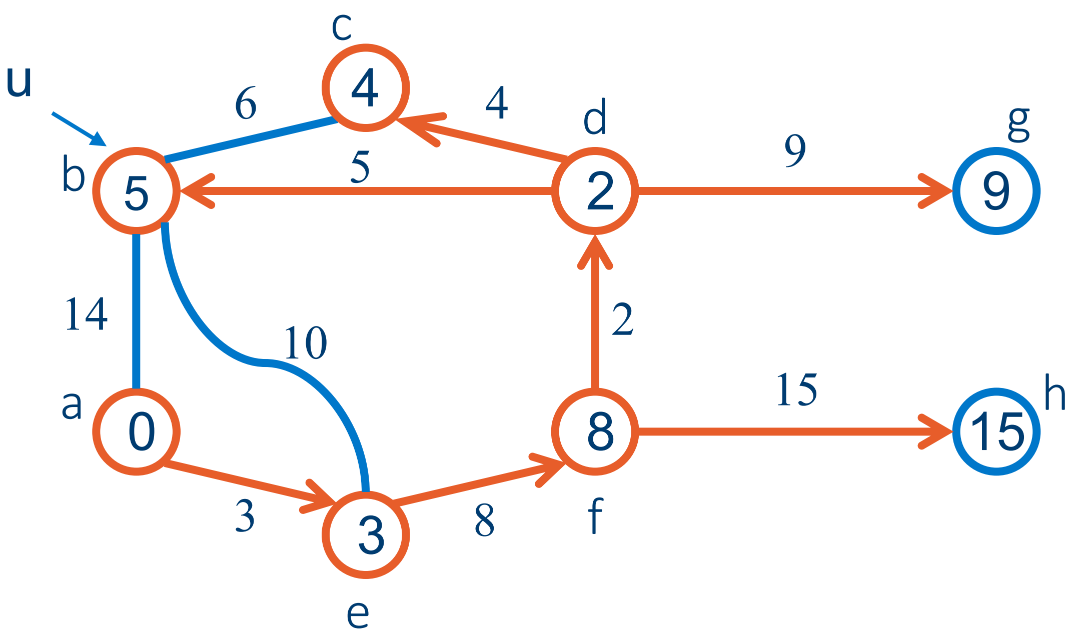
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Q: ~~a(0)~~ b(5) ~~c(4)~~ ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ g(9) h(15)

Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

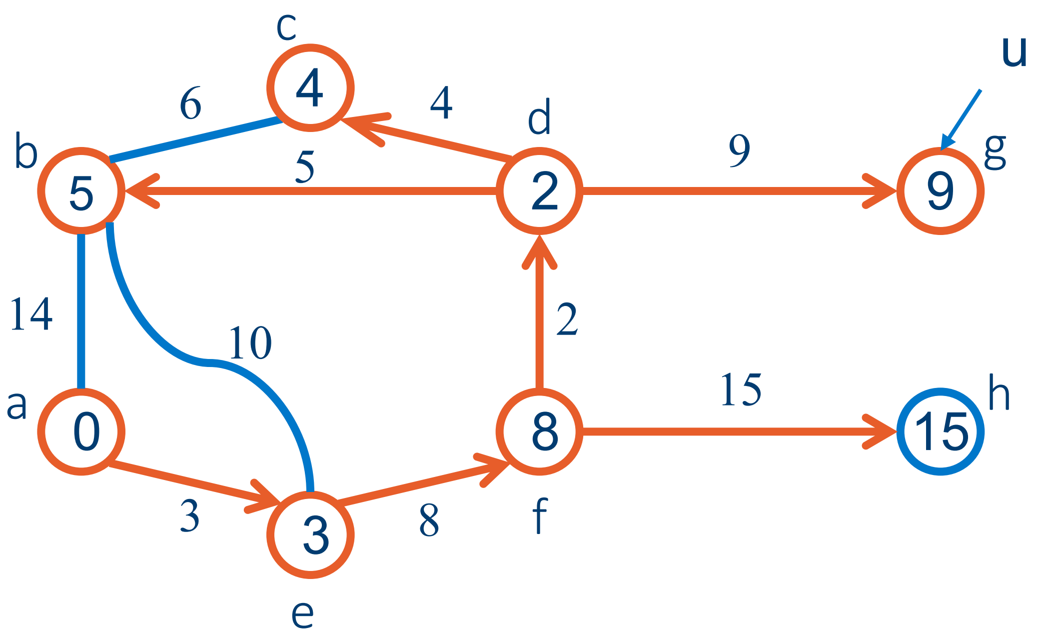


Q: ~~a(0)~~ ~~b(5)~~ ~~c(4)~~ ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ g(9) h(15)

Prim's Algorithm

MST-PRIM(G, w, r)

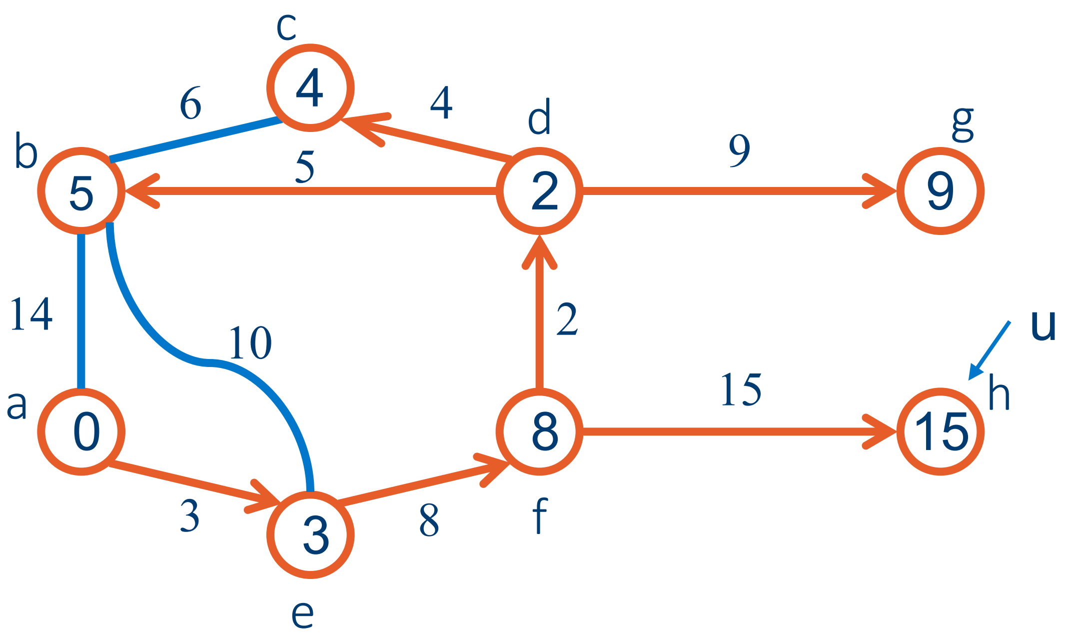
```
for each u ∈ G.V
        u.key = ∞
        u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
                if v ∈ Q and w(u, v) < v.key
                        v.π = u
                        v.key = w(u, v)
```



Q: ~~a(0)~~ ~~b(5)~~ ~~c(4)~~ ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ ~~g(9)~~ h(15)

Prim's Algorithm

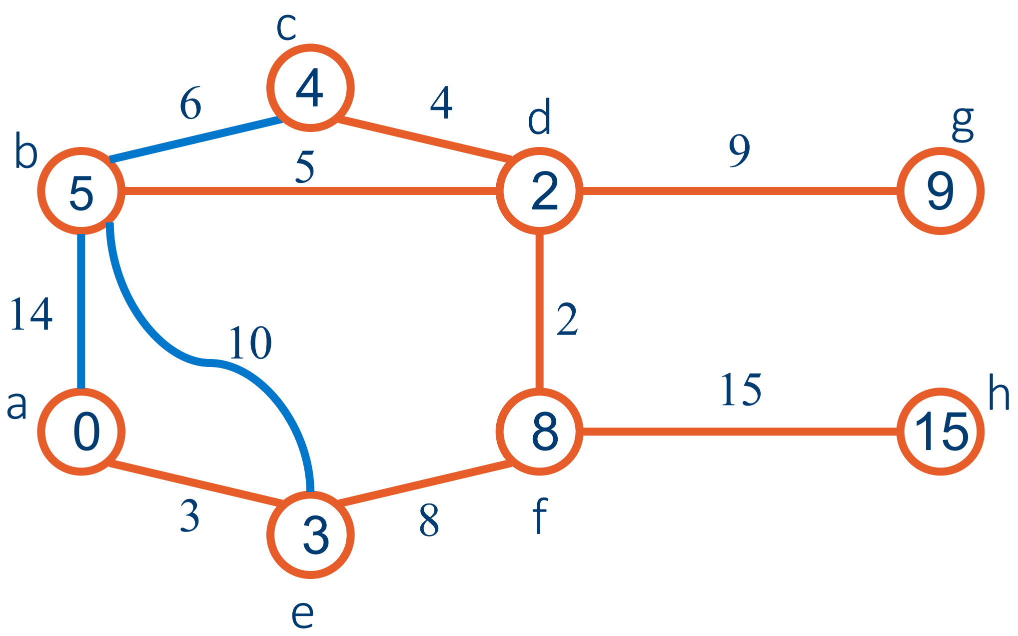
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Q: ~~a(0)~~ ~~b(5)~~ ~~c(4)~~ ~~d(2)~~ ~~e(3)~~ ~~f(8)~~ ~~g(9)~~ ~~h(15)~~

Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Prim's Algorithm

MST-PRIM(G, w, r)

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

$O(V)$

$O(V)$

$O(V \log V)$

$O(E \log V)$

Involves an implicit DECREASE-KEY operation on the min-heap, which a binary min-heap supports in O(log V)

Time Complexity

$O(E \log V)$

Wrap-up

- We learned some interesting applications of DFS:
  - Topological Sort
  - Strongly Connected Components
- Minimum Spanning Tree
  - Kruskal
  - Prim