Lecture #22: Graph

School of Computer Science and Engineering
Kyungpook National University (KNU)

Woo-Jeoung Nam

KNU KYUNGPOOK NATIONAL UNIVERSITY

Agenda

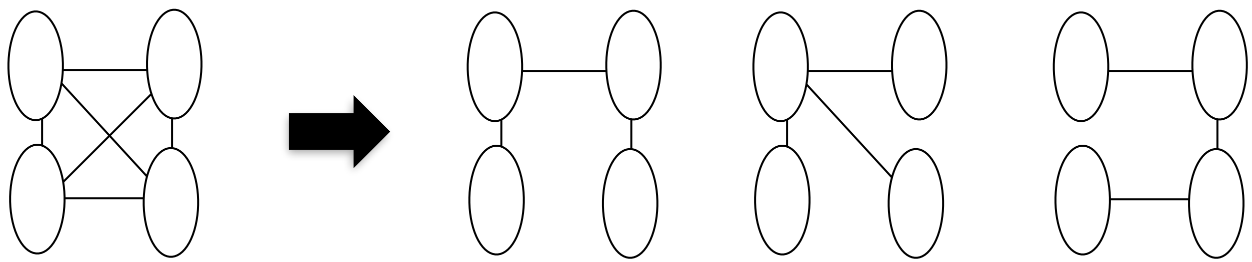
- Graph elementary
  - Definitions & Terminologies
  - Properties
  - Connected components
- Graph algorithms
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Minimum Spanning Trees (MST)
    - Prims & Kruskals
  - Shortest Paths and Transitive Closure
    - Dijkstra's algorithm

Spanning Trees

- When the Graph G is connected, a DFS(v) or BFS(v) starting at any vertex V visits all the vertices in G
- Implicitly, the search operation partitions the edges in G into two sets: Tree-edges (T) and non-tree edges (N)
- T is the set of edges used or traversed during searching and N is the set of remaining edges
- We can determine the set of tree edges by adding a statement to the if clause of DFS() or BFS() that inserts the edge (v,w) into a linked list of edges.
- Let T be the head of this linked list.
- The edges in T form a tree that includes all vertices of G.

Spanning Trees – Examples

- A spanning tree is any tree that consists of edges in G and that include all the vertices in G
  1. if we add a non-tree edge into a spanning tree → cycle
  2. spanning tree is a minimal subgraph, G’, of G such that V(G)=V(G’) and G’ is connected



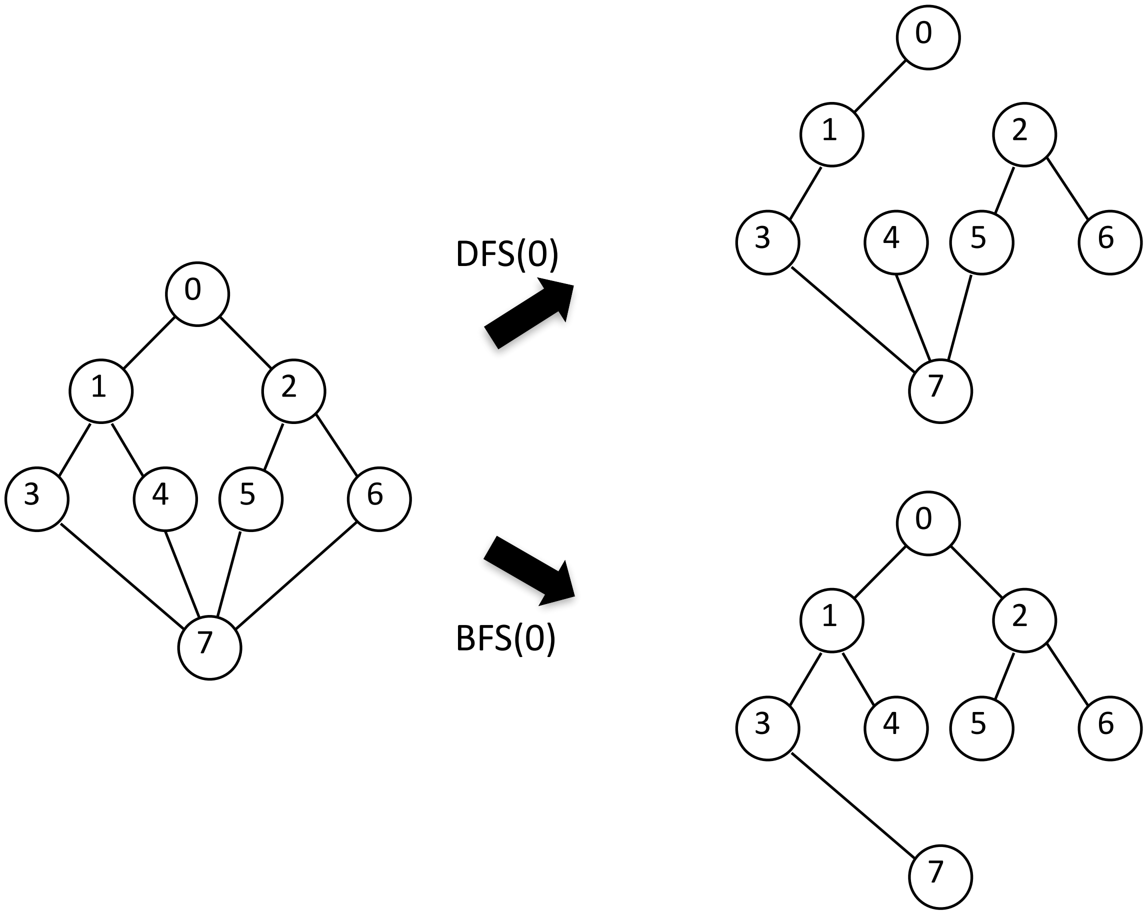
A complete graph G

Three spanning trees of graph G

- Hence, we can use BFS() to get breadth first spanning tree or DFS() to create depth first spanning tree

KNU KYUNGPOOK NATIONAL UNIVERSITY

Spanning Trees – Examples



Two Major Properties of Spanning Tree

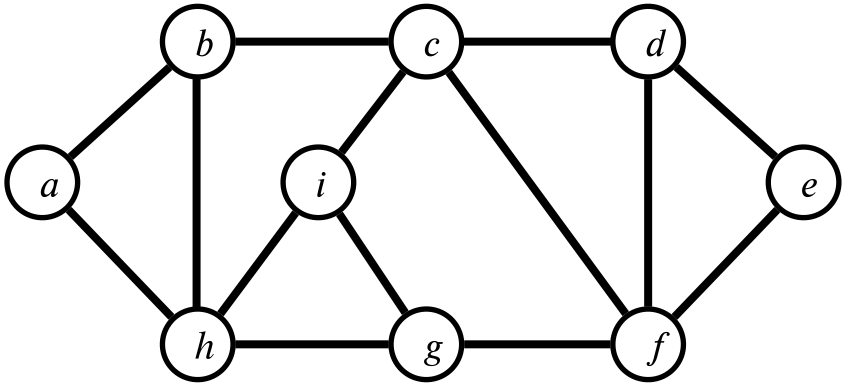
1. If we add a non-tree edge, (v,w), into a spanning tree, we get a cycle that consists of the edge (v,w) and all the edges on the path from w to v in T.
   - For example, if we add non-tree edge (7,6) to DFS spanning tree ,the resulting cycle will be 7,6,2,5,7.
   - This property is used to get independent set of circuit equations for a given electrical network.

minimal "subgraph" V(G)=V(G')

2. A spanning tree is a minimal subgraph(G') of G such that V(G)=V(G') and G' is connected. Hence, a minimal subgraph can be defined as one with the fewest number of edges.
   - Any connected graph with n vertices must have at least n-1 edges and all the connected graphs with n-1 edges are trees. Hence a spanning three has n-1 edges.
   - Constructing minimal subgraphs is applicable in the design of communication networks. The minimal number of links needed to connect n cities is n-1.
   - Constructing the spanning trees of G help us to select the one with lowest total cost or the lowest overall length by using weighted graph.

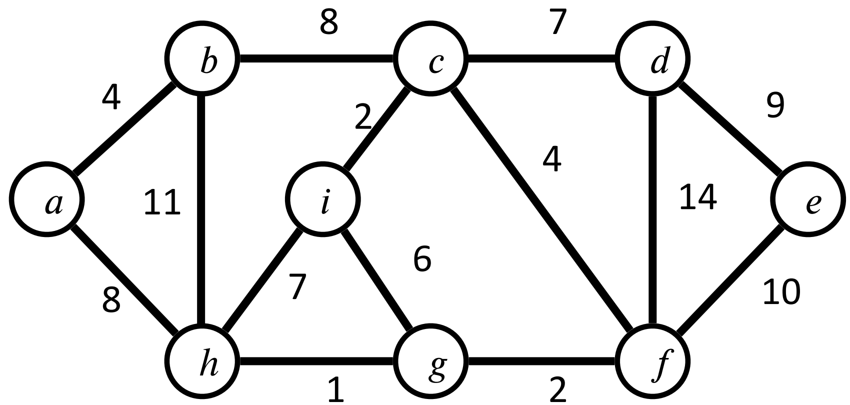
Minimum Spanning Trees

- Given n pins, and n-1 wires, how could we connect all the pins?
- Simple solution: Use all wires and connect them whenever possible until a combination is found where all vertices are connected
- Below, all possible connections between pins are shown



Minimum Spanning Trees (MST)

- Given n pins, and n-1 wires, how could we connect all the pins?
- Now, each possible connection has a cost. Since resource is limited, choose the best combination of connections that minimizes the cost (optimization problem)



Minimum Spanning Trees – Definition

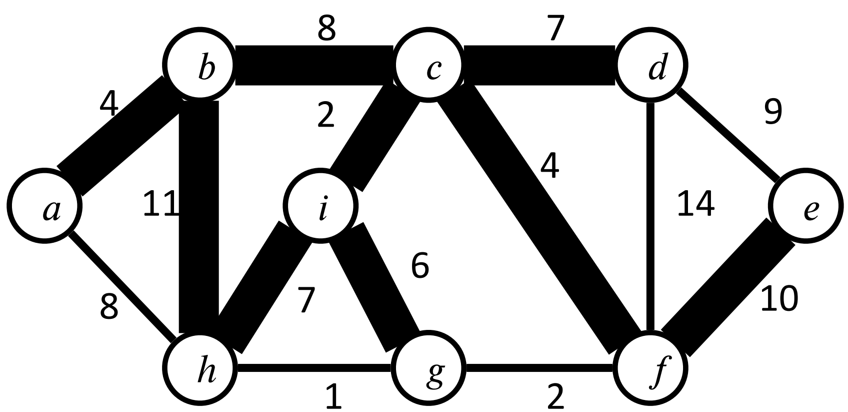
- Given an undirected graph G=(V, E), where V is the set of pins, E is the set of possible connections, and for each edge (u,v)∈E, we have a weight w(u,v) that is the cost for connecting u and v
- We want to find an acyclic subset $T \subseteq E$ that connects all of the vertices whose total weight w(T) is minimized
  - Minimize $w(T) = \sum_{(u,v)\in T} w(u,v)$

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

- Observation
  - Since T is acyclic, it must form a tree, which we call a spanning tree
  - Finding T is called the minimum (cost) spanning tree problem
  - The Kruskal's algorithm and Prim's algorithm is able to solve the minimum spanning tree problem

Recap – BFS/DFS vs Minimum Spanning Tree

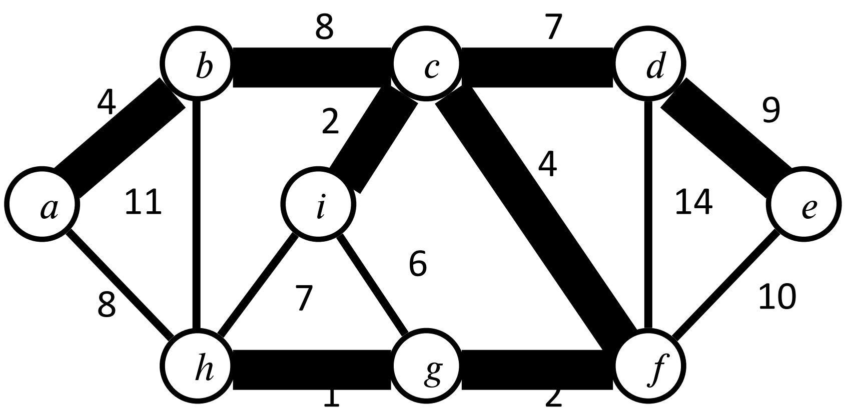
BFS(c)



Cost=59

Recap – BFS/DFS vs Minimum Spanning Tree

MST



Cost=37

Growing a Minimum Spanning Tree (MST)

- MST is grown by one edge at a time, which is maintained by a set of edges A
- A is grown while satisfying the following loop invariant
  - Prior to each iteration, A is a subset of some MST

- At each step, we determine an edge (u,v) that we can add to A without violating the invariant so that $A \cup \{(u, v)\}$ is also a subset of the MST → such edge (u,v) is called a safe edge

$A = \emptyset$

while "A does not form a spanning tree,"

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

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

```
GENERIC-MST(G, w)
  A = Ø
  while A does not form a spanning tree
      find an edge (u, v) that is safe for A
      A = A ∪ {(u, v)}
  return A
```

→ Tricky part

→ When returned, A must be a MST (guaranteed)

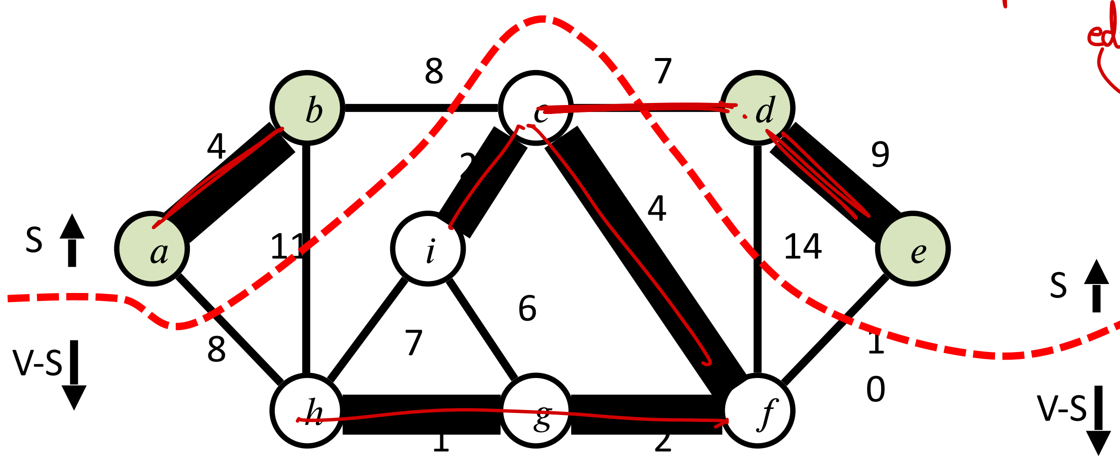
Some notions – cut, cross, light edge

- A cut (S, V-S) of an undirected graph G=(V, E) is a partition of V

"u∈S and v∈(V−S)"

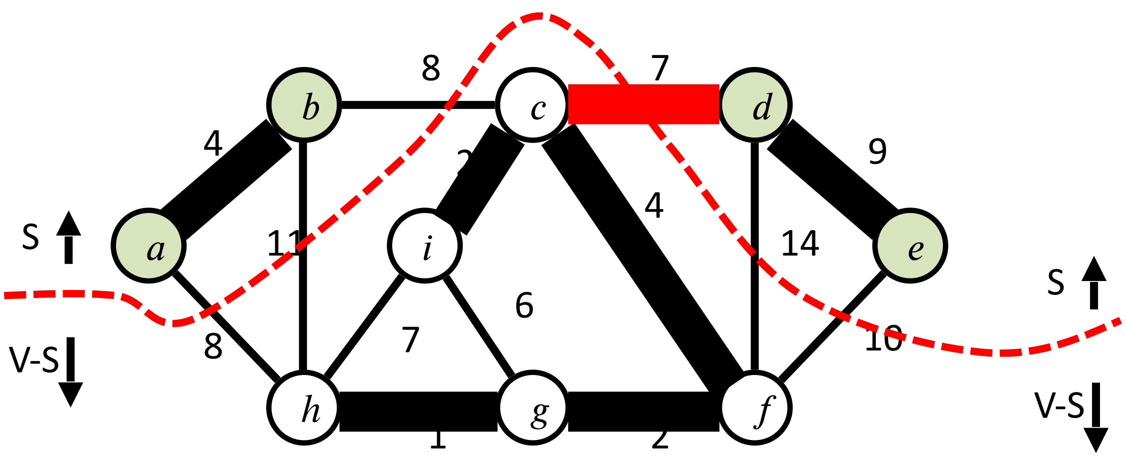
- An edge (u,v) **crosses** the cut (S, V-S) if $u \in S$ *and* $v \in (V - S)$
- The cut (S, V-S) **respects** A if no edge in A crosses the cut
- An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut
    - 교차하는 간선 중 가중치가 최소면
- **A light edge is a safe edge**

light edge는 cut을 지나는 가장 작은 가중치를 가지는 edge를 말함



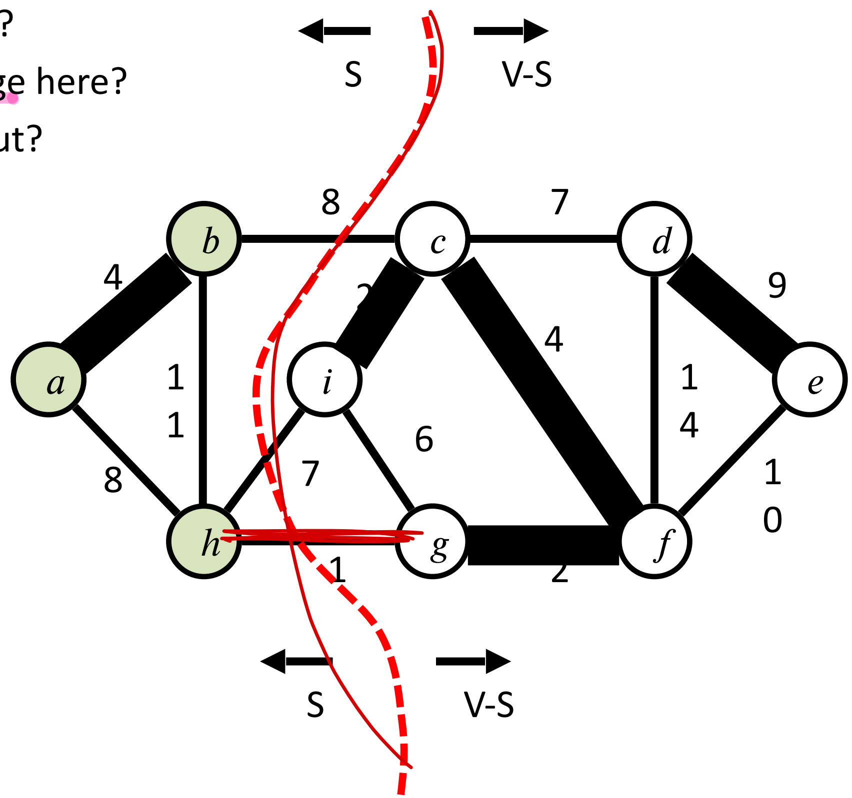
Some notions – cut, cross, light edge

- A cut (S, V-S) of an undirected graph G=(V, E) is a partition of V
- An edge (u,v) crosses the cut (S, V-S) if $u \in S$ *and* $v \in (V - S)$
- The cut (S, V-S) respects A if no edge in A crosses the cut
- An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut
- **A light edge is a safe edge (c,d)**



Some notions – cut, cross, light edge

- Does the cut matter?
- What is the light edge here?
- How to define the cut?



Minimum Cost Spanning Tree

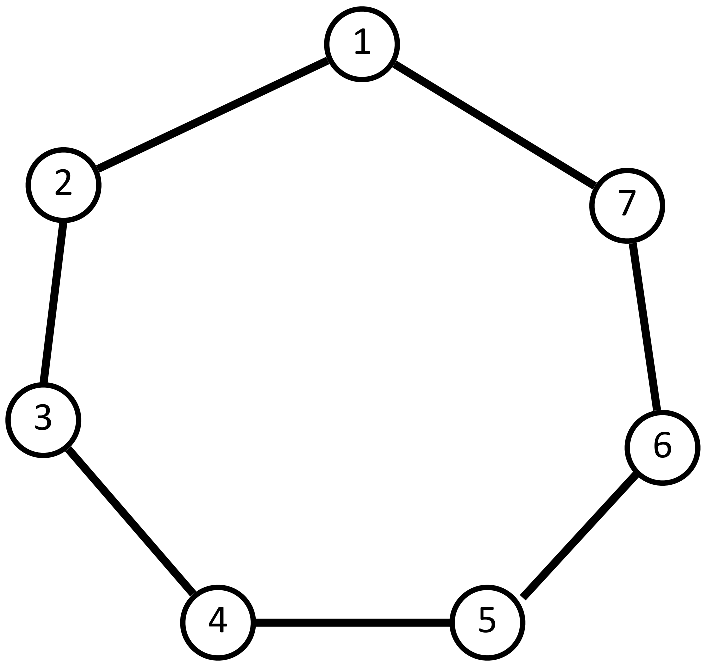
- Minimum cost spanning tree of a connected undirected graph
  - It is a spanning tree of least cost
  - Three greedy algorithm to compute it : Kruskal's, Prim's, and Sollin's algorithms
- In greedy method, we construct an optimal solution in stages
  - At each stage, make the best decision using some criterion (local optimum)
  - When the algorithm terminates, we hope that the local optimum is equal to the global optimum (not guaranteed)

→ 끝났으면 local optimum이 global optimum과 같은 경우일 것이다.

- Spanning tree construction constraints
  - Only use edges within the graph
  - Only use exactly n-1 edges
  - Spanning tree must be acyclic

KNU KYUNGPOOK NATIONAL UNIVERSITY

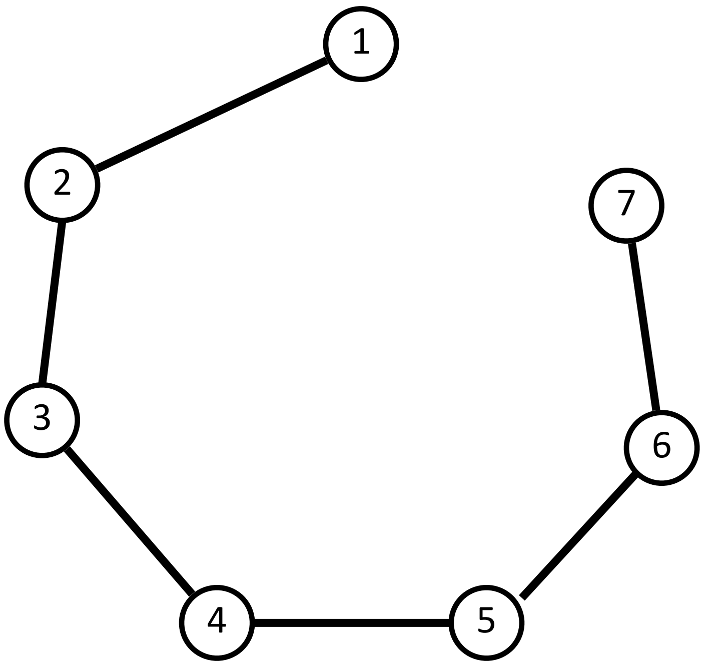
Using n-1 edges



Connected? Yes

Minimum Spanning Tree? No

Using n-1 edges



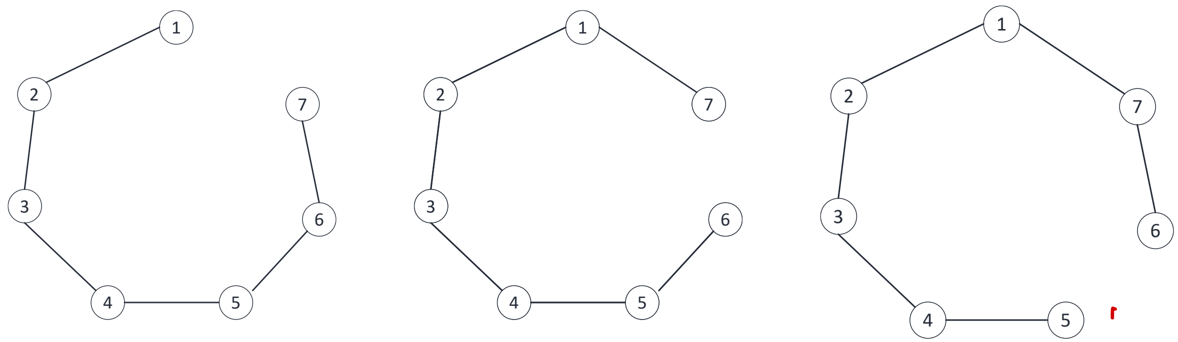
Connected? Yes
Minimum Spanning Tree? Yes

Min. # of edges = n-1, where n is the # of nodes

Using n-1 edges

Connected? Yes
Minimum Spanning Tree? Yes

How many MSTs can there be?



…

$$\partial = \binom{\Gamma}{\partial}$$

Minimum Cost Spanning Tree Algorithms

- Kruskal's algorithm
  - Joseph B. Kruskal, Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," Proceedings of the American Mathematical Society, pp. 48–50 ,1956

- Prim's algorithm
  - V. Jarník, "O jistém problému minimálním [About a certain minimal problem]," Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63. (in Czech)
  - R. C. Prim, "Shortest connection networks and some generalizations.," Bell System Technical Journal, 36 (1957), pp. 1389–1401

- Sollin's algorithm
  - It was first published in 1926 by Otakar Borůvka as a method of constructing an efficient electricity network for Moravia. The algorithm was rediscovered by Choquet in 1938; again by Florek, Łukasiewicz, Perkal, Steinhaus, and Zubrzycki in 1951; and again by Sollin in 1965. (http://wikipedia.org)

Quick comparison

- Problem: Given G=(V,E), find S=(V,T), where $T \subseteq E$ and $w(T)$ is minimum
- Kruskal's algorithm
  - The set A is a forest whose vertices is in T
  - The safe edge added to A is always a least-weighted edge in G that connects two distinct components (merges)

두 개의 다른 컴포넌트를 합침.

- Prim's algorithm
  - The set A forms a single tree
  - The safe edge added to A is always a least-weighted edge in G that connects the tree to a vertex not in the tree (grows)

Kruskal's Algorithm – The psuedocode

G.E 를 오름차순 정렬

```
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
```

- Initialize A
- Create |V| trees, where each tree contains only the vertex v (at the beginning each vertex is a tree)
- Order edges from low to high cost
- Check whether to merge two trees. If u and v belong to different trees then merge. Otherwise don't merge
- Merge two trees

for (u, v) ∈ G.E. take 오름차순

if Find-set(u) ≠ Find-Set(v)

A = A ∪ {(u, v)}

UNION(u, v) → merge two trees.

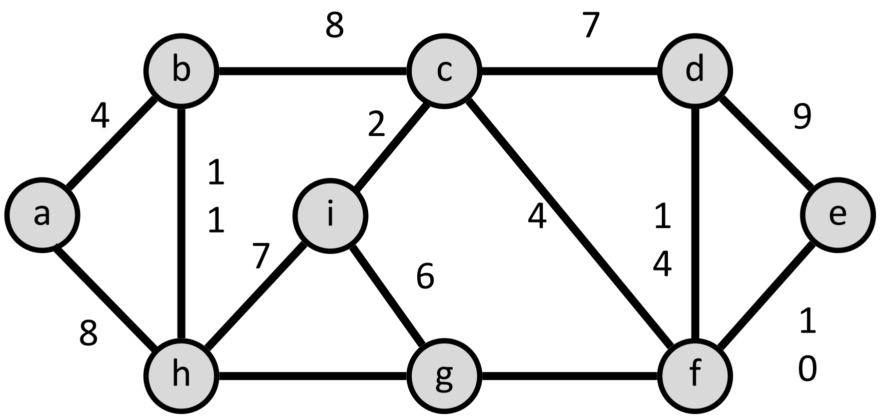
return A.

Kruskal's Algorithm – Example

9개의 set을 만듦

```
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
```

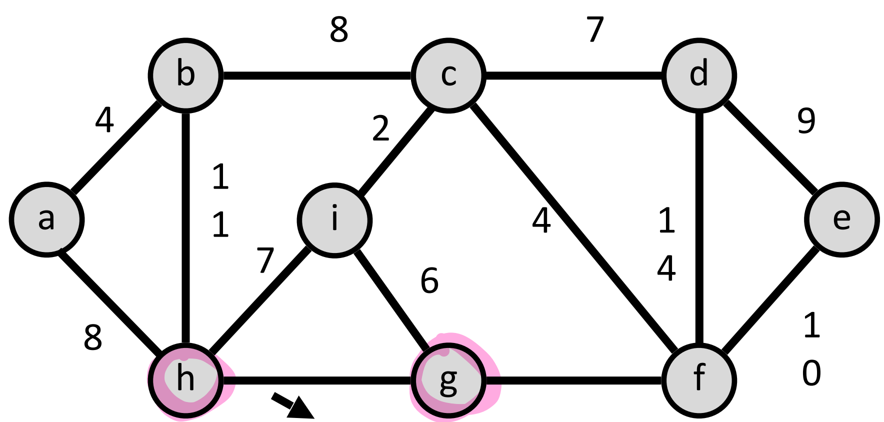
9 trees created



Kruskal’s Algorithm – Example

```
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
```

Start at edge with lowest cost, min w(u, v)



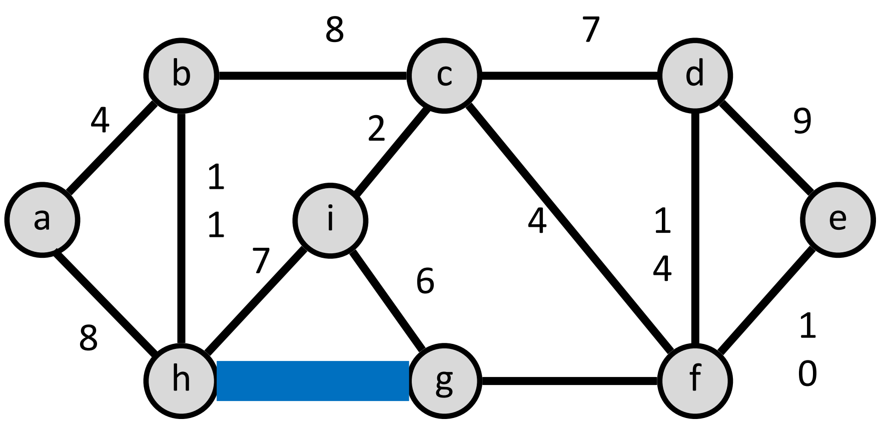
Do vertices h and g belong to different trees?

Yes! So merge the two trees.

Kruskal’s Algorithm – Example

```
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
```

Merge two trees →새로운 tree.



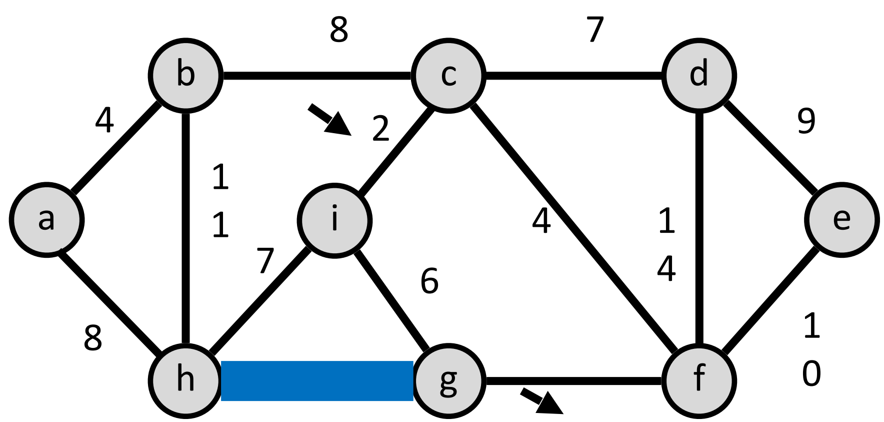
Do vertices h and g belong to different trees?

Yes! So merge the two trees.

Kruskal's Algorithm – Example

```
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
```

next min w(u, v)

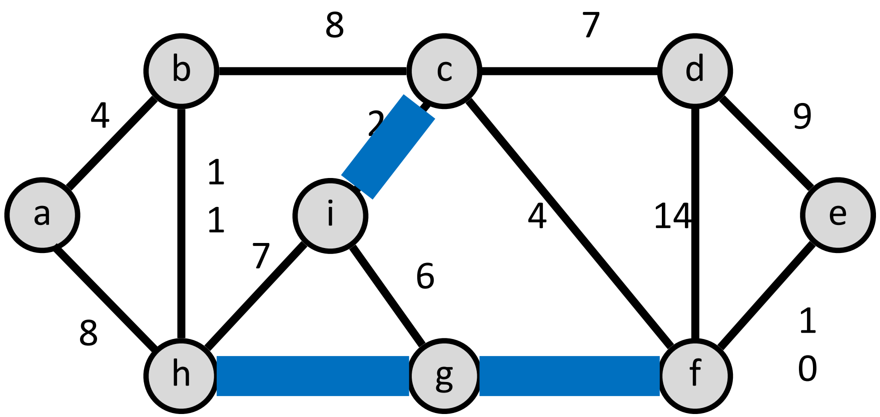


Do vertices c and i belong to different trees?
Do vertices g and f belong to different trees?

Kruskal's Algorithm – Example

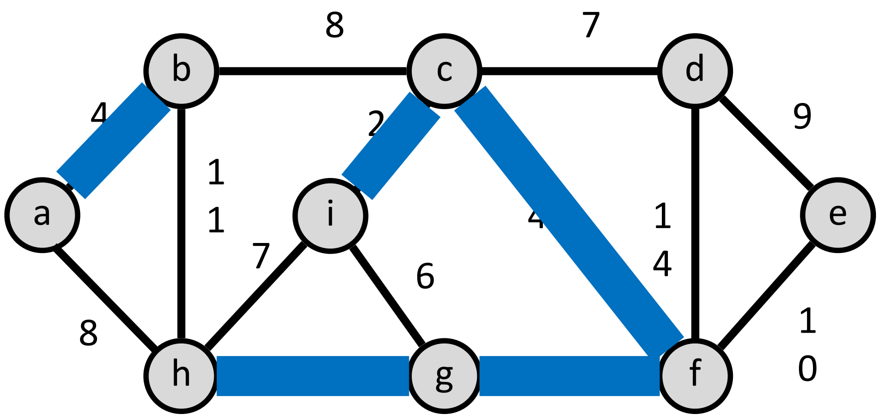
```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ G.V
      MAKE-SET(v)
  sort the edges of G.E into nondecreasing order by weight w
  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

Merge two trees



Kruskal’s Algorithm – Example

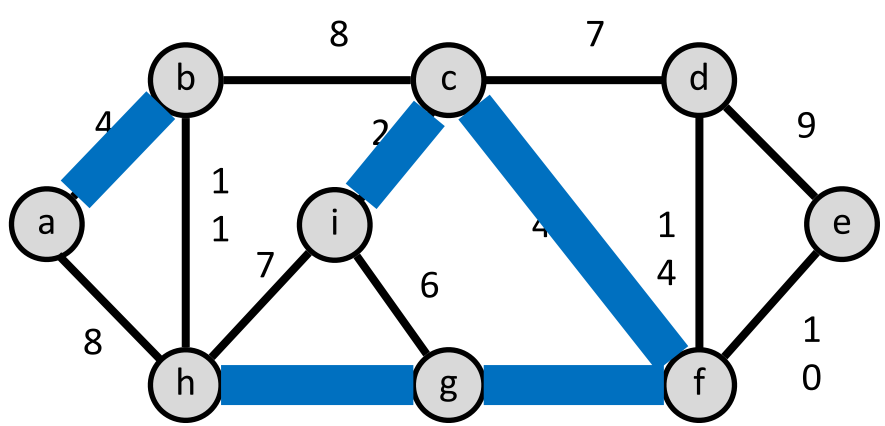
min(u,v)=4



KNU KYUNGPOOK NATIONAL UNIVERSITY

Kruskal's Algorithm – Example

min(u,v)=6

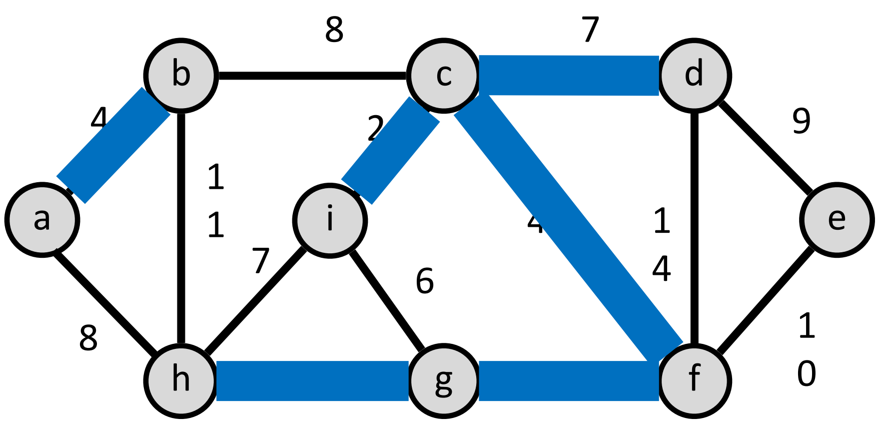


Do vertices i and g belong to different trees?

KNU KYUNGPOOK NATIONAL UNIVERSITY

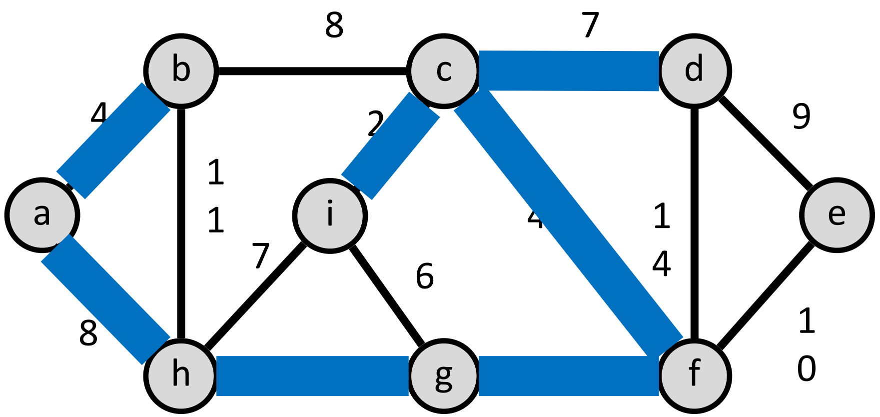
Kruskal’s Algorithm – Example

min(u,v)=7



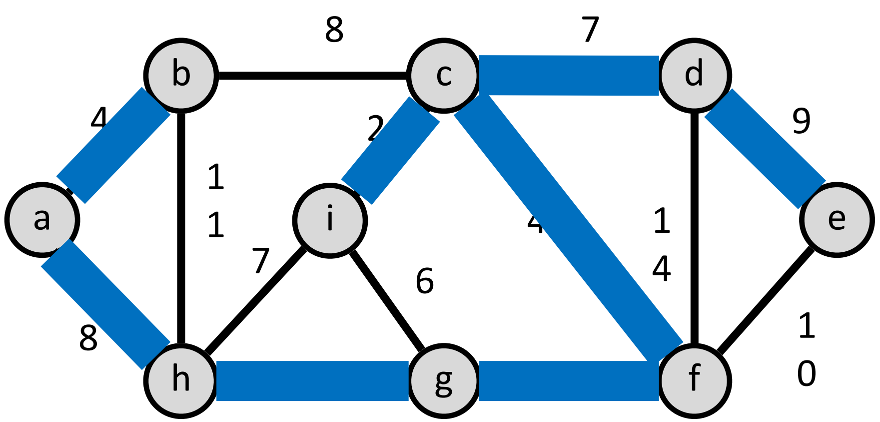
Kruskal's Algorithm – Example

min(u,v)=8



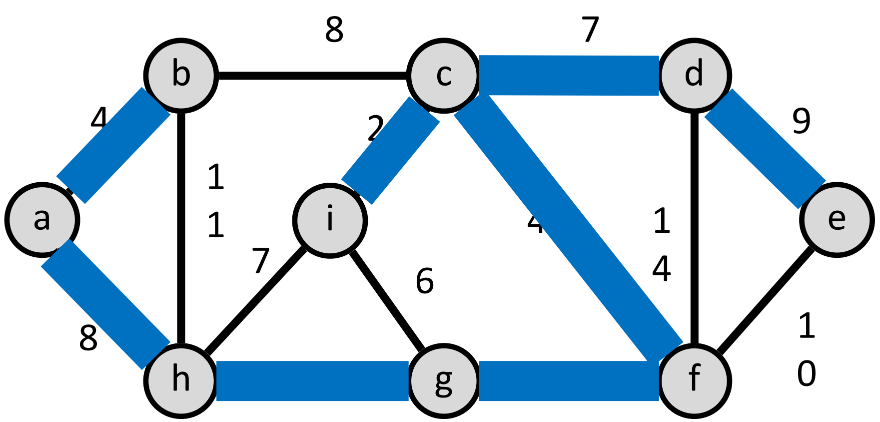
Kruskal’s Algorithm – Example

min(u,v)=9



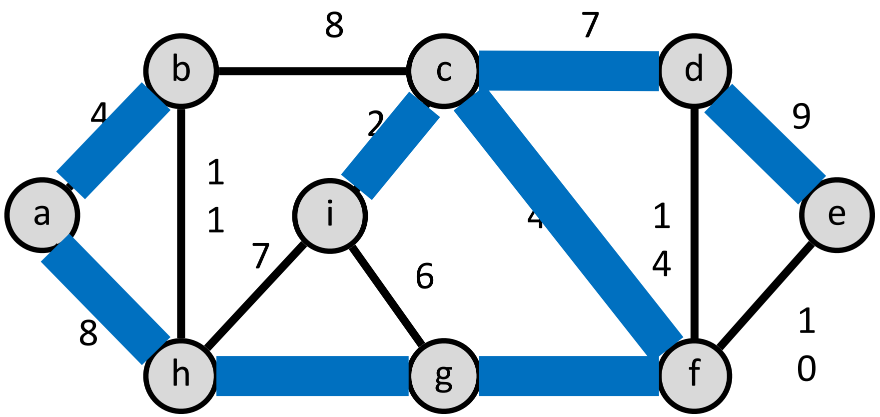
Kruskal's Algorithm – Example

min(u,v)=10

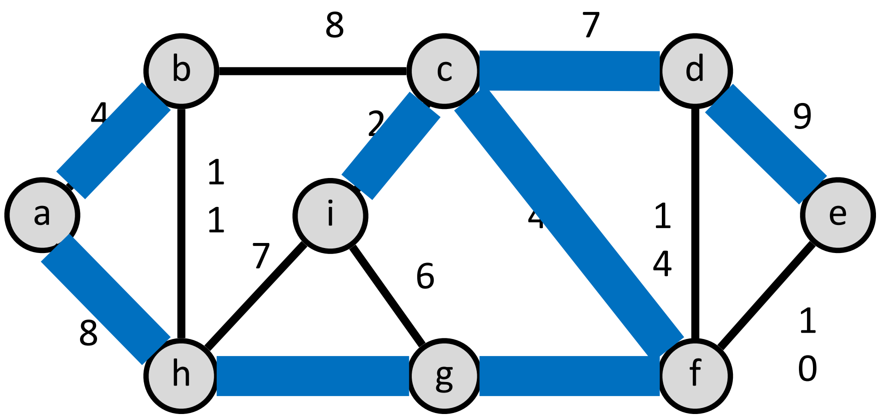


Kruskal's Algorithm – Example

min(u,v)=11,
14



Kruskal's Algorithm – Example



Cost=37

Kruskal's Algorithm – Analysis

- Running time of Kruskal's algorithm for a graph G=(V,E) depends on how we implement the function for finding the disjoint sets (to decide whether to merge two trees or not)

O(1) + O(|V|) + O(|E|log|E|)

```
MST-KRUSKAL(G, w)
A = ∅                                                        ---> O(1) - constant
for each vertex v ∈ G.V                                      ---> O(|V|)
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w   ---> O(E lg(E))
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)                             ---> O(E)
        A = A ∪ {(u, v)}
        UNION(u, v)                                          ---> Merge two trees
return A
```

$O(E\alpha(E))$

$O(E\alpha(E))$

$\alpha(E)$: 역아커만 함수

O(1)

Total running time is $O(E\lg(E))$

Prim's Algorithm – The pseudocode

Kruskal 에서는 각 정점들 모두 집합으로 만들었음

- Set A forms a single tree
- Prim's algorithm starts at an arbitrary root vertex r
- During the procedure, all edges not in the tree (or A) are in Q

means no edge to A
u.key is the min(u, v) where v is in A

root
r key = 0
"Q = G.V"

```
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)
```

Initialize all vertices and Q

간선 선택하고, 연결된 간선들은 후보 리스트에 삽입

Get vertex u that connects to tree A with minimum cost.
(A light or safe edge)

Update keys of v in Q in respect to the selected u

A와 연결된 가중치 가장 작은

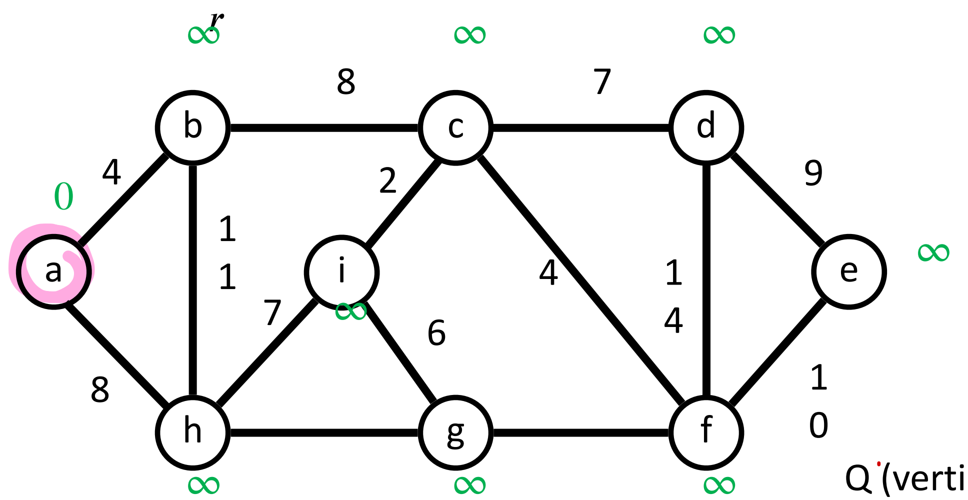
여기 가중치가 가장 작은 최고 우선순위 정점

KNU KYUNGPOOK NATIONAL UNIVERSITY

Prim's Algorithm – Example

```
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)
```

Vertex **a** is chosen as r



Q (vertices not in A)

a	b	c	d	e	f	g	h	i

A (MST)

for each v ∈ G.Adj[u]

if v∈Q and w(u,v) < v.key

v.π = u

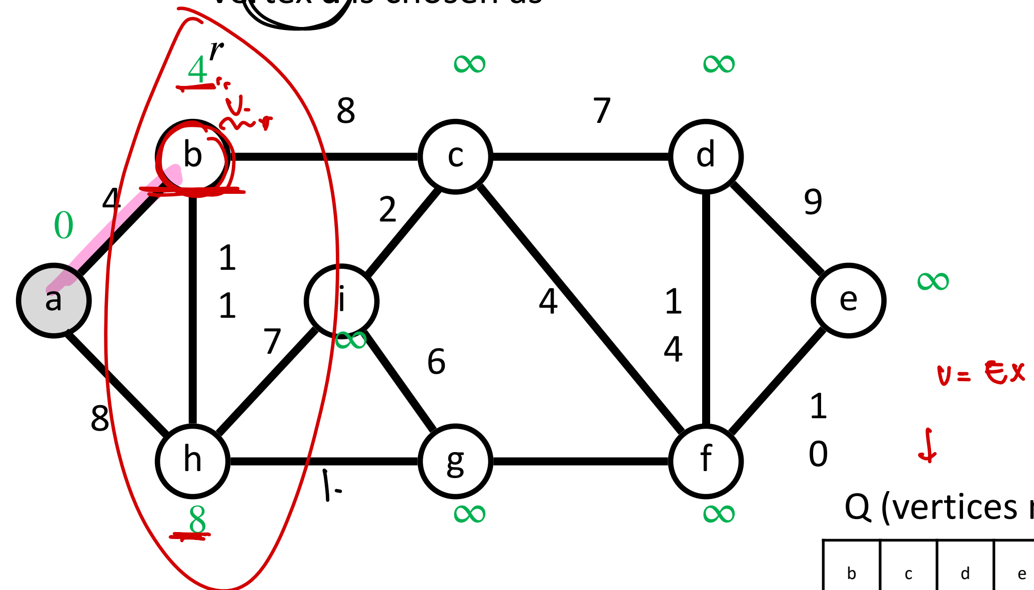
v.key = w(u,v).

Prim's Algorithm – Example

"A와 연결된 애들이랑"

```
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)
```

Vertex **a** is chosen as



Q (vertices not in A)

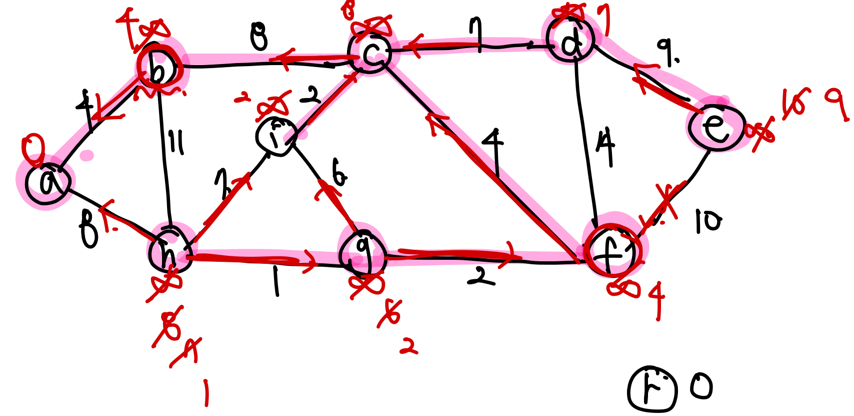
b	c	d	e	f	g	h	i

A (MST)

a							

KNU KYUNGPOOK NATIONAL UNIVERSITY

Prim's Algorithm – Example



A

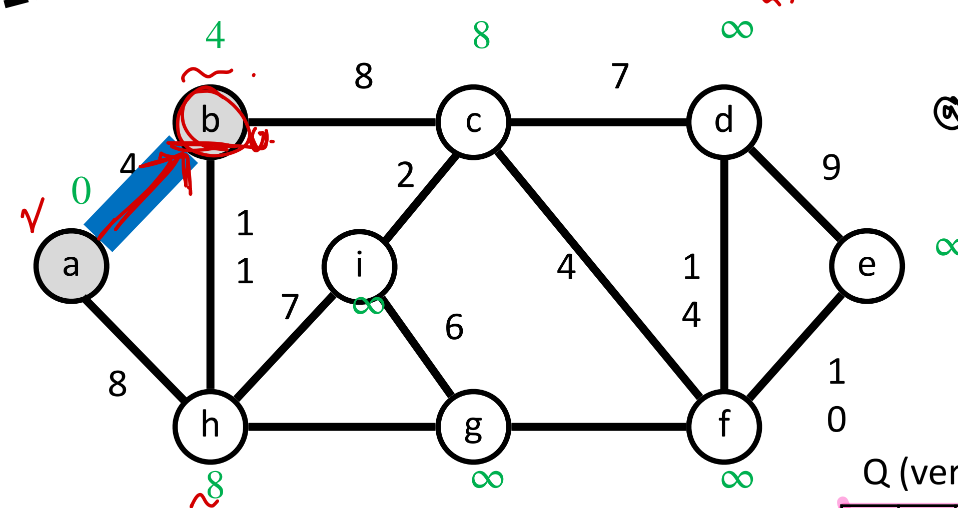
```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```

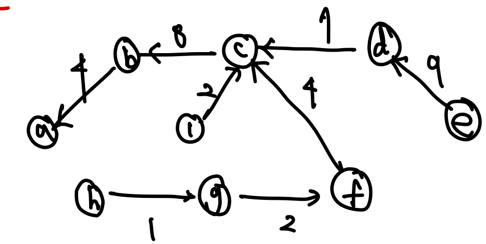
a는 A에 있고
b는 Q에 있는

if $v \in Q$ and
$w(u,v) < v.key$

(a,b) has smallest cost, a is in A and b is in Q.
We want to keep growing the tree A.

Q





$4+8+7+9+2+4+1+2$
$= 12+16+6+3$
$= 28+9 = 37$

Q (vertices not in A)

c	d	e	f	g	h	i	

A (MST)

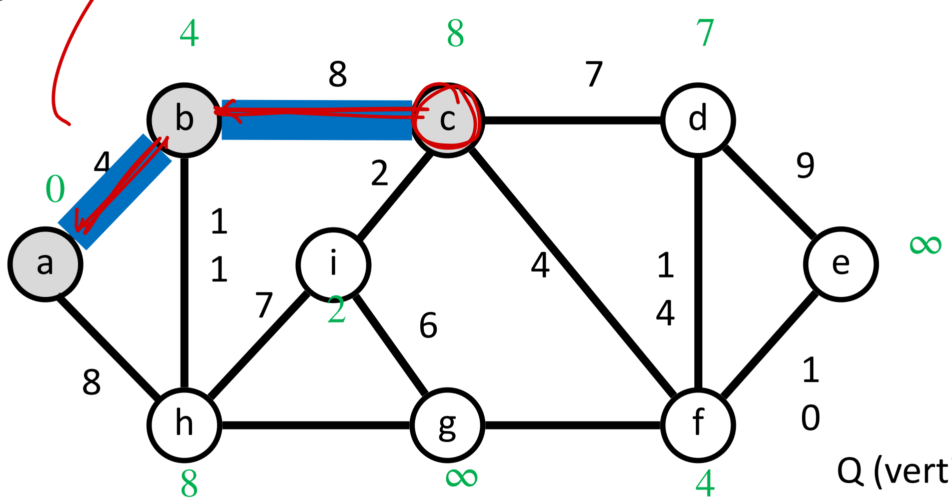
a	b						

Prim's Algorithm – Example

for each v ∈ G.Adj

```
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)
```

(b,c) and (a,h) has smallest cost with 8



Q (vertices not in A)

d	e	f	g	h	i		

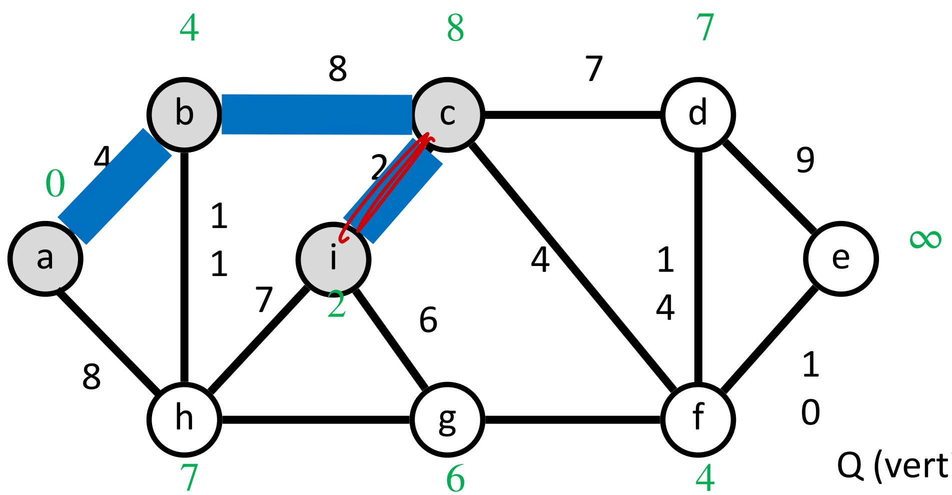
A (MST)

a	b	c					

Prim's Algorithm – Example

```
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)
```

Repeat until Q is empty



Q (vertices not in A)

d	e	f	g	h			

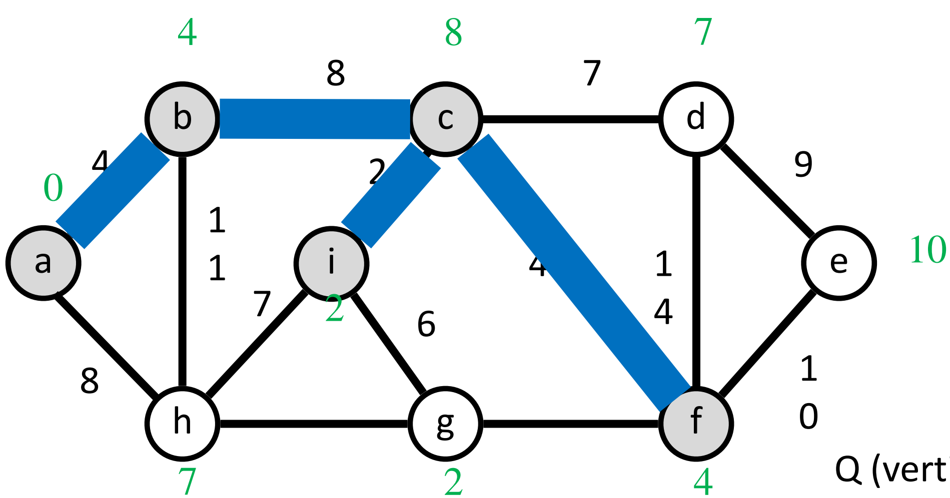
A (MST)

a	b	c	i				

Prim's Algorithm – Example

```
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)
```

Repeat until Q is empty



Q (vertices not in A)

d	e	g	h				

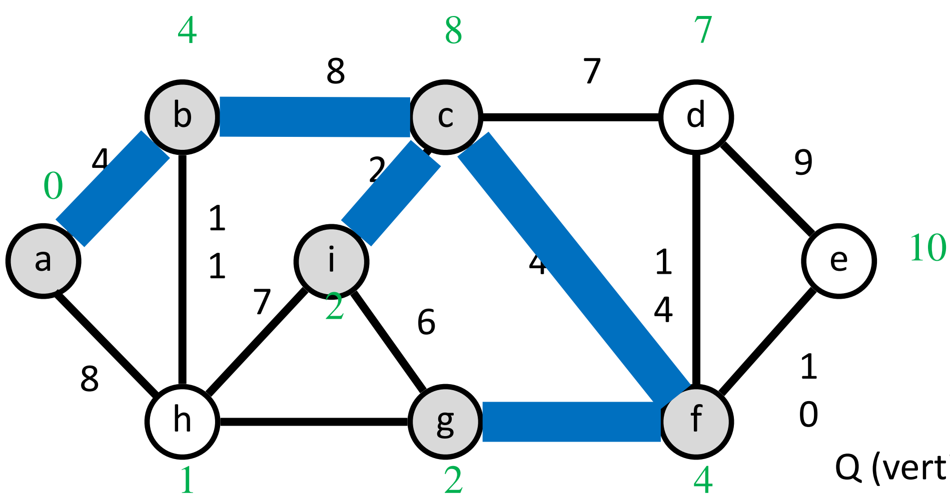
A (MST)

a	b	c	i	f			

Prim’s Algorithm – Example

```
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)
```

Repeat until Q is empty



Q (vertices not in A)

d	e	h					

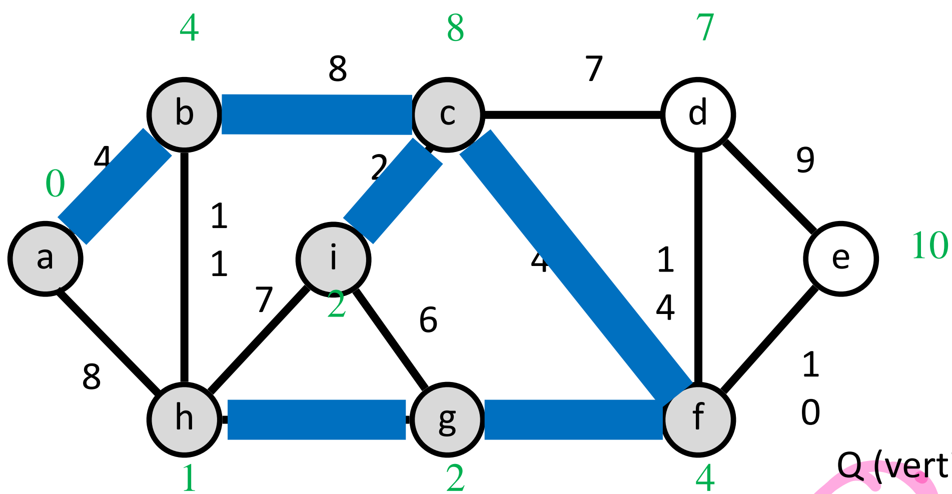
A (MST)

a	b	c	i	f	g		

Prim’s Algorithm – Example

```
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)
```

Repeat until Q is empty



Q (vertices not in A)

d	e						

A (MST)

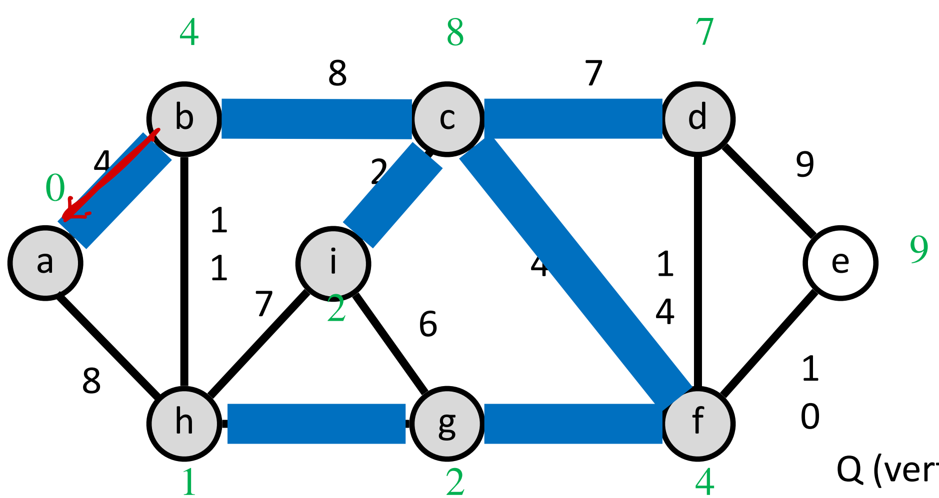
a	b	c	i	f	g	g	

Prim's Algorithm – Example

Q는 A에 포함되지 않은.

```
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)
```

Repeat until Q is empty



Q (vertices not in A)

e							

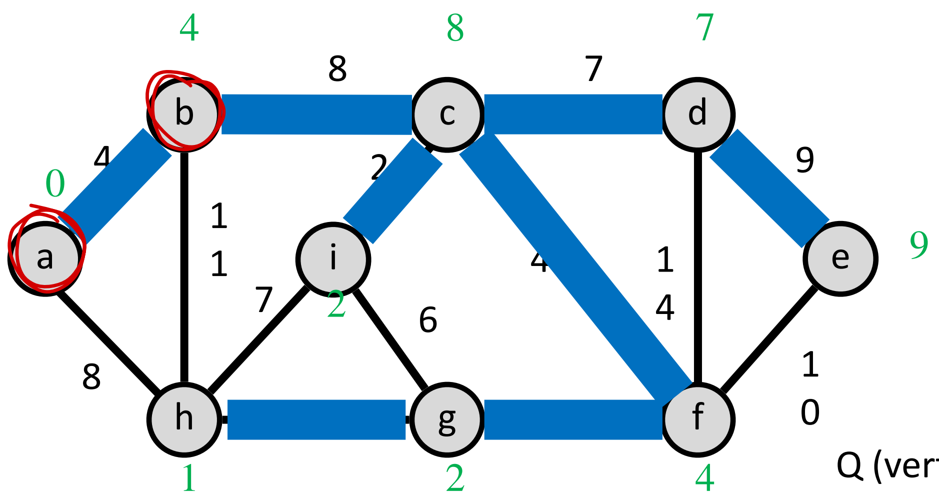
A (MST)

a	b	c	i	f	g	h	d

Prim's Algorithm – Example

```
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)
```

Repeat until Q is empty



Q (vertices not in A)

A (MST)

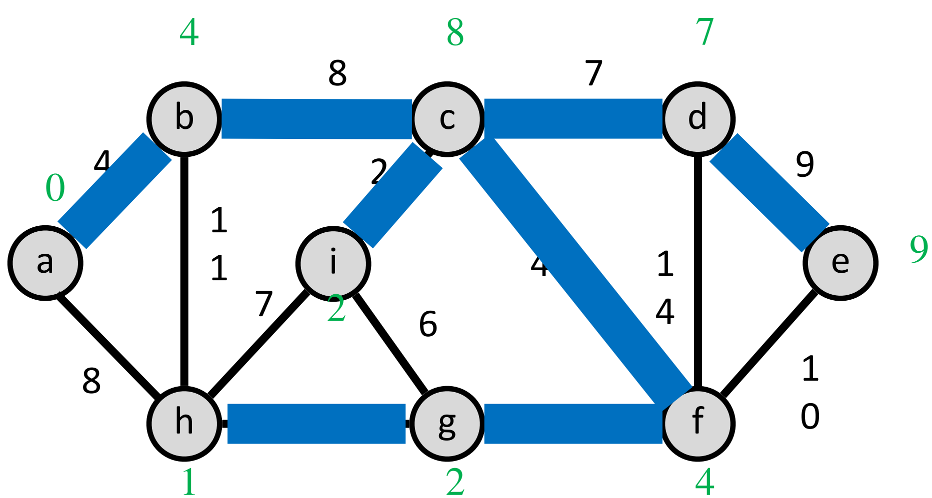
a	b	c	i	f	g	h	d	e

Prim's Algorithm – Example

Cost=37

This is the same when adding all the green numbers

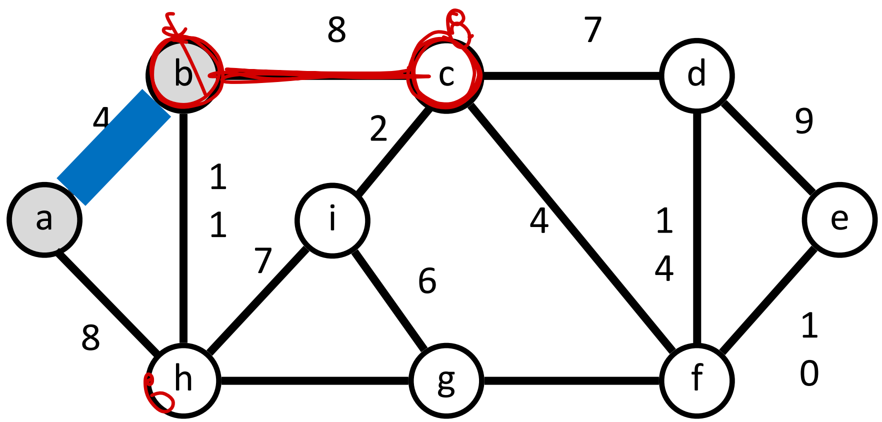
$$\sum_{i \in v} v.key = 37$$



Prim’s Algorithm – Discussion

$Q = V - A$

- How is the Q used and what data structure should we use for optimal performance?
- First, Q includes vertices not in A. Thus Q=V-A, which means that they are candidates for A



Q (vertices not in A)

c	d	e	f	g	h	i	

A (MST)

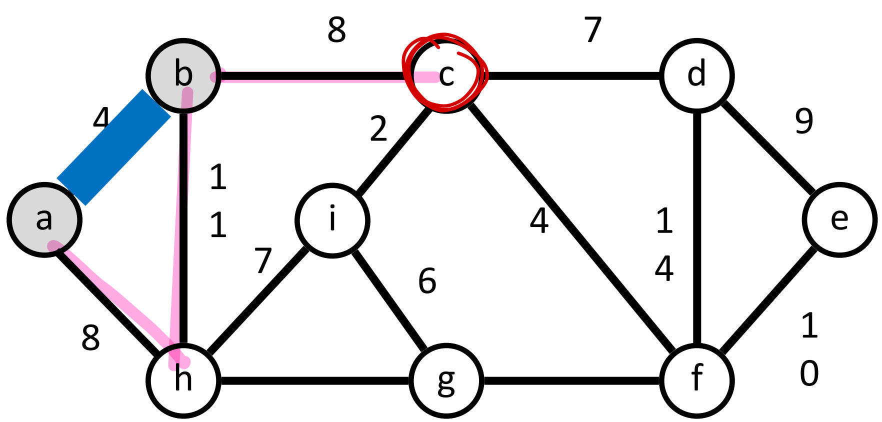
a	b						

Prim's Algorithm – Discussion

candidate

v's –

- From Q, we want to find a v that is adjacent to A and whose distance to a vertex in A is minimum
- If A={a, b}, there are three candidate v's – (b,c), (a,h) and (b,h)



①

Q에서 A와 인접한 vertex 중 값이 가장 작은 것을 고름

Q (vertices not in A)

c	d	e	f	g	h	i	

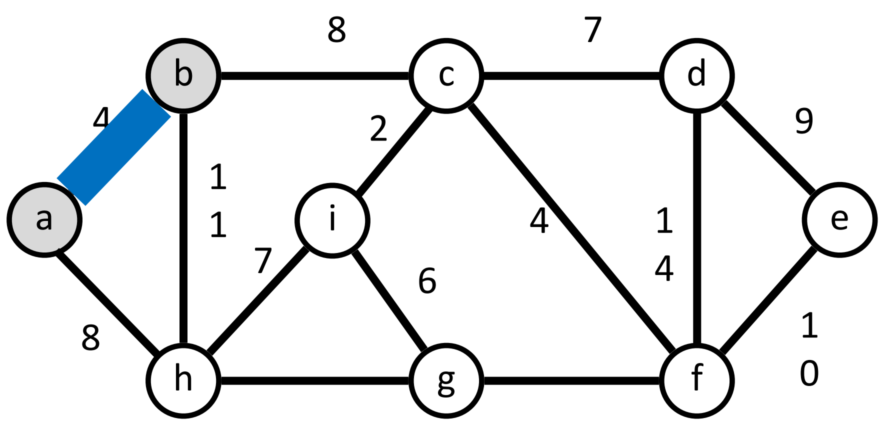
A (MST)

a	b						

Prim’s Algorithm – Discussion

min-heap

- To always choose a minimum edge from the candidates, we can build a min-priority queue!
- Very fast, $O(V\,lg(V))$
- Actually, the EXTRACT-MIN function defines the running time of Prim’s algorithm



Q (vertices not in A)

c	d	e	f	g	h	i	

A (MST)

a	b						

Prim's Algorithm – Analysis

Q

logV

```
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)
```

$O(V)$

모든 정점 마

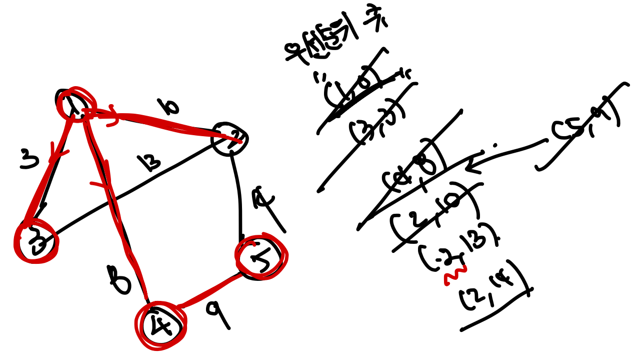
$O(V \lg(V))$

$O(E)$

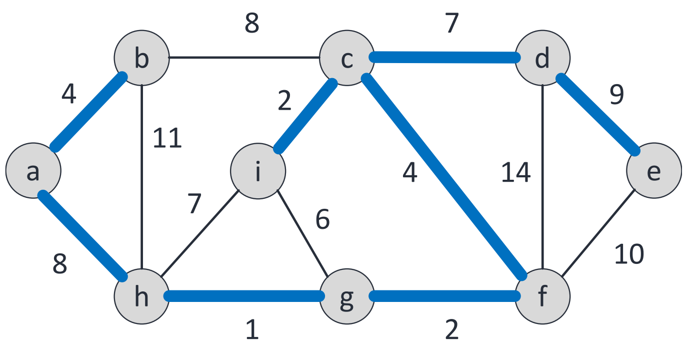
$V\log V + E$

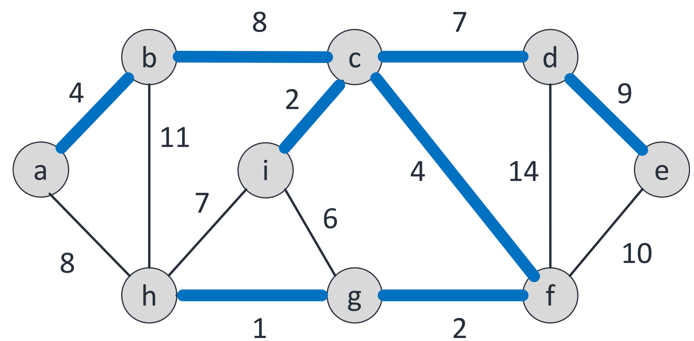
Running time is $O(E + V\lg(V))$

Kruskal's vs Prim's



Kruskal's MST	Prim's MST





Cost=37

$O(E \lg(E))$

Cost=37

$O(E \lg(E)) \rightarrow O(E + V \lg(V))$

(When using Fibonacci heaps)