Algorithms

Ok-Ran Jeong

School of Computing, Gachon University

5. Graph Algorithms I

Contents

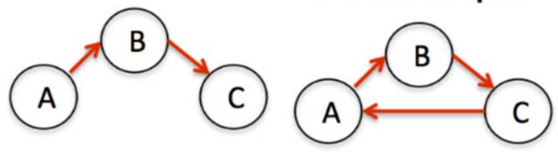
- Graph representation
- Minimum spanning trees
- Prim's algorithm
- Kruskal's algorithm

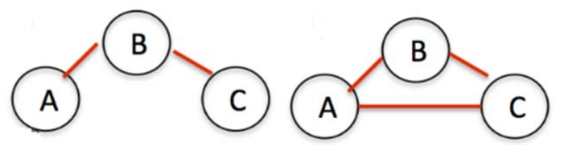
- Problem 9: Saving ink

Graphs (review)

- **Definition.** A *directed graph (digraph)*

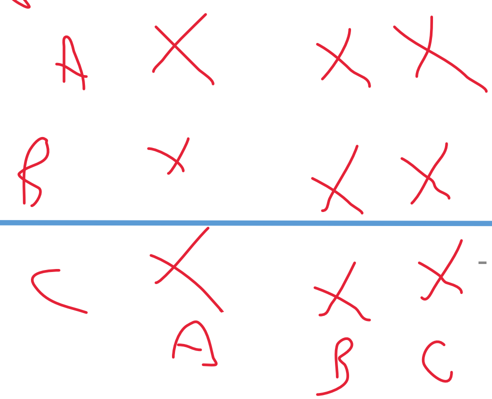
$G = (V, E)$ is an ordered pair consisting of
  - a set V of *vertices* (singular: *vertex*),
  - a set $E \subseteq V \times V$ of *edges.*

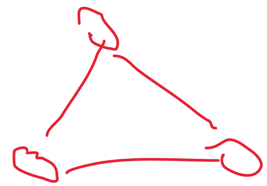




- In an *undirected graph* $G = (V, E)$, the edge set E consists of *unordered* pairs of vertices.
- In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$.

E : edge의 크기



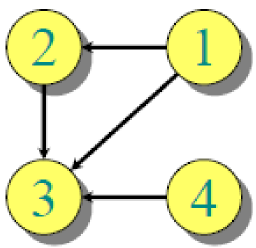


Adjacency-matrix representation

The ***adjacency matrix*** **of a graph *G = (V, E), where***
V = {1, 2, …, n}, is the matrix *A[1 . . n, 1 . . n]* given by

$$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{if } (i,j) \notin E. \end{cases}$$

edge가 존재한다면

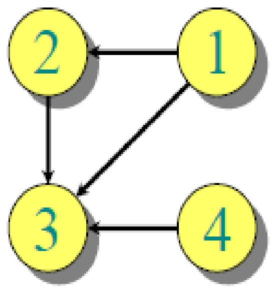


A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ storage
⇒ ***dense***
representation.

Adjacency-list representation

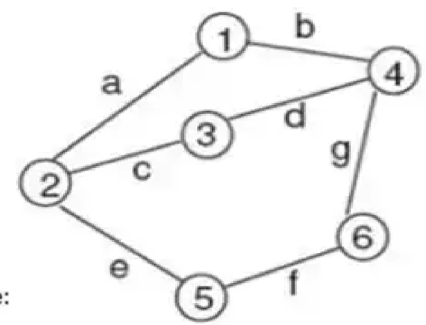
An ***adjacency list*** **of a vertex** ***v* ∈ *V*** is the list ***Adj[v]***
of vertices adjacent to *v*.



$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

degree : 2

degree : 3



For undirected graphs, **|Adj[v]|** ***= degree(v).***
For digraphs, **|Adj[v]|** ***= out-degree(v).***
Handshaking Lemma: $\Sigma_{v \in V} = 2|E|$ for undirected graphs ⇒ adjacency lists use $\Theta(V + E)$ storage — a **sparse** representation (for either type of graph).

Right Hand Side:
|E| = 7
2 * |E| = 14

Left Hand side:

d(1) = 2 d(4) = 3 d(2) = 3
d(5) = 2 d(3) = 2 d(6) = 2

d(1) + d(2) + d(3) + d(4) + d(5) + d(6) = 2+3+2+3+2+2=14

희박한

edge에 2를 곱한 값이 전체 verices의 수와 같다

Minimum spanning trees

Input: A connected, undirected graph ***$G = (V, E)$***
with weight function $w : E \to \mathbb{R}$

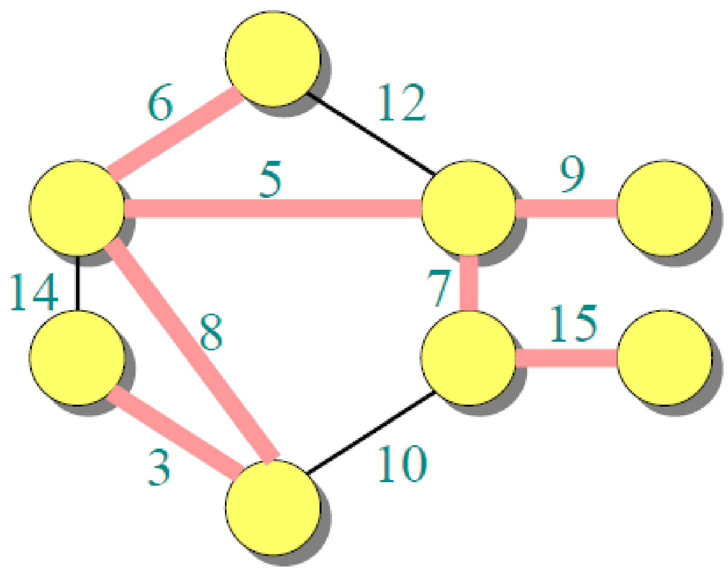
- For simplicity, assume that all edge weights are distinct.

Output: A *spanning tree T* — a tree that connects all vertices of minimum weight:

최소 weight(가중치)를 가지는 tree

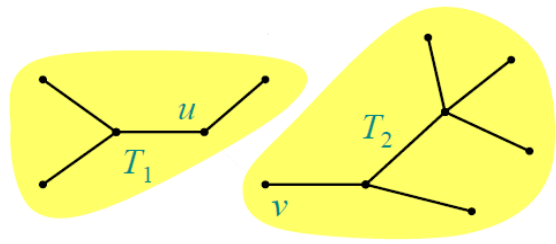
$$w(T) = \sum_{(u,v) \in T} w(u,v) .$$

Example of MST



Optimal substructure

MST *T*:



(Other edges of *G* are not shown.)

Remove any edge $(u, v) \in T$. *Then, T is partitioned* into two subtrees T_1 *and* T_2.

Theorem. The subtree T_1 is an MST of $G_1 = (V_1, E_1)$, the subgraph of *G* ***induced*** by the vertices of T_1:

V_1 = vertices of T_1,

$E_1 = \{ (x, y) \in E : x, y \in V1 \}$.

v1을 연결하는 edge

Similarly for T_2.

Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G.

Do we also have overlapping subproblems?

- *Yes*

dynamic programming can work

Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G.

Do we also have overlapping subproblems?

Yes

Great, then dynamic programming may work!

- *Yes, but MST exhibits another powerful property which leads to an efficient algorithm.*

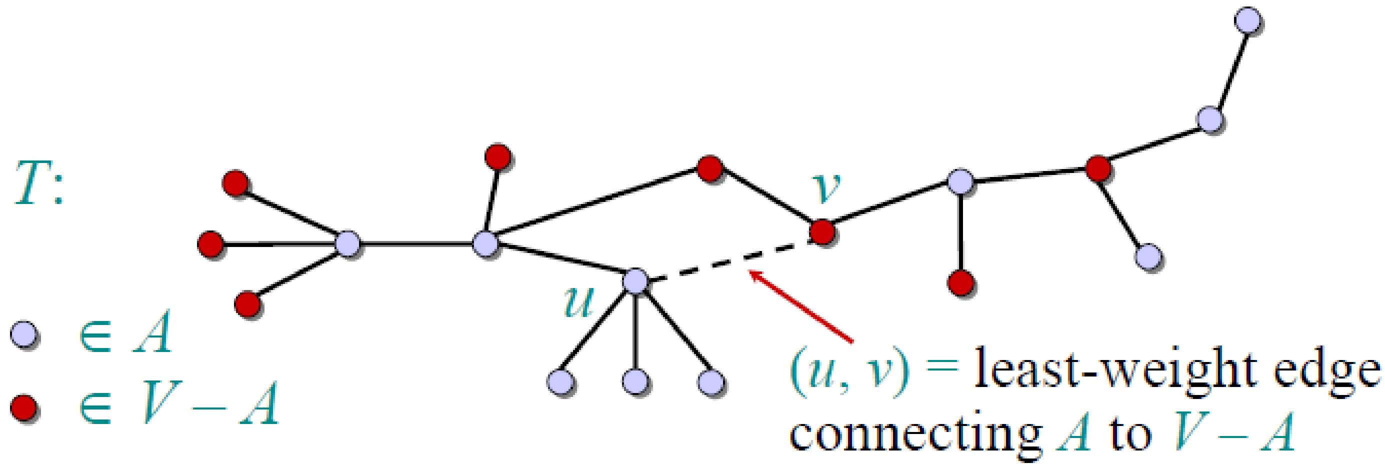
Hallmark for “greedy” algorithms

Greedy-choice property
A locally optimal choice is globally optimal.

Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.

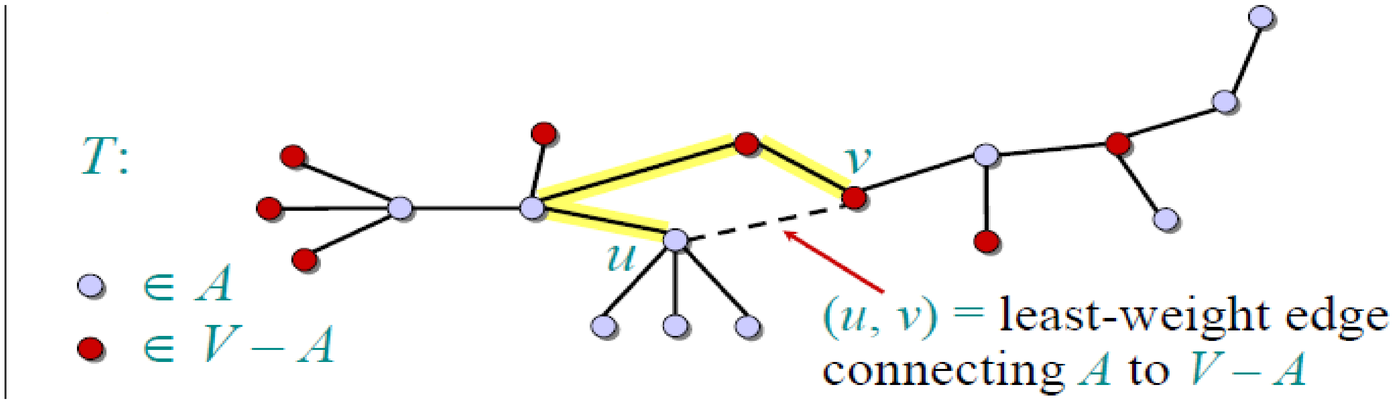
Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.



Proof of theorem

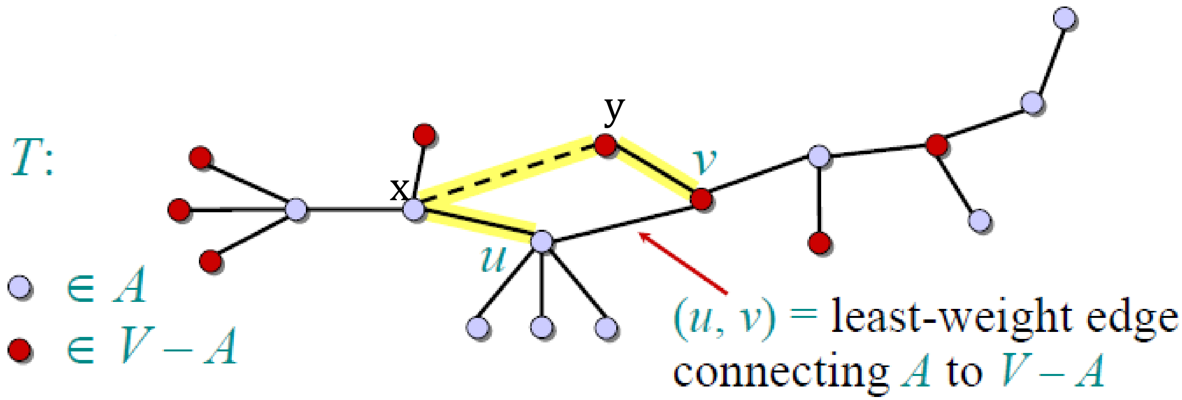
Proof. Suppose $(u, v) \notin T$. Cut and paste.



Consider the unique simple path from u to v in T.

Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.

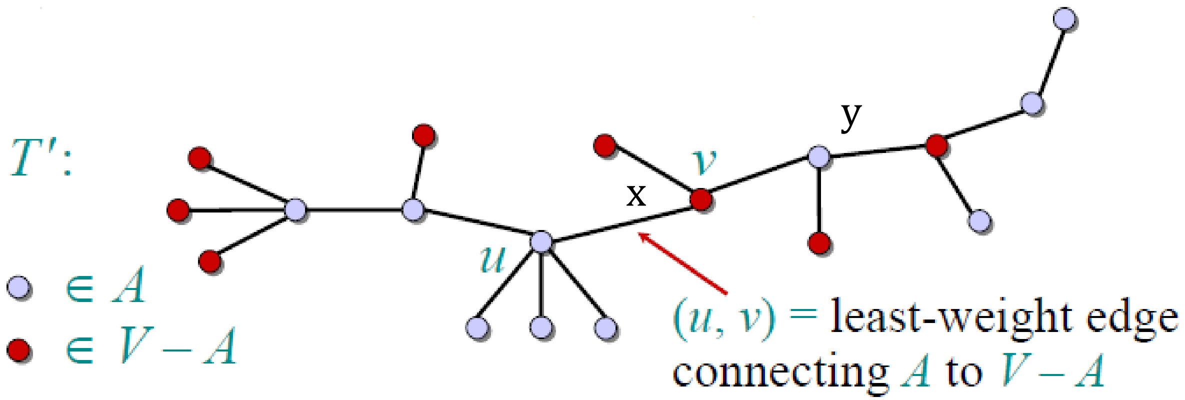


Consider the unique simple path from u to v in T.

Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.

Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.



Consider the unique simple path from u to v in T.

Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.

A lighter-weight spanning tree than T results.

$T' = T - \{(x, y)\} \cup \{(u, v)\}$

$w(u, v) \le w(x, y)$
$w(T') = w(T) - w(x, y) + w(u, v) \le w(T)$
$w(T) \le w(T')$
$\therefore$ (u, v) *is safe for A*

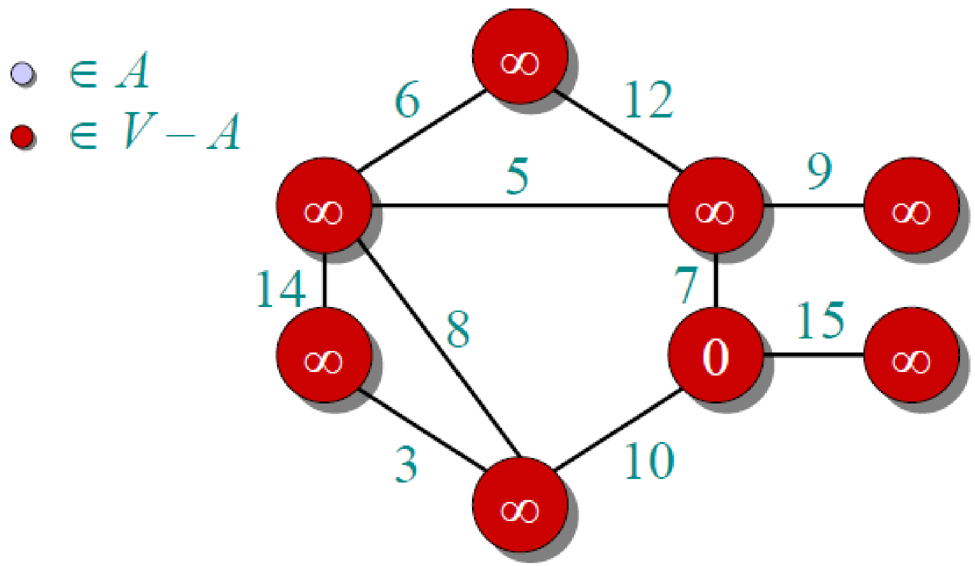
Prim's algorithm

IDEA: Maintain $V - A$ *as a priority queue* Q*. Key* each vertex in Q with the weight of the least-weight edge connecting it to a vertex in $A(MST)$.

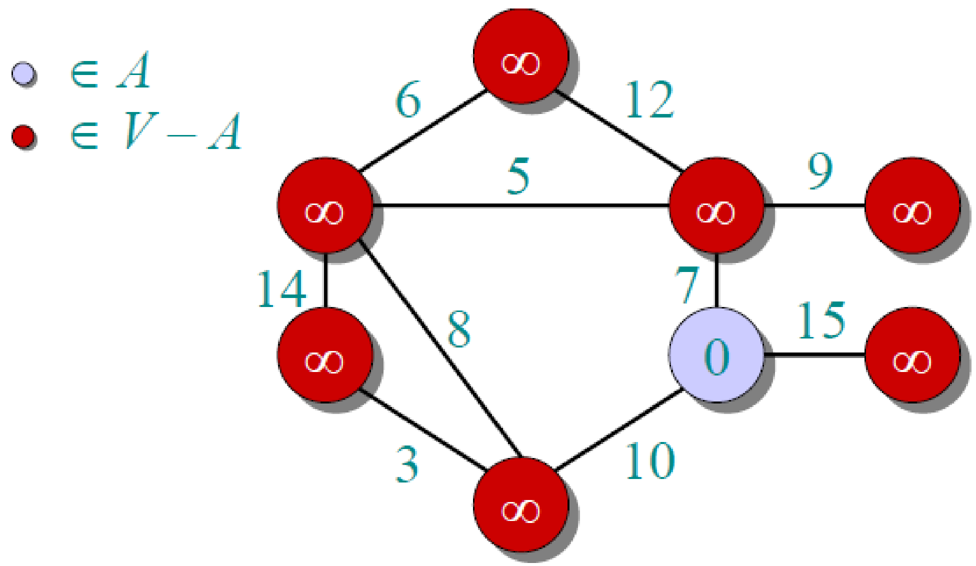
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q) Q를 교차하는 least weight edge 찾기
   for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
          $\pi[v] \leftarrow u$

At the end, $\{(v, \pi[v])\}$ forms the MST.

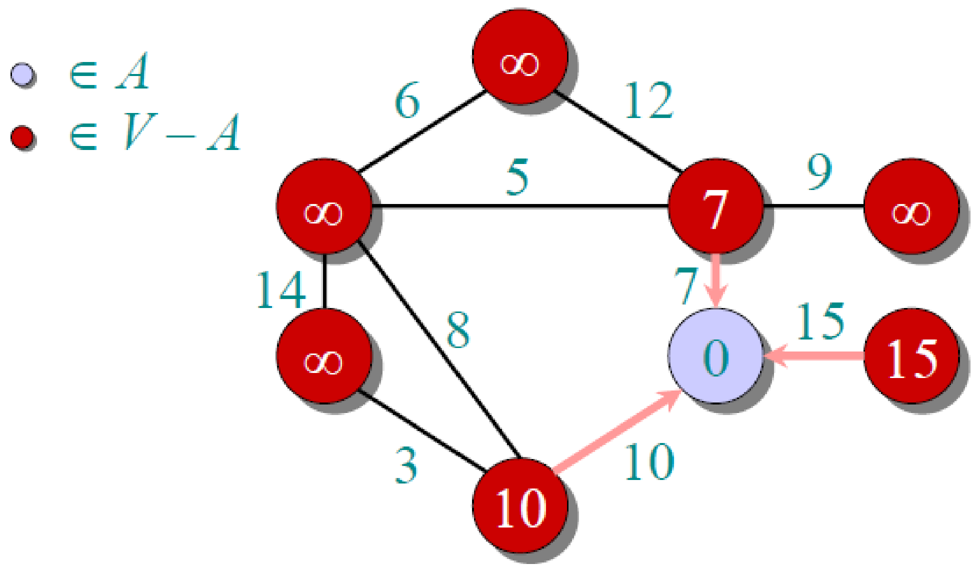
Example of Prim's algorithm



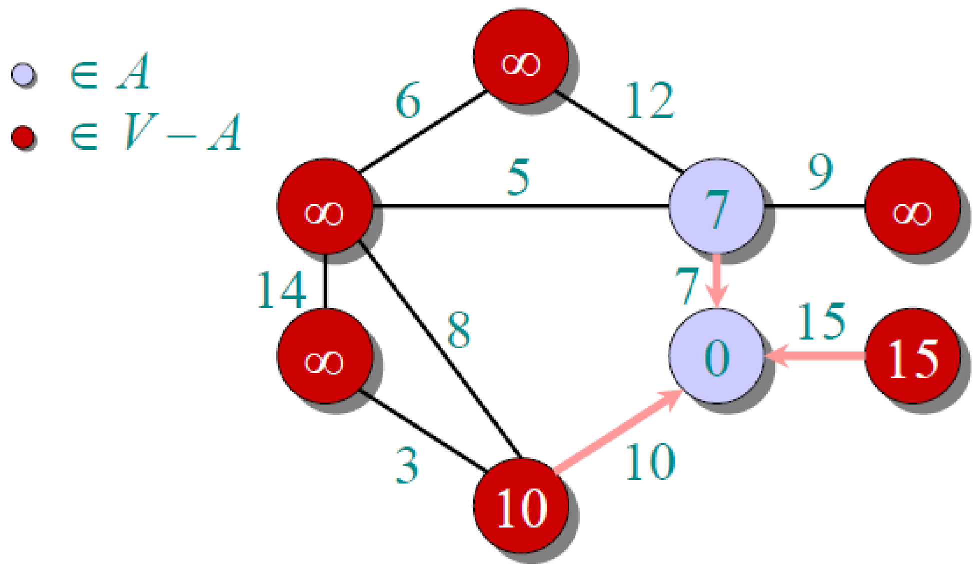
Example of Prim’s algorithm



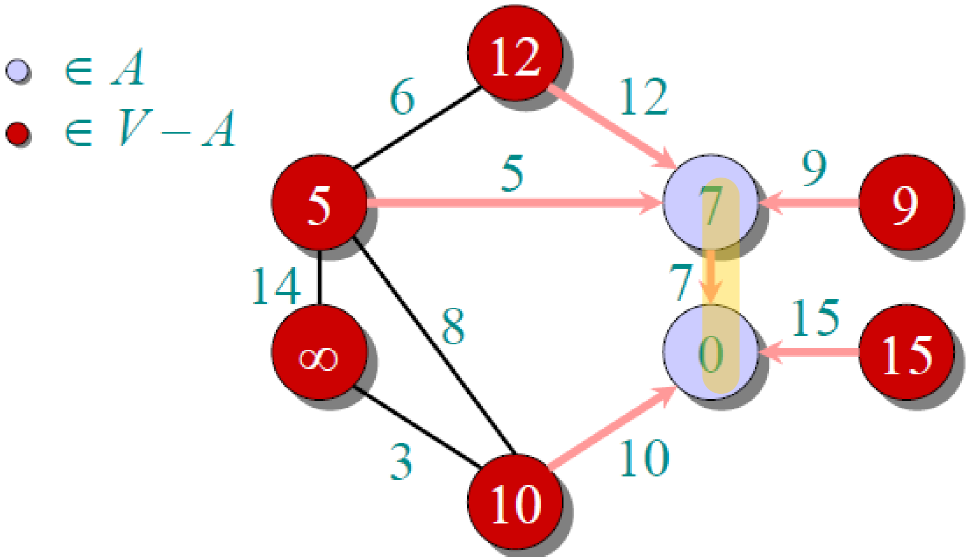
Example of Prim’s algorithm



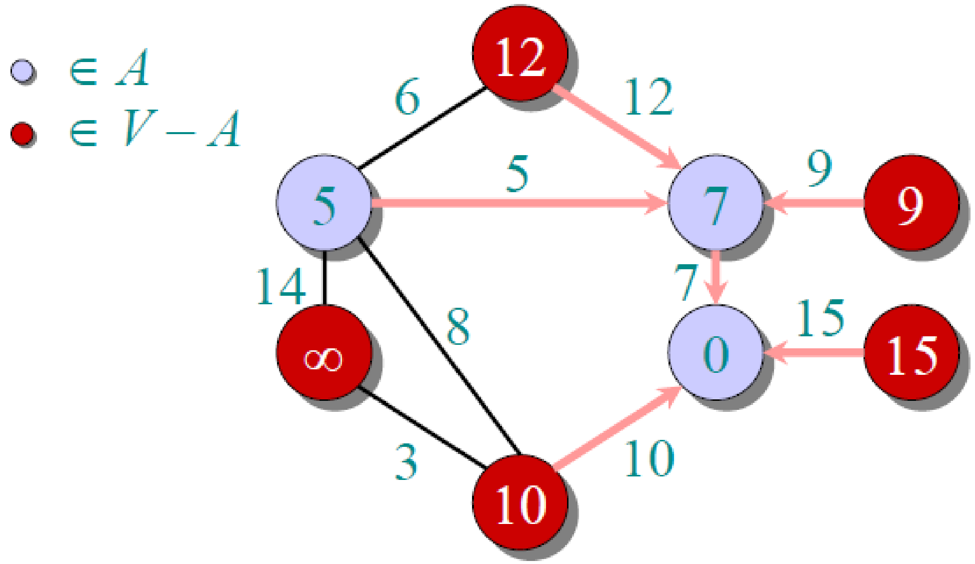
Example of Prim's algorithm



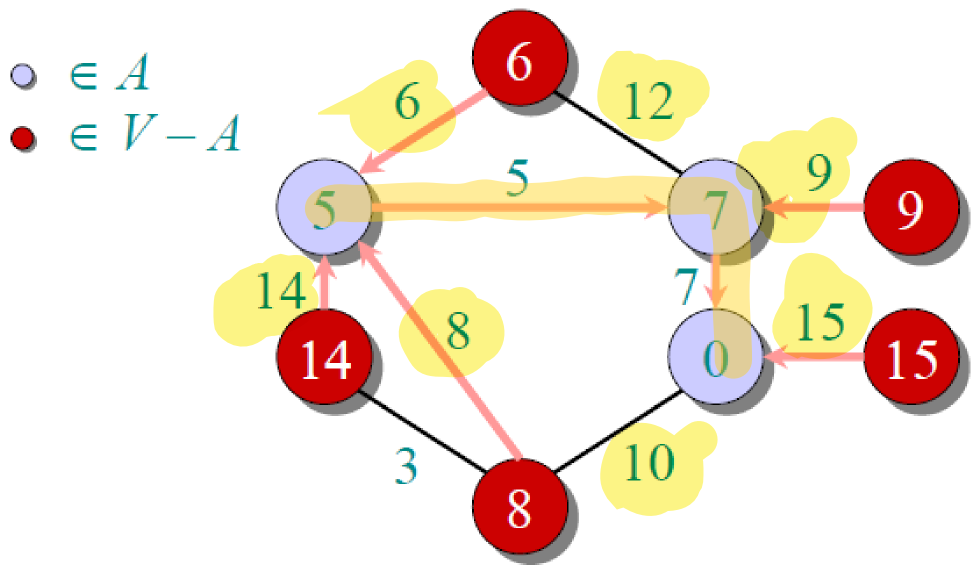
Example of Prim's algorithm



Example of Prim's algorithm

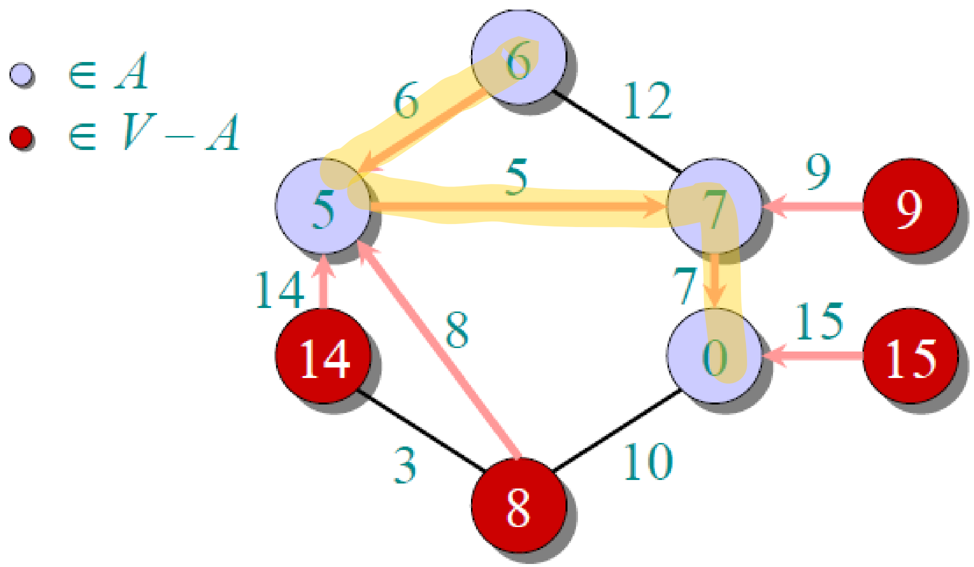


Example of Prim's algorithm



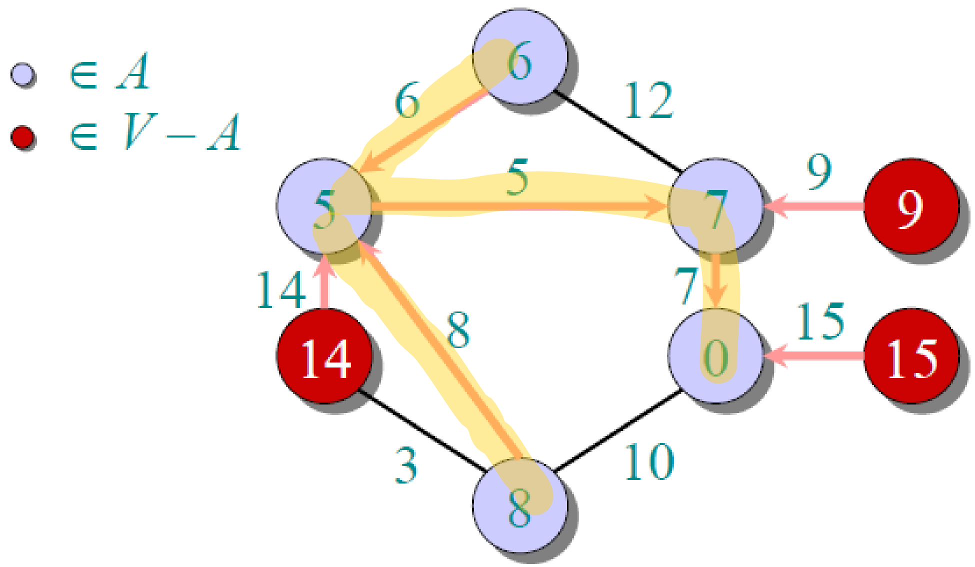
6 선택

Example of Prim's algorithm

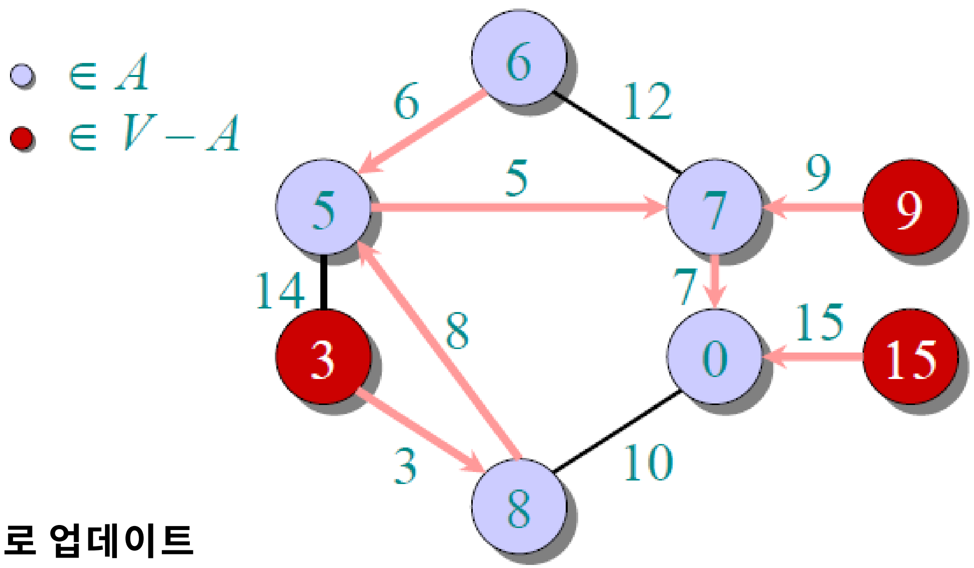


14 8 15 9

Example of Prim's algorithm

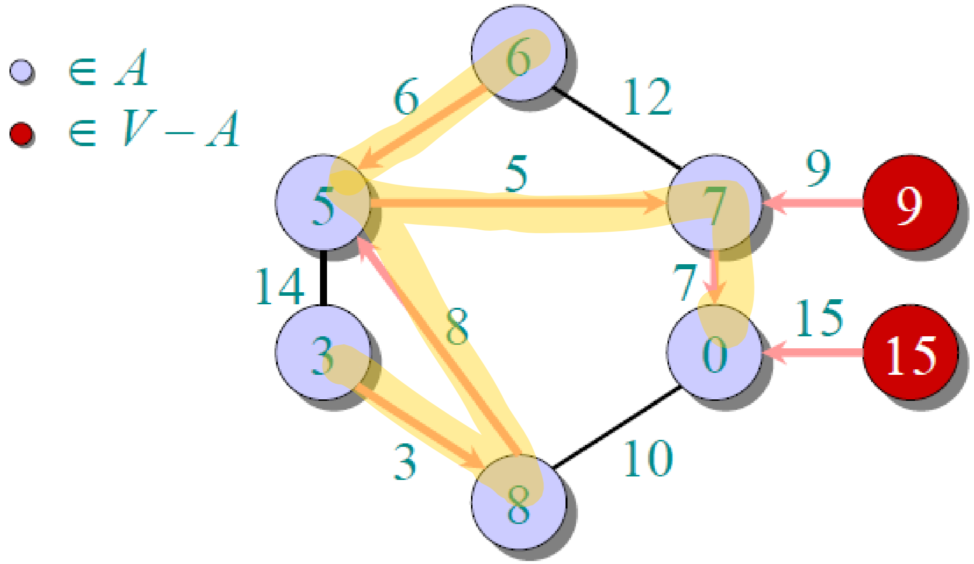


Example of Prim's algorithm

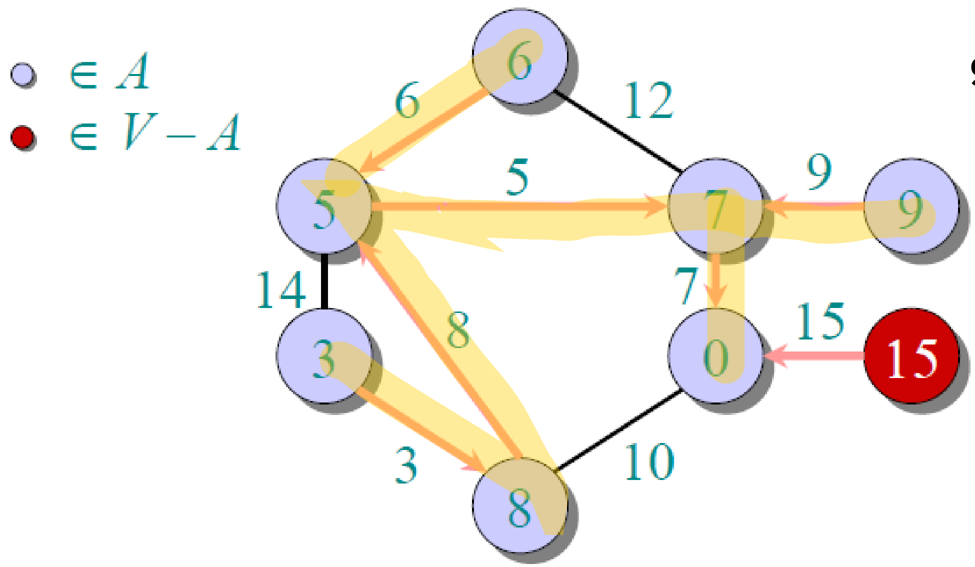


14가 3보다 크므로 3으로 업데이트

Example of Prim's algorithm

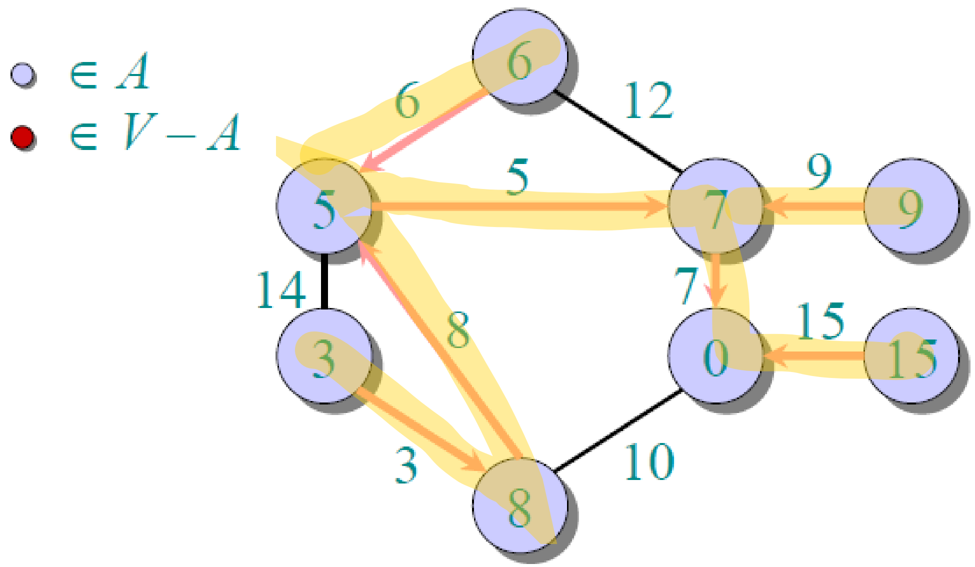


Example of Prim's algorithm



9와 15가 비교돼서 9 선택

Example of Prim’s algorithm



Analysis of Prim

$\Theta(V)$ total

$Q \leftarrow V$

$key[v] \leftarrow \infty$ for all $v \in V$

$key[s] \leftarrow 0$ for some arbitrary $s \in V$

$|V|$ times

while $Q \neq \emptyset$

do $u \leftarrow$ EXTRACT-MIN(Q) ***Using Binary Heap***

$degree(u)$ times

for each $v \in Adj[u]$

do if $v \in Q$ and $w(u, v) < key[v]$

then $key[v] \leftarrow w(u, v)$

$\pi[v] \leftarrow u$

Handshaking Lemma $\Rightarrow$ O(E) implicit DECREASE-KEY's.

Time = O(V)$\cdot T_{\text{EXTRACT-MIN}}$ + $O(E) \cdot T_{\text{DECREASE-KEY}}$ = $O(V\log V) + O(E\log V) = O(E\log V)$

Kruskal's algorithm

Idea: Start with a forest of single-node trees. Grow MST by repeatedly adding a light edge from all edges connecting two different trees in the forest. Such a light edge is safe by corollary.

반복적으로 가장 작은 edge를 추가하여 MST를 grow함.

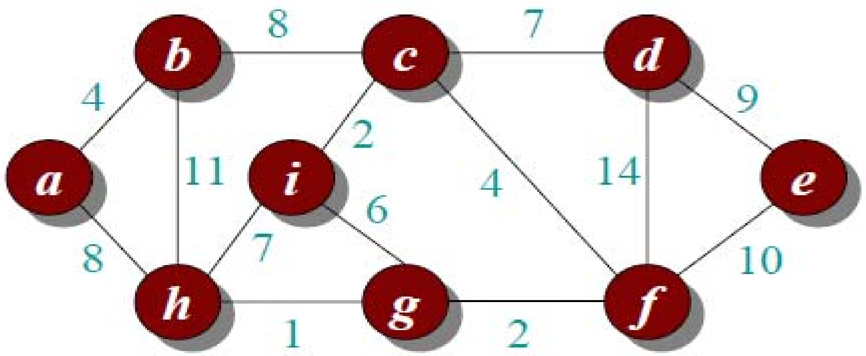
오름차순

Implementation: Sort the edges in non-decreasing order of weights. Using a *disjoint-set (union-find) data structure*, *u* and *v* are vertices of the same tree if and only if FIND-SET(u) = FINDSET (v).

서로 겹치는게 없는 서로소 집합

이 경우에만 동일한 크기의 depth가 됨

Kruskal’s algorithm




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

같은 트리에 속하지 않는다면

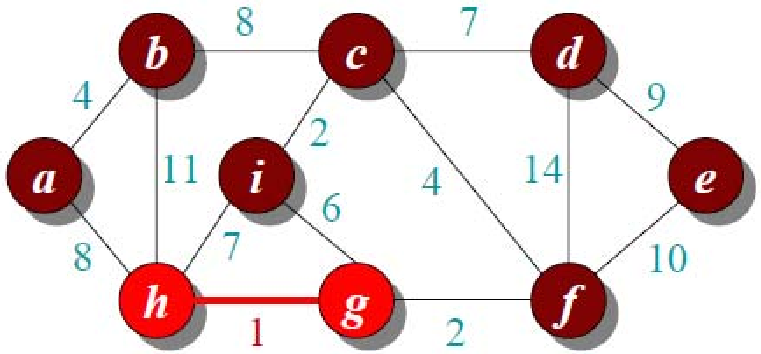
그 edge를 추가

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (g, h) is safe.



MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

{1, 2, 2, 4, 4, 6, 7, 7, 8, 8, 9, 10, 11, 14}

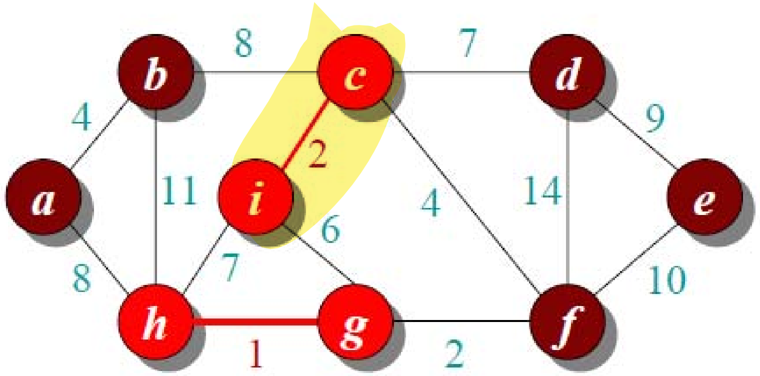
같은 set이 아니기 때문에

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (c, i) is safe.



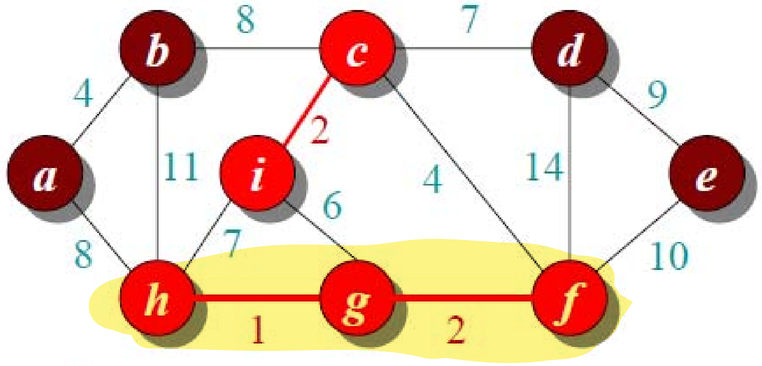
MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (f, g) is safe.



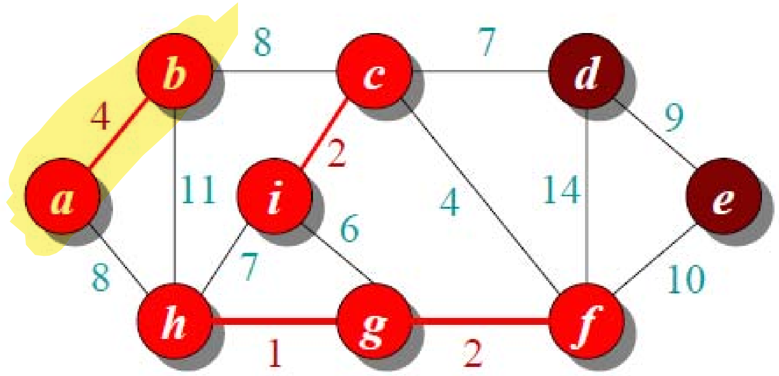
MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (a, b) is safe.



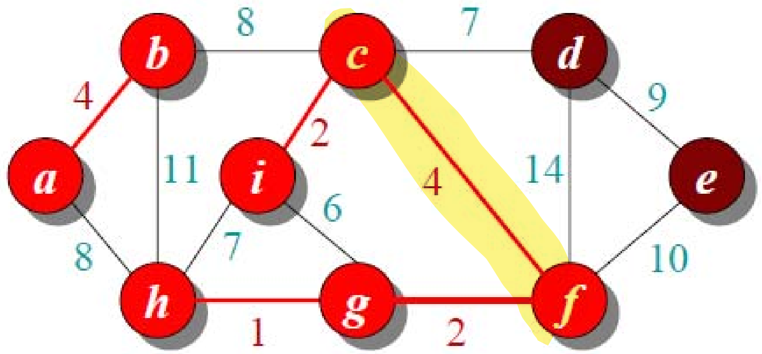
MST-KRUSKAL(G, w)
  $A \leftarrow \emptyset$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal’s algorithm

- Edge (c, f) is safe.



MST-KRUSKAL(G, w)
  $A \leftarrow \emptyset$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

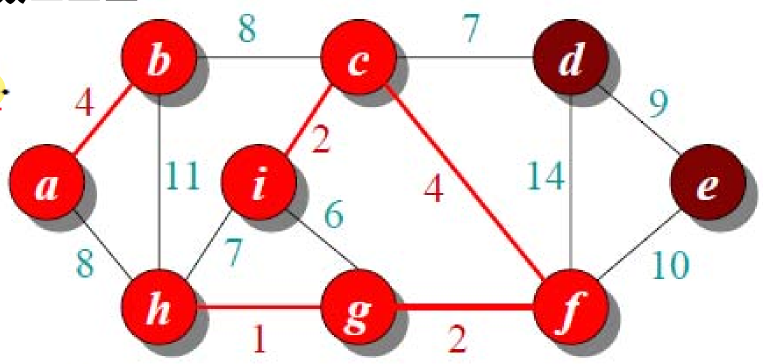
Disjoint-set collection:

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

Kruskal's algorithm

같은 set에 포함되어 있으므로

- Edge (g, i) is unsafe.



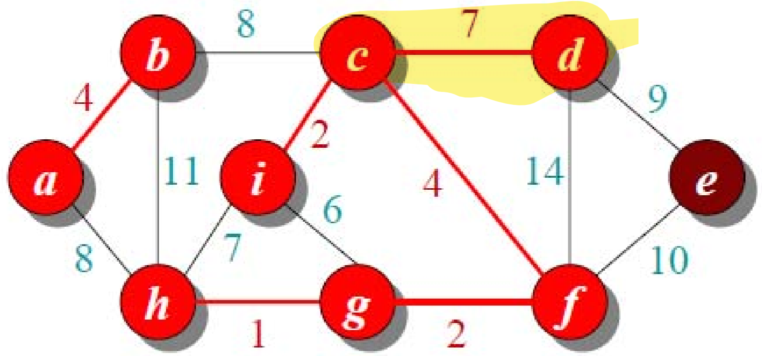
MST-KRUSKAL(G, w)
  $A \leftarrow \emptyset$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
   if FIND-SET$(u) \neq$ FIND-SET(v) **then**
     $A \leftarrow A \cup \{(u, v)\}$
     UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (c, d) is safe.



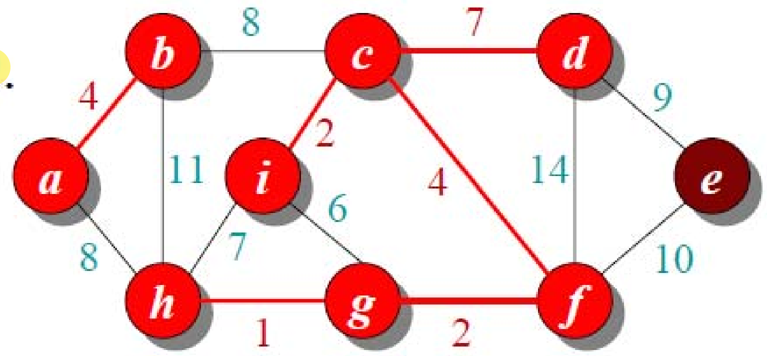
MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (h, i) is unsafe.



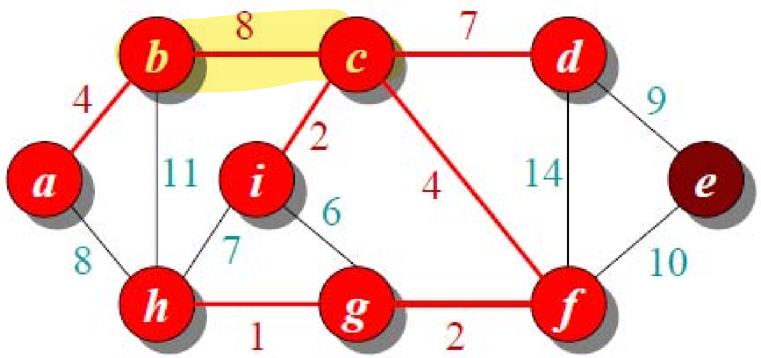
MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (b, c) is safe.



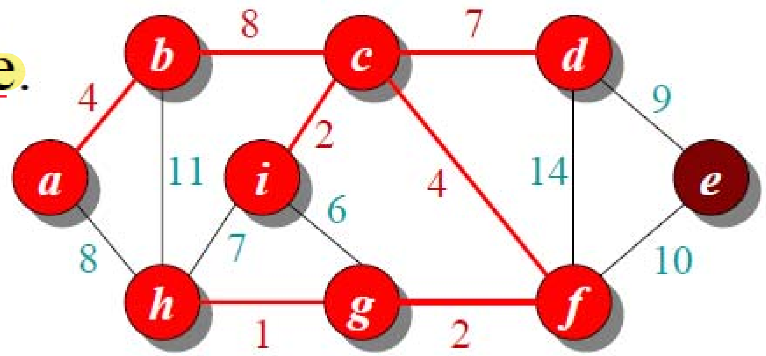
MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (a, h) is unsafe.



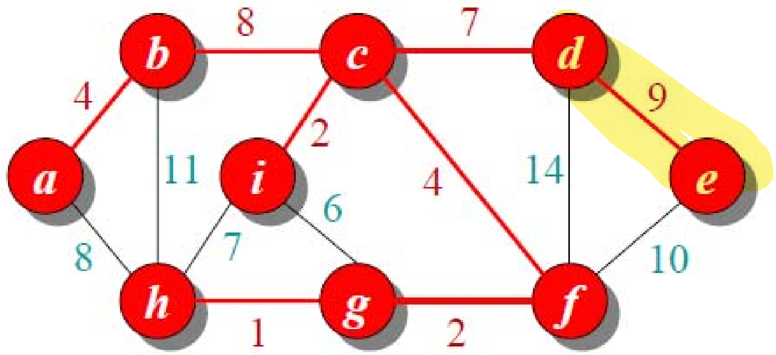
MST-KRUSKAL(G, w)
  $A \leftarrow \emptyset$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (d, e) is safe.



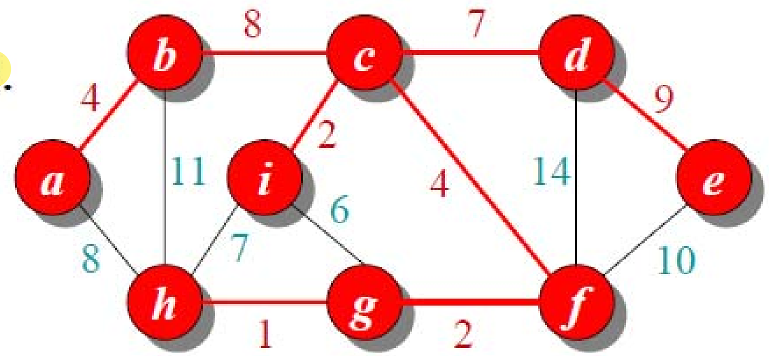
MST-KRUSKAL(G, w)
  $A \leftarrow \varnothing$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
   if FIND-SET$(u) \neq$ FIND-SET(v) **then**
     $A \leftarrow A \cup \{(u, v)\}$
     UNION(u, v)
  return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (e, f) is unsafe.



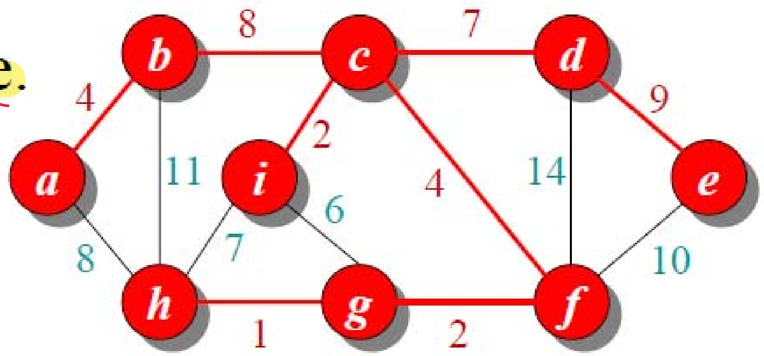

```
MST-KRUSKAL(G, w)
  A ← ∅
  for each vertex v ∈ V[G] do
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge (u, v) ∈ E, in sorted order, do
    if FIND-SET(u) ≠ FIND-SET(v) then
      A ← A ∪ {(u, v)}
      UNION(u, v)
  return A
```

Disjoint-set collection:

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

Kruskal’s algorithm

- Edge (b, h) is unsafe.



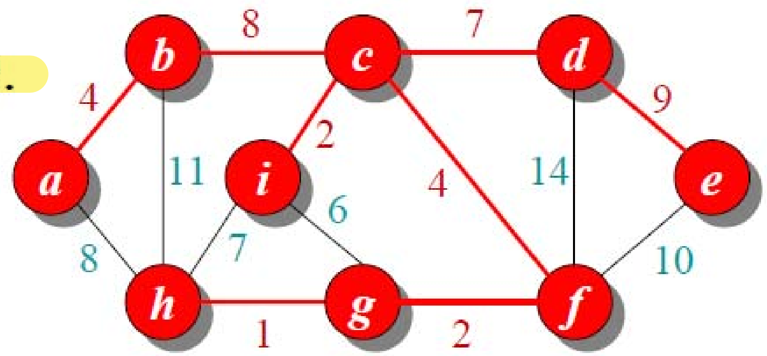
MST-KRUSKAL(G, w)
$A \leftarrow \emptyset$
for each vertex $v \in V[G]$ **do**
MAKE-SET(v)
sort E in nondecreasing order of w values
for each edge $(u, v) \in E$, in sorted order, **do**
if FIND-SET$(u) \neq$ FIND-SET(v) **then**
$A \leftarrow A \cup \{(u, v)\}$
UNION(u, v)
return A

Disjoint-set collection:

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

Kruskal's algorithm

- Edge (d, f) is unsafe.



MST-KRUSKAL(G, w)
  $A \leftarrow \emptyset$
  for each vertex $v \in V[G]$ **do**
    MAKE-SET(v)
  sort E in nondecreasing order of w values
  for each edge $(u, v) \in E$, in sorted order, **do**
    if FIND-SET$(u) \neq$ FIND-SET(v) **then**
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  return A

Disjoint-set collection:

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

MST-KRUSKAL Pseudocode

- MST-KRUSKAL(G, w)

1. A = { }
2. For each vectex v G.V
3.   MAKE-SET(v)
4.   sort the edges of G.E into nondecreasing order by weight w
5. For each edge (u, v) G.E, taken in nondecreasing order by weight
6.   if FIND-SET(u) ≠ FIND-SET(v)   같은 트리에 속하지 않는다면
7.   A = A U {(u,v)}
8.   UNION(u,v)
9. return A

Kruskal's algorithm

Theorem On a disjoint-set forest with union by rank and path compression, a sequence of m MAKE-SET, FIND-SET, and UNION operations, n *of* which are MAKE-SET operations, take $O(m\ \alpha(n))$ time, where α is a very slowly growing function.

Kruskal's algorithm makes $|V|$ MAKE-SET operations, $2|E|$ FIND-SET operations, and $|V| - 1$ UNION operations. Since G is connected, $|E| \geq |V| - 1$, and so the total number of disjoint set operations is $2|E|+2|V|-1 = O(E)$. Running time is thus $O(E\ \alpha(V)) = O(E \lg V) = O(E \lg E)$.

Problem 9: Saving ink

- Susan likes to make a line drawing with ink. There‘re several dots on drawing paper. Your job is to tell Susan how to connect the dots so as to minimize the amount of ink used.
  - Susan connects the dots by drawing straight lines between pairs, possibly lifting the pen between lines.
  - When Susan is done there must be a sequence of connected lines from any dot to any other dot.
- Test using 3 different data sets (test cases).

Problem 9: Saving ink

- Input
  - The input begins with a single positive integer on a line by itself indicating the number of dots (0<n<30) on drawing paper. For each dots, a line follows; each following line contains two real numbers indicating the (x, y) coordinates of the dots.
- Output
  - Your program must print a single real number to two decimal places: the minimum total length of ink lines that can connect all the dots.

Problem 9: Saving ink

- Sample input
  - 3
  - 1.0 1.0
  - 2.0 2.0
  - 2.0 4.0

- Sample output
  - ✓3.41

Problem 9: Data sets for Saving ink

- Data sets

3	
1	1
2	2
2	4

3.41

4	
1	4
2	5
3	6
1	1

5.83

5	
1	1
1	2
2	1
2	2
5	5

7.24

THANK YOU