Lecture 5:
Greedy algorithm

Algorithm

Jeong-Hun Kim

Remind

- Divide and conquer
  - Three steps
    - Divide
      - Basecase
    - Conquer
    - Combine

- Representative algorithms
  - Merge sort algorithm, Strassen algorithm

- Recurrence formula
  - Recurrence relation, Substitute method, Master method

Table of Contents

Part 1

GREEDY ALGORITHMS

Greedy Algorithms

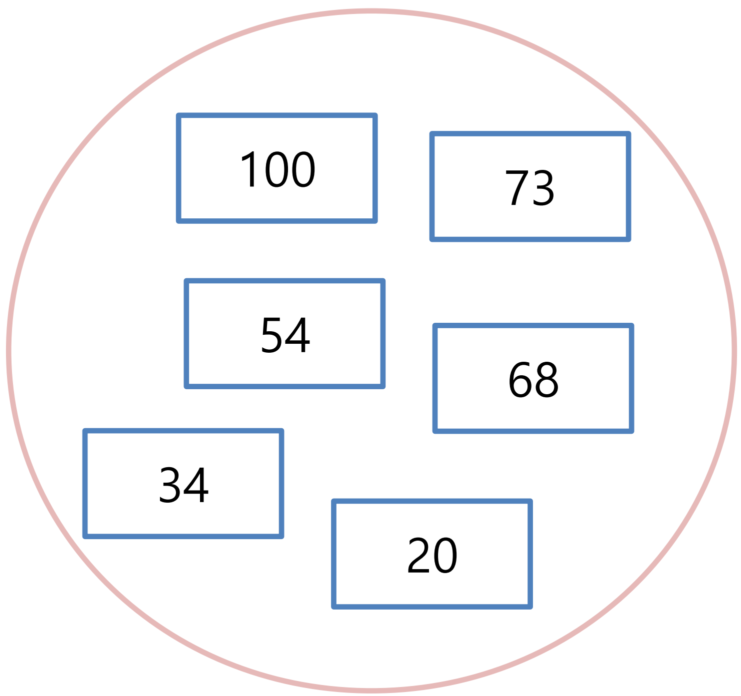
- What is the greedy algorithm?
  - An algorithm commonly used in the optimization problem
  - It calculates optimal solution at each step by considering current states
    - Locally-optimal choice
  - Doesn’t guarantee a global optimum for most problems
    - It (approximately) guarantees a global optimum under constraints

Part 2

WARMING UP: NUMBER SELECTION

Warming up: Number Selection

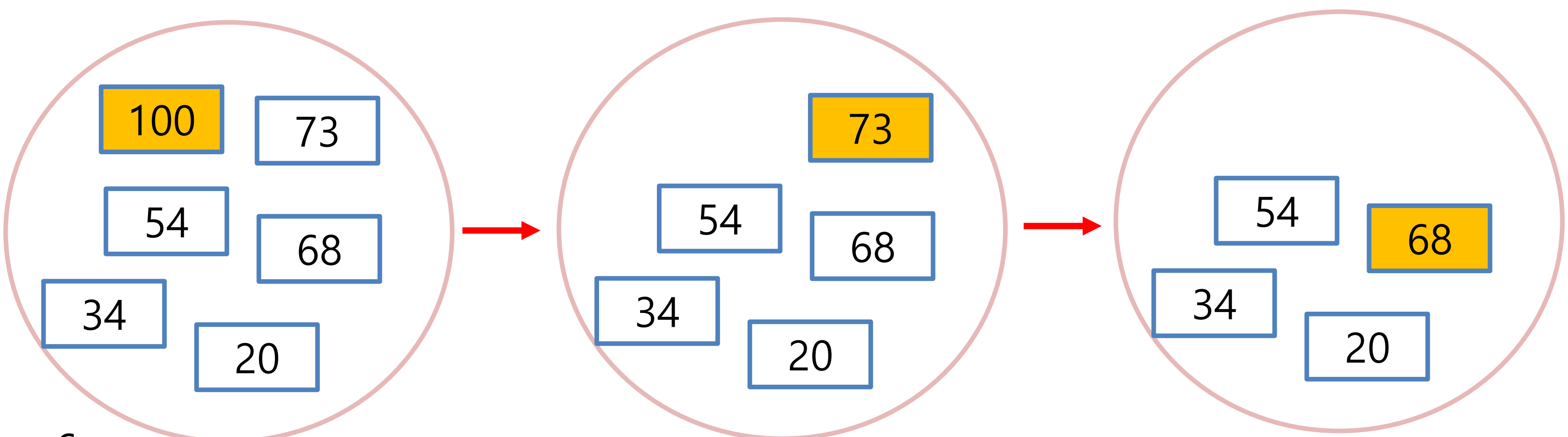
- Number selection problem
  - Basket A contains N numbers
  - Select x numbers to maximize the sum of the selected numbers ($x \leq N$)



N = 6
x = 3

Warming up: Number Selection

- Number selection problem (cont'd)
  - Basket A contains N numbers
  - Select x numbers to maximize the sum of the selected numbers ($x \leq N$)
  - Solution)
    - Greedy algorithm:
      - At each step, select the greatest number from A (repeat x times)



N = 6
x = 3

최종 선택 : {100, 73, 68}

Warming up: Number Selection

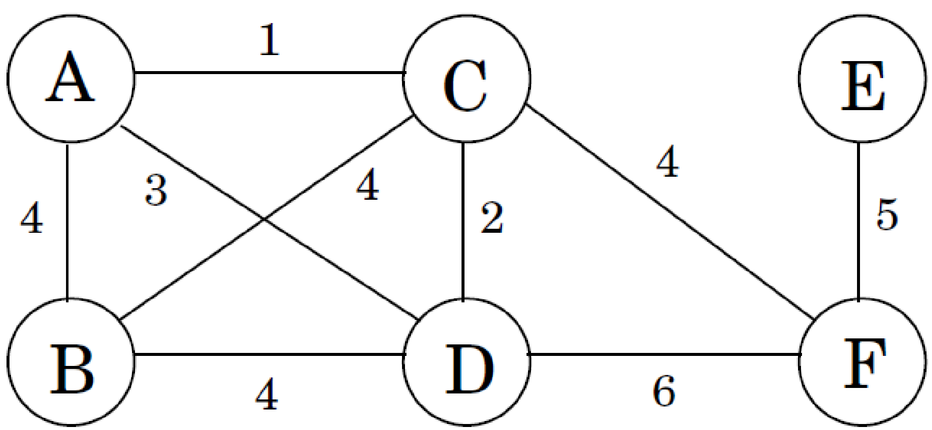
- Number selection problem (cont'd)
  - Solution)
    - Greedy algorithm:
      - At each step, select the greatest number from A (repeat x times)
  - Claim: the greedy algorithm is an optimal solution
  - Proof by contradiction
    - Let's consider a set $S=\{n_1, \ldots, n_x\}$ as the selected x numbers, and assume that the sum of the elements in S is not maximum
    - At the i-th step, there must exist the largest number, n_i, that has not been selected ($i \leq x$) $\leftarrow$ Contradiction

Part 3

MINIMUM SPANNING TREE

Minimum Spanning Tree

- Minimum spanning tree (MST) problem
  - Let G=(V(G), E(G)) be a weighted connected graph
    - |V(G)|=n, |E(G)|=m, w(e) is a cost (weight) of edge e ∈ E(G)
    - All costs are greater than zero
  - Find G′=(V(G), T) that satisfies the following conditions:
    - G′ is a connected graph
    - The sum of all costs in G′ is minimized



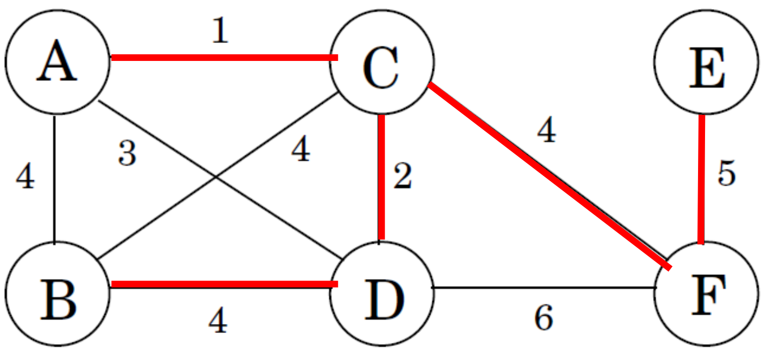
Minimum Spanning Tree

- Minimum spanning tree (MST) problem (cont'd)
  - Find G'=(V(G), T) that satisfies the following conditions:
    - G' is a connected graph
    - The sum of all costs in G' is minimized
  - When G' contains a cycle, removing any edge from the cycle will still result in G' being a connected graph
    - However, the sum of costs must decrease
  - Therefore, G' always becomes an acyclic + connected graph (= tree)
  - G' that satisfies the above conditions is referred to as a MST

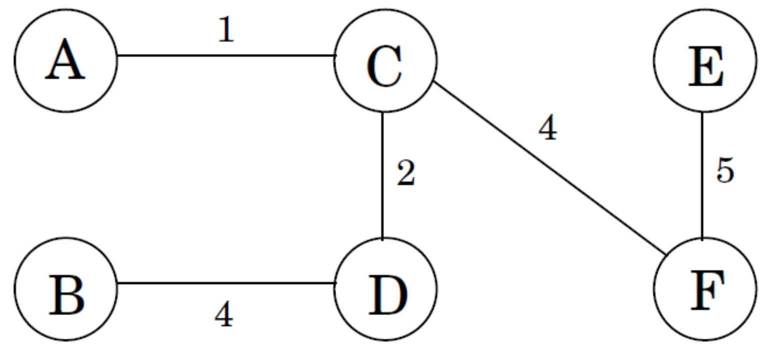
Minimum Spanning Tree

❖ Minimum spanning tree (MST) problem (cont'd)

- Example)



Graph G



MST of G

- Application) network design

Part 4

KRUSKAL'S ALGORITHM

Kruskal's Algorithm

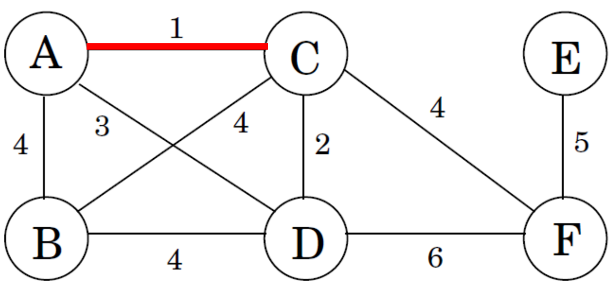
- Kruskal's algorithm
  - Representative solution of MST problem
  - Greedy algorithm
  - Methodology
    - At each step, add the edge with the smallest cost among the edges that, when added to T, do not create a cycle (feasible solution)
    - Repeat this process until V(G) is connected

Kruskal’s Algorithm

❖ Kruskal’s algorithm (cont’d)

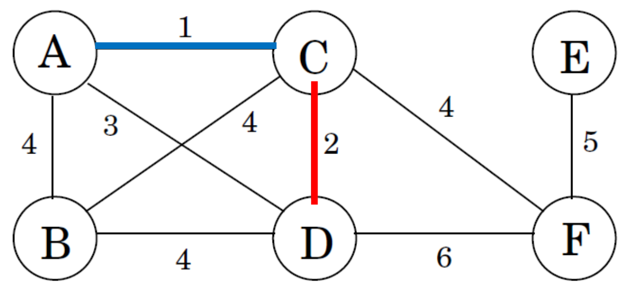
- Methodology
  - At each step, add the edge with the smallest cost among the edges that, when added to T, do not create a cycle (feasible solution)
  - Repeat this process until V(G) is connected

(1)



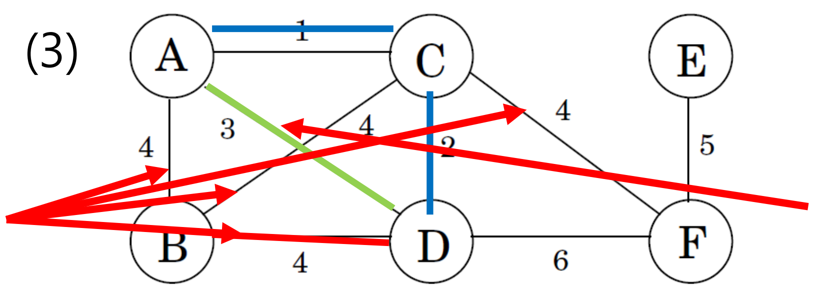
T = {(A, C)}

(2)



T = {(A, C), (C, D)}

(3)



T = {(A, C), (C, D), (B, D)}

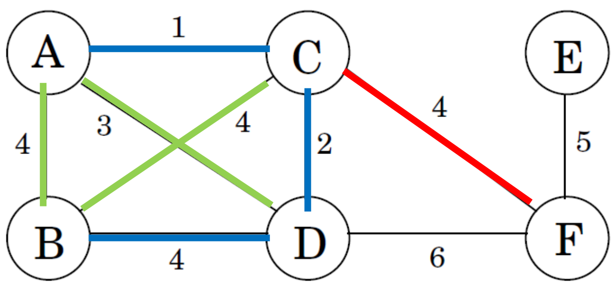
Freely order edges with the same cost
(in this case, there are multiple MSTs)

When adding the edge(A,D), it creates a cycle A-C-D, so this edge cannot be added

Kruskal’s Algorithm

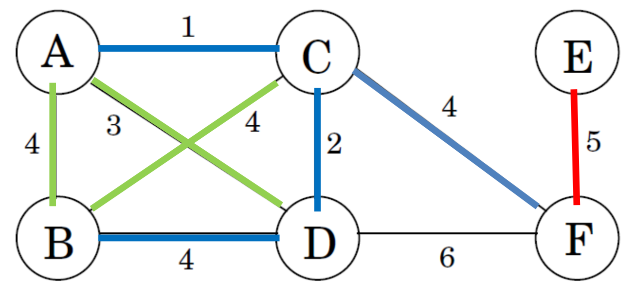
❖ Kruskal’s algorithm (cont’d)

(4)



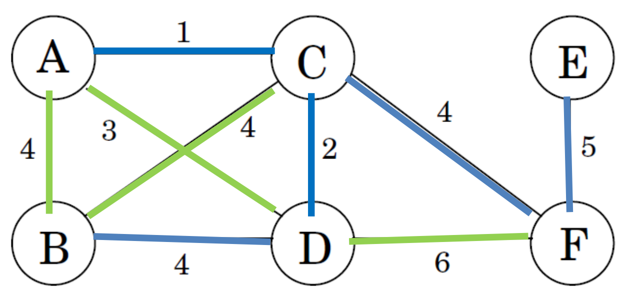
T = {(A, C), (C, D), (B, D), (C, F)}

(5)



T = {(A, C), (C, D), (B, D), (C, F), (E, F)}

(6)



Set of edges in MST

T = {(A, C), (C, D), (B, D), (C, F), (E, F)}

Kruskal’s Algorithm

- Kruskal’s algorithm (cont’d)
  - Lemma 1 (feasibility)
    - The result of Kruskal’s algorithm, G’=(V(G), T), is always a connected graph
  - Proof
    - Let’s assume that G’ is not connected, and V_1, V_2, …, V_r are the connected components of G’
    - Since G is connected, there is always at least one edge, e, connecting vertices from V_1 and vertices from $V(G) – V_1$
    - Adding this edge e will still keep the acyclic
    - Therefore, Kruskal’s algorithm will add edge e, and the same logic applies to V_2, …, V_r

Kruskal's Algorithm

- Kruskal's algorithm (cont'd)
  - Lemma 2 (optimality)
    - Kruskal's algorithm returns a unique MST when all the edges in G have distinct costs
  - Proof
    - Let T and T' be the edge sets for Kruskal's algorithm and the optimal solution, respectively
    - If T ≠ T', then there must be some edge e=(u, v) belonging to T'–T
    - In this case, the costs of the edges forming the path(u, v) in Kruskal's algorithm is all less than the cost of edge e
    - Therefore, the total cost of T' – {e} ∪ {e'} is less than the total cost of T', and these edges still form a spanning tree
      - The assumption that T' is optimal leads to a contradiction

Kruskal's Algorithm

- Kruskal's algorithm (cont'd)
  - Pseudo-code

```
Sort edges by weight and assume w1 ≤ w2 ≤ … ≤ wm          → (1)
T is empty /* T will store edges of a MST */
for i=1 to m do
   if (T + i is feasible (does not contain a cycle))       → (2)
   add i to T
return the set T
```

  - Time complexity
    - (1): $O(m \log m)$
    - (2) Perform DFS on $G' = (V(G), T)$
      - For each edge added, $O(n+m)$
    - Total: $O(m \log m) + O(m(n+m)) = O(m^2)$

Part 5

UNION-FIND DATA STRUCTURE

Union-Find Data Structure

- Motivation
  - Storing intermediate results as a forest, which is a set of trees
  - If the edge to be added connects two trees, T1 and T2, in the forest, then adding that edge merges T1 and T2 into one tree
  - An edge is not adeed if it is incident to two vertices within the same tree in the forest
    - Adding such an edge would result in an infeasible solution

Union-Find Data Structure

- Union-find data structure
  - A data structure for managing disjoint sets
  - Operations
    - makeset(x): create a set consisting of a single element x
    - find(x): return the set that contains x
    - union(x, y):
      - Combine the set containing x and the set containing y into a single set

Union-Find Data Structure

- Union-find data structure (cont'd)

  makeset

  - Storing intermediate results as a forest, which is a set of trees
  - If the edge to be added connects two trees, T1 and T2, in the forest, then adding that edge merges T1 and T2 into one tree → merge
  - An edge is not adeed if it is incident to two vertices within the same tree in the forest → find
    - Adding such an edge would result in an infeasible solution

Union-Find Data Structure

- Union-find data structure (cont'd)
  - Implementing Kruskal's algorithm using union-find
    - Each set consists of vertices from G, and initially, there are n sets, each containing a single, different vertex from G
    - If an edge is incident to vertices belonging to two different sets, then those two sets are unioned together
    - If an edge is incident to vertices belonging to the same set, then that edge is not added

Union-Find Data Structure

- Union-find data structure (cont'd)
  - Implementing Kruskal's algorithm using union-find
    - Pseudo-code

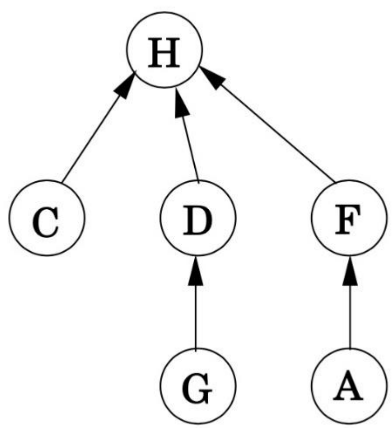
Figure 5.4 Kruskal's minimum spanning tree algorithm.

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

Union-Find Data Structure

❖ Union-find data structure (cont'd)

- Union by rank
  - Storing each vertex set using directed trees
  - Each node in the tree has an edge in the direction of its parent node, and the element of the root node becomes the set name



Set H = {H, C, D, F, G, A}

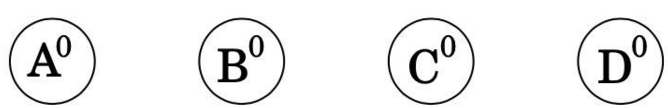
Rank 0 node = C, G, A
Rank 1 node = D , F
Rank 2 node = H

Union-Find Data Structure

- Union-find data structure (cont'd)
  - Union by rank
    - makeset (x)

<u>procedure makeset</u>(x)
$\pi(x) = x$
$\texttt{rank}(x) = 0$

    - π(x) represents the parent node of x
      - E.g., if x = π(x), x is the set name (root node)



makeset(A)
makeset(B)
makeset(C)
makeset(D)

Union-Find Data Structure

- Union-find data structure (cont'd)
  - Union by rank
    - find(x)

```
function find(x)
while x ≠ π(x):  x = π(x)
return x
```

    - The name of the set that includes x becomes the root node of the tree containing x
    - To reach the root node, find(x) traverse the parent pointers

Union-Find Data Structure

- Union-find data structure (cont'd)
  - Union by rank
    - union(x, y)

$\underline{\texttt{procedure union}}(x, y)$
$r_x = \texttt{find}(x)$
$r_y = \texttt{find}(y)$
$\texttt{if } r_x = r_y\texttt{:} \quad \texttt{return}$
$\texttt{if rank}(r_x) > \texttt{rank}(r_y)\texttt{:}$
$\quad \pi(r_y) = r_x$

Root node of r_y becomes r_x

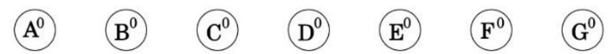
$\texttt{else:}$
$\quad \pi(r_x) = r_y$
$\quad \texttt{if rank}(r_x) = \texttt{rank}(r_y): \ \texttt{rank}(r_y) = \texttt{rank}(r_y) + 1$

If ry and rx have the same rank, the rank of the new set's root node increases by 1

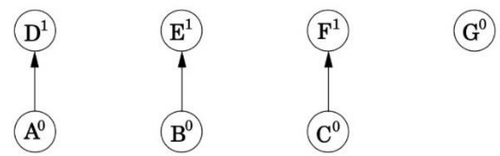
Union-Find Data Structure

- Union-find data structure (cont'd)
  - Union by rank

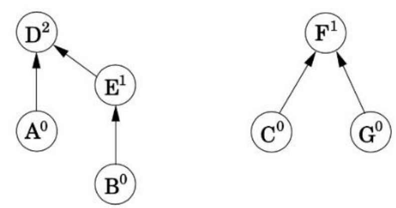
After makeset(A), makeset(B), $\ldots$, makeset(G):



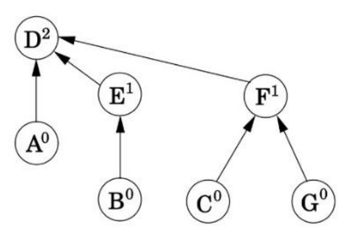
After union(A, D), union(B, E), union(C, F):



After union(C, G), union(E, A):



After union(B, G):

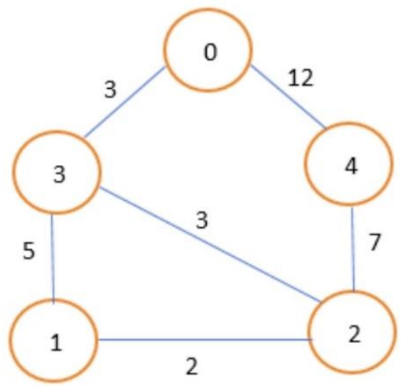


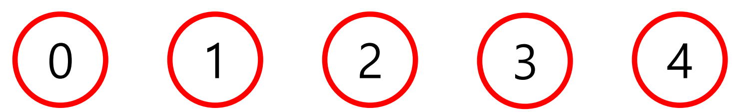
Union-Find Data Structure

❖ Union-find data structure (cont'd)

- Example of Kruskal's algorithm using union-find

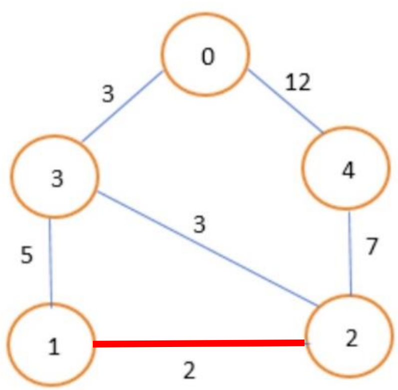
(1)

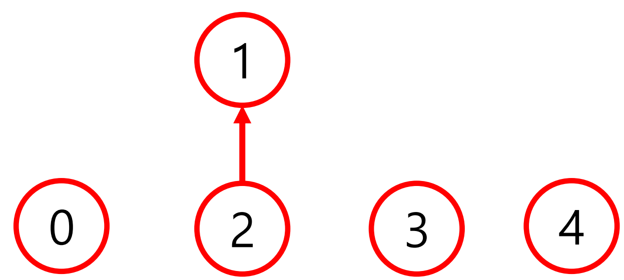




makeset (0), makeset(1), .. , makeset (4)

(2)





Since find(1) and find(2) are different, union(1, 2) is performed

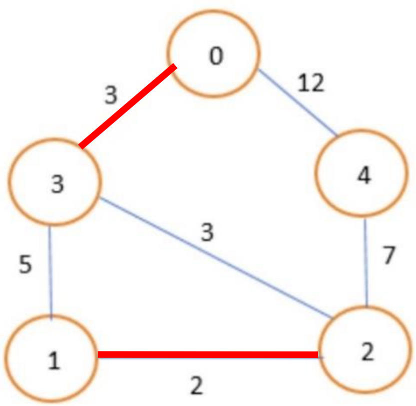
Add the edge(1, 2) to T

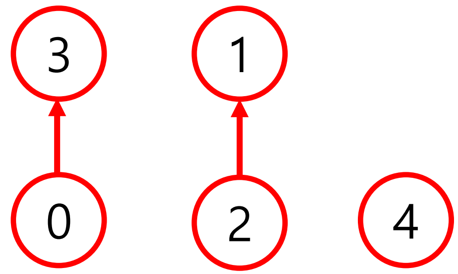
Union-Find Data Structure

❖ Union-find data structure (cont'd)

- Example of Kruskal's algorithm using union-find

(3)

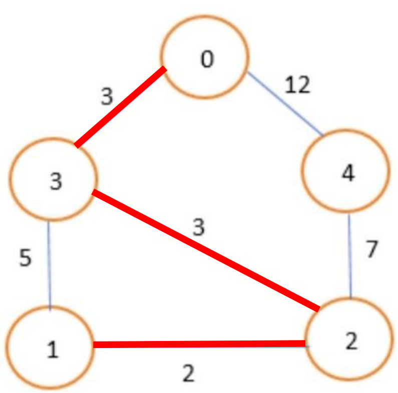


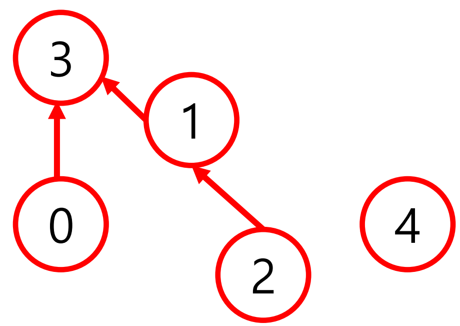


Since find(3) and find(0) are different, union(3, 0) is performed

Add the edge(3, 0) to T

(4)





Since find(3) and find(2) are different, union(3, 2) is performed

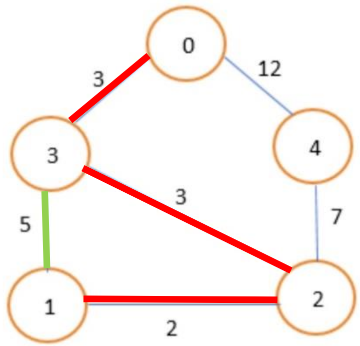
Add the edge(3, 2) to T

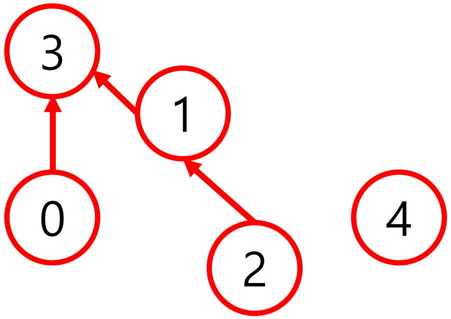
Union-Find Data Structure

❖ Union-find data structure (cont'd)

- Example of Kruskal's algorithm using union-find

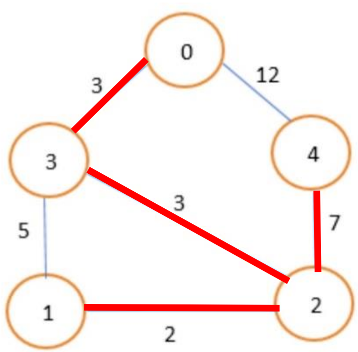
(5)

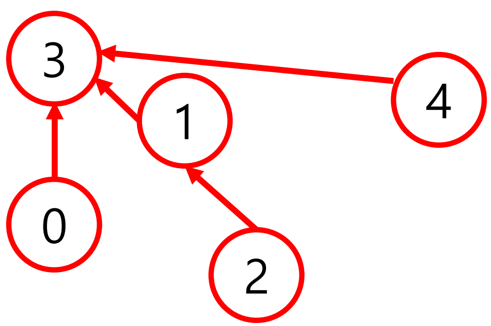




Since find(3) and find(1) are the same, do not add the edge (3, 1)

(6)





Since find(4) and find(2) are different, union(4, 2) is performed

Add the edge(4, 2) to T

Union-Find Data Structure

- Union-find data structure (cont’d)
  - Time complexity of Kruskal’s algorithm using union-find data structure
    - Property 1
      - No node has a rank smaller than its parent’s rank
    - Property 2
      - A root node with rank k has at least 2^k nodes as descendants
    - Property 3
      - A node with rank k has at most $n/2^k$ nodes

Union-Find Data Structure

- Union-find data structure (cont'd)
  - Time complexity of Kruskal's algorithm using union-find data structure

Figure 5.4 Kruskal's minimum spanning tree algorithm.

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)                          (1)

X = {}
Sort the edges E by weight             (2)
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X             (3)
      union(u, v)
```

- (1) makeset (n times): O(n)
- (2) edge sorting: m log m
- (3) find / union (up to m times): O(m log m)
- Total: O(m log m)

Part 6

PRIM’S ALGORITHM

Prim's Algorithm

- Prim's algorithm
  - Lemma 3. cut property
    - Edge e=(u, v) belongs to the edge set T of MST ↔

      e is the edge with the smallest cost among the edges that connect vertex set S containing u and the complement set V(G) - S

    - Proof
      - Let's assume that the MST has another e' connecting S and V(G)-S instead of e
      - Due to the properties of e, the total cost of T – {e'} ∪ {e} becomes smaller than the total cost of the original T
      - Therefore, the assumption that T is the edge set of the MST leads to a contradiction

Prim’s Algorithm

- Prim’s algorithm
  - Lemma 3. cut property
    - Edge e=(u, v) belongs to the edge set T of MST ↔

      e is the edge with the smallest cost among the edges that connect vertex set S containing u and the complement set V(G) - S

    - Proof
      - Let’s assume that the MST has another e’ connecting S and V(G)-S instead of e
      - Due to the properties of e, the total cost of T – {e’} ∪ {e} becomes smaller than the total cost of the original T
      - Therefore, the assumption that T is the edge set of the MST leads to a contradiction

Prim's Algorithm

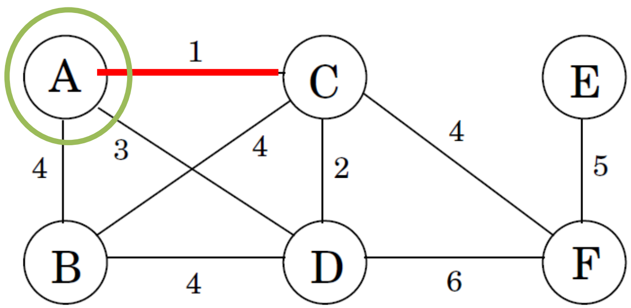
- Prim's algorithm
  - The cut property states that for any arbitrary vertex set $S \subset V(G)$, the MST always includes the edge with the smallest cost among the edges that connect S and V(G) – S
  - Therefore, by gradually adding vertices to S while satisfying the cut property by selecting edges, MST can be constructed

  - Methodology
    - $S=\{v\}$, $v \in V(G)$
    - Add $e=(v, w)$, which has minimum cost, into T
      - $w \in V(G) - S$
    - Add w into S
    - Repeat above process until $S = V(G)$

Prim's Algorithm

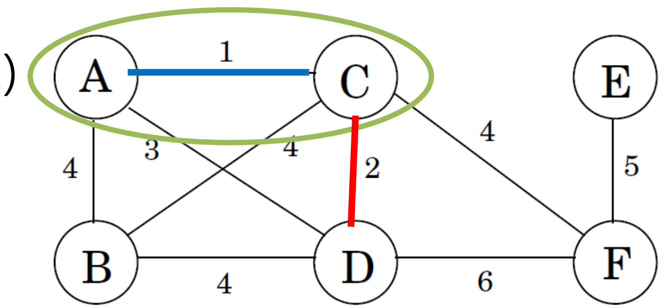
❖ Prim's algorithm

(1)



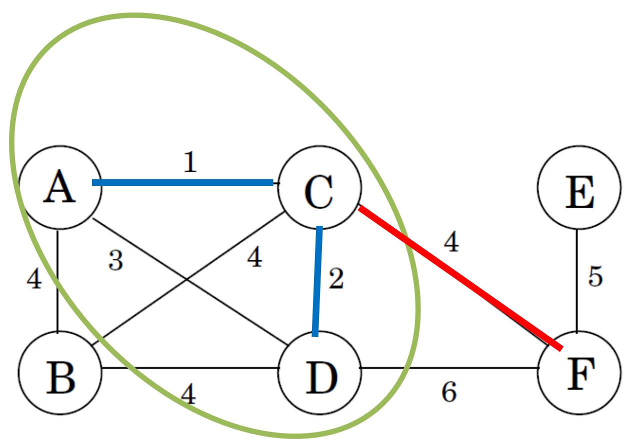
S = {(A)}

(2)



S = {(A, C)}

(3)

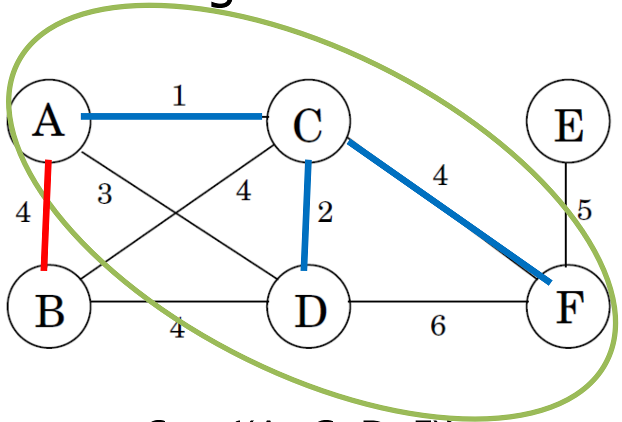


S = {(A, C, D)}

Prim's Algorithm

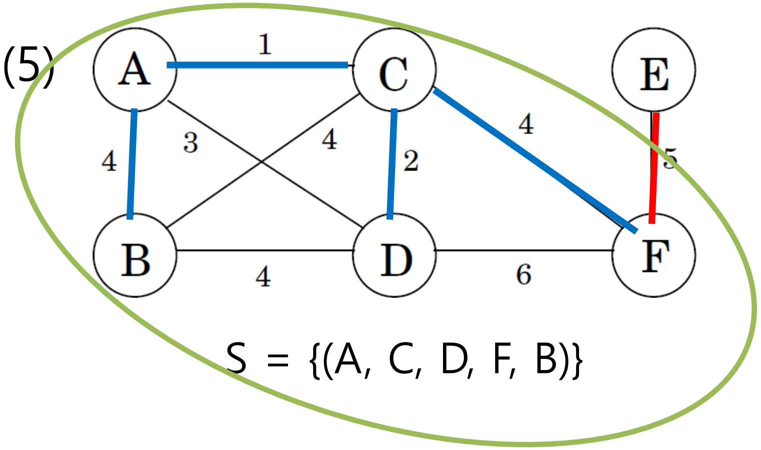
❖ Prim's algorithm

(4)



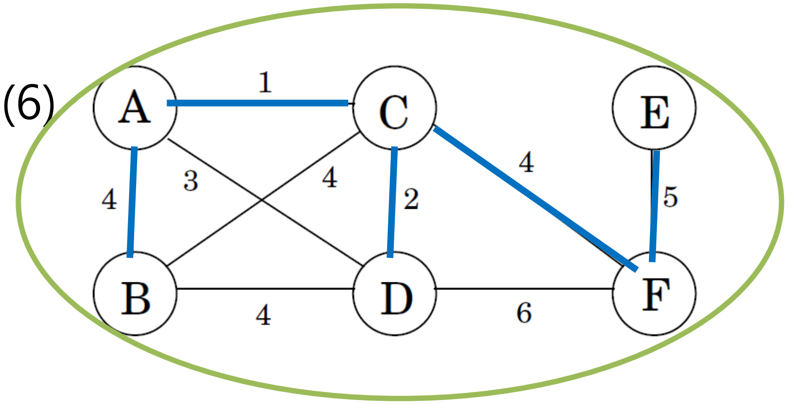
S = {(A, C, D, F)}

(5)



S = {(A, C, D, F, B)}

(6)



S = {(A, C, D, F, B, E)}

Prim's Algorithm

❖ Prim's algorithm

$S = \{v_1\}$
$T = \varnothing$ (* T will store edges of a MST *)
while $S \neq V(G)$ (1)
    pick $e = (v,w)$ in $E(G)$ such that, $v \in S$ and $w \in V(G) - S$, and
    e has minimum cost (2)
    $T = T \cup \{e\}$
    $S = S \cup \{w\}$
return the set T

- Time complexity
    - (1): n iterations
    - (2): O(m)
    - Total: O(nm)

Summary

- Greedy algorithm
  - Number selection problem
  - Minimum spanning tree (MST) problem

- MST problem
  - Kruskal's algorithm
    - Improved algorithm: union-find data structure
  - Prim's algorithm

Questions?

SEE YOU NEXT TIME!