

Data Structure

# 자료구조



## 가중치 그래프와 최소비용 신장 트리



한국기술교육대학교  
온라인평생교육원

## 학습내용

- 가중치 그래프
- Kruskal의 최소비용 신장 트리
- Prim의 최소비용 신장 트리

## 학습목표

- 가중치 그래프의 개념을 설명할 수 있다.
- 최소비용 신장 트리를 설명할 수 있다.
- Prim과 Kruskal의 최소비용 신장 트리 알고리즘을 설명할 수 있다.

# 가중치 그래프



## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 1 가중치 그래프의 개념

##### 가중치 그래프(Weighted Graph)

- 간선에 가중치가 할당된 그래프
- 예 주요 도시들 사이의 도로망

▶ 수학적 표현

$$G = (V, E, w)$$

w: 비용, 가중치(Weight), 길이

경로 p의 길이

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$



## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 1 가중치 그래프의 개념

● 응용 분야

네트워크 구축

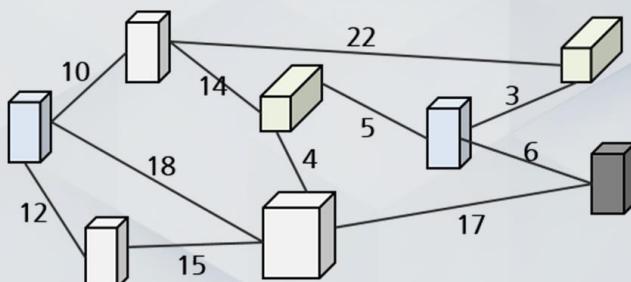
사이트 사이의 연결 방법에 따른 속도 등을 표현

도로망 구축

각 도시 사이의 연결 관계와 소요 시간, 비용, 거리 등을 표현

##### 문제

- 최소비용 신장 트리
- 최단 거리 문제





# 가중치 그래프

## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 1 가중치 그래프의 개념

##### ○ 가중치 그래프의 표현 방법

###### 인접 행렬

- 행렬의 각 요소에 0이나 1이 아니라 가중치 값을 저장
- 만약 간선이 없으면? 무한대 값의 가중치로 표시

###### 인접 리스트

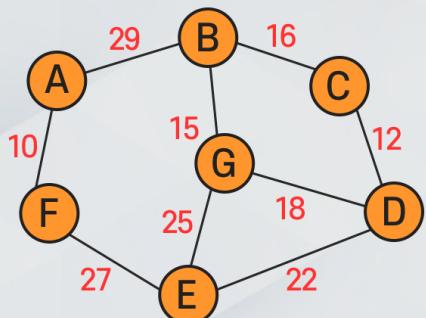
- 인접 리스트의 각 노드에 가중치를 추가

## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 2 인접 행렬을 이용한 표현

##### ○ 인접 행렬을 이용한 가중치 그래프의 표현



인접 행렬 표현 →

	A	B	C	D	E	F	G
A	0	29	$\infty$	$\infty$	$\infty$	10	$\infty$
B	29	0	16	$\infty$	$\infty$	$\infty$	15
C	$\infty$	16	0	12	$\infty$	$\infty$	$\infty$
D	$\infty$	$\infty$	12	0	22	$\infty$	18
E	$\infty$	$\infty$	$\infty$	22	0	27	25
F	10	$\infty$	$\infty$	$\infty$	27	0	$\infty$
G	$\infty$	15	$\infty$	18	25	$\infty$	0

무방향 그래프

대각선 기준으로 대칭 행렬



# 가중치 그래프

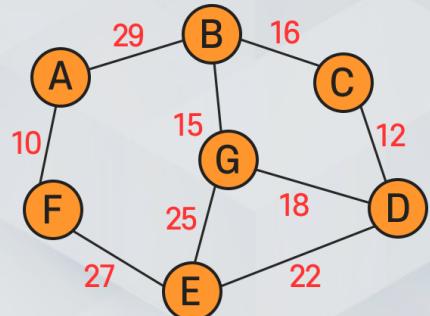
## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 2 인접 행렬을 이용한 표현

##### 파이썬에서 행렬(2차원 배열)의 표현

- ▶ 리스트의 리스트를 사용
- ▶ 간선이 없으면 None
- ▶ 인접 행렬로 표현한 가중치 그래프의 예



```

vertex = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G' ]           # 정점 리스트
weight = [ [ None, 29, None, None, None, 10, None ],      # 인접 행렬
          [ 29, None, 16, None, None, 15, None ],
          [ None, 16, None, 12, None, None, None ],
          [ None, None, 12, None, 22, None, 18 ],
          [ None, None, None, 22, None, 27, 25 ],
          [ 10, None, None, None, 27, None, None ],
          [ None, 15, None, 18, 25, None, None ] ]
  
```

## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 2 인접 행렬을 이용한 표현

##### 인접 행렬: 간단한 연산

##### 예 인접 행렬에서의 가중치의 합 계산

```

def weightSum(vlist, W):                      # 정점 리스트, 인접 행렬
    sum = 0                                     # 가중치의 합
    for i in range(len(vlist)):                  # 모든 정점에 대해 (i: 0, ..., N-1)
        for j in range(i+1, len(vlist)):          # 하나의 행에 대해 (삼각 영역)
            if W[i][j] != None:                   # 만약 간선이 있으면
                sum += W[i][j]                     # sum에 추가
    return sum                                    # 전체 가중치 합을 반환

print('AM : weight sum = ', weightSum(vertex, weight))
  
```

C:\#WINDOWS#\system32#\cmd.exe

AM : weight sum = 174



# 가중치 그래프

## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 2 인접 행렬을 이용한 표현

##### ● 인접 행렬: 간단한 연산

**예** 인접 행렬에서의 모든 간선 출력

```
def printAllEdges(vlist, W):
    for i in range(len(vlist)):
        for j in range(i+1, len(W[i])):
            if W[i][j] != None and W[i][j] != 0:
                print( "(%s,%s,%d)" % (vlist[i], vlist[j], W[i][j]), end=' ')
    print()

printAllEdges(vertex, weight)
```

## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 3 인접 리스트를 이용한 표현

##### ● 인접 리스트를 이용한 가중치 그래프의 표현

인접 리스트

연결 리스트, 파이썬 리스트(배열 구조), 집합 등

간선에 가중치 정보도 포함되어야 함(튜플 사용)

딕셔너리 등도 이용 가능

# 가중치 그래프



## 가중치 그래프와 최소비용 신장 트리

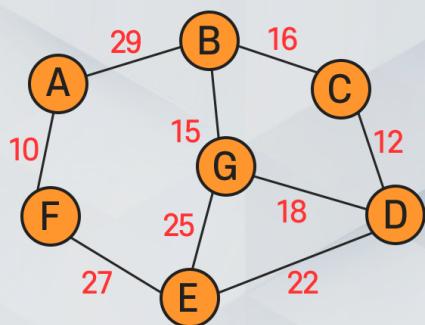
### 가중치 그래프

#### 3 인접 리스트를 이용한 표현

##### ● 인접 리스트를 이용한 표현 예

▶ 딕셔너리, 집합, 튜플, 리스트를 이용한 그래프 표현

```
graphAL = { 'A' : {('B',29),('F',10)},  
            'B' : {('A',29),('C',16), ('G',15)},  
            'C' : {('B',16),('D',12)},  
            'D' : {('C',12),('E',22), ('G',18)},  
            'E' : {('D',22),('F',27), ('G',25)},  
            'F' : {('A',10),('E',27)},  
            'G' : {('B',15),('D',18), ('E',25)} }
```



## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 3 인접 리스트를 이용한 표현

##### ● 인접 행렬: 간단한 연산

##### 예 인접 리스트에서의 가중치의 합 계산

```
def weightSum(graph):          # 가중치의 총 합을 구하는 함수  
    sum = 0  
    for v in graph:  
        for e in graph[v]:  
            sum += e[1]  
    return sum // 2              # 그레프의 모든 정점 v에 대해: 'A', 'B', ...  
                                # v의 모든 간선 e에 대해: ('B', 29), ...  
                                # sum에 추가  
                                # 하나의 간선이 두 번 더해지므로 2로 나눔  
  
print('AL : weight sum = ', weightSum(graphAL))
```

```
C:\#WINDOWS\system32#cmd.exe  
AM : weight sum = 174
```



# 가중치 그래프

## 가중치 그래프와 최소비용 신장 트리

### 가중치 그래프

#### 3 인접 리스트를 이용한 표현

##### ● 인접 행렬: 간단한 연산

##### 예 인접 리스트에서의 모든 간선 출력

```

def printAllEdges(graph):
    for v in graph:
        for e in graph[v]:
            if v <= e[0]:
                print("%s,%s,%d" % (v,e[0],e[1]), end=' ')
printAllEdges(graphAL)

```

C:\WINDOWS\system32\cmd.exe - □ ×  
(A,F,10) (A,B,29) (B,G,15) (B,C,16) (C,D,12) (D,G,18) (D,E,22) (E,G,25) (E,F,27)



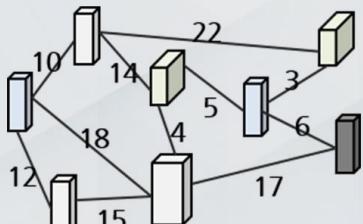
# Kruskal의 최소비용 신장 트리

## 가중치 그래프와 최소비용 신장 트리

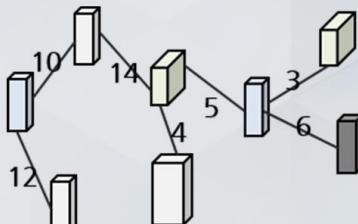
### Kruskal의 최소비용 신장 트리

#### 1 최소비용 신장 트리

신장 트리(Spanning Tree)	최소비용 신장 트리 (Minimum Spanning Tree, MST)
<ul style="list-style-type: none"> <li>연결 그래프 내의 모든 정점을 포함하는 트리</li> <li><math>(n-1)</math>개의 간선 사용, 사이클이 포함되면 안 됨</li> </ul>	<ul style="list-style-type: none"> <li>간선들의 가중치 합이 최소인 신장 트리</li> </ul>



(a) 사이트 사이의 연결 비용



(b) 최소비용 신장 트리의 예

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 1 최소비용 신장 트리

##### » MST의 응용

도로, 통신, 배관 건설	모두 연결하면서 길이/비용을 최소화
전기 회로	단자를 모두 연결하면서 전선의 길이를 최소화

##### » MST 알고리즘

Kruskal 알고리즘

Prim 알고리즘

# Kruskal의 최소비용 신장 트리



## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 2 Kruskal의 MST 알고리즘

##### 탐욕적 기법(Greedy Method)

- “그 순간에 최적”을 반복적으로 선택
- 항상 최적의 해답을 주지는 않음

비싸다고 느끼한 것을  
먼저 먹으면 결국  
많이 못 먹을 수도 있는데…



본전을 뽑으려면  
가장 비싼 것부터  
먼저 먹어야지…

##### 탐욕적인 방법

결정을 할 때마다 항상 “그 순간에 최적”이라고 생각되는 것을 선택

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 2 Kruskal의 MST 알고리즘

##### Kruskal의 MST 알고리즘

- 탐욕적 기법(Greedy Method) 사용
- “**그 순간에 최적**”이라고 생각되는 것을 선택
- 각 단계에서 최선의 답을 선택 → 최종적인 해답에 도달
- 항상 최적의 해답을 주는지 검증 필요  
→ Kruskal의 MST 알고리즘은 최적의 해답임이 증명



# Kruskal의 최소비용 신장 트리

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 2 Kruskal의 MST 알고리즘

### ● Kruskal의 최소비용 신장 트리 알고리즘

Kruskal()

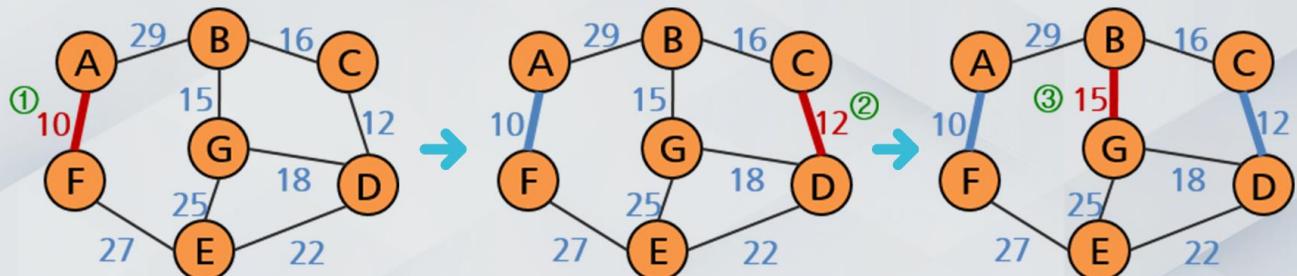
- 1 그래프의 모든 간선을 가중치에 따라 오름차순으로 정렬
- 2 가장 가중치가 가장 작은 간선  $e$ 를 뽑기
- 3  $e$ 를 신장 트리에 넣었을 때 사이클이 생기면 넣지 않고 2번으로 이동
- 4 사이클이 생기지 않으면 최소 신장 트리에 삽입
- 5  $N-1$ 개의 간선이 삽입될 때까지 2번으로 이동

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 2 Kruskal의 MST 알고리즘

### ● Kruskal의 알고리즘 처리 과정



AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29



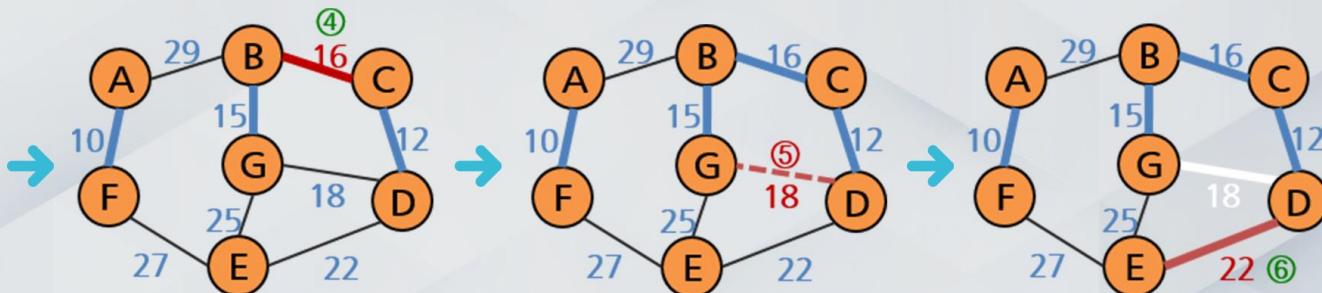
# Kruskal의 최소비용 신장 트리

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 2 Kruskal의 MST 알고리즘

● Kruskal의 알고리즘 처리 과정



AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

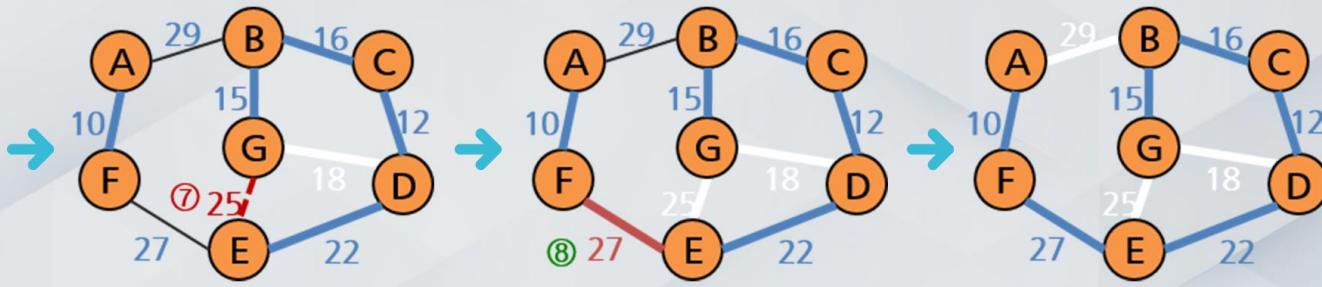
AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 2 Kruskal의 MST 알고리즘

● Kruskal의 알고리즘 처리 과정



AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29

AF	CD	BG	BC	DG	DE	EG	EF	AB
10	12	15	16	18	22	25	27	29



# Kruskal의 최소비용 신장 트리

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

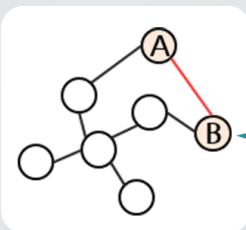
## 2 Kruskal의 MST 알고리즘

### ● 사이클 검사

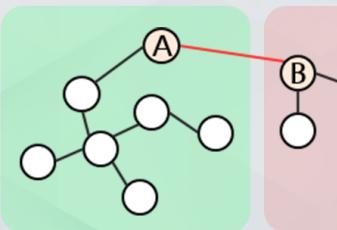
▶ 현재 신장 트리에 간선 (A, B)를 추가할 때 사이클이 생길까?

A와 B가 같은 연결 그래프에 속하는 경우 → 사이클 발생 O

A와 B가 다른 연결 그래프에 속하는 경우 → 사이클 발생 X



A와 B가 같은  
집합에 속함  
→ 사이클 발생 O



A와 B가 다른  
집합에 속함  
→ 사이클 발생 X

→ union-find 알고리즘 사용

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 3 Union-Find 알고리즘

# union-find 알고리즘

서로소(Disjoint)인 부분 집합을 다루는 알고리즘

### Union 연산

- 두 집합의 합집합을 만드는 연산
- $\text{union}(x, y)$ : 원소 x와 y가 속한 집합을 하나로 뭉침

### Find 연산

- 원소가 속한 집합을 찾는 연산
- 원소 x가 속한 집합  
→ 보통은 “대표 원소” 반환



# Kruskal의 최소비용 신장 트리

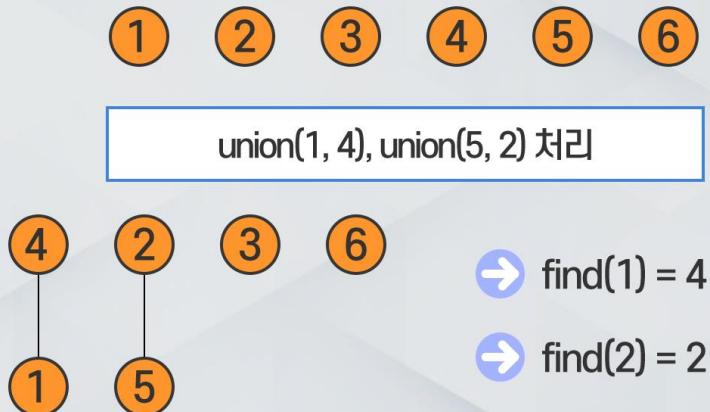
가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 3 Union-Find 알고리즘

### ◎ union과 find 연산 예

예 서로소인 부분 집합 {1}, {2}, {3}, {4}, {5}, {6}



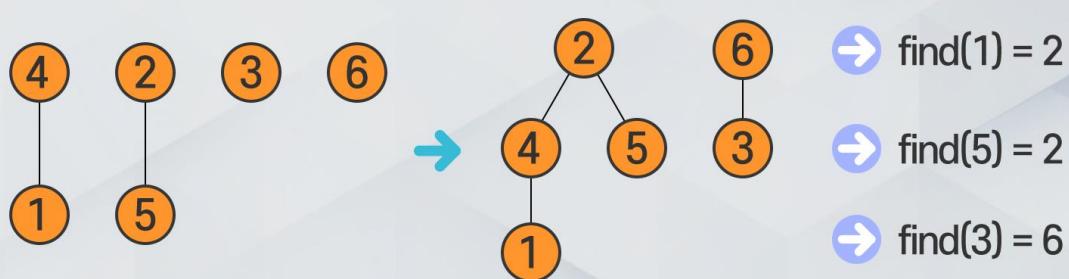
가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 3 Union-Find 알고리즘

### ◎ union과 find 연산 예

예 서로소인 부분 집합 {1}, {2}, {3}, {4}, {5}, {6}





# Kruskal의 최소비용 신장 트리

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 3 Union-Find 알고리즘

##### ◎ union-find 구현

```

parent = []                      # 각 노드의 부모노드 인덱스
set_size = 0                       # 전체 부분 집합의 개수

def init_set(nSets):
    global set_size, parent
    set_size = nSets
    for i in range(nSets):
        parent.append(-1)          # 맨 처음에는 각각이 고유의 집합이 됨
    
```

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 3 Union-Find 알고리즘

##### ◎ union-find 구현

```

def find(id):                  # 정점 id가 속한 집합의 대표 번호 탐색
    while (parent[id] >= 0):   # 부모가 있으면(대표 노드가 아님)
        id = parent[id]        # 부모로 올라감(id 갱신)
    return id                  # 최종 id는 트리의 루트(부모가 -1)가 됨

def union(s1, s2):             # 두 서로소 집합을 병합하는 연산
    global set_size
    parent[s1] = s2            # 노드 s1을 s2에 병합시킴
    set_size = set_size - 1     # 집합의 개수가 줄어듦
    
```



# Kruskal의 최소비용 신장 트리

가중치 그래프와 최소비용 신장 트리

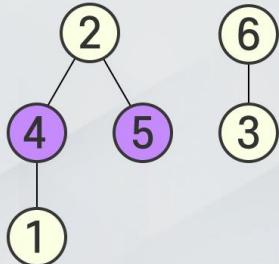
Kruskal의 최소비용 신장 트리

## 3 Union-Find 알고리즘

### ◎ 사이클 검사 예

▶ 간선 (4, 5)의 사이클을 검사

$\text{find}(4) == \text{find}(5)$  → 사이클 발생 → MST에 삽입 불가



$\text{find}(4) \rightarrow 2$   
 $\text{find}(5) \rightarrow 2$   
 같은 집합에 속하므로  
 간선 (4, 5)를 삽입할 수 없음

가중치 그래프와 최소비용 신장 트리

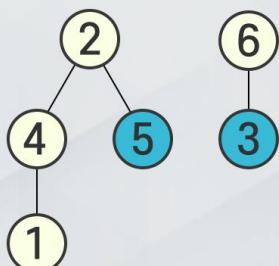
Kruskal의 최소비용 신장 트리

## 3 Union-Find 알고리즘

### ◎ 사이클 검사 예

▶ 간선 (3, 5)의 사이클을 검사

$\text{find}(3) != \text{find}(5)$  → 사이클 발생 X → MST에 삽입 가능



$\text{find}(3) \rightarrow 6$   
 $\text{find}(5) \rightarrow 2$   
 같은 집합에 속하므로  
 간선 (3, 5)를 삽입할 수 없음

# Kruskal의 최소비용 신장 트리



## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 4 Kruskal 알고리즘 구현

##### ● 초기화 과정

```

def MSTKruskal(vertex, adj):
    vsize = len(vertex)                                # 정점의 개수
    init_set(vsize)                                    # 정점 집합 초기화
    eList = []                                         # 간선 리스트

    for i in range(vsize-1):                           # 모든 간선을 리스트에 넣음
        for j in range(i+1, vsize):
            if adj[i][j] != None:
                eList.append((i,j,adj[i][j]))           # 튜플로 저장

```

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 4 Kruskal 알고리즘 구현

##### ● 간선 선택 과정(n-1개)

```

# (1) 간선 리스트를 가중치의 내림차순으로 정렬
eList.sort(key= lambda e : e[2], reverse=True)

edgeAccepted = 0
while (edgeAccepted < vsize - 1):                  # n - 1개의 간선
    # (2) 가장 작은 가중치를 가진 간선을 뽑는다
    e = eList.pop(-1)                                 # 가장 작은 가중치를 가진 간선
    uset = find(e[0])                                # 두 정점이 속한 집합 번호
    vset = find(e[1])

```



# Kruskal의 최소비용 신장 트리

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 4 Kruskal 알고리즘 구현

##### ● 간선 선택 과정(n-1개)

```

# (3) 사이클이 생기면 루프를 다시 반복함
# (4) 사이클이 생기지 않으면 MST에 추가
if uset != vset:                                # 두 정점이 다른 집합 원소이면
    print("간선 추가 : (%s, %s, %d)" %      # MST에 간선 추가
          (vertex[e[0]], vertex[e[1]], e[2]))
    union(uset, vset)                            # 두 집합을 합함
    edgeAccepted += 1                           # 간선이 하나 추가됨
# (5) n-1개의 간선이 삽입될 때 까지 반복

```

## 가중치 그래프와 최소비용 신장 트리

### Kruskal의 최소비용 신장 트리

#### 4 Kruskal 알고리즘 구현

##### ● 테스트 프로그램

```

vertex = [    'A', 'B', 'C', 'D', 'E', 'F', 'G' ]
weight = [ [ None, 29, None, None, None, 10, None ],
           [ 29, None, 16, None, None, 15, None ],
           [ None, 16, None, 12, None, None, None ],
           [ None, None, 12, None, 22, None, 18 ],
           [ None, None, None, 22, None, 27, 25 ],
           [ 10, None, None, None, 27, None, None ],
           [ None, 15, None, 18, 25, None, None ] ]
print("MST By Kruskal's Algorithm")
MSTKruskal(vertex, weight)

```

C:\>WINDOWS\system32\cmd.exe -

MST By Kruskal's Algorithm

간선 추가 : (A, F, 10)  
 간선 추가 : (C, D, 12)  
 간선 추가 : (B, G, 15)  
 간선 추가 : (B, C, 16)  
 간선 추가 : (D, E, 22)  
 간선 추가 : (E, F, 27)

계속하려면 아무 키나 누르십시오.



# Kruskal의 최소비용 신장 트리

가중치 그래프와 최소비용 신장 트리

Kruskal의 최소비용 신장 트리

## 4 Kruskal 알고리즘 구현

### ● 시간 복잡도

Kruskal 알고리즘:  $O(e \log e)$

- 대부분 간선들을 정렬하는 시간에 좌우
- 간선  $e$ 개를 정렬하는 시간
- 정점의 수에 비해 간선이 적은 희박한 그래프(Sparse Graph)에 유리



# Prim의 최소비용 신장 트리

## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 1 Prim의 MST 알고리즘

#### Prim의 MST 알고리즘

- 하나의 정점에서부터 시작하여 트리를 단계적으로 확장
- 현재의 신장 트리 집합에 인접한 정점 중 최저 간선으로 연결된 정점을 선택하여 신장 트리 집합에 추가  
→ 탐욕적 기법
- 이 과정을  $n-1$ 개의 간선을 가질 때까지 반복

## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 1 Prim의 MST 알고리즘

##### ◎ Prim의 최소비용 신장 트리 알고리즘

Prim()

1 그래프에서 시작 정점을 선택하여 초기 트리를 만듦

2 현재 트리의 정점들과 인접한 정점들 중에서 간선의 가중치가 가장 작은 정점  $v$ 를 선택

3 정점  $v$ 와 이때의 간선을 트리에 추가

4 모든 정점이 삽입될 때까지 2번으로 이동

# Prim의 최소비용 신장 트리

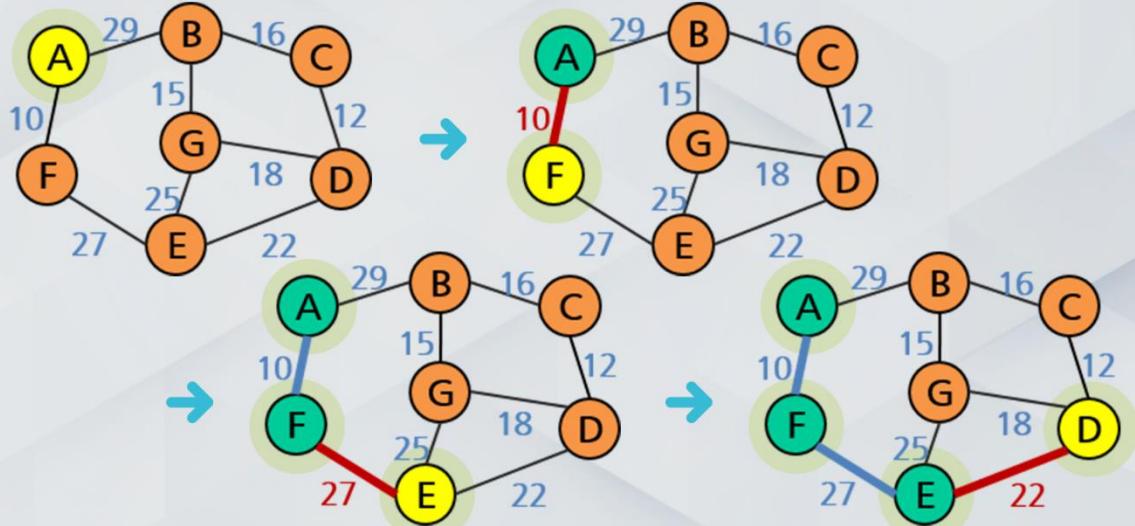


## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 1 Prim의 MST 알고리즘

● Prim의 알고리즘 처리 과정

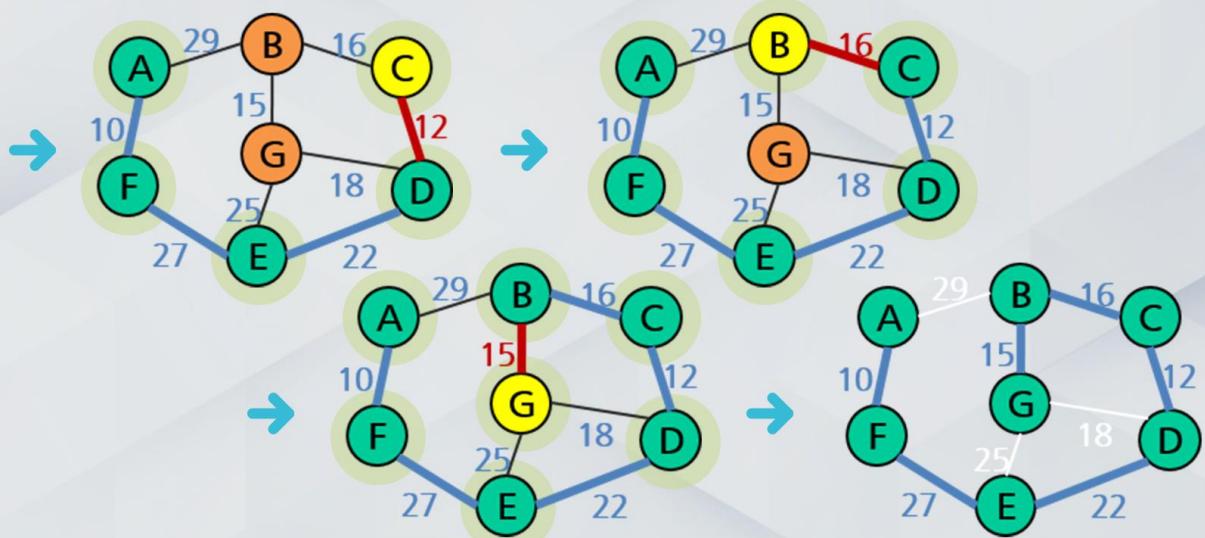


## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 1 Prim의 MST 알고리즘

● Prim의 알고리즘 처리 과정



# Prim의 최소비용 신장 트리



## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 2 Prim 알고리즘 구현

현재까지 만들어진 MST의 인접 정점들 중에서  
간선 가중치가 최소인 정점을 반복적으로 선택해 트리를 확장

##### ▶ 간선 가중치가 최소인 정점 선택

```

INF = 9999
def getMinVertex(dist, selected):
    minv = 0
    mindist = INF
    for v in range(len(dist)):
        if selected[v] == False and dist[v] < mindist:
            mindist = dist[v]
            minv = v
    return minv
    
```

# 무한대  
# dist[v]: 지금까지 알려진 거리  
# dist가 최소인 정점의 인덱스 저장  
# 초기화  
# 모든 정점에 대해  
# 선택되지 않은 정점 중에서 dist가 최소인 정점 선택  
# dist가 최소인 정점의 인덱스 반환

## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 2 Prim 알고리즘 구현

```

def MSTPrim(vertex, adj):
    vsize = len(vertex)
    dist = [INF] * vsize
    selected = [False] * vsize
    dist[0] = 0

    for i in range(vsize):
        u = getMinVertex(dist, selected)
        selected[u] = True
        print(vertex[u], end=' ')
        
```

# 정점의 수  
# 정점과 MST와의 거리  
# 정점의 선택(MST) 여부  
# 0번 정점에서 시작

# n개의 모든 정점이 선택되어야 함  
# 최소 간선을 가진 정점 u 선택  
# MST에 u를 포함시킴

for v in range(vsize):
 if (adj[u][v] != None):
 # v가 MST에 포함되지 않았고, u를 거친 거리가 더 짧으면 dist[v] 갱신
 if not selected[v] and adj[u][v] < dist[v]:
 dist[v] = adj[u][v]

print()

# Prim의 최소비용 신장 트리



## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 2 Prim 알고리즘 구현

##### ● 테스트 프로그램

```
rvertex = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G' ]
weight = [
    [ None, 29, None, None, None, 10, None ],
    [ 29, None, 16, None, None, None, 15 ],
    [ None, 16, None, 12, None, None, None ],
    [ None, None, 12, None, 22, None, 18 ],
    [ None, None, None, 22, None, 27, 25 ],
    [ 10, None, None, None, 27, None, None ],
    [ None, 15, None, 18, 25, None, None ] ]

print("MST By Prim's Algorithm")
MSTPrim(vertex, weight)
```

```
C:\WINDOWS\system32\cmd.exe
MST By Prim's Algorithm
A F E D C B G
```

## 가중치 그래프와 최소비용 신장 트리

### Prim의 최소비용 신장 트리

#### 2 Prim 알고리즘 구현

##### ● 시간 복잡도

**Prim 알고리즘:  $O(n^2)$**

- 주 반복문이  $n$ 번, 내부 반복문이  $n$ 번 반복
- 완전 그래프와 같이 밀집한 그래프에 유리

**Kruskal 알고리즘:  $O(e \log e)$**

- 희박한 그래프(Sparse Graph)에 유리