CHAPTER.18

그래프의 응용

학습 내용

[1] 그래프의 응용

학습 목표

- 그래프로 활용되는 응용 프로그램을 작성할 수 있다.

01 그래프의 응용

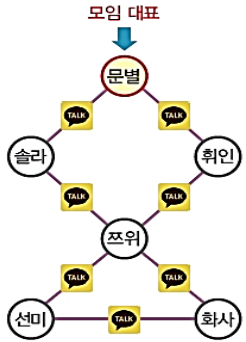
1) 친구가 연락되는지 확인
2) 최소 비용으로 자전거 도로 연결

1] 친구가 연락되는지 확인

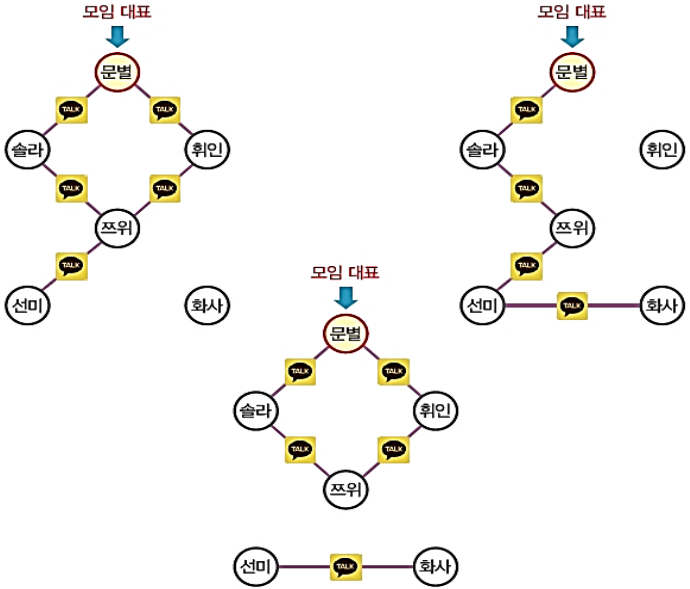
세종사이버대학교

친구들의 비상연락망에서 특정 친구가 연락되는지 확인하는 방법

- 비상 연락망의 예



비상 연락망의 기본 형태

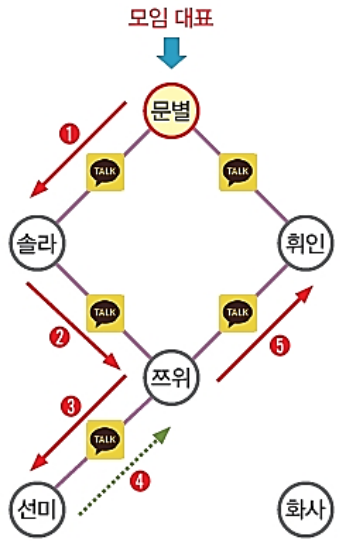


비상 연락망의 연결이 끊긴 경우

1] 친구가 연락되는지 확인

비상연락망 연결이 끊긴 경우

- ‘화사’만 동떨어진 예




```
gSize = 6
G1 = Graph(gSize)
G1.graph[문별][솔라] = 1; G1.graph[문별][휘인] = 1
G1.graph[솔라][문별] = 1; G1.graph[솔라][쯔위] = 1
G1.graph[휘인][문별] = 1; G1.graph[휘인][쯔위] = 1
G1.graph[쯔위][솔라] = 1; G1.graph[쯔위][휘인] = 1; G1.graph[쯔위][선미] = 1
G1.graph[선미][쯔위] = 1;
```

1] 친구가 연락되는지 확인

비상연락망 연결이 끊긴 경우

- 시작 정점인 문별부터 방문을 시작하여 연결된 모든 친구에게 연락하고 그 결과를 visitedAry에 저장

```
stack = []
visitedAry = []    # 방문한 정점

current = 0        # 시작 정점
stack.append(current)
visitedAry.append(current)

while (len(stack) != 0) :
    next = None
    for vertex in range(gSize) :
        if G1.graph[current][vertex] != 0 :
            if vertex in visitedAry :   # 방문한 적이 있는 정점이면 탈락
                pass
```

비상연락망 연결이 끊긴 경우

- 시작 정점인 문별부터 방문을 시작하여 연결된 모든 친구에게 연락하고 그 결과를 visitedAry에 저장

```
            else :                          # 방문한 적이 없으면 다음 정점으로 지정
                next = vertex
                break

    if next != None :                       # 다음에 방문할 정점이 있는 경우
        current = next
        stack.append(current)
        visitedAry.append(current)
    else :                                  # 다음에 방문할 정점이 없는 경우
        current = stack.pop()
```

1] 친구가 연락되는지 확인

비상연락망 연결이 끊긴 경우

- 순회 결과인 visitedAry에 '화사'가 들어 있는지 확인하면 그래프에 연결된 상태인지 알 수 있음

```
if 화사 in visitedAry :
    print('화사가 연락이 됨')
else :
    print('화사가 연락이 안됨 ㅠ')
```

1] 친구가 연락되는지 확인

특정 정점이 그래프에 연결되어 있는지 확인하는 함수

```
1  gSize = 6
2  def findVertex(g, findVtx) :
3      stack = []
4      visitedAry = []    # 방문한 정점
5
6      current = 0        # 시작 정점
7      stack.append(current)
8      visitedAry.append(current)
9
10     while (len(stack) != 0) :
11         next = None
12         for vertex in range(gSize) :
13             if g.graph[current][vertex] != 0 :
```

1] 친구가 연락되는지 확인

특정 정점이 그래프에 연결되어 있는지 확인하는 함수

```
14                  if vertex in visitedAry :
15                        pass
16                   else :
17                        next = vertex
18                        break
19
20          if next != None :
21               current = next
22               stack.append(current)
23               visitedAry.append(current)
24          else :
25               current = stack.pop()
```

#방문한적이있는정점이면탈락

#방문한적이없으면다음정점으로 지정

#다음에방문할정점이있는경우

#다음에방문할정점이없는경우

1] 친구가 연락되는지 확인

특정 정점이 그래프에 연결되어 있는지 확인하는 함수

```
26
27      if findVtx in visitedAry :
28          return True
29      else :
30          return False
```

실행 결과

없음

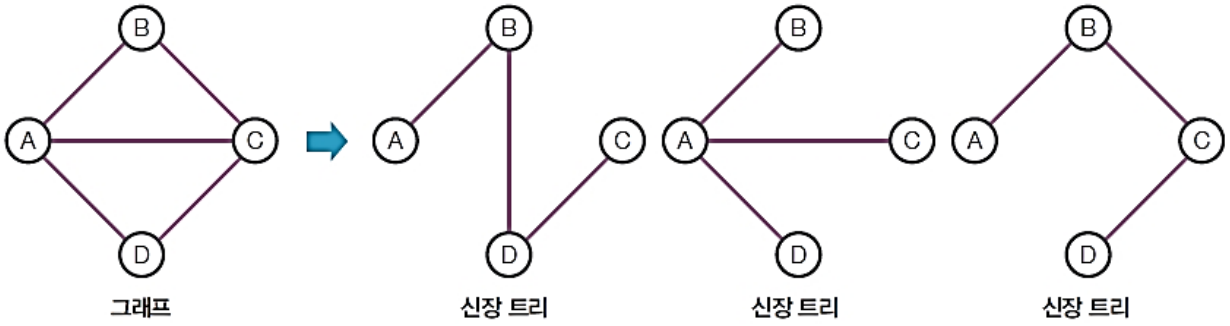
2] 최소 비용으로 자전거 도로 연결

최소 신장 트리의 개념

신장 트리 (Spanning Tree)

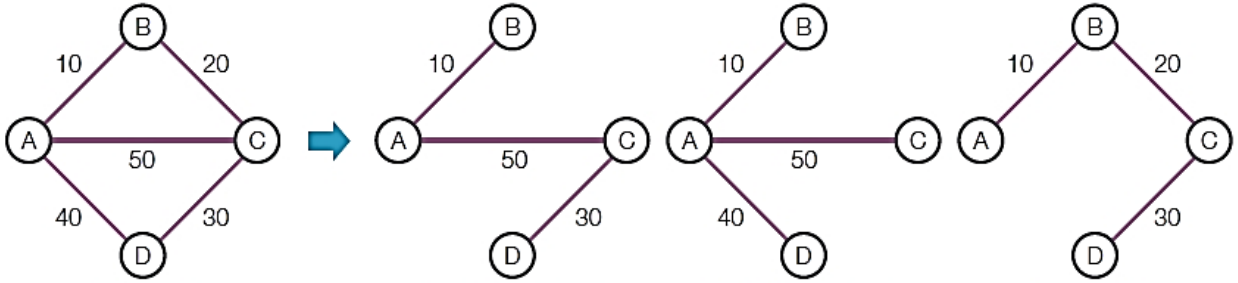
최소 간선으로 그래프의 모든 정점이 연결되는 그래프

그래프와 다양한 신장 트리



최소 신장 트리의 개념

가중치 그래프와 다양한 신장 트리



2] 최소 비용으로 자전거 도로 연결

최소 비용 신장 트리

최소 비용 신장 트리

가중치 그래프에서 만들 수 있는 신장 트리 중
합계가 최소인 것

구현하는 방법은 프림(Prim) 알고리즘, 크루스컬(Kruskal) 알고리즘 등이 있음

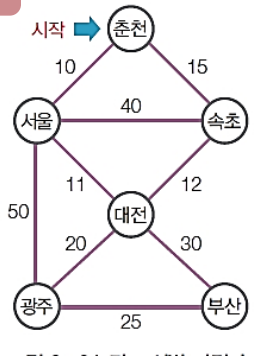
2] 최소 비용으로 자전거 도로 연결

최소 비용 신장 트리

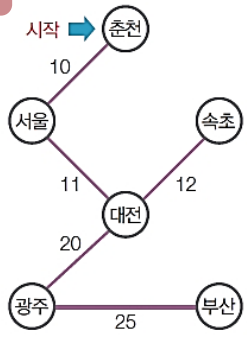
구현하는 방법은 프림(Prim) 알고리즘, 크루스컬(Kruskal) 알고리즘 등이 있음

- 최소 비용 신장 트리를 활용하여 자전거 도로를 최소 비용으로 연결하는 예(크루스컬 알고리즘을 활용)

그래프에 가중치를 추가한 형태



최소 비용 신장 트리로 구성한 상태



전체 자전거 도로를 위한 가중치 그래프 구현

전체 비용이 나와 있는 가중치 그래프 구현 예

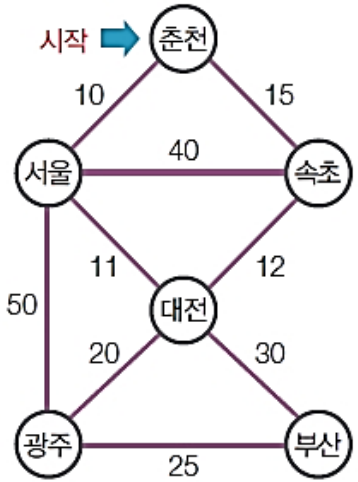
```
G1 = None
nameAry = ['춘천', '서울', '속초', '대전', '광주', '부산']
춘천, 서울, 속초, 대전, 광주, 부산 = 0, 1, 2, 3, 4, 5

gSize = 6
G1 = Graph(gSize)
G1.graph[춘천][서울] = 10; G1.graph[춘천][속초] = 15
G1.graph[서울][춘천] = 10; G1.graph[서울][속초] = 40; G1.graph[서울][대전] = 11; G1.graph[서울][광주] = 50
G1.graph[속초][춘천] = 15; G1.graph[속초][서울] = 40; G1.graph[속초][대전] = 12
G1.graph[대전][서울] = 11; G1.graph[대전][속초] = 12; G1.graph[대전][광주] = 20; G1.graph[대전][부산] = 30
G1.graph[광주][서울] = 50; G1.graph[광주][대전] = 20; G1.graph[광주][부산] = 25
G1.graph[부산][대전] = 30; G1.graph[부산][광주] = 25
```

2] 최소 비용으로 자전거 도로 연결

전체 자전거 도로를 위한 가중치 그래프 구현

전체 비용이 나와 있는 가중치 그래프 구현 예

- 각 도시별 자전거 도로 연결 계획도



	춘천	서울	속초	대전	광주	부산
춘천	0	10	15	0	0	0
서울	10	0	40	11	50	0
속초	15	40	0	12	0	0
대전	0	11	12	0	20	30
광주	0	50	0	20	0	25
부산	0	0	0	30	25	0

가중치와 간선 목록 생성

가중치와 간선을 별도 배열로 만드는 예

- 생성된 가중치와 간선 목록

```
edgeAry = []
for i in range(gSize) :
    for k in range(gSize) :
        if G1.graph[i][k] != 0 :
            edgeAry.append([G1.graph[i][k], i, k])
```

```
[[10, '춘천', '서울'], [15, '춘천', '속초'], [10, '서울', '춘천'], [40, '서울', '속초'], [11,
'서울', '대전'], [50, '서울', '광주'], [15, '속초', '춘천'], [40, '속초', '서울'], [12, '속초',
'대전'], [11, '대전', '서울'], [12, '대전', '속초'], [20, '대전', '광주'], [30, '대전', '부산'],
[50, '광주', '서울'], [20, '광주', '대전'], [25, '광주', '부산'], [30, '부산', '대전'], [25, '부
산', '광주']]
```

간선 정렬

가중치를 기준으로 내림차순으로 간선 정렬

- 정렬된 가중치와 간선 목록

```
from operator import itemgetter
edgeAry = sorted(edgeAry, key=itemgetter(0), reverse=True)
```

```
[[50, '서울', '광주'], [50, '광주', '서울'], [40, '서울', '속초'], [40, '속초', '서울'], [12, '속초', '대전'], [12, '대전', '속초'], [25, '광주', '부산'], [25, '부산', '광주'], [15, '춘천', '속초'], [15, '속초', '춘천'], [20, '대전', '광주'], [20, '광주', '대전'], [30, '대전', '부산'], [30, '부산', '대전'], [11, '서울', '대전'], [11, '대전', '서울'], [10, '춘천', '서울'], [10, '서울', '춘천']]
```

중복 간선 제거

중복 간선 제거

- 중복을 제거한 가중치와 간선 목록

```
newAry = []
for i in range(0, len(edgeAry), 2) :
    newAry.append(edgeAry[i])
```

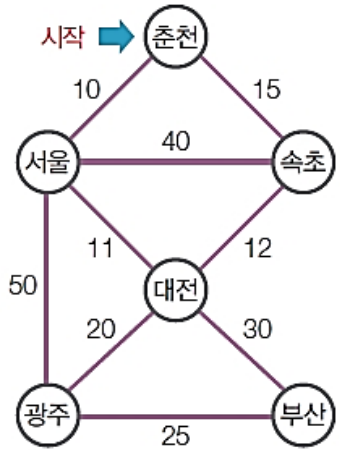
```
[[50, '서울', '광주'], [40, '서울', '속초'], [30, '대전', '부산'], [25, '광주', '부산'], [20,
'대전', '광주'], [15, '춘천', '속초'], [12, '속초', '대전'], [11, '서울', '대전'], [10, '춘천',
'서울']]
```

중복 간선 제거

중복 간선 제거

- 가중치표와 자전거 도로 연결망

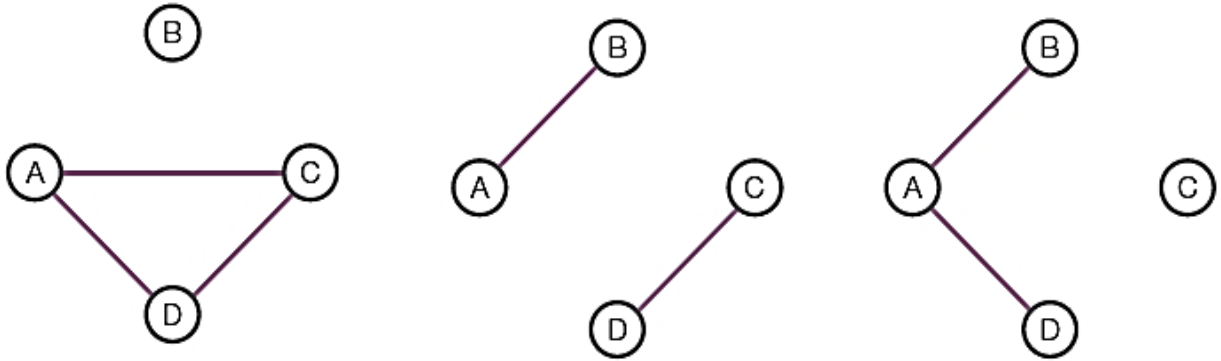
가중치	간선
50	서울 – 광주
40	서울 – 속초
30	대전 – 부산
25	광주 – 부산
20	대전 – 광주
15	춘천 – 속초
12	속초 – 대전
11	서울 – 대전
10	춘천 – 서울



2] 최소 비용으로 자전거 도로 연결

가중치가 높은 간선부터 제거

> **도시가 연결되지 않는 것은 허용하지 않음**



2] 최소 비용으로 자전거 도로 연결

가중치가 높은 간선부터 제거

1 서울 - 광주 간선 제거

❶ index = 0

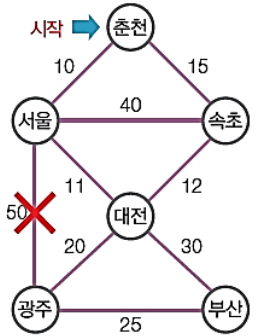
```
start = newAry[index][1]    # 서울
end = newAry[index][2]      # 광주
```

❷
```
G1.graph[start][end] = 0
G1.graph[end][start] = 0
```

❸ del(newAry[index])

가중치 배열(newAry)

	가중치	간선
index → 0	~~50~~	~~서울 – 광주~~
1	40	서울 – 속초
2	30	대전 – 부산
3	25	광주 – 부산
4	20	대전 – 광주
5	15	춘천 – 속초
6	12	속초 – 대전
7	11	서울 – 대전
8	10	춘천 – 서울



2] 최소 비용으로 자전거 도로 연결

세종사이버대학교

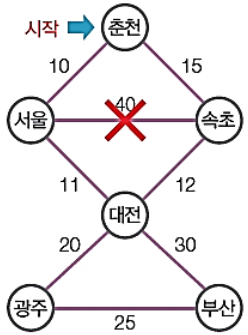
가중치가 높은 간선부터 제거

2 서울 – 속초 간선 제거

```
❶ index = 0

   start = newAry[index][1]    # 서울
   end = newAry[index][2]      # 속초

❷ G1.graph[start][end] = 0
   G1.graph[end][start] = 0

❸ del(newAry[index])
```

가중치 배열(newAry)

	가중치	간선
	~~50~~	~~서울 – 광주~~
index → 0	~~40~~	~~서울 – 속초~~
1	30	대전 – 부산
2	25	광주 – 부산
3	20	대전 – 광주
4	15	춘천 – 속초
5	12	속초 – 대전
6	11	서울 – 대전
7	10	춘천 – 서울



2] 최소 비용으로 자전거 도로 연결

가중치가 높은 간선부터 제거

3 대전 – 부산 간선 제거

❶ index = 0

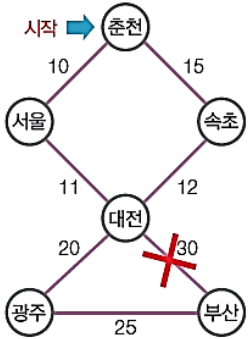
```
start = newAry[index][1]    # 대전
end = newAry[index][2]      # 부산
```

❷
```
G1.graph[start][end] = 0
G1.graph[end][start] = 0
```

❸ del(newAry[index])

가중치 배열(newAry)

	가중치	간선
	~~50~~	~~서울 – 광주~~
	~~40~~	~~서울 – 속초~~
index ➡	0	~~30~~ 대전 – 부산
	1	25 광주 – 부산
	2	20 대전 – 광주
	3	15 춘천 – 속초
	4	12 속초 – 대전
	5	11 서울 – 대전
	6	10 춘천 – 서울



가중치가 높은 간선부터 제거

4 광주 - 부산 간선의 제거 시도와 원상 복구

```
❶ index = 0

   start = newAry[index][1]        # 광주
   end = newAry[index][2] # 부산
❷ saveCost = newAry[index][0]

❸ G1.graph[start][end] = 0
   G1.graph[end][start] = 0

❹ startYN = findVertex(G1, start)
   endYN = findVertex(G1, end)
```

가중치가 높은 간선부터 제거

4 광주 - 부산 간선의 제거 시도와 원상 복구

```
    if startYN and endYN : # 두 정점 모두 그래프와 연결되어 있다면
        del(newAry[index])# 가중치 배열에서 완전히 제거
❺   else :
        G1.graph[start][end] = saveCost
        G1.graph[end][start] = saveCost
❻   index += 1
```

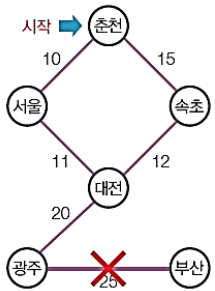
2] 최소 비용으로 자전거 도로 연결

가중치가 높은 간선부터 제거

4 광주 - 부산 간선의 제거 시도와 원상 복구

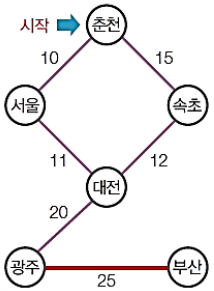
가중치 배열(newAry)

	가중치	간선
	~~50~~	~~서울 – 광주~~
	~~40~~	~~서울 – 속초~~
	~~30~~	~~대전 – 부산~~
index 0	~~25~~	~~광주 – 부산~~
1	20	대전 – 광주
2	15	춘천 – 속초
3	12	속초 – 대전
4	11	서울 – 대전
5	10	춘천 – 서울



가중치 배열(newAry)

	가중치	간선
	~~50~~	~~서울 – 광주~~
	~~40~~	~~서울 – 속초~~
	~~30~~	~~대전 – 부산~~
0	25	광주 – 부산
index 1	20	대전 – 광주
2	15	춘천 – 속초
3	12	속초 – 대전
4	11	서울 – 대전
5	10	춘천 – 서울



2] 최소 비용으로 자전거 도로 연결

세종사이버대학교

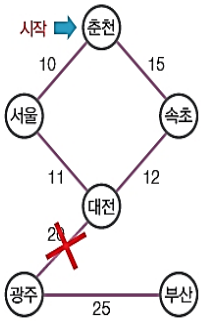
가중치가 높은 간선부터 제거

5 대전 - 광주 간선의 제거 시도와 원상 복구

```
index = 1    # 배열의 1번째 위치(대전-광주). 앞 단계에서 index를 1 증가시켰음

## 나머지는 앞 코드와 동일
```

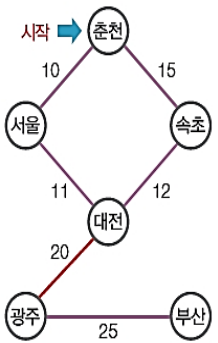
가중치 배열(newAry)

		가중치	간선
		~~50~~	~~서울 – 광주~~
		~~40~~	~~서울 – 속초~~
		~~30~~	~~대전 – 부산~~
0		25	광주 – 부산
index ➡	1	~~20~~	~~대전 – 광주~~
2		15	춘천 – 속초
3		12	속초 – 대전
4		11	서울 – 대전
5		10	춘천 – 서울



가중치 배열(newAry)

		가중치	간선
		~~50~~	~~서울 – 광주~~
		~~40~~	~~서울 – 속초~~
		~~30~~	~~대전 – 부산~~
0		25	광주 – 부산
1		20	대전 – 광주
index ➡	2	15	춘천 – 속초
3		12	속초 – 대전
4		11	서울 – 대전
5		10	춘천 – 서울



2] 최소 비용으로 자전거 도로 연결

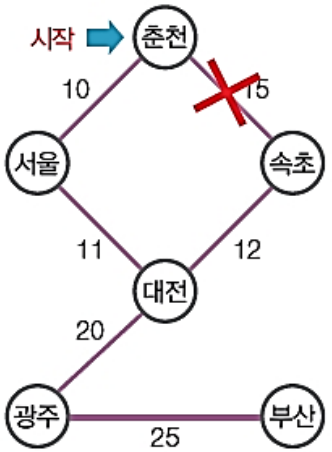
세종사이버대학교

가중치가 높은 간선부터 제거

6 춘천 – 속초 간선 제거

가중치 배열(newAry)

	가중치	간선
	~~50~~	~~서울 – 광주~~
	~~40~~	~~서울 – 속초~~
	~~30~~	~~대전 – 부산~~
0	25	광주 – 부산
1	20	대전 – 광주
index → 2	~~15~~	~~춘천 – 속초~~
3	12	속초 – 대전
4	11	서울 – 대전
5	10	춘천 – 서울



자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

```
1  ## 클래스와 함수 선언 부분 ##
2  class Graph() :
3      def __init__ (self, size) :
4          self.SIZE = size
5          self.graph = [[0 for _ in range(size)] for _ in range(size)]
6
7  def printGraph(g) :
…      # 생략(      앞쪽의      8~17행과 동일)
18
19 def findVertex(g, findVtx) :
…      # 생략(      앞쪽의      3~30행과 동일)
49
50
```

자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

```
51 ## 전역 변수 선언 부분 ##
52 G1 = None
53 nameAry = ['춘천', '서울', '속초', '대전', '광주', '부산']
54 춘천, 서울, 속초, 대전, 광주, 부산 = 0, 1, 2, 3, 4, 5
55
56
57 ## 메인 코드 부분 ##
58 gSize = 6
59 G1 = Graph(gSize)
60 G1.graph[춘천][서울] = 10; G1.graph[춘천][속초] = 15
61 G1.graph[서울][춘천] = 10; G1.graph[서울][속초] = 40; G1.graph[서울][대전] = 11;
   G1.graph[서울][광주] = 50
62 G1.graph[속초][춘천] = 15; G1.graph[속초][서울] = 40; G1.graph[속초][대전] = 12
```

자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

```
63 G1.graph[대전][서울] = 11; G1.graph[대전][속초] = 12; G1.graph[대전][광주] = 20;
   G1.graph[대전][부산] = 30
64 G1.graph[광주][서울] = 50; G1.graph[광주][대전] = 20; G1.graph[광주][부산] = 25
65 G1.graph[부산][대전] = 30; G1.graph[부산][광주] = 25
66
67 print('## 자전거 도로 건설을 위한 전체 연결도 ##')
68 printGraph(G1)
69
70 # 가중치 간선 목록
71 edgeAry = []
72 for i in range(gSize) :
73     for k in range(gSize) :
74         if G1.graph[i][k] != 0 :
```

자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

```
75                edgeAry.append([G1.graph[i][k], i, k])
76
77 from operator import itemgetter
78 edgeAry = sorted(edgeAry, key=itemgetter(0), reverse=True)
79
80 newAry = []
81 for i in range(0, len(edgeAry), 2) :
82     newAry.append(edgeAry[i])
83
84 index = 0
85 while (len(newAry) > gSize-1) :   # 간선 개수가 '정점 개수-1'일 때까지 반복
86     start = newAry[index][1]
87     end = newAry[index][2]
```

자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

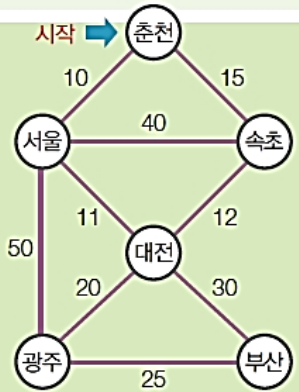
```
88      saveCost = newAry[index][0]
89
90      G1.graph[start][end] = 0
91      G1.graph[end][start] = 0
92
93      startYN = findVertex(G1, start)
94      endYN = findVertex(G1, end)
95
96      if startYN and endYN :
97          del(newAry[index])
98      else :
99          G1.graph[start][end] = saveCost
100         G1.graph[end][start] = saveCost
101         index += 1
```

자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

```
102
103 print('## 최소 비용의 자전거 도로 연결도 ##')
104 printGraph(G1)
```

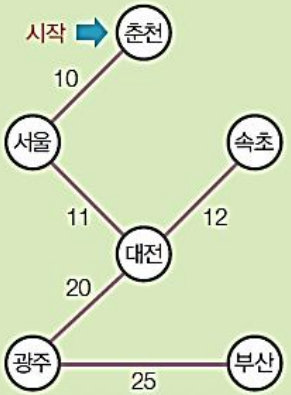
실행 결과

```
## 자전거 도로 건설을 위한 전체 연결도 ##
      춘천  서울  속초  대전  광주  부산
춘천    0    10    15     0     0     0
서울   10     0    40    11    50     0
속초   15    40     0    12     0     0
대전    0    11    12     0    20    30
광주    0    50     0    20     0    25
부산    0     0     0    30    25     0
```



자전거 도로 건설을 위한 최소 비용 신장 트리의 전체 코드

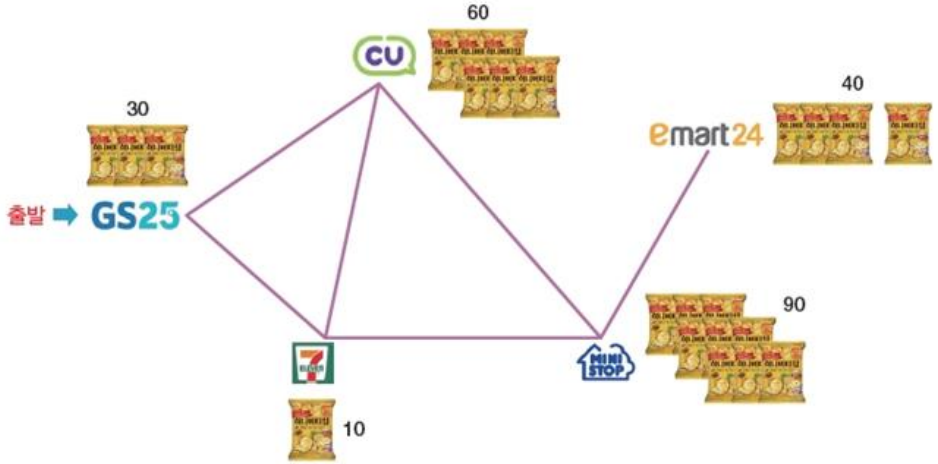
```
## 최소 비용의 자전거 도로 연결도 ##
      춘천 서울 속초 대전 광주 부산
춘천   0   10   0   0   0   0
서울  10    0   0  11   0   0
속초   0    0   0  12   0   0
대전   0   11  12   0  20   0
광주   0    0   0  20   0  25
부산   0    0   0   0  25   0
```



2] 최소 비용으로 자전거 도로 연결

예제 실습

2014년에 출시한 허니버터칩은 한동안 상당한 인기로
구하기가 하늘의 별 따기만큼 어려울 정도였다. 우리 동네에서
허니버터칩 재고가 가장 많은 편의점을 알아내려고 한다.
편의점끼리는 서로 그래프 형태로 이어져 있다고 가정하자.



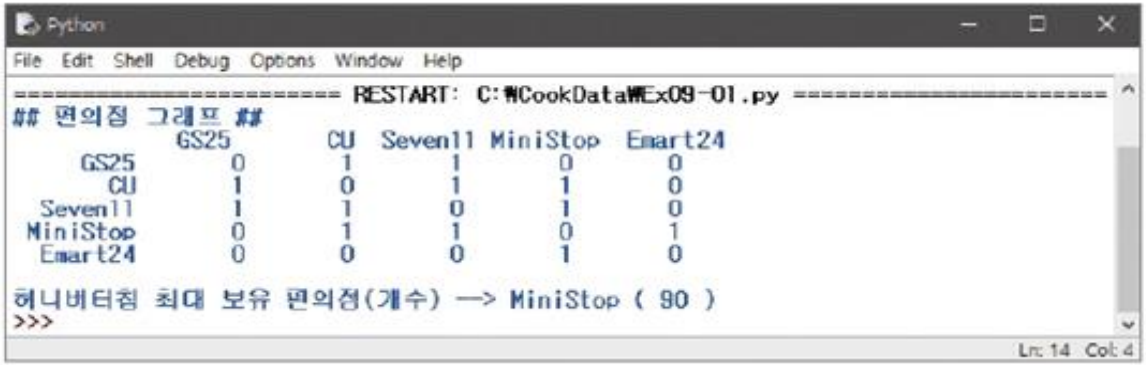
2] 최소 비용으로 자전거 도로 연결

세종사이버대학교

예제 실습

2014년에 출시한 허니버터칩은 한동안 상당한 인기로 구하기가 하늘의 별 따기만큼 어려울 정도였다. 우리 동네에서 허니버터칩 재고가 가장 많은 편의점을 알아내려고 한다. 편의점끼리는 서로 그래프 형태로 이어져 있다고 가정하자.

실행 결과



2] 최소 비용으로 자전거 도로 연결

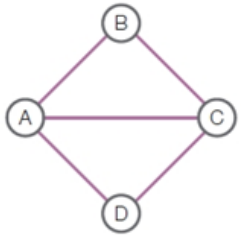
예제 실습

학습평가

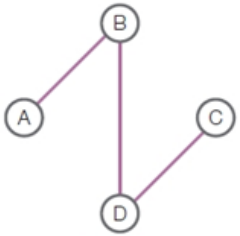
Q1

Q1 다음 중 신장 트리가 아닌 것은?

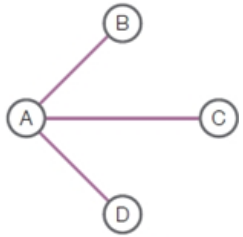
1



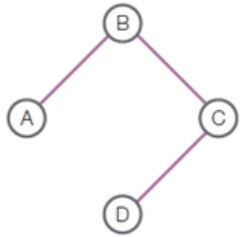
2



3



4

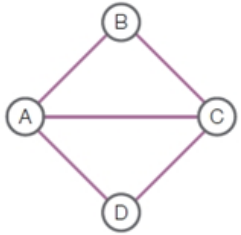


학습평가

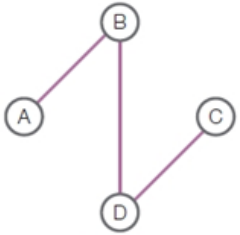
Q1

Q1 다음 중 신장 트리가 아닌 것은?

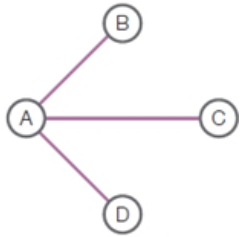
1 ✓



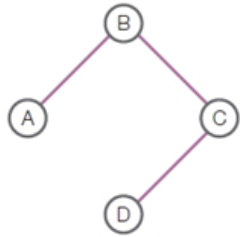
2



3



4

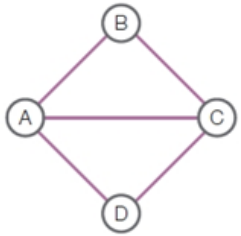


학습평가

Q1

Q1 다음 중 신장 트리가 아닌 것은?

1



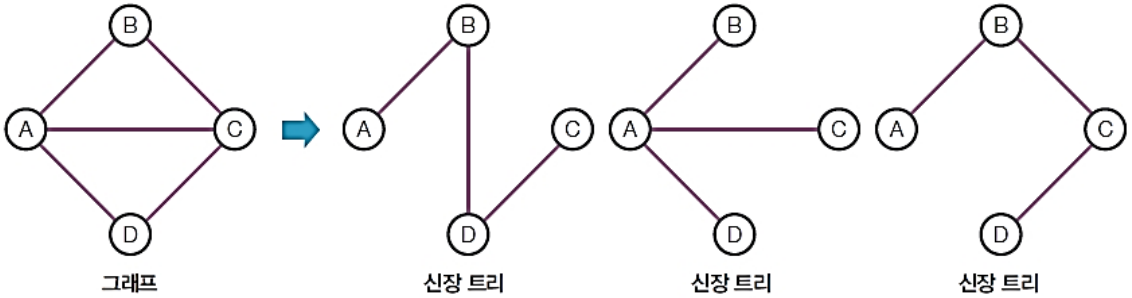
정 답 1

해 설 1번은 그래프로 모든 정점이 서로 연결되어 있습니다.

그래프의 응용

최소 신장 트리의 개념

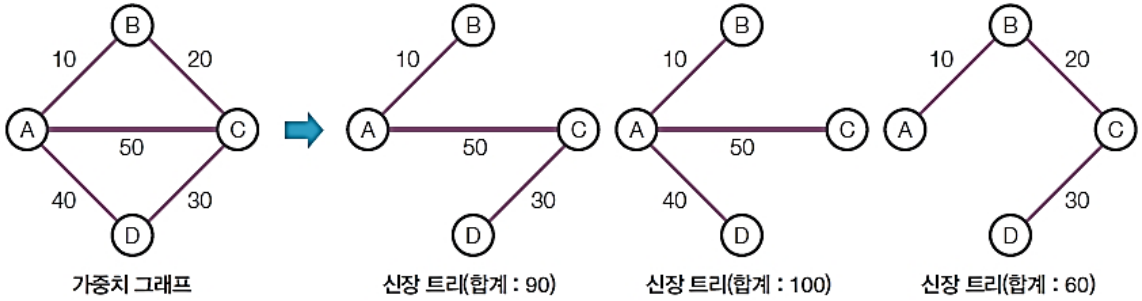
- 신장 트리(Spanning Tree)는 최소 간선으로 그래프의 모든 정점이 연결되는 그래프
- 그래프와 다양한 신장 트리



그래프의 응용

최소 신장 트리의 개념

- 가중치 그래프와 신장 트리



그래프의 응용

최소 비용 신장 트리

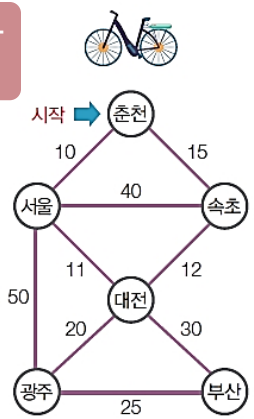
- 가중치 그래프에서 만들 수 있는 신장 트리 중 합계가 최소인 것
- 구현하는 방법은 프림(Prim) 알고리즘, 크루스컬(Kruskal) 알고리즘 등이 있음

그래프의 응용

최소 비용 신장 트리

- 각 도시별 자전거 도로 연결도

그래프에 가중치를 추가한 형태



최소 비용 신장 트리로 구성한 상태