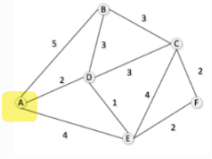
과제3 32203660 이현정

1-a. 인접행렬

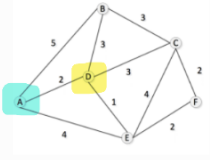
	A	B	C	D	E	F
A	0	5	∞	2	4	∞
B	5	0	3	3	∞	∞
C	∞	3	0	3	4	2
D	2	3	3	0	1	∞
E	4	∞	4	1	0	2
F	∞	∞	2	∞	2	0

1-b. Prim's 알고리즘

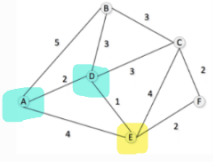
1) 엣지 $F=\phi$
노드 $Y=\{A\}$



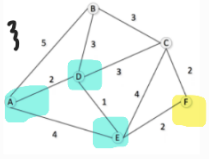
2) $F=\{(A,D)\}$
$Y=\{A,B\}$



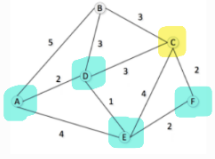
3) $F=\{(A,D),(D,E)\}$
$Y=\{A,B,E\}$



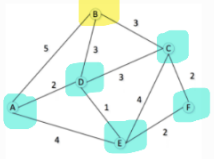
4) $F=\{(A,D),(D,E),(E,F)\}$
$Y=\{A,B,E,F\}$



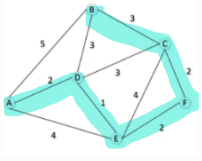
5) $F=\{(A,D),(D,E),(E,F),(F,C)\}$
$Y=\{A,B,E,F,C\}$



5) $F=\{(A,D),(D,E),(E,F),(F,C),(C,B)\}$
$Y=\{A,B,E,F,C,B\}$



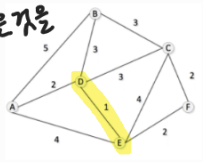
→ minimum spanning tree는
2+1+2+2+3=10인



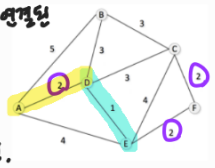
입니다.

1-c. Kruskal's 알고리즘

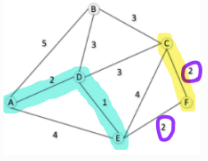
1) 엣지의 크기가 가장 작은것을 선택하면
그림과 같은 트리 1개가 생성됨.



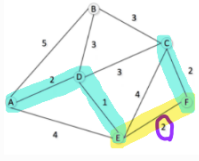
2) 인접해진 트리와 연결될 수 있는 노드와 연결된 다음으로 작은 크기를 가진 엣지를 1개를 선택하면
D 노드로 연결이 되어 1개트리가 됨.



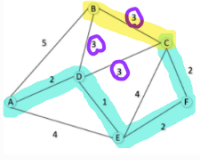
3) 다음 작은 엣지로
서브트리가 1개 추가로 생성됨.



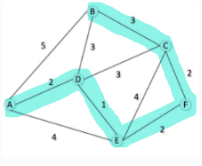
4) 다음 작은 엣지로
2개의 서브트리가
연결되어 트리 1개가 됨.



5) 마지막 노드다 연결되어져서
트리가 완성됨.



→ minimum spanning tree는
2+1+2+2+3=10인



2. 프로그래밍 과제 dijsktra

GitHub 주소 : https://github.com/12hyeon/Hyeon-Algorithm/blob/main/lec-algorithm/dijkstra.py

```python
INF = 10000

def dijkstr(g, s, n):
    node = [0] * n  # 방문한 노드 1, 방문 전 노드 0
    d = [0] * n  # 거리 기록
    node[s] = 1  # 방문 노드
    for i in range(n):
        d[i] = g[s][i] # 시작 노드에서부터 각 노드까지의 거리

    for i in range(n-1) : # 나머지 노드 처리
        now = INF # 거리
        min = 0 # 인덱스
        for j in range(n):
            if (node[j] == 0) and (d[j] < now):
                now = d[j]  # 방문전 노드까지 거리의 최소
                min = j
        node[min] = 1 # 노드 방문

        for j in range(n):
            if (node[j] == 0):
                if (d[min]+g[min][j] < d[j]) :
                    d[j] = d[min] + g[min][j]
                    # 현재 선택한 노드를 지나는 길이 가장 값이 적으면 업데이트
    return d

n = int(input("입력할 그래프의 노드 수를 입력하세요 >> "))
g = [[0]]*n # 그래프

cnt = 0
while 1 :
    print("그래프를 입력하시오. 단, 무한대는 10000으로 나타내시오.")
    cnt = 0
    for x in range(n):
        print("  ", chr(65+x), end="  ")
    print()
    for x in range(n):
        print(chr(65+x), end=": ")
        g[x] = list(map(int, input().rstrip().split()))
        cnt += len(g[x])
    if cnt == n*n:
        break
    else:
        print('\n입력된 정보가 잘못 되었습니다.')

distance = dijkstr(g, 0, n)

print('\n최단 경로')
for i in range(n):
    print(chr(65+i), ":", distance[i])
```

<코드 및 자료조>

∞를 표현하기 위해 10000 이라는 수를 ∞ 대신으로 이용합니다.

입력할 그래프의 노드 개수를 받아서 해당 크기만큼 그래프를 담을 g를 0으로 초기화후

1번씩 줄들을 띄어쓰기 단위로 구분해서 g에 넣게 되고, 해당 개수가 n*n개가 맞는지 확인후

dijstra 함수에다가 A를 가르키는 시작노드 (0)을 기준으로 돌아가게 됩니다.

방문한 노드와 아닌 노드를 구분하기 위해 node 리스트로 1(방문o), 0(미방문)를 구분할 수 있게 하였고,

d리스트에는 각 노드까지의 거리가 업데이트되서 저장될 수 있게 됩니다.

우선 시작노드에서부터 각 노드까지거리를 d리스트에 담고, for문에서 나머지 노드를 방문한 경우를 처리합니다.

아직 방문하기 전 노드 중 가장 거리가 짧은 것을 for문을 통해 찾아서 해당 노드를 방문했다고 저장해 둡니다.

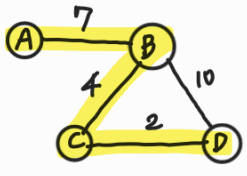
그리고 방금 새롭게 방문한 노드 (인덱스: min)를 거쳐는 경로가 현재 d에 담긴 각 노드까지의 거리를 비교해서

위 노드를 거치는 경로가 짧으면, 해당 값을 d에 넣어서 최단 거리로 업데이트하는 것이 for문 (시작노드제외) 마다 적용됩니다.

이러한 과정으로 노드들이 모두 방문된 경우가 확인되면, 모든 경로의 최단 경로가 d에 저장됩니다.

그러면 마지막으로 시작노드에서부터 각 노드까지의 최단거리가 출력됩니다.

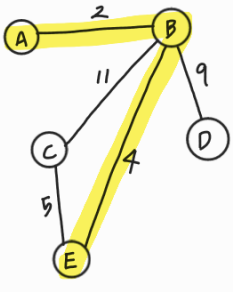
test case 1)




```
입력할 그래프의 노드 수를 입력하세요 >> 4
그래프를 입력하시오. 단, 무한대는 10000으로 나타내시오.
   A     B     C     D
A: 0     7     10000 10000
B: 7     0     4     10
C: 10000 4     0     2
D: 10000 10    2     0

최단 경로
A : 0
B : 7
C : 11
D : 13
```

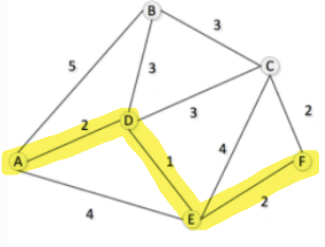
test case 2)




```
입력할 그래프의 노드 수를 입력하세요 >> 5
그래프를 입력하시오. 단, 무한대는 10000으로 나타내시오.
   A     B     C     D     E
A: 0     2     10000 10000 10000
B: 2     0     11    9     4
C: 10000 11    0     10000 5
D: 10000 9     10000 0     10000
E: 10000 4     5     10000 0

최단 경로
A : 0
B : 2
C : 11
D : 11
E : 6
```

test case 3)




```
입력할 그래프의 노드 수를 입력하세요 >> 6
그래프를 입력하시오. 단, 무한대는 10000으로 나타내시오.
   A     B     C     D     E     F
A: 0     5     10000 2     4     10000
B: 5     0     3     3     10000 10000
C: 10000 3     0     3     4     2
D: 2     3     3     0     1     10000
E: 4     10000 4     1     0     2
F: 10000 10000 2     10000 2     0

최단 경로
A : 0
B : 5
C : 5
D : 2
E : 3
F : 5
```