25-1 이니로
알고리즘 멘토링

멘토 - 김수성

동전 2 / 2294

백준 2294 / https://www.acmicpc.net/problem/2294

문제

n가지 종류의 동전이 있다. 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다. 그러면서 동전의 개수가 최소가 되도록 하려고 한다. 각각의 동전은 몇 개라도 사용할 수 있다.

입력

첫째 줄에 n, k가 주어진다. (1 ≤ n ≤ 100, 1 ≤ k ≤ 10,000) 다음 n개의 줄에는 각각의 동전의 가치가 주어진다. 동전의 가치는 100,000보다 작거나 같은 자연수이다. 가치가 같은 동전이 여러 번 주어질 수도 있다.

출력

첫째 줄에 사용한 동전의 최소 개수를 출력한다. 불가능한 경우에는 -1을 출력한다.

동전 2 / 2294

N개의 동전이 주어지고, 동전을 사용해서 M원을 만들 때 사용하는 동전의 최소 개수를 구하는 문제

5 + 5 + 5 = 15

예제 입력 1 복사

```
3 15
1
5
12
```

예제 출력 1 복사

```
3
```

동전 2 / 2294

각 동전은 여러 번 사용 할 수 있음
-> 냅색 문제가 아님

어차피 한 동전을 여러 번 사용 할 수 있으니
동전에 대한 정보는 dp에 필요가 없음

DP[i][j] = i 번째 동전까지 사용 했을 때 j 원을 만들기 위한 최소 동전
-> DP[j] = j 원을 만들기 위한 최소 동전

동전 2 / 2294

DP[j] = j 원을 만들기 위한 최소 동전

각 배열 A의 값 A[i]에 대해서 각 동전을 j원을 만들기 위해서
쓴다고 생각해보자

예제에서는 A = {1, 5, 12}
A[1]을 사용해서 j원을 만들기 위해서는 j - A[1]에서 A[1]을 더함
A[2]를 사용해서 j원을 만들기 위해서는 j - A[1]에서 A[2]를 더함
...

동전 2 / 2294

DP[j] = j 원을 만들기 위한 최소 동전

예제에서는 A = {1, 5, 12}
A[1]을 사용해서 j원을 만들기 위해서는 j - A[1]에서 A[1]을 더함
A[2]를 사용해서 j원을 만들기 위해서는 j - A[1]에서 A[2]를 더함

DP[j] = MIN(DP[j], DP[j - A[i]] + 1)
j - A[i]에서 동전을 1개 더 사용 했으므로 +1 을 해줘야 함
DP의 식에서 MIN값을 사용하므로 INF값으로 DP 초기화

동전 2 / 2294

C++

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];
    for(int i = 0;i < MAX;i++) dp[i] = -1;

    // 값이 INF면 M원을 만들 수 없음
    cout << (solve(m) == INF ? -1 : solve(m));

    return 0;
}
```

```cpp
const ll MAX = 10101;
const ll INF = 1e12;
ll dp[MAX], n, m, a[101];

ll solve(ll cur){
    if(cur < 0) return INF;
    ll& ret = dp[cur];
    if(ret != -1) return ret;
    // min값을 구하기 위해서 큰 수로 초기화
    ret = INF;

    // Base Case
    if(!cur) return ret = 0;

    // 각 동전을 1개 씩 사용해서 cur을 만들 때 최솟값
    for(int i = 1;i <= n;i++){
        ret = min(ret, solve(cur - a[i]) + 1);
    }

    return ret;
}
```

동전 2 / 2294

Python

```python
n, m = map(int, input().split())
a = [0] * (n+1)
for i in range(1, n+1):
    a[i] = int(input())

dp = [-1] * (m+1)

result = solve(m)
# 값이 INF면 M원을 만들 수 없음
print(-1 if result == INF else result)
```

```python
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

INF = 10**12
def solve(cur):
    if cur < 0:
        return INF

    if dp[cur] != -1:
        return dp[cur]

    # Base case
    if cur == 0:
        dp[cur] = 0
        return 0

    # min값을 구하기 위해서 큰 수로 초기화
    dp[cur] = INF

    # 각 동전을 1개 씩 사용해서 cur을 만들 때 최솟값
    for i in range(1, n+1):
        dp[cur] = min(dp[cur], solve(cur - a[i]) + 1)

    return dp[cur]
```

질문?

가장 긴 바이토닉 부분 수열 / 11054

백준 11054 / https://www.acmicpc.net/problem/11054

문제

수열 S가 어떤 수 S_k를 기준으로 $S_1 < S_2 < ... S_{k-1} < S_k > S_{k+1} > ... S_{N-1} > S_N$을 만족한다면, 그 수열을 바이토닉 수열이라고 한다.

예를 들어, {10, 20, **30**, 25, 20}과 {10, 20, 30, **40**}, {**50**, 40, 25, 10} 은 바이토닉 수열이지만, {1, 2, 3, 2, 1, 2, 3, 2, 1}과 {10, 20, 30, 40, 20, 30} 은 바이토닉 수열이 아니다.

수열 A가 주어졌을 때, 그 수열의 부분 수열 중 바이토닉 수열이면서 가장 긴 수열의 길이를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 수열 A의 크기 N이 주어지고, 둘째 줄에는 수열 A를 이루고 있는 A_i가 주어진다. ($1 \le N \le 1,000$, $1 \le A_i \le 1,000$)

출력

첫째 줄에 수열 A의 부분 수열 중에서 가장 긴 바이토닉 수열의 길이를 출력한다.

가장 긴 바이토닉 부분 수열 / 11054

가장 긴 바이토닉 부분 수열의 길이를 구하는 문제

수열 S가 어떤 수 S_k를 기준으로 $S_1 < S_2 < ... S_{k-1} < S_k > S_{k+1} > ... S_{N-1} > S_N$을 만족한다면, 그 수열을 바이토닉 수열이라고 한다.

1 5 2 1 4 3 4 5 2 1
증가하다가 어느 순간부터 감소하는 수열

예제 입력 1 복사

```
10
1 5 2 1 4 3 4 5 2 1
```

예제 출력 1 복사

```
7
```

가장 긴 바이토닉 부분 수열 / 11054

1 5 2 1 4 3 4 5 2 1
증가하다가 어느 순간부터 감소하는 수열

저번에 푼 가장 긴 증가하는 부분 수열 문제와 비슷함
DP[i] = DP[j] + 1 (j < i && A[j] < A[i])

증가하는 부분 수열에 감소하는 부분 수열을 붙인 것으로 생각

가장 긴 바이토닉 부분 수열 / 11054

증가하는 부분 수열에 감소하는 부분 수열을 붙인 것으로 생각

현재 수열이 증가하고 있으면 계속 증가하거나, 감소하는 상태로 바뀜
현재 수열이 감소하고 있으면 계속 감소해야 함

-> 현재 수열이 증가하고 있는지, 감소하고 있는지 알아야 함

가장 긴 바이토닉 부분 수열 / 11054

현재 수열이 증가하고 있으면 계속 증가하거나, 감소하는 상태로 바뀜
현재 수열이 감소하고 있으면 계속 감소해야 함

-> 현재 수열이 증가하고 있는지, 감소하고 있는지 알아야 함

DP[i][0] = 현재 수열이 증가하고 i 번째 인덱스가 수열의 마지막
일 때 최대 길이

DP[i][1] = 현재 수열이 감소하고 i 번째 인덱스가 수열의 마지막
일 때 최대 길이

가장 긴 바이토닉 부분 수열 / 11054

DP[i][0] = 현재 수열이 증가하고 i 번째 인덱스가 수열의 마지막 일 때 최대 길이

DP[i][1] = 현재 수열이 감소하고 i 번째 인덱스가 수열의 마지막 일 때 최대 길이

DP[i][0] = DP[j][0] + 1 (j < i && A[j] < A[i])
DP[i][1] = MAX(DP[j][0], DP[j][1]) + 1 (j < i && A[j] > A[i])

가장 긴 바이토닉 부분 수열 / 11054

DP[i][0] = DP[j][0] + 1 (j < i && A[j] < A[i])
DP[i][1] = MAX(DP[j][0], DP[j][1]) + 1 (j < i && A[j] > A[i])

Base Case

DP[1][0] = DP[1][1] = 1

. {10, 20, **30**, 25, 20}과 {10, 20, 30, **40**}, {**50**, 40, 25, 10} 은 바이토닉 수열이지만,

문제의 조건에 따라서 DP[i][0]도 정답이 될 수 있음

가장 긴 바이토닉 부분 수열 / 11054

C++

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n;
    for(int i = 1;i <= n;i++) cin >> a[i];
    for(int i = 0;i < MAX;i++){
        for(int j = 0;j <= 1;j++) dp[i][j] = -1;
    }

    ll result = 0;
    for(int i = 1;i <= n;i++){
        // DP[i][0], DP[i][1] 중 최댓값이 정답
        result = max(result, max(solve(i, 0), solve(i, 1)));
    }
    cout << result;
    return 0;
}
```

```cpp
const ll MAX = 1010;
ll dp[MAX][2], a[MAX], n;

ll solve(ll cur, ll cnt){
    ll& ret = dp[cur][cnt];
    if(ret != -1) return ret; ret = 1;

    // 감소하는 상태
    if(cnt) for(int i = 1;i < cur;i++){
        // 이전 값이 현재 값보다 작으면 건너 뜀
        if(a[i] <= a[cur]) continue;
        ret = max(ret, solve(i, 0) + 1);
        ret = max(ret, solve(i, 1) + 1);
    }
    // 증가하는 상태
    else for(int i = 1;i < cur;i++){
        // 이전 값이 현재 값보다 크면 건너 뜀
        if(a[i] >= a[cur]) continue;
        ret = max(ret, solve(i, 0) + 1);
    }

    return ret;
}
```

가장 긴 바이토닉 부분 수열 / 11054

Python

```python
n = int(input())
A = list(map(int, input().split()))
a = [0] + A

dp = [[-1] * 2 for _ in range(n + 1)]

ans = 0
for i in range(1, n + 1):
    # DP[i][0], DP[i][1]중 최댓값이 정답
    ans = max(ans, solve(i, 0), solve(i, 1))

print(ans)
```

```python
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

def solve(cur, cnt):
    if dp[cur][cnt] != -1:
        return dp[cur][cnt]

    ret = 1
    # 감소하는 상태
    if cnt == 1:
        for i in range(1, cur):
            # 이전 값이 현재 값보다 작으면 건너 뜀
            if a[i] <= a[cur]:
                continue
            ret = max(ret, solve(i, 0) + 1, solve(i, 1) + 1)
    # 증가하는 상태
    else:
        for i in range(1, cur):
            # 이전 값이 현재 값보다 크면 건너 뜀
            if a[i] >= a[cur]:
                continue
            ret = max(ret, solve(i, 0) + 1)

    dp[cur][cnt] = ret
    return ret
```

질문?

내리막 길 / 1520

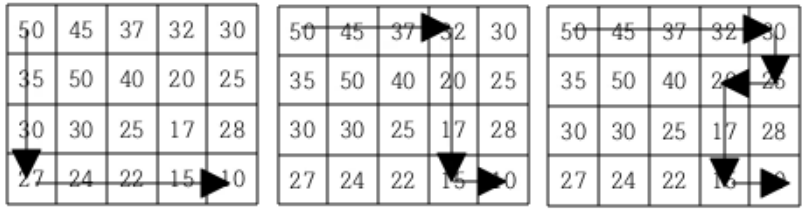
백준 1520 / https://www.acmicpc.net/problem/1520

문제

여행을 떠난 세준이는 지도를 하나 구하였다. 이 지도는 아래 그림과 같이 직사각형 모양이며 여러 칸으로 나뉘어져 있다. 한 칸은 한 지점을 나타내는데 각 칸에는 그 지점의 높이가 쓰여 있으며, 각 지점 사이의 이동은 지도에서 상하좌우 이웃한 곳끼리만 가능하다.

50	45	37	32	30
35	50	40	20	25
30	30	25	17	28
27	24	22	15	10

현재 제일 왼쪽 위 칸이 나타내는 지점에 있는 세준이는 제일 오른쪽 아래 칸이 나타내는 지점으로 가려고 한다. 그런데 가능한 힘을 적게 들이고 싶어 항상 높이가 더 낮은 지점으로만 이동하여 목표 지점까지 가고자 한다. 위와 같은 지도에서는 다음과 같은 세 가지 경로가 가능하다.



지도가 주어질 때 이와 같이 제일 왼쪽 위 지점에서 출발하여 제일 오른쪽 아래 지점까지 항상 내리막길로만 이동하는 경로의 개수를 구하는 프로그램을 작성하시오.

내리막 길 / 1520

(1,1)에서 출발해서 항상 감소하는 숫자로 이동 할 때 (n,m)에 도착하는 경우의 수를 구하는 문제

예제 입력 1 복사

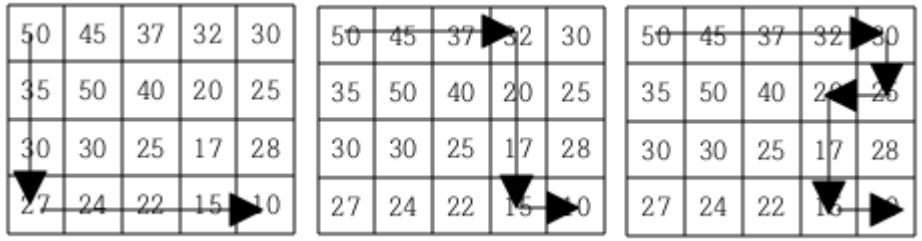
```
4 5
50 45 37 32 30
35 50 40 20 25
30 30 25 17 28
27 24 22 15 10
```

예제 출력 1 복사

```
3
```

내리막 길 / 1520

각 칸을 정점, 각 칸에서 내리막으로 가는 길을 간선으로 가진
그래프를 생각해보자



각 칸에 도달 할 수 있는 경우의 수는 각 칸에 연결되어 있는
이전의 칸의 경우의 수를 더해서 구할 수 있음

내리막 길 / 1520

각 칸에 도달 할 수 있는 경우의 수는 각 칸에 연결되어 있는
이전의 칸의 경우의 수를 더해서 구할 수 있음

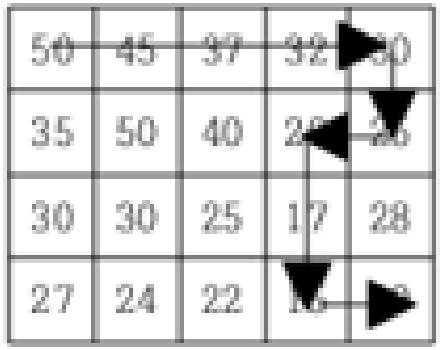
DP[i][j] = (0,0)에서 내리막 길만 사용해서 (i,j)에 도달하는 경우의 수
DP[i][j] = DP[i – 1][j] + DP[i + 1][j] + DP[i][j – 1] + DP[i][j + 1]

물론 이전의 값이 더 클 때만 더해야 함

내리막 길 / 1520

DP[i][j] = (0,0)에서 내리막 길만 사용해서 (i,j)에 도달하는 경우의 수
DP[i][j] = DP[i – 1][j] + DP[i + 1][j] + DP[i][j – 1] + DP[i][j + 1]

또한 이 문제는 바텀 업으로 풀기 어려움
바텀 업으로 DP 테이블을 채우는 순서가
일정하지 않음



DP 테이블을 채우는 순서를 구해야 함
-> 위상정렬
2주 뒤에 배움

내리막 길 / 1520

C++

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= m;j++){
            cin >> a[i][j];
            dp[i][j] = -1;
        }
    }
    cout << solve(n, m);

    return 0;
}
```

```cpp
const ll MAX = 505;
ll n, m, dp[MAX][MAX], a[MAX][MAX];
ll dx[4] = {0, 0, 1, -1}, dy[4] = {1, -1, 0, 0};

bool outrange(ll cy, ll cx){
    return cx <= 0 || cy <= 0 || cx > m || cy > n;
}

ll solve(ll cy, ll cx){
    ll& ret = dp[cy][cx];
    if(ret != -1) return ret; ret = 0;
    // Base Case
    if(cy == 1 && cx == 1) return ret = 1;

    for(int i = 0;i < 4;i++){
        ll ny = cy + dy[i], nx = cx + dx[i];
        // 격자 밖이면 건너 뜀
        if(outrange(ny, nx)) continue;
        // 내리막 길이 아니면 건너 뜀
        if(a[ny][nx] <= a[cy][cx]) continue;
        ret += solve(ny, nx);
    }

    return ret;
}
```

내리막 길 / 1520

Python

```python
n, m = map(int, input().split())
a  = [[0] * (m+1) for _ in range(n+1)]
dp = [[-1] * (m+1) for _ in range(n+1)]

for i in range(1, n+1):
    row = list(map(int, input().split()))
    for j in range(1, m+1):
        a[i][j] = row[j-1]

print(solve(n, m))
```

```python
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def solve(cy, cx):
    if dp[cy][cx] != -1:
        return dp[cy][cx]

    # Base Case
    if cy == 1 and cx == 1:
        dp[cy][cx] = 1
        return 1

    ret = 0
    for i in range(4):
        ny = cy + dy[i]
        nx = cx + dx[i]
        # 격자 밖이면 건너 뜀
        if not (1 <= ny <= n and 1 <= nx <= m):
            continue

        # 내리막 길이 아니면 건너 뜀
        if a[ny][nx] <= a[cy][cx]:
            continue
        ret += solve(ny, nx)

    dp[cy][cx] = ret
    return ret
```

질문?

7주차 - 다익스트라

다익스트라

다익스트라
    최단 경로 알고리즘

최단 경로 알고리즘
    다익스트라, 벨만 포드, 플로이드, SPFA …

위의 알고리즘 중에서 다익스트라가 가장 많이 쓰임

다익스트라

다익스트라

최단 경로 알고리즘

어떤 그래프의 한 정점에 대해서 다른 모든 정점에 대한
최단 경로를 구할 수 있음

단 모든 간선의 비용이 음수이면 안됨
-> 시간 복잡도 보장이 안됨
-> 음수의 사이클이 존재하면 무한 루프가 돌음

다익스트라

다익스트라

0-1. 거리 배열을 큰 값으로 초기화

0-2. 시작 정점을 방문할 수 있는 정점으로 처리함

1. 방문할 수 있는 정점 중 가장 최단 거리인 정점을 방문

2. 현재 정점과 인접한 정점들을 방문함

2-1. 인접한 정점의 거리가 원래 거리보다 최단거리이면
방문할 수 있는 정점으로 처리함

2-2. 원래 거리가 인접한 정점의 거리보다 최단거리이면
이미 최단거리이므로 건너 뜀

다익스트라

다익스트라

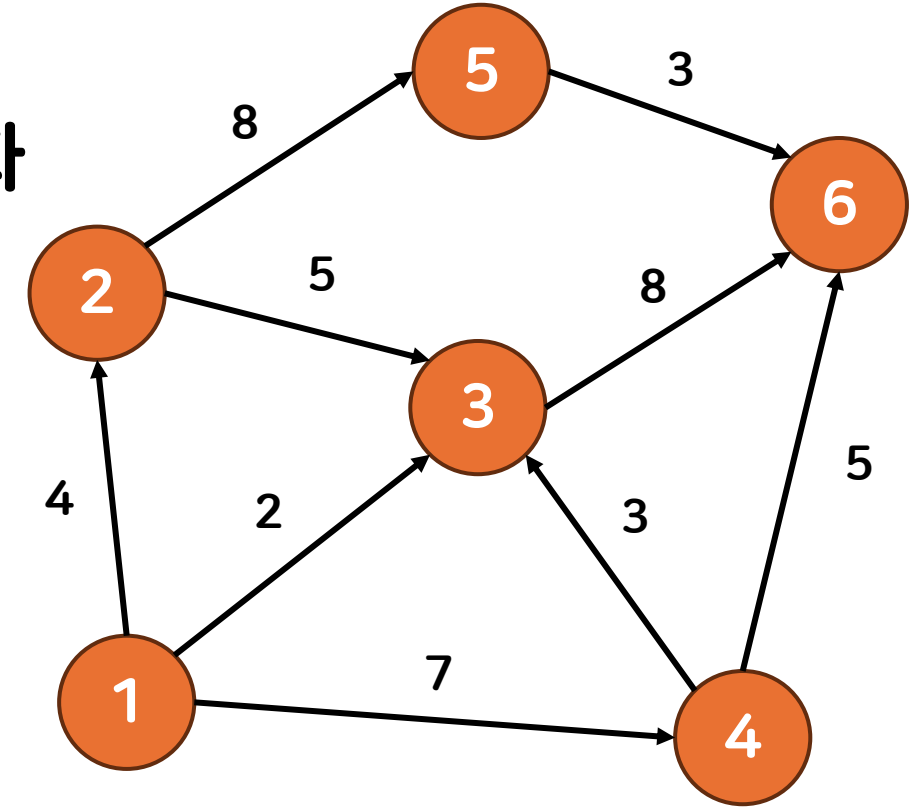
1. 방문할 수 있는 정점 중 가장 최단 거리인 정점을 방문

가장 최단 거리인 정점을 알아야 함
-> 우선순위 큐 사용

다익스트라

다익스트라

0-1. 모든 경로를 큰 값으로 초기화



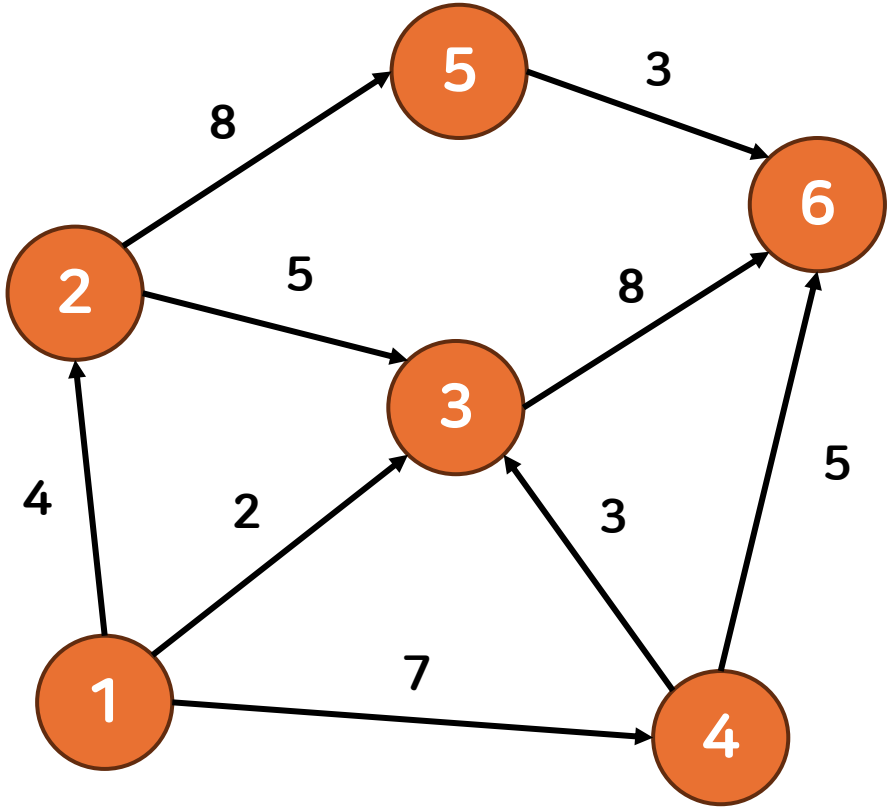
1	2	3	4	5	6
INF	INF	INF	INF	INF	INF

다익스트라

다익스트라

0-2. 시작 정점을 방문 할 수 있는 정점으로 처리함

시작 정점은 거리가 0



(0, 1)

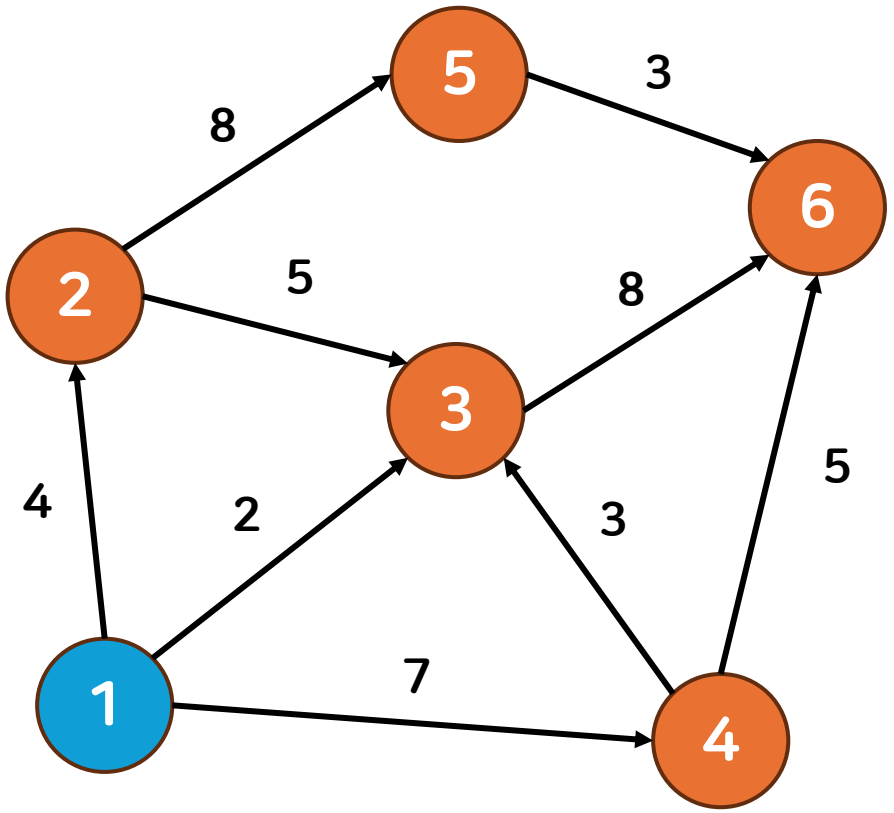
1	2	3	4	5	6
INF	INF	INF	INF	INF	INF

다익스트라

다익스트라

1. 방문할 수 있는 정점 중 가장 최단 거리인 정점을 방문

거리 배열의 값이 우선순위 큐의 값보다 작으므로 거리 갱신



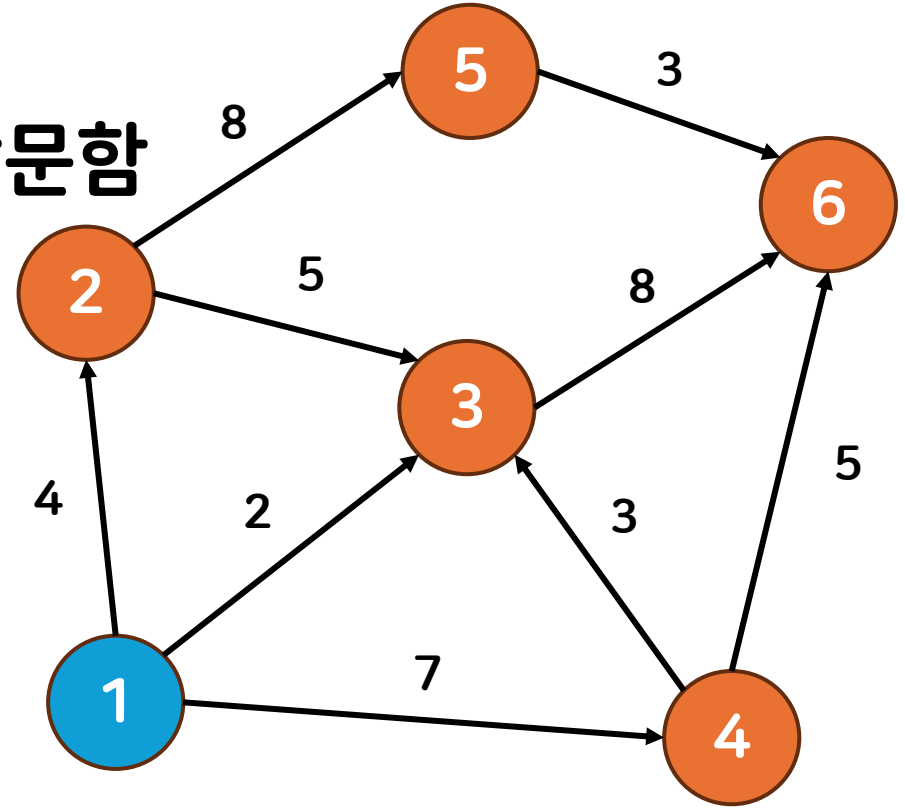
1	2	3	4	5	6
0	INF	INF	INF	INF	INF

다익스트라

다익스트라

2. 현재 정점과 인접한 정점들을 방문함

2-1. 인접한 정점의 거리가 원래 거리보다 최단거리이면 방문할 수 있는 정점으로 처리함



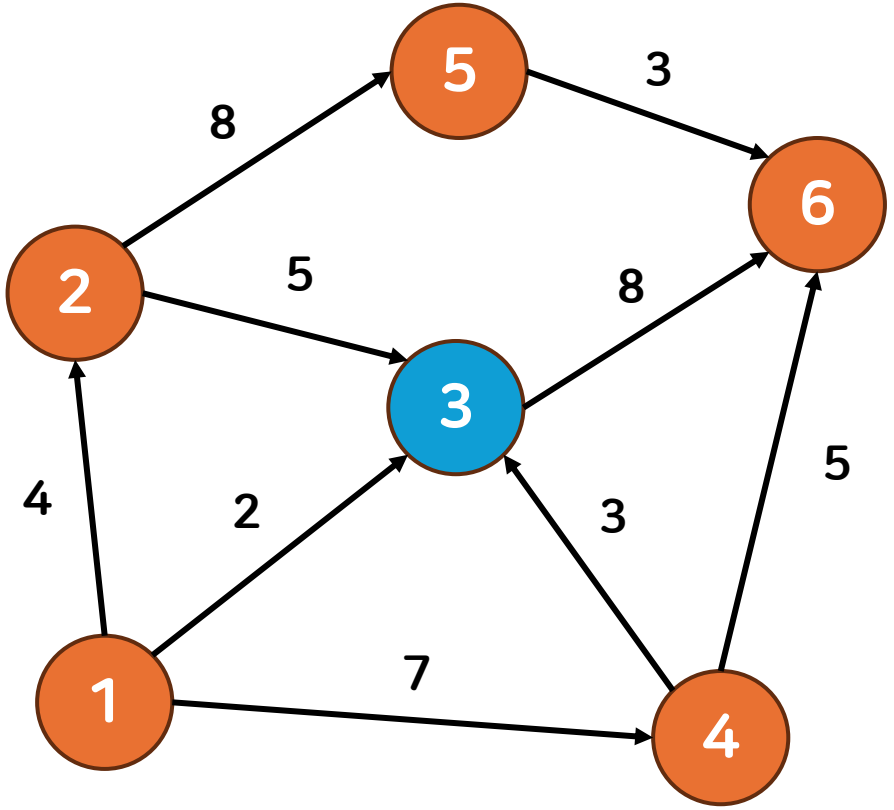
(2, 3) (4, 2) (7, 4)

1	2	3	4	5	6
0	INF	INF	INF	INF	INF

다익스트라

다익스트라

1. 방문할 수 있는 정점 중 가장 최단 거리인 정점을 방문



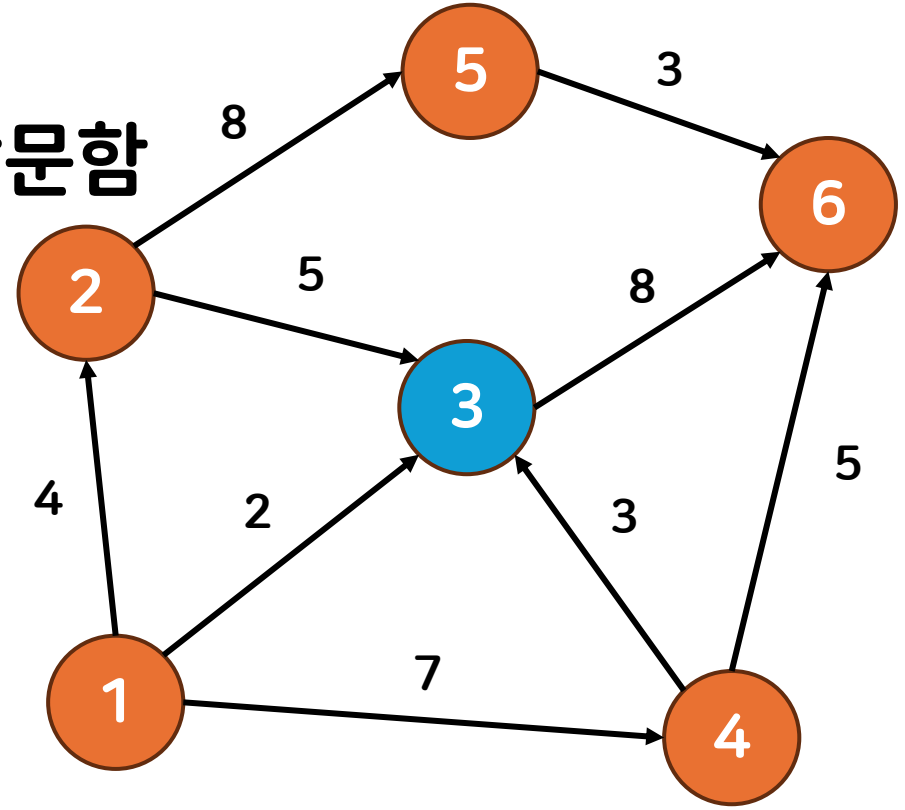
(4, 2) (7, 4)

1	2	3	4	5	6
0	INF	2	INF	INF	INF

다익스트라

다익스트라

2. 현재 정점과 인접한 정점들을 방문함



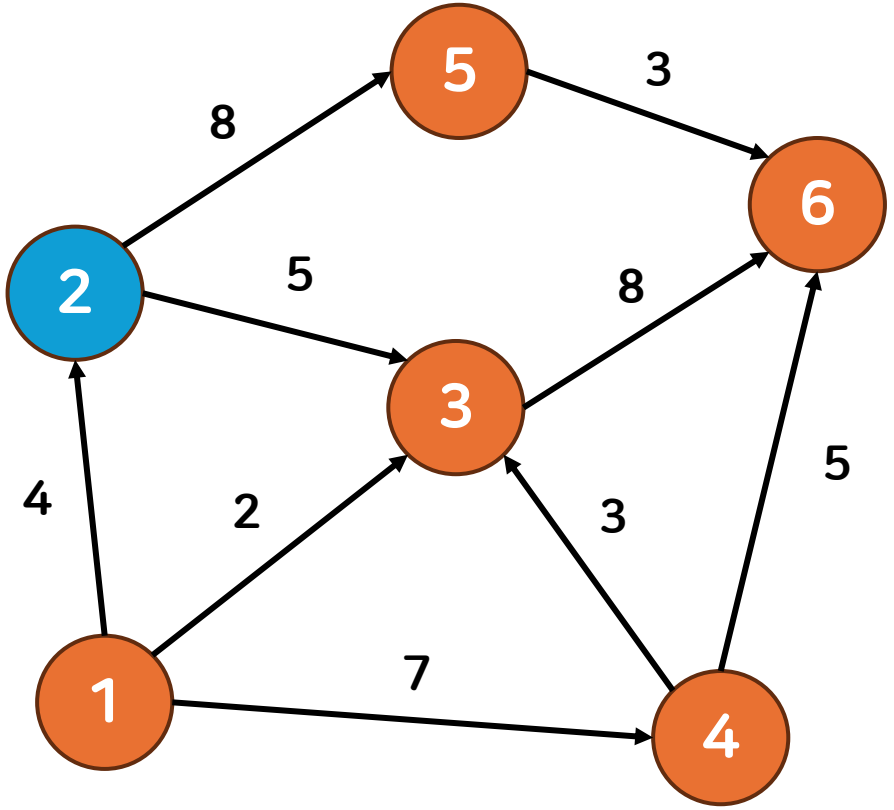
(4, 2)	(7, 4)	(10, 6)

1	2	3	4	5	6
0	INF	2	INF	INF	INF

다익스트라

다익스트라

1. 방문할 수 있는 정점 중 가장 최단 거리인 정점을 방문



(7, 4) (10, 6)

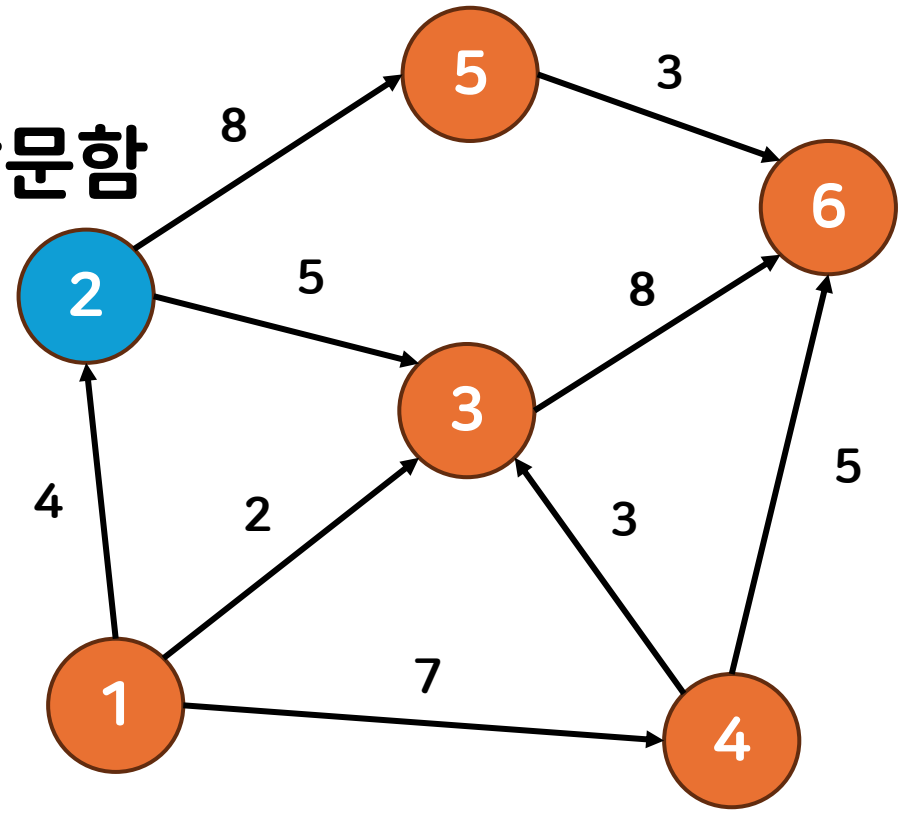
1	2	3	4	5	6
0	4	2	INF	INF	INF

다익스트라

다익스트라

2. 현재 정점과 인접한 정점들을 방문함

정점 3은 이미 최단거리 이므로 갱신 X

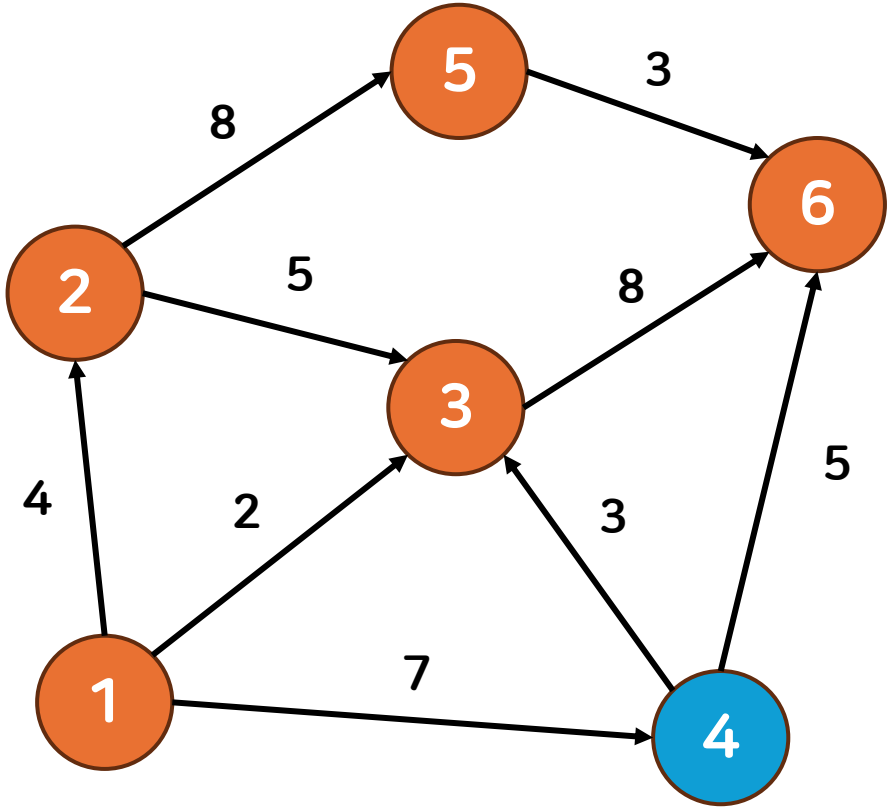


(7, 4)	(10, 6)	(12, 5)

1	2	3	4	5	6
0	4	2	INF	INF	INF

다익스트라

다익스트라

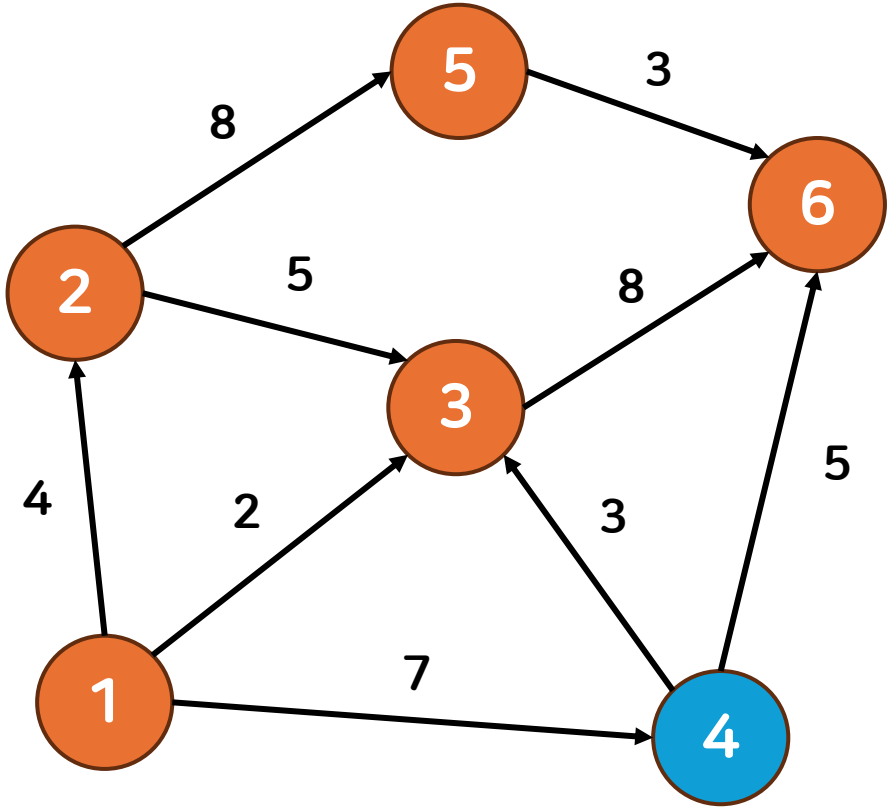


(10, 6) (12, 5)

1	2	3	4	5	6
0	4	2	7	INF	INF

다익스트라

다익스트라

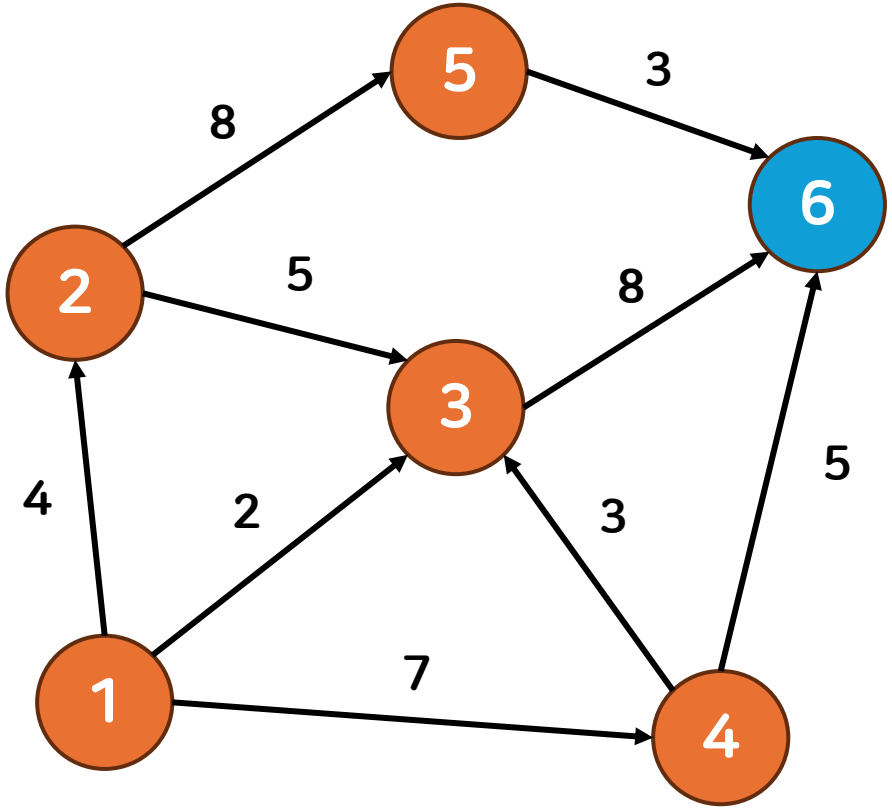


(10, 6)	(12, 5)	(12, 6)

1	2	3	4	5	6
0	4	2	7	INF	INF

다익스트라

다익스트라

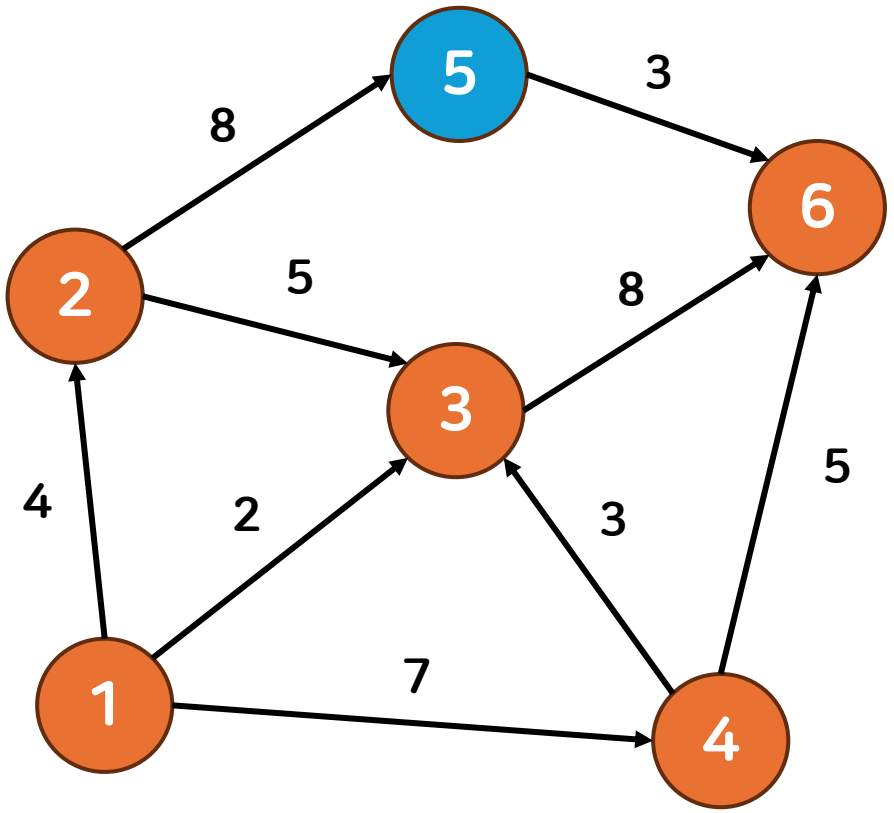


(12, 5) (12, 6)

1	2	3	4	5	6
0	4	2	7	INF	10

다익스트라

다익스트라

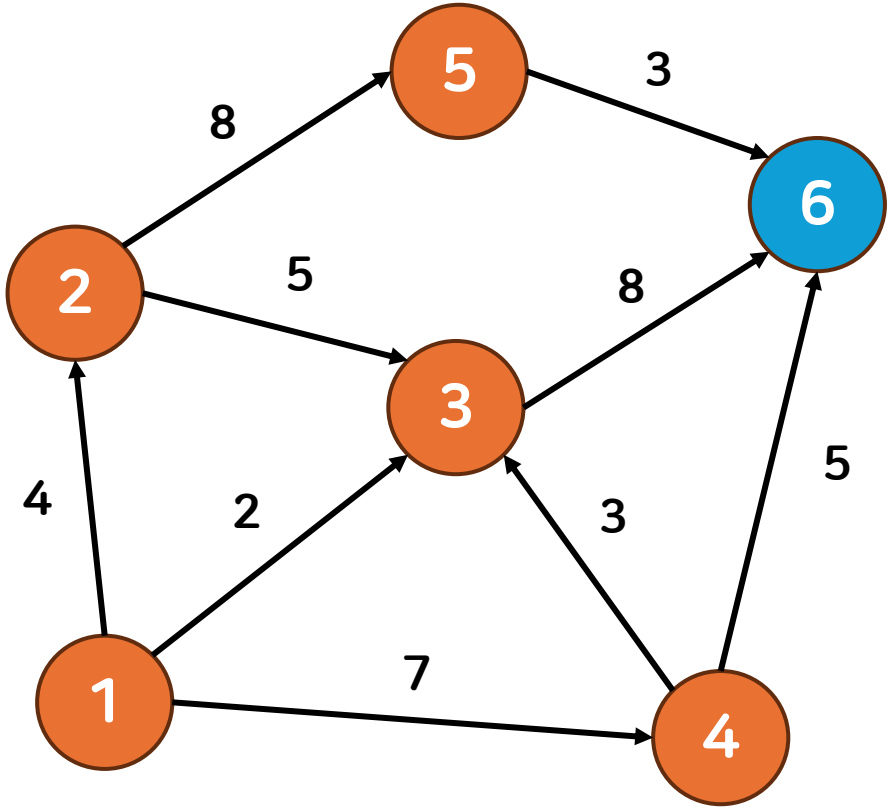


(12, 6)

1	2	3	4	5	6
0	4	2	7	12	10

다익스트라

다익스트라



1	2	3	4	5	6
0	4	2	7	12	10

다익스트라

시간 복잡도?

간선의 개수 -> E, 정점의 개수 -> V

항상 우선순위 큐에서 제일 거리가 작은 값을 빼서 사용함
우선순위 큐에 값이 아무리 많아도 V ^ 2를 넘을 수 없음

우선순위 큐 연산 -> O(log (V ^ 2)) = O(2 log V)
-> O(log V)

다익스트라

시간 복잡도?

간선의 개수 -> E, 정점의 개수 -> V

우선순위 큐 연산 -> O(log V)

어떤 정점을 방문할 때는 항상 최단거리인 상태에서 방문하게 됨

-> 각 정점은 최대 한번 방문함

-> 각 간선도 최대 한번 방문함

우선순위 큐에 값은 최대 E번 들어가게 됨

다익스트라

시간 복잡도?

간선의 개수 -> E, 정점의 개수 -> V

우선순위 큐 연산 -> O(log V)

우선순위 큐에 값은 최대 E번 들어가게 됨

-> O(E log V)

최소비용 구하기 / 1916

백준 1916 / https://www.acmicpc.net/problem/1916

문제

N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다. 우리는 A번째 도시에서 B번째 도시까지 가는데 드는 버스 비용을 최소화 시키려고 한다. A번째 도시에서 B번째 도시까지 가는데 드는 최소비용을 출력하여라. 도시의 번호는 1부터 N까지이다.

입력

첫째 줄에 도시의 개수 N(1 ≤ N ≤ 1,000)이 주어지고 둘째 줄에는 버스의 개수 M(1 ≤ M ≤ 100,000)이 주어진다. 그리고 셋째 줄부터 M+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 그리고 그 다음에는 도착지의 도시 번호가 주어지고 또 그 버스 비용이 주어진다. 버스 비용은 0보다 크거나 같고, 100,000보다 작은 정수이다.

그리고 M+3째 줄에는 우리가 구하고자 하는 구간 출발점의 도시번호와 도착점의 도시번호가 주어진다. 출발점에서 도착점을 갈 수 있는 경우만 입력으로 주어진다.

출력

첫째 줄에 출발 도시에서 도착 도시까지 가는데 드는 최소 비용을 출력한다.

최소비용 구하기 / 1916

N개의 정점과 M개의 간선이 주어짐
마지막 줄에 시작점 S와 도착점 E가 주어졌을 때
S -> E의 최단 경로를 구하는 문제

예제 입력 1 복사

```
5
8
1 2 2
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
1 5
```

예제 출력 1 복사

```
4
```

최소비용 구하기 / 1916

C++

```cpp
const ll MAX = 1010;
const ll INF = 1e12;
ll n, m, d[MAX];
vector <pair<ll, ll>> adj[MAX];

using pll = pair<ll, ll>;
priority_queue <pll, vector<pll>, greater<pll>> pq;

int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    while(m--){
        ll s, e, c; cin >> s >> e >> c;
        adj[s].push_back({e, c});
    }
    ll s, e; cin >> s >> e;
```

```cpp
    // 거리 배열 큰 값으로 초기화
    for(int i = 0;i < MAX;i++) d[i] = INF;
    // 시작 정점을 방문 가능한 정점으로 처리
    pq.push({0, s});

    while(!pq.empty()){
        // 방문 가능한 정점 중 최단 거리인 정점 방문
        auto[cd, cur] = pq.top(); pq.pop();
        // 이미 최단거리이면 건너 뜀
        if(d[cur] <= cd) continue;
        // 거리 갱신
        d[cur] = cd;

        for(auto& [nxt, co] : adj[cur]){
            // 다음 정점이 이미 최단거리이면 건너 뜀
            if(d[nxt] <= cd + co) continue;
            // 방문 가능한 정점에 포함
            pq.push({cd + co, nxt});
        }
    }

    cout << d[e];

    return 0;
}
```

최소비용 구하기 / 1916

Python

```python
import sys
import heapq
input = sys.stdin.readline

INF = 10**12

n = int(input().rstrip())
m = int(input().rstrip())
adj = [[] for _ in range(n + 1)]
for _ in range(m):
    s, e, c = map(int, input().split())
    adj[s].append((e, c))

s, e = map(int, input().split())

# 거리 배열 큰 값으로 초기화
dist = [INF] * (n + 1)
```

```python
pq = []
# 시작 정점을 방문 가능한 정점으로 처리
heapq.heappush(pq, (0, s))

while pq:
    # 방문 가능한 정점 중 최단 거리인 정점 방문
    cd, cur = heapq.heappop(pq)
    # 이미 최단거리이면 건너 뜀
    if dist[cur] <= cd:
        continue

    # 거리 갱신
    dist[cur] = cd
    for nxt, co in adj[cur]:
        nd = cd + co
        # 현재 거리가 최단 거리이면
        if dist[nxt] > nd:
            # 방문 가능한 정점에 포함
            heapq.heappush(pq, (nd, nxt))

print(dist[e])
```

질문?

파티 / 1238

백준 1238 / https://www.acmicpc.net/problem/1238

문제

N개의 숫자로 구분된 각각의 마을에 한 명의 학생이 살고 있다.

어느 날 이 N명의 학생이 X (1 ≤ X ≤ N)번 마을에 모여서 파티를 벌이기로 했다. 이 마을 사이에는 총 M개의 단방향 도로들이 있고 i번째 길을 지나는데 T_i(1 ≤ T_i ≤ 100)의 시간을 소비한다.

각각의 학생들은 파티에 참석하기 위해 걸어가서 다시 그들의 마을로 돌아와야 한다. 하지만 이 학생들은 워낙 게을러서 최단 시간에 오고 가기를 원한다.

이 도로들은 단방향이기 때문에 아마 그들이 오고 가는 길이 다를지도 모른다. N명의 학생들 중 오고 가는데 가장 많은 시간을 소비하는 학생은 누구일지 구하여라.

입력

첫째 줄에 N(1 ≤ N ≤ 1,000), M(1 ≤ M ≤ 10,000), X가 공백으로 구분되어 입력된다. 두 번째 줄부터 M+1번째 줄까지 i번째 도로의 시작점, 끝점, 그리고 이 도로를 지나는데 필요한 소요시간 T_i가 들어온다. 시작점과 끝점이 같은 도로는 없으며, 시작점과 한 도시 A에서 다른 도시 B로 가는 도로의 개수는 최대 1개이다.

모든 학생들은 집에서 X에 갈수 있고, X에서 집으로 돌아올 수 있는 데이터만 입력으로 주어진다.

출력

첫 번째 줄에 N명의 학생들 중 오고 가는데 가장 오래 걸리는 학생의 소요시간을 출력한다.

파티 / 1238

정점의 개수 N, 간선의 개수 M, 정점 X가 주어짐
Dist[i][X] + Dist[X][i]의 최댓값을 구하는 문제

예제 입력 1 복사

```
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3
```

예제 출력 1 복사

```
10
```

파티 / 1238

Dist[i][X] + Dist[X][i]의 최댓값을 구하는 문제

다익스트라는 한 정점에 대한 최단 거리만 구할 수 있음
Dist[X][i]는 X을 기준으로 다익스트라를 구하면 됨

Dist[i][X]를 구하는 방법?

파티 / 1238

Dist[i][X]를 구하는 방법?

간단하게 생각하면 모든 정점에 대해서 다익스트라를 돌리면 됨
그러면 모든 정점 쌍의 최단거리를 구할 수 있음

시간 복잡도?

다익스트라 O(M log N)
정점이 N개이므로 O(NM log N)
= 1000 * 1000 * 10 = 1억

파티 / 1238

시간 복잡도?

다익스트라 O(M log N)

정점이 N개이므로 O(NM log N)

= 1000 * 1000 * 10 = 1억

제한 시간이 1초이므로 애매하긴 하지만 돌아가긴 함

-> 이 풀이가 정해는 아님

파티 / 1238

Dist[i][X]를 구하는 방법?

Dist[i][X]는 i -> X로 가는 최단 거리

거꾸로 생각해보자
-> 모든 간선들을 뒤집어보자

파티 / 1238

Dist[i][X]는 i -> X로 가는 최단 거리

모든 간선들의 방향을 뒤집었을 때
Dist[i][X] = X -> i로 가는 최단 거리

모든 역방향 간선들에 대해서 X를 기준으로 둔 다익스트라

Dist[i][X]와 Dist[X][i]를 구했으므로
둘의 합의 최댓값을 찾으면 됨

파티 / 1238

C++

```cpp
const ll MAX = 1010;
const ll INF = 1e12;
ll n, m, x, d[MAX][2];
vector <pair<ll, ll>> adj[MAX][2];

using pll = pair<ll, ll>;
priority_queue <pll, vector<pll>, greater<pll>> pq;

int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m >> x;
    while(m--){
        ll s, e, c; cin >> s >> e >> c;
        // 정방향 간선
        adj[s][0].push_back({e, c});
        // 역방향 간선
        adj[e][1].push_back({s, c});
    }

    for(int i = 0;i < MAX;i++){
        for(int j = 0;j < 2;j++) d[i][j] = INF;
    }
```

```cpp
    for(int i = 0;i <= 1;i++){
        while(!pq.empty()) pq.pop();
        pq.push({0, x});

        while(!pq.empty()){
            auto[cd, cur] = pq.top(); pq.pop();
            if(d[cur][i] <= cd) continue;
            d[cur][i] = cd;

            for(auto& [nxt, co] : adj[cur][i]){
                if(d[nxt][i] <= cd + co) continue;
                pq.push({cd + co, nxt});
            }
        }
    }

    ll result = 0;
    for(int i = 1;i <= n;i++){
        // 역방향 간선 + 정방향 간선의 최댓값
        result = max(result, d[i][0] + d[i][1]);
    }
    cout << result;

    return 0;
}
```

파티 / 1238

Python

```python
import sys
import heapq
input = sys.stdin.readline

INF = 10**12
n, m, x = map(int, input().split())

adj = [[[] for _ in range(2)] for _ in range(n+1)]
for _ in range(m):
    s, e, c = map(int, input().split())
    # 정방향 간선
    adj[s][0].append((e, c))
    # 역방향 간선
    adj[e][1].append((s, c))
```

```python
d = [[INF, INF] for _ in range(n+1)]
for i in range(2):
    pq = []
    heapq.heappush(pq, (0, x))
    while pq:
        cd, cur = heapq.heappop(pq)
        if d[cur][i] <= cd:
            continue
        d[cur][i] = cd
        for nxt, co in adj[cur][i]:
            nd = cd + co
            if d[nxt][i] > nd:
                heapq.heappush(pq, (nd, nxt))

result = 0
for i in range(1, n+1):
    # 역방향 간선 + 정방향 간선의 최댓값
    result = max(result, d[i][0] + d[i][1])

print(result)
```

질문?

질문?

기본 과제

최소비용 구하기 – https://www.acmicpc.net/problem/1916

파티 - https://www.acmicpc.net/problem/1238

해킹 – https://www.acmicpc.net/problem/10282

소가 길을 건너간 이유 7 -
https://www.acmicpc.net/problem/14461

수열과 개구리 - https://www.acmicpc.net/problem/32294

고생하셨습니다