JooHyun – Lee (comkiwer)

꿀꿀이 축제

Hancom Education Co. Ltd.

Problem

Problem

꿀꿀이 축제

N개의 마을에 꿀꿀이들이 한 마리씩 살고 있다.

이 N마리의 꿀꿀이들이 X (1 ≤ X ≤ N)번 마을에 모여 축제를 열기로 했다.

마을 사이에는 총 M개의 단방향 도로들이 있고 i번 길을 걸어서 통과하는데 Ti(1 ≤ Ti ≤ 100)의 시간이 소비된다.

각 꿀꿀이들은 파티에 참석하기 위해 X번 마을까지 걸어가서 축제를 즐긴 후

다시 자신의 마을로 돌아온다.

걷는 것이 힘든 꿀꿀이들은 가능한 짧은 시간에 오고 가기를 원하므로 최단 경로를 이용한다고 한다.

도로들은 단방향이기 때문에 꿀꿀이들이 오고 가는 길이 다를 수 있다.

모든 꿀꿀이 각각은 축제에 다녀오는 최단 경로가 존재한다.

N마리의 꿀꿀이들 중 오고 가는데 가장 많은 시간이 걸리는 꿀꿀이의 소요시간을 구하시오.

Problem

꿀꿀이 축제

[입력 형식]

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 Ti가 들어온다.

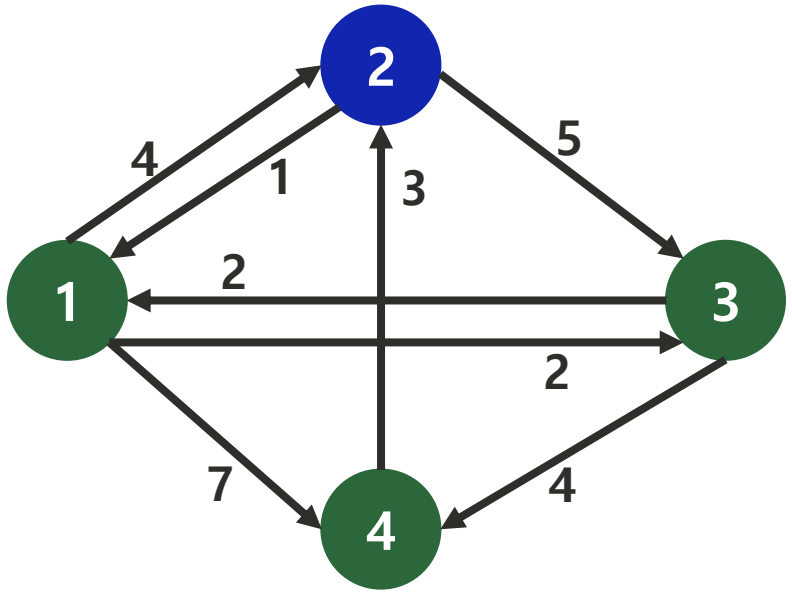
[출력 형식]

첫 번째 줄에 N명의 꿀꿀이들 중 오고 가는데 가장 오래 걸리는 꿀꿀이의 소요시간을 출력한다.

Problem analysis

Problem analysis : 예제

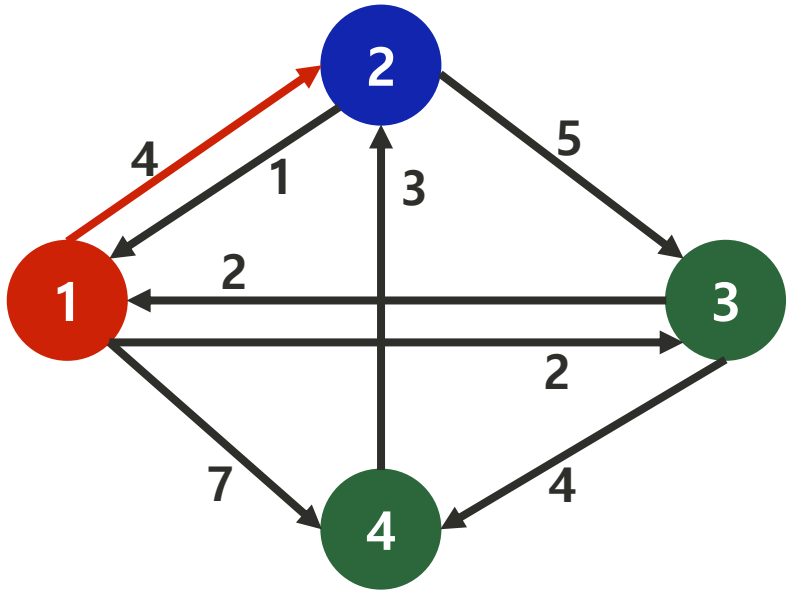
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3



4 개의 마을이 있고
8 개의 길이 주어지는데
2 번 마을이 축제가 열리는 마을이다.

Problem analysis : 예제

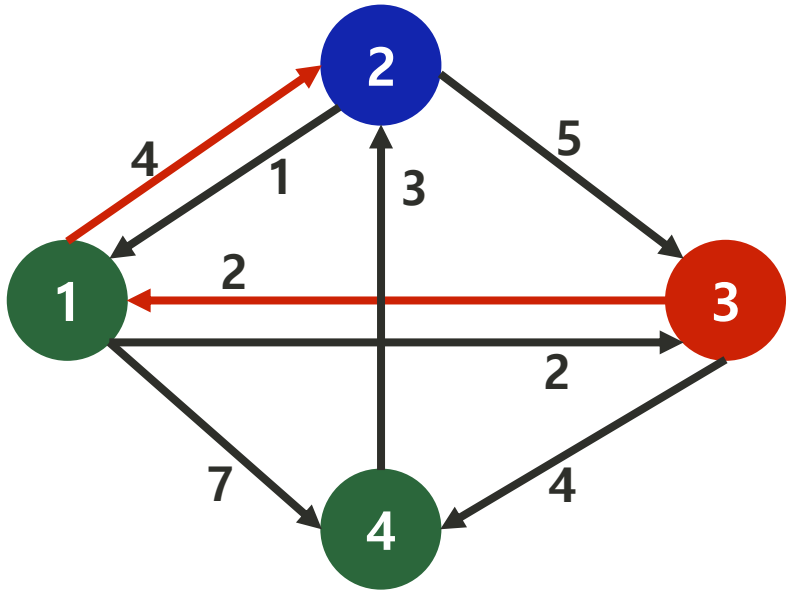
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3



1->2 : 4
1번 마을에서 2번 마을까지
최단 거리는 4이다.

Problem analysis : 예제

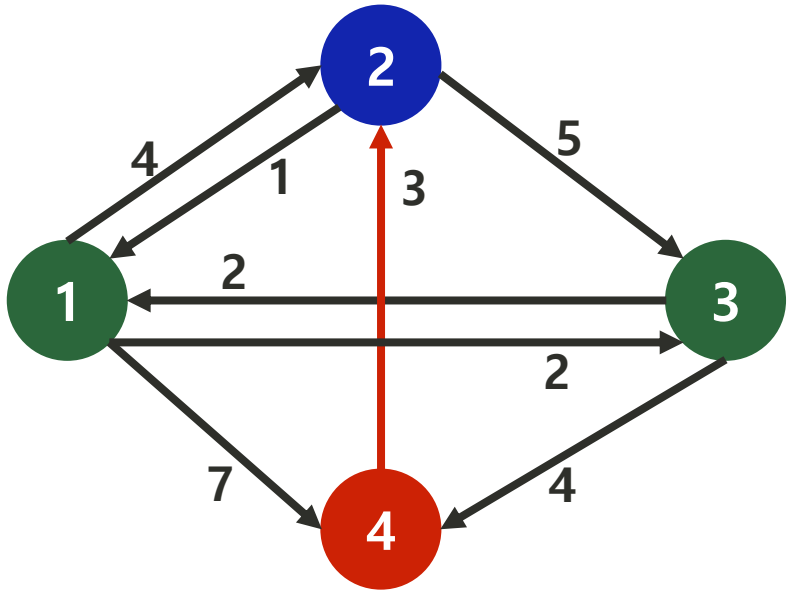
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3



1->2 : 4
3->2 : 6
3번 마을에서 2번 마을까지
최단 거리는6이다.

Problem analysis : 예제

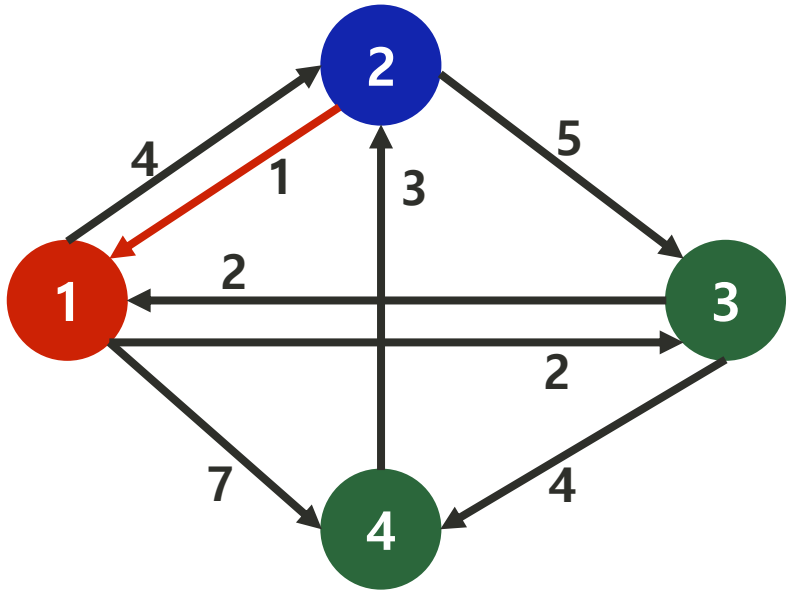
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3



1->2 : 4
3->2 : 6
4->2 : 3
4번 마을에서 2번 마을까지
최단 거리는3이다.

Problem analysis : 예제

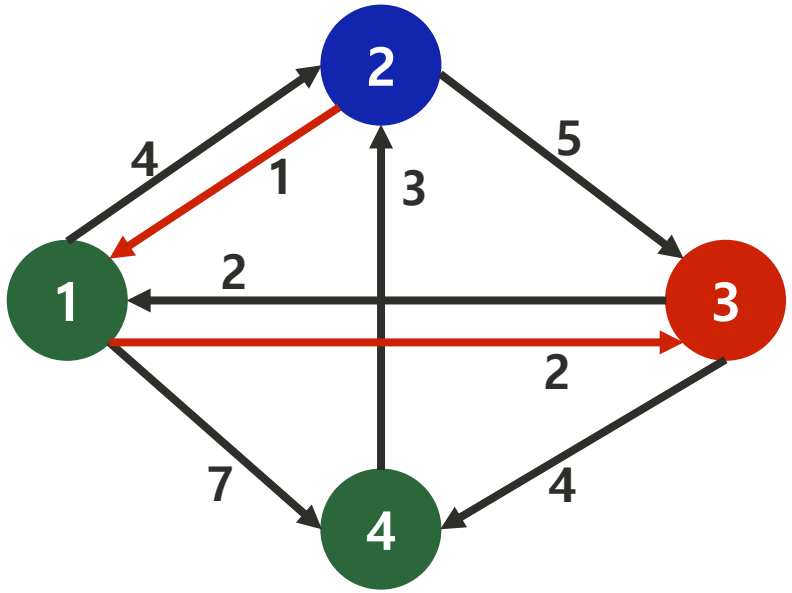
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3



2->1 : 1
2번 마을에서 1번 마을까지
최단 거리는 1이다.

Problem analysis : 예제

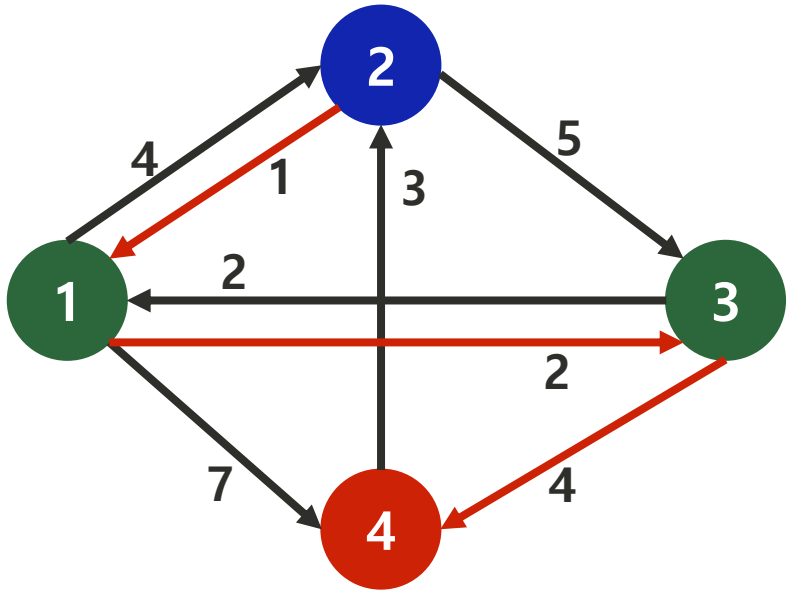
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3



2->1 : 1
2->3 : 3
2번 마을에서 3번 마을까지
최단 거리는 3이다.

Problem analysis : 예제

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

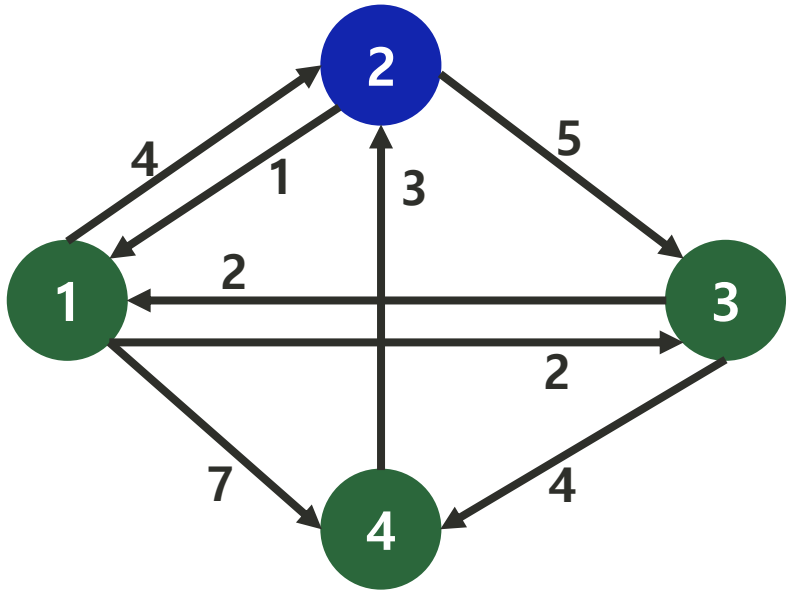


2->1 : 1
2->3 : 3
2->4 : 7

2번 마을에서 4번 마을까지
최단 거리는 7이다.

Problem analysis : 예제

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



1->2->1 : 4 + 1 = 5
3->2->3 : 6 + 3 = 9
4->2->4 : 3 + 7 = 10

이므로
가장 먼 거리는 10이다.

Problem analysis : 제약 조건

- 마을의 수는 최대 1,000 개, 길의 수는 최대 10,000 개이다.
- 마을을 노드로 길은 방향과 가중치가 있는 간선으로 생각하여 그래프 문제로 다룰 수 있다.
- 각 마을 별로 축제 장으로 가는 최단거리 + 돌아오는 최단거리를 구하고
  이들 중에 최대값을 답으로 하면 된다.
- 축제가 열리는 마을에서 각자의 집으로 돌아가는 경우는
  출제가 열리는 마을로부터 나머지 마을까지 최단거리를 구하는 문제로 생각할 수 있다.

  이때 시간복잡도는 $O(|V|^2)$ 또는 $O(|E| \log(|V|))$ 이다.

Problem analysis : 제약 조건

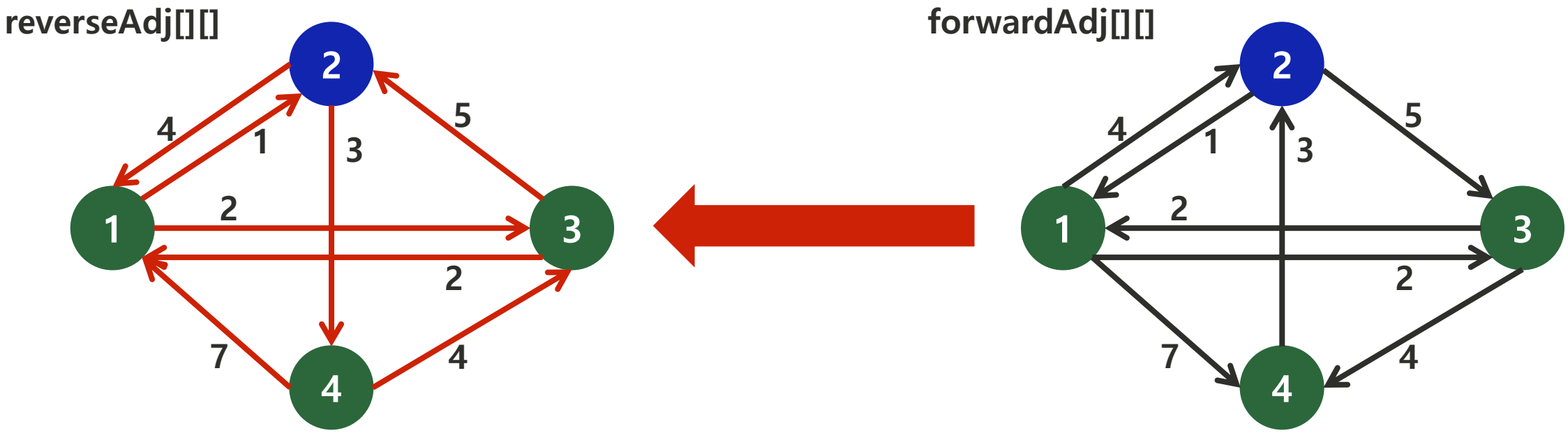
- 그런데 각 마을로부터 축제장까지의 최단 거리는
  각 마을 별로 최단거리를 구해야 하므로
  이때 시간복잡도는 $O(|V| * |V|^2)$ 또는 $O(|V| * |E| \log(|V|))$ 이 된다.

  너무 비효율적으로 보인다.
  다른 방법이 있을까?

Solution sketch

Solution sketch

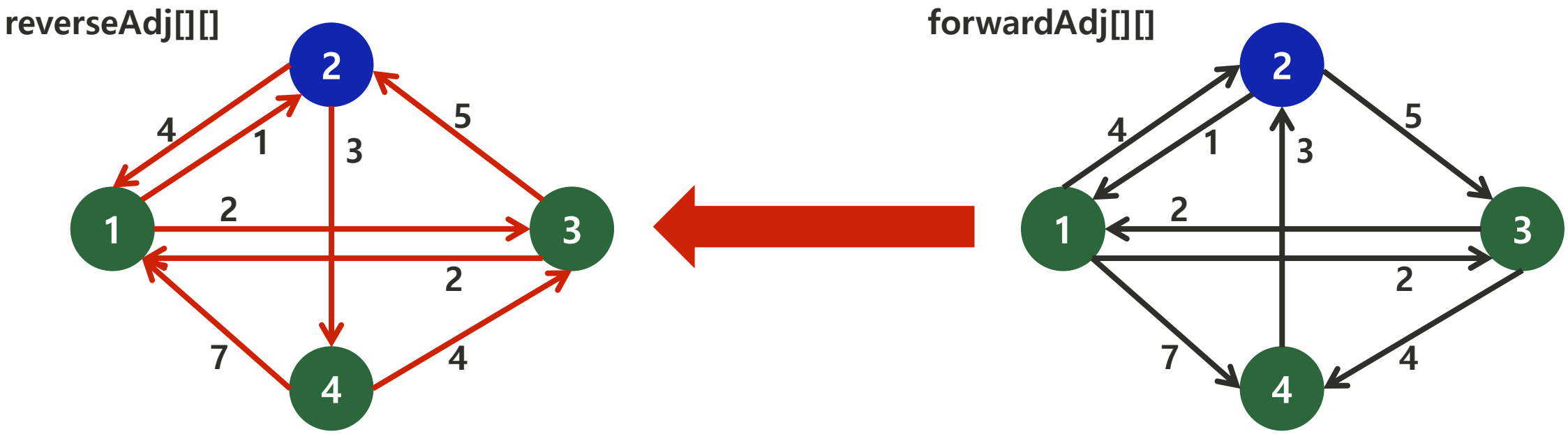
- 각 마을에서 축제 장으로 가는 문제를
  축제 장에서 각 마을로 가는 문제로 바꾸어 생각할 수 있다.
- 이 경우 비용은 각 마을에서 축제 장으로 가는 것으로 계산되어야 하므로
  그래프의 방향을 반대로 뒤집어 생각해야 한다.
  따라서 예제의 오른쪽 그래프는 왼쪽처럼 바꾸어 계산한다.



Solution sketch

- reverseAdj[][]를 이용하여 각 마을에서 축제의 장으로 가는 최단 거리를 계산하고 forwardAdj[][]를 이용하여 축제의 장에서 각 마을로 돌아가는 최단 거리를 계산하여 문제를 해결 할 수 있다.

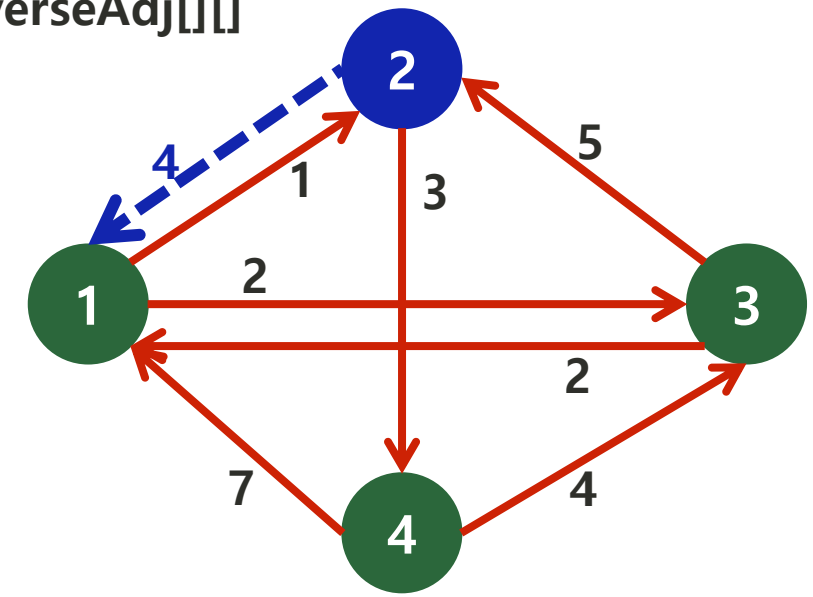
  이때 시간복잡도는 둘 다 $O(|V|^2)$ 또는 $O(|E| \log(|V|))$ 이다.



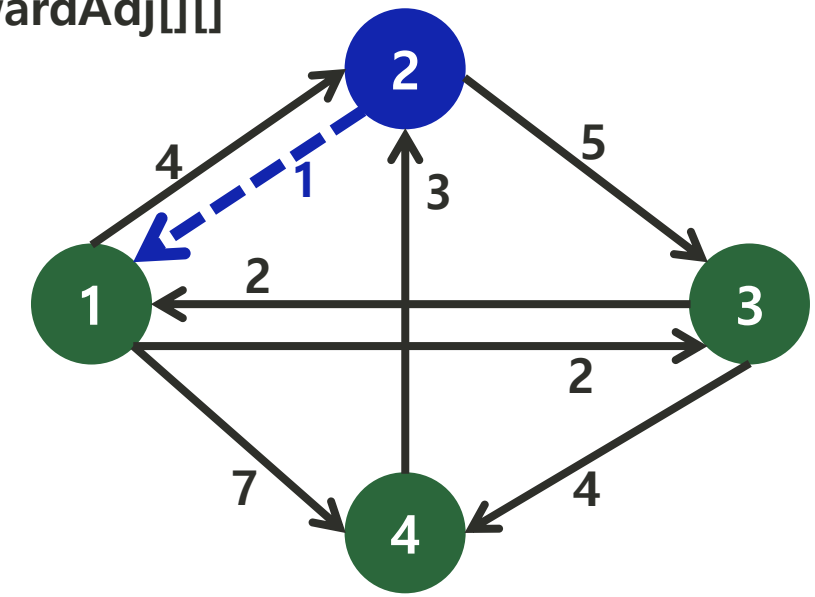
Solution sketch

- 1번 마을에서 2번 마을 축제의 장으로 가는 최단 경로는 reverseAdj[][]를 이용하여 2번 마을에서 1번 마을로 가는 것으로 생각하여 구할 수 있다. : 최단 거리 4
- 2번 마을에서 1번 마을로 돌아가는 최단 경로는 forwardAdj[][]를 이용하여 구할 수 있다. : 최단 거리 1

reverseAdj[][]



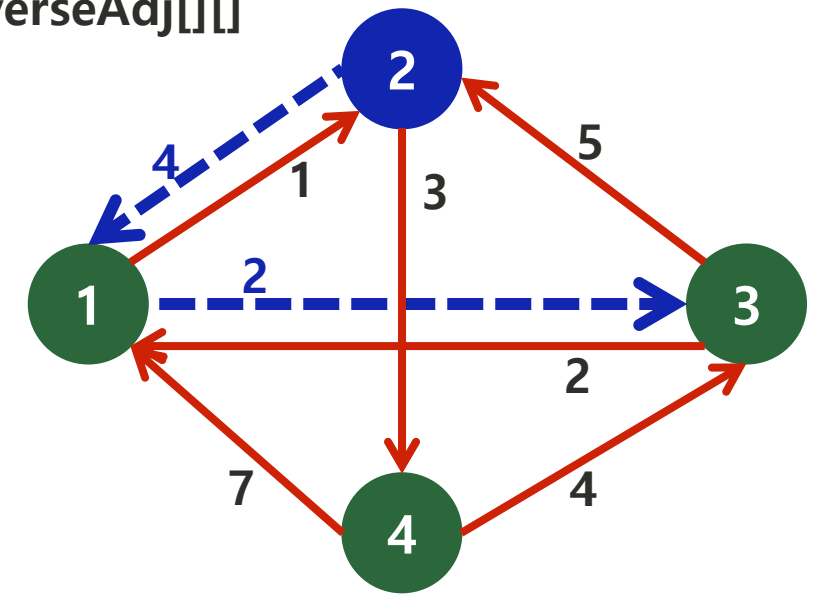
forwardAdj[][]



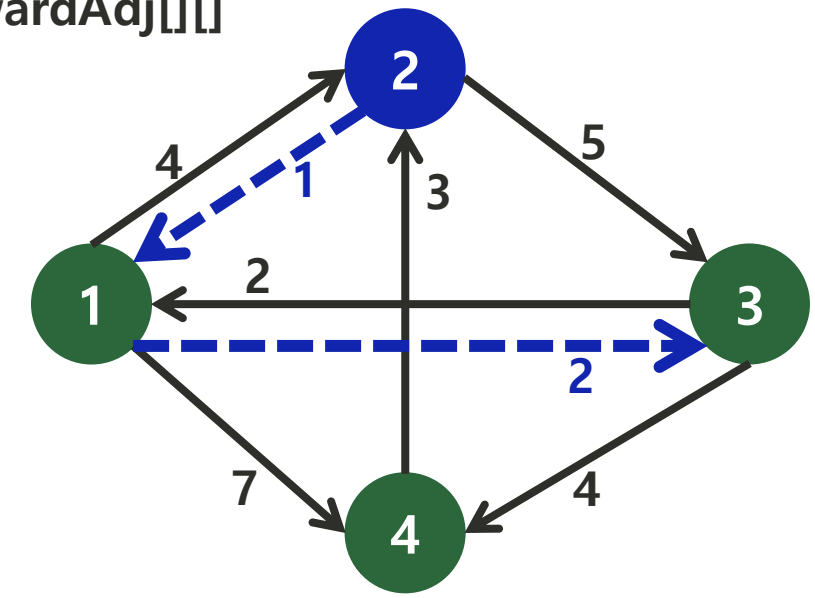
Solution sketch

- 3번 마을에서 2번 마을 축제의 장으로 가는 최단 경로는 reverseAdj[][]를 이용하여 2번 마을에서 3번 마을로 가는 것으로 생각하여 구할 수 있다. : 최단 거리 6
- 2번 마을에서 3번 마을로 돌아가는 최단 경로는 forwardAdj[][]를 이용하여 구할 수 있다. : 최단 거리 3

reverseAdj[][]



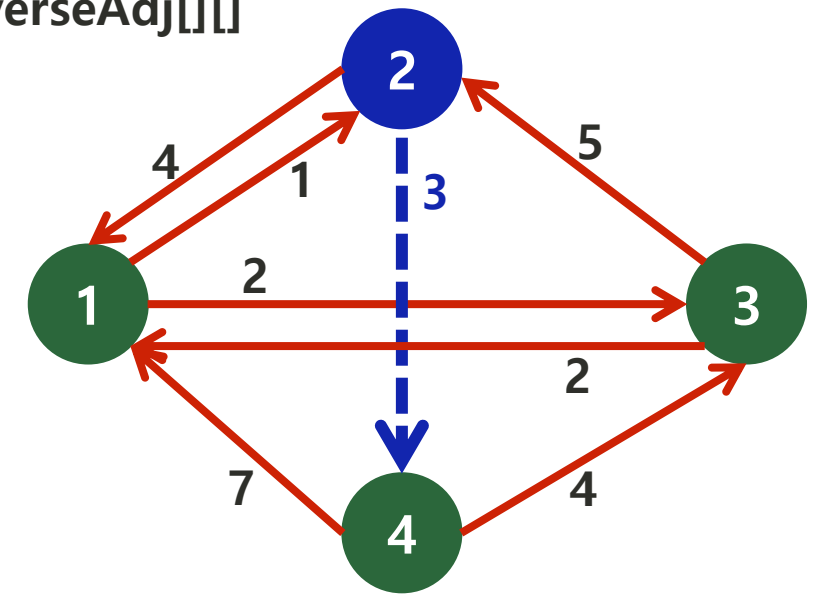
forwardAdj[][]



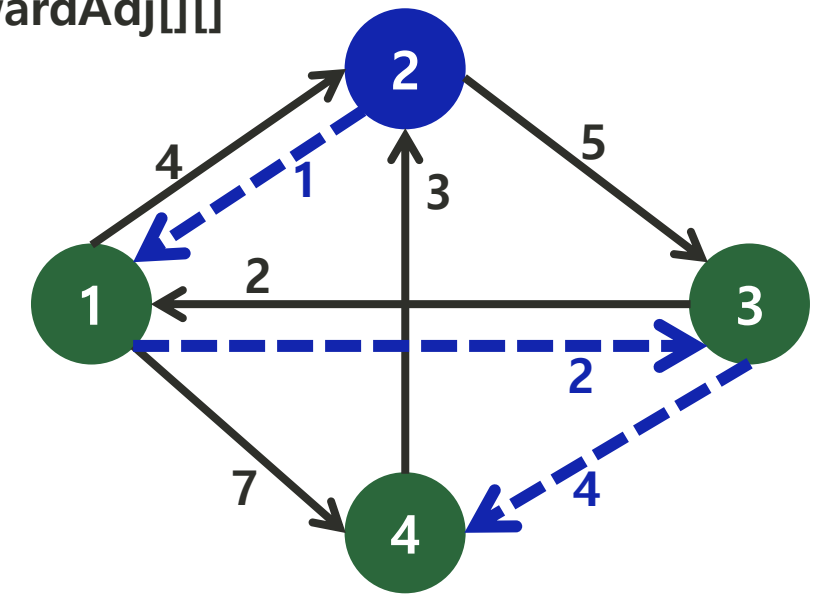
Solution sketch

- 4번 마을에서 2번 마을 축제의 장으로 가는 최단 경로는 reverseAdj[][]를 이용하여 2번 마을에서 4번 마을로 가는 것으로 생각하여 구할 수 있다. : 최단 거리 3
- 2번 마을에서 1번 마을로 돌아가는 최단 경로는 forwardAdj[][]를 이용하여 구할 수 있다. : 최단 거리 7

reverseAdj[][]



forwardAdj[][]



Solution sketch

- 필요한 자료구조를 선언하면 다음과 같다.

```
using pii = pair<int, int>;
const int INF = 1 << 20;
const int LM = 1004;
int N, M, X;
vector<pii> forwardAdj[LM], reverseAdj[LM];  // 인접 배열
int forwardDist[LM];    // 축제 장으로부터 각 마을까지 최단거리
int reverseDist[LM];    // 각 마을로부터 축제 장까지 최단거리
int visited[LM], vcnt;  // 최단거리가 구해진 마을을 체크할 때 사용
```

Solution sketch

- 입력 후 인접 배열을 만드는 과정은 다음과 같다.

```
void input() {
    scanf("%d %d %d", &N, &M, &X);
    int s, e, w, i;
    for (i = 0; i < M; ++i) {
        scanf("%d %d %d", &s, &e, &w);
        forwardAdj[s].push_back({ e, w });
        reverseAdj[e].push_back({ s, w });
    }
}
```

Solution sketch

- 두 인접 배열을 이용하여 최단 거리 함수는 다음과 같다.

```
void Dijkstra(int*dist, vector<pii>*adj) {
    for (int i = 1; i <= N; ++i) dist[i] = INF;
    dist[X] = 0;
    ++vcnt;   // visited[]를 초기화 하는 효과

    for (int i = 1; i <= N; ++i) {
        // extract minNode & minDist
        int minNode = INF, minDist = INF;
        for (int j = 1; j <= N; ++j) {
            if (visited[j] < vcnt && minDist > dist[j])
                minNode = j, minDist = dist[j];
        }

        if (minNode == INF) break;
        // check minNode
        visited[minNode] = vcnt;

        // relaxation
        for (auto&p : adj[minNode]) {
            if (dist[p.first] > minDist + p.second)
                dist[p.first] = minDist + p.second;
        }
    }
}
```

```
void Dijkstra(vector<pii> adj[], int dist[]) {
    // init : implement here
    for (int i = 1; i <= N; ++i) dist[i] = INF;
    dist[X] = 0;

    pq = {}, ++vcnt;
    pq.push({ 0, X });  // pair<int:dist, int: node>

    while (!pq.empty()) {
        // 1. extract minNode & minDist
        int minDist = pq.top().first, minNode = pq.top().second;
        pq.pop();
        if (visited[minNode] == vcnt) continue;

        visited[minNode] = vcnt; // 2. checkNode : implement here

        for (auto&pa : adj[minNode]) {  // 3. relaxation
            int tdist = minDist + pa.second;
            if (dist[pa.first] > tdist) {
                dist[pa.first] = tdist;
                pq.push({ tdist, pa.first });
            }
        }
    }
}
```

Solution sketch

- 메인과 출력함수는 다음과 같다.

```
void output() {
    int res = 0;
    for (int i = 1; i <= N; ++i) {
        res = max(res, forwardDist[i] + reverseDist[i]);
    }
    printf("%d\n", res);
}

int main() {
    input();
    Dijkstra(forwardDist, forwardAdj);
    Dijkstra(reverseDist, reverseAdj);
    output();
    return 0;
}
```

Solution sketch

[Summary]

- 최단 경로를 구하는 그래프 문제를 Dijkstra 알고리즘으로 해결할 수 있다.

- 출발지가 여러 곳이고 도착지가 한 곳인 문제를
  출발지가 한 곳, 도착지가 여러 곳인 문제로 바꾸어 해결 할 수 있다.

Code example 1

: $O(|V|^2)$

Code example 1


```cpp
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

using pii = pair<int, int>;
const int INF = 1 << 20;
const int LM = 1004;
int N, M, X;
vector<pii> forwardAdj[LM], reverseAdj[LM];  // 인접배열
int forwardDist[LM], reverseDist[LM];
int visited[LM], vcnt;

void input() {
    scanf("%d %d %d", &N, &M, &X);
    int s, e, w, i;
    for (i = 0; i < M; ++i) {
        scanf("%d %d %d", &s, &e, &w);
        forwardAdj[s].push_back({ e, w });
        reverseAdj[e].push_back({ s, w });
    }
}
```

Code example 1


```
void Dijkstra(int*dist, vector<pii>*adj) {
    for (int i = 1; i <= N; ++i) dist[i] = INF;
    dist[X] = 0;
    ++vcnt;   // visited[]를 초기화 하는 효과

    for (int i = 1; i <= N; ++i) {
        // extract minNode & minDist
        int minNode = INF, minDist = INF;
        for (int j = 1; j <= N; ++j) {
            if (visited[j] < vcnt && minDist > dist[j])
                minNode = j, minDist = dist[j];
        }

        if (minNode == INF) break;
        // check minNode
        visited[minNode] = vcnt;

        // relaxation
        for (auto&p : adj[minNode]) {
            if (dist[p.first] > minDist + p.second)
                dist[p.first] = minDist + p.second;
        }
    }
}
```

Code example 1


```
void output() {
    int res = 0;
    for (int i = 1; i <= N; ++i) {
        res = max(res, forwardDist[i] + reverseDist[i]);
    }
    printf("%d\n", res);
}

int main() {
    input();
    Dijkstra(forwardDist, forwardAdj);
    Dijkstra(reverseDist, reverseAdj);
    output();
    return 0;
}
```

Code example 2

: O(|E| log(|V|))

Code example 2


```cpp
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

using pii = pair<int, int>;
const int LM = 1005;
const int INF = 1 << 25;
int N, E, X;
vector<pii> forwardAdj[LM], reverseAdj[LM];
int forwardDist[LM], reverseDist[LM];
int visited[LM], vcnt;

priority_queue<pii, vector<pii>, greater<pii>> pq;  // minheap
```

Code example 2


```
void Dijkstra(vector<pii> adj[], int dist[]) {
    // init : implement here
    for (int i = 1; i <= N; ++i) dist[i] = INF;
    dist[X] = 0;

    pq = {}, ++vcnt;
    pq.push({ 0, X });  // pair<int:dist, int: node>

    while (!pq.empty()) {
        // 1. extract minNode & minDist
        int minDist = pq.top().first, minNode = pq.top().second;
        pq.pop();
        if (visited[minNode] == vcnt) continue;

        visited[minNode] = vcnt; // 2. checkNode : implement here

        for (auto&pa : adj[minNode]) {  // 3. relaxation
            int tdist = minDist + pa.second;
            if (dist[pa.first] > tdist) {
                dist[pa.first] = tdist;
                pq.push({ tdist, pa.first });
            }
        }
    }
}
```

Code example 2


```
void output() {
    int ansDist = 0;
    for (int i = 1; i <= N; ++i) {
        if (i == X) continue;
        if (ansDist < forwardDist[i] + reverseDist[i]) {
            ansDist = forwardDist[i] + reverseDist[i];
        }
    }
    printf("%d\n", ansDist);
}

int main() {
    int i, s, e, w;
    scanf("%d %d %d", &N, &E, &X);
    for (i = 0; i < E; ++i) {
        scanf("%d %d %d", &s, &e, &w);
        forwardAdj[s].push_back({ e, w });
        reverseAdj[e].push_back({ s, w });
    }
    Dijkstra(forwardAdj, forwardDist);
    Dijkstra(reverseAdj, reverseDist);
    output();
    return 0;
}
```

Thank you