2022-2 HI-ARC 중급스터디

6주차. 최단 경로2

이지은 (leeju1013)

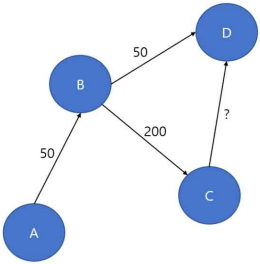
목차

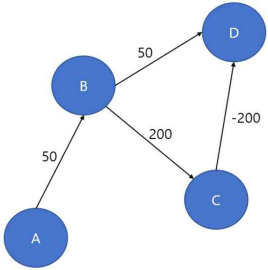
1. 벨만-포드
  – 11657. 타임머신

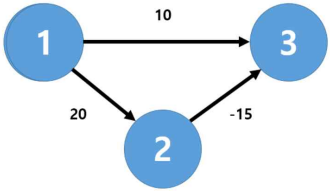
2. 플로이드-워셜
  – 11404. 플로이드

최단경로 알고리즘

- BFS
- 다익스트라
- **벨만-포드**
- 플로이드-와샬







- 가중치가 음수인 간선

'거리'가 음수..?
어쨌든 최단거리가 작으면 작을수록 좋다고 했을 때,
그게 음수가 되더라도 값을 정확히 구해주는 알고리즘

가중치가 거리가 아니라 이동시간(소요시간)이라고
생각해보면, 타임머신을 타고 과거로 돌아가는 느낌이랄까..

'최단거리' 정의 : 최소의 cost를 들여서 오는 경로의 cost 합

- 다익스트라

최소 비용을 가지는 간선만 우선적으로 뽑으면서
경우의 수를 줄여나가며 아직 방문하지 않은
정점의 거리를 갱신함.
1번 정점에서 3번 정점까지의 최단거리는 10

- 벨만-포드

모든 경우의 수를 전부 탐색함.
1번 정점에서 3번 정점까지의 최단거리는 20+(-15)=5

1. 벨만-포드(Bellman–Ford)

- What?

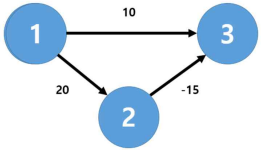
- 그래프의 한 정점에서 다른 정점들까지의 *최단거리*를 구하는 알고리즘

- 가중치가 **음수인 간선**이 존재할 때도 사용 가능!

- 시간복잡도 **O(VE)**

- **음의 사이클**(negative cycle)이 존재하면 제대로 동작하지 않음.

1. 벨만-포드(Bellman–Ford)



- How?

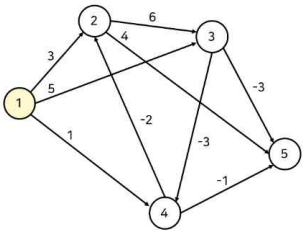
1. 모든 간선 E개를 하나씩 확인한다.

2. 각 간선을 거쳐 다른 정점으로 가는 비용을 계산하여
   최단 거리 테이블을 갱신한다.

3. 1과 2를 **v-1번** 반복한다.

   V개의 정점과 E개의 간선이 있는 가중 그래프에서
   정점 A에서 정점 B까지의 최단 거리는 최대 V-1개의 정점을 지나기 때문

[v-1번 반복 예정..]



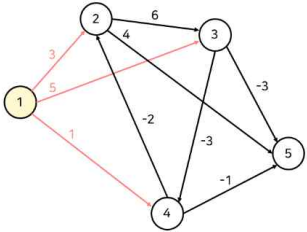
1	2	3	4	5
0	INF	INF	INF	INF

시작점인 1번 정점에서
나머지 정점들로의 최단거리 구하기!

초기값으로 시작점은 0,
나머지는 max num으로 설정.

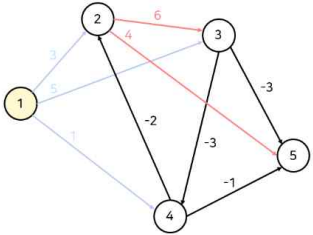
모든 간선을 하나씩 확인해보자.

[1번째 iteration]



1	2	3	4	5
0	3	5	1	INF

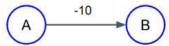
1번 노드에서 나가는 3개의 간선에
의해 dist[2], dist[3], dist[4]가 갱신됨.



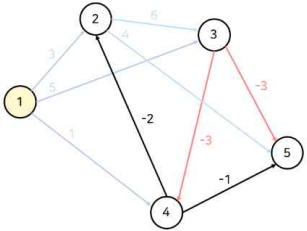
1	2	3	4	5
0	3	5	1	7

2번 노드에서 나가는 2개의 간선에
의해 dist[5]가 7로 갱신되었고,
dist[3]은 유지되었음.

* 이때, dist[cur]!=INF (cur정점이 이미 한번 방문해서
최단거리를 갱신했음)일 때만 갱신해줌.
즉, 도달할 수 있는 정점과 연결된 간선에 대해서만 탐색.

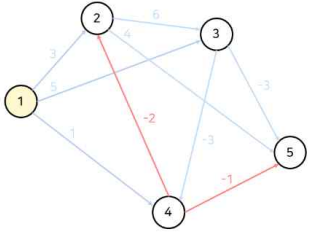


INF-10 으로
갱신되는 걸
막아야



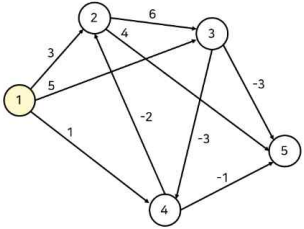
1	2	3	4	5
0	3	5	1	2

3번 노드에서 나가는 2개의 간선에 의해 dist[5]가 2으로 갱신되었고, dist[4]는 유지되었음.



1	2	3	4	5
0	-1	5	1	0

4번 노드에서 나가는 2개의 간선에 의해 dist[2]가 -1으로 갱신되었고, dist[5]가 0으로 갱신되었음.



1	2	3	4	5
0	-1	5	1	0

5번 노드에서 나가는 간선이 없으므로 첫번째 iteration 종료.

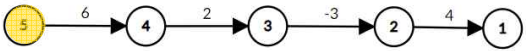
이러한 iteration 과정을 총 v-1번 반복함.

이 예제에서는 더 이상 갱신이 발생하지 않음.

- iteration 과정을 총 v-1번 반복하는 이유?

방금 예시처럼 단 한 번의 iteration 만에
모든 정점으로의 최단 거리를 구할 수도 있음.
그런 일은 최적의 순서로 갱신을 했을 때만 일어남.

다음과 같은 경우에는 4번(=v-1)의 iteration을 전부 돌아야
시작점인 5번 정점에서 나머지 모든 정점으로의 최단 거리를 구할 수 있음.



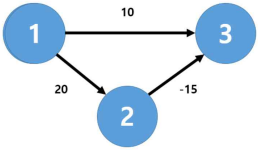
	1	2	3	4	5
	INF	INF	INF	INF	0
1번째 iteration	1	2	3	4	5
	INF	INF	INF	6	0
2번째 iteration	1	2	3	4	5
	INF	INF	8	6	0
3번째 iteration	1	2	3	4	5
	INF	5	8	6	0
4번째 iteration	1	2	3	4	5
	9	5	8	6	0

1. 벨만-포드(Bellman–Ford)

- How?

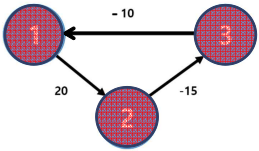
1. 모든 간선 E개를 하나씩 확인한다.

2. 각 간선을 거쳐 다른 정점으로 가는 비용을 계산하여 최단 거리 테이블을 갱신한다.



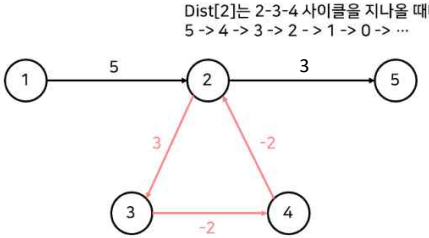
3. 1과 2를 **v-1번** 반복한다.
   만약 V개의 정점을 지났는데 최단 경로가 갱신이 된다면 음의 사이클이 발생한 것이며 비용이 무한하게 갱신이 되기 때문에 최단 경로를 구할 수 없음.

만약 음의 사이클 발생 여부를 체크하고 싶다면
1~2번 과정을 한 번 더 수행함.
이때 최단 거리 테이블이 갱신된다면 음의 사이클이 존재.



- 음의 사이클?

: 가중치 합이 음수인 사이클



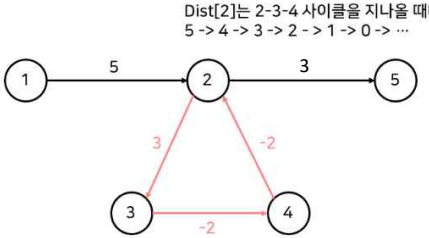
	1	2	3	4	5
	0	INF	INF	INF	INF
1번째 iteration	1	2	3	4	5
	0	4	8	6	8
2번째 iteration	1	2	3	4	5
	0	3	7	5	7
3번째 iteration	1	2	3	4	5
	0	2	6	4	6
4번째 iteration	1	2	3	4	5
	0	1	5	3	5
5번째 iteration	1	2	3	4	5
	0	0	4	2	4

2-3-4 사이클의 가중치 합(3-2-2 = -1)이 음수이므로 음의 사이클임.
해당 사이클을 무한히 반복하며 지난다면 2, 3, 4, 5번 정점까지의 최단거리가 음의 무한대까지 내려감.

간선을 V-1개를 초과하여 지날수록 더 짧은 거리로 갈 수 있게 됨.
즉, '정점 A에서 정점 B까지의 최단 거리는 최대 V-1개의 정점을 지난다는 가정'이 성립하지 않음.

- 음의 사이클?

: 가중치 합이 음수인 사이클



	1	2	3	4	5
	0	INF	INF	INF	INF
1번째 iteration	1	2	3	4	5
	0	4	8	6	8
2번째 iteration	1	2	3	4	5
	0	3	7	5	7
3번째 iteration	1	2	3	4	5
	0	2	6	4	6
4번째 iteration	1	2	3	4	5
	0	1	5	3	5
5번째 iteration	1	2	3	4	5
	0	0	4	2	4

=> 음의 사이클을 발견했다면 최단 경로가 존재하지 않는다고 결론지음.

v-1번까지의 iteration이후 더 많은 iteration을 돌렸을 때(최소 v번), 최단 거리 값이 갱신된다면
(v-1개의 간선보다 더 많은 간선을 통해 최단 경로를 구할 수 있다면)
음의 사이클이 존재한다고 판단함.

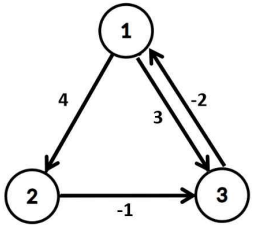
타임머신

성공

골드 IV

시간 제한

1 초



예제 입력 1 복사

```
3 4
1 2 4
1 3 3
2 3 -1
3 1 -2
```

예제 출력 1

```
4
3
```

예제 입력 2 복사

```
3 4
1 2 4
1 3 3
2 3 -4
3 1 -2
```

예제 출력 2

```
-1
```

문제

N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 버스가 M개 있다. 각 버스는 A, B, C로 나타낼 수 있는데, A는 시작도시, B는 도착도시, C는 버스를 타고 이동하는데 걸리는 시간이다. 시간 C가 양수가 아닌 경우가 있다. C = 0인 경우는 순간 이동을 하는 경우, C < 0인 경우는 타임머신으로 시간을 되돌아가는 경우이다.

1번 도시에서 출발해서 나머지 도시로 가는 가장 빠른 시간을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 N (1 ≤ N ≤ 500), 버스 노선의 개수 M (1 ≤ M ≤ 6,000)이 주어진다. 둘째 줄부터 M개의 줄에는 버스 노선의 정보 A, B, C (1 ≤ A, B ≤ N, -10,000 ≤ C ≤ 10,000)가 주어진다.

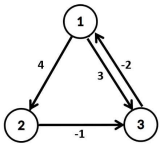
출력

만약 1번 도시에서 출발해 어떤 도시로 가는 과정에서 시간을 무한히 오래 전으로 되돌릴 수 있다면 첫째 줄에 -1을 출력한다. 그렇지 않다면 N-1개 줄에 걸쳐 각 줄에 1번 도시에서 출발해 2번 도시, 3번 도시, ..., N번 도시로 가는 가장 빠른 시간을 순서대로 출력한다. 만약 해당 도시로 가는 경로가 없다면 대신 -1을 출력한다.

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <climits>

using namespace std;
const int MAX_V=501;
int n,m,a,b,c,negative_cycle=0;
vector<pair<int,int>> adj[MAX_V];
long long dist[MAX_V];

void bellman_ford(int st){
    fill(dist, dist+MAX_V, INT_MAX);
    dist[st]=0;
    for(int iter=1; iter<=n; iter++){
        // n번의 루프 : 모든 간선 확인작업 n-1번 + 음의사이클 유무 확인 1번
        for(int cur=1; cur<=n; cur++){
            if(dist[cur]==INT_MAX) continue; //아직 이어지지 않은 정점은 패스
            for(auto &u:adj[cur]){
                int next=u.first, cost=u.second;
                if(dist[next]>dist[cur]+cost){ //갱신
                    dist[next]=dist[cur]+cost;
                    if(iter==n) // n번째 루프에 갱신되면
                        negative_cycle=1; // 음의 사이클이 존재한다는 뜻
                }
            }
        }
    }
}
```

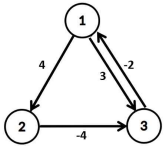
O(V) O(E)

시간복잡도 O(VE)



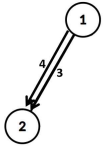
예제 출력 1

```
4
3
```



예제 출력 2

```
-1
```



예제 출력 3

```
3
-1
```

```cpp
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    cin>>n>>m;
    for(int i=0; i<m; i++){
        cin>>a>>b>>c;
        adj[a].push_back({b,c});
    }

    bellman_ford(1);

    if(negative_cycle) cout<<-1;
    else{
        for(int i=2; i<=n; i++){
            if(dist[i]==INT_MAX) cout<<-1<<'\n';
            else cout<<dist[i]<<'\n';
        }
    }
    return 0;
}
```

쉬는 시간

1. 벨만-포드
   – 11657. 타임머신

2. 플로이드-워셜
   – 11404. 플로이드

홍익대학교 프로그래밍 대회 개최

✓일시: 11월 18일 금요일 3시-5시(2시간)

✓참여대상: 홍익대학교 재학생,휴학생(대학원생 포함)

✓장소: 홍익대학교 C동 816호

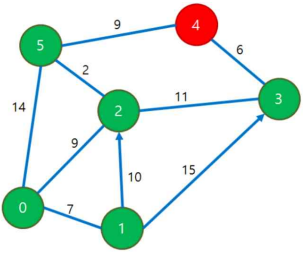
✓언어제한: c, c++, python, java (java로 풀었을때는 정답 보장은 안됨)

✓신청기간: 11/4(금)~11/13(일)

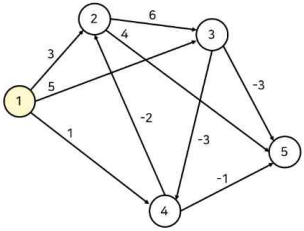
✓신청 링크 : https://forms.gle/9WUV7yL1vUCZPNxd7

최단경로 알고리즘

- BFS
- 다익스트라
- 벨만-포드
- **플로이드-와샬**



0	1	2	3	4	5
0	7	9	20	20	11



1	2	3	4	5
0	-1	5	1	0

2. 플로이드-워셜(Floyd-Warshall)

- What?

- 그래프의 **모든** 정점에서 다른 정점들까지의 *최단거리*를 구하는 알고리즘
  -> 2차원 배열에 dist 저장

- 가중치가 **음수인 간선**이 존재할 때도 사용 가능!

- 시간복잡도 **O(V^3)**

- **음의 사이클**(negative cycle)이 존재하면 제대로 동작하지 않음.

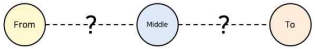
2. 플로이드-워셜(Floyd-Warshall)

- How?
  : DP 방식으로 각 정점쌍의 최단거리 갱신

1. DP 테이블 정의
   dist[i][j] = 정점 i로부터 정점 j 까지의 최단 거리

2. 점화식 찾기
   dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]) (즉, i : 출발지, k : 경유지, j : 도착지)

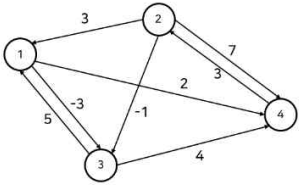
3. 초기값 설정
   dist[i][i] = 0 (출발지와 도착지가 같은 경우 dist값은 0), 나머지는 INF



노란색은 i(출발지),
빨간색은 j(도착지),
파란색은 k(경유지)을 의미함.

각 노드들을 경유지로 두는
모든 경우를 탐색함.
즉, 각각의 경유지 노드 k에 대해,
모든 (i,j)쌍의 dist를 비교함.

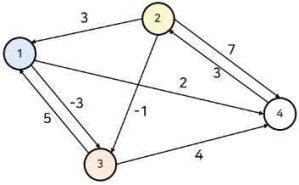
dist[i][j]에는 현재까지 찾은
i~j의 최단거리가 저장되어 있음.



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

예를들어 dist[1][4]=3이라고 할 때,
1->2->4를 거친 가중치 총합이
3이어서 이후 다른 탐색에서 사용
되는 dist[1][4]는 1->2->4 경로를
내포하는 것임.
따라서 dist[1][5]를 갱신할 때
k로 4를 경유하였다면,
dist[1][5]는 1->2->4->5 경로를
내포하는 것임.

From = 2
Middle = 1
To = 3



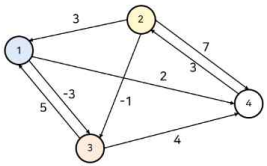
새로운 dist값 : dist[2][1] + dist[1][3] = 0

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

From = 2
Middle = 1
To = 3

원래 2 -> 3 경로의 가중치인 -1과,
중간 노드인 1번 노드를 거쳐가는
2 -> 1 -> 3 경로의 가중치 0을 비교함.

중간 노드를 거치는 경로의 가중치가
더 크므로, dist[2][3]은 갱신되지 않음.

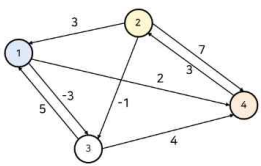


새로운 dist값 : dist[2][1] + dist[1][3] = 0

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

From = 2
Middle = 1
To = 4

원래 2 -> 4 경로의 가중치인 dist[2][4]와
중간 노드인 1번 노드를 거쳐가는
2 -> 1 -> 4 경로의 가중치인
dist[2][1] + dist[1][4]를 비교함.
중간노드를 거치는 경로의 가중치가
더 작기 때문에, dist[2][4]를 5로 갱신함.

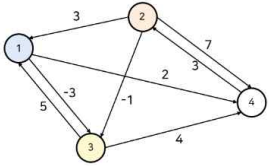


새로운 dist값 : dist[2][1] + dist[1][4] = 5

	1	2	3	4
1	0	INF	-3	2
2	3	0	-1	5
3	5	INF	0	4
4	INF	3	INF	0

From = 3
Middle = 1
To = 2

중간 노드인 1에서 2번 노드로 연결된
간선이 현재까지 없으므로, (dist[1][2] = INF)
dist 값을 갱신하지 않음.

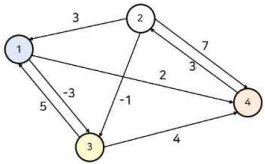


새로운 dist값 : dist[3][1] + dist[1][2] = INF

	1	2	3	4
1	0	INF	-3	2
2	3	0	-1	5
3	5	INF	0	4
4	INF	3	INF	0

From = 3
Middle = 1
To = 4

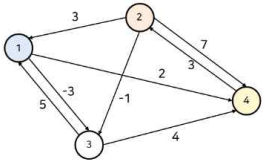
dist[3][1] + dist[1][4]값이
기존 dist[3][4]보다 크므로,
갱신하지 않음.



새로운 dist값 : dist[3][1] + dist[1][4] = 7

	1	2	3	4
1	0	INF	-3	2
2	3	0	-1	5
3	5	INF	0	4
4	INF	3	INF	0

From = 4
Middle = 1
To = 2

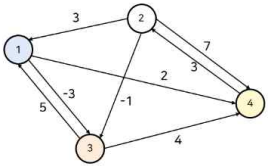


새로운 dist값 : dist[4][1] + dist[1][2] = INF

	1	2	3	4
1	0	INF	-3	2
2	3	0	-1	5
3	5	INF	0	4
4	INF	3	INF	0

From = 4
Middle = 1
To = 3

이렇게 한 노드에 대한
중간 노드 루프가 끝났다면,
다음 노드를 중간 노드로 지정해
지금까지의 과정을 반복함.(다음 페이지)



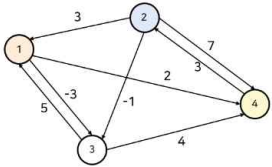
새로운 dist값 : dist[4][1] + dist[1][3] = INF

	1	2	3	4
1	0	INF	-3	2
2	3	0	-1	5
3	5	INF	0	4
4	INF	3	INF	0

From = 4
Middle = 2
To = 1

2번 노드를 Middle로 정했을 때
dist 배열의 값이 변경되는 한 경우 예시임.

이처럼 모든 노드를 경유지로 정해서
최단거리를 갱신함.



새로운 dist값 : dist[4][2] + dist[2][1] = 6

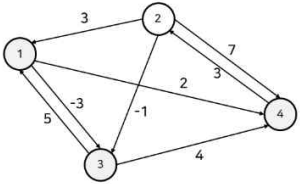
	1	2	3	4
1	0	INF	-3	2
2	3	0	-1	5
3	5	INF	0	4
4	6	3	INF	0

. . .

최종

아래는 플로이드-와샬 알고리즘이
완료된 후의 dist배열임.

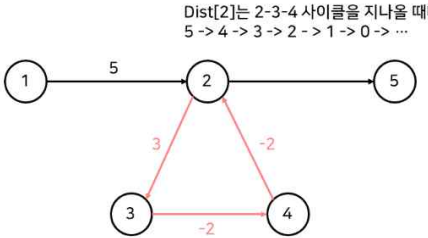
결국 플로이드 알고리즘은,
가능한 모든 경로를 체크하여
모든 최단거리를 구하는 알고리즘임.



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

- 음의 사이클?

: 가중치 합이 음수인 사이클



벨만-포드 알고리즘과 마찬가지로 플로이드-와샬 알고리즘도 음의 사이클이 존재할 때 최단 거리를 구할 수 없음.

벨만-포드에서는 원래의 v-1번 iteration에서 추가적인 iteration 진행을 통해 음의 사이클 존재 여부를 확인했음.
플로이드-와샬에서는 어떻게 찾아낼까???

처음에 dist[i][i] = 0으로 초기화했었음.
자기 자신에서 출발해 자기 자신에게 돌아오는 경로의 가중치는 절대 0보다 작아질 수 없기 때문.

하지만 음의 사이클이 존재한다면, 그 사이클에 포함된 노드는 dist[i][i]가 음수 값을 가짐.

따라서 플로이드-와샬 알고리즘 적용 후,
모든 노드의 번호 i에 대해 dist[i][i]가 0인지 확인하여 음의 사이클 존재 여부를 판단 가능.

플로이드

성공

골드 IV

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	46410	19167	13561	41.808%

문제

n(2 ≤ n ≤ 100)개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 m(1 ≤ m ≤ 100,000)개의 버스가 있다. 각 버스는 한 번 사용할 때 필요한 비용이 있다.

모든 도시의 쌍 (A, B)에 대해서 도시 A에서 B로 가는데 필요한 비용의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 n이 주어지고 둘째 줄에는 버스의 개수 m이 주어진다. 그리고 셋째 줄부터 m+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 버스의 정보는 버스의 시작 도시 a, 도착 도시 b, 한 번 타는데 필요한 비용 c로 이루어져 있다. 시작 도시와 도착 도시가 같은 경우는 없다. 비용은 100,000보다 작거나 같은 자연수이다.

시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다.

출력

n개의 줄을 출력해야 한다. i번째 줄에 출력하는 j번째 숫자는 도시 i에서 j로 가는데 필요한 최소 비용이다. 만약, i에서 j로 갈 수 없는 경우에는 그 자리에 0을 출력한다.

예제 입력 1

```
5
14
1 2 2
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
3 5 10
3 1 8
1 4 2
5 1 7
3 4 2
5 2 4
```

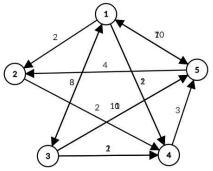
예제 출력 1 복사

```
0 2 3 1 4
12 0 15 2 5
8 5 0 1 1
10 7 13 0 3
7 4 10 6 0
```

```cpp
#include <iostream>
#include <climits>

using namespace std;
const int MAX_V=101;
int dist[MAX_V][MAX_V];

int main(){
    ios_base::sync_with_stdio(false); cin.

    int n,m,x,y,z;
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            dist[i][j]=INT_MAX;
        }
    }

    for(int i=0; i<m; i++){
        cin>>x>>y>>z;
        if(dist[x][y]>z) //시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다.
            dist[x][y]=z;
    }
```

O(V^3)

```cpp
    for(int k=1; k<=n; k++){ //경유지
        for(int i=1; i<=n; i++){ //출발지
            for(int j=1; j<=n; j++){ //도착지
                if(dist[i][k]!=INT_MAX && dist[k][j]!=INT_MAX) //오버플로우 방지
                    dist[i][j]=min(dist[i][j], dist[i][k]+dist[k][j]);
            }
        }
    }

    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            //i에서 j로 갈 수 없는 경우에는 그 자리에 0을 출력한다.
            if(i==j || dist[i][j]==INT_MAX) cout<<0<<" ";
            else cout<<dist[i][j]<<" ";
        }
        cout<<'\n';
    }
    return 0;
}
```



예제 출력 1 복사

```
0 2 3 1 4
12 0 15 2 5
8 5 0 1 1
10 7 13 0 3
7 4 10 6 0
```

감사합니다

- 필수 문제
  - 11657. 타임머신
  - 11404. 플로이드
- 연습 문제
  - 1865. 웜홀
  - 1738. 골목길
  - 3860. 할로윈 묘지
  - 13317. 한 번 남았다
  - 11403. 경로 찾기
  - 1956. 운동
  - 14938. 서강그라운드
  - 11562. 백양로 브레이크
- 11월 16일(수요일) 저녁 6시 T702