

2022 AlgoLive 8th Study

Graph III : Floyd-Warshall

Floyd-Warshall Algorithm

- 모든 최단 경로를 구하는 알고리즘

Dijkstra

하나의 정점에서 다른 모든
정점까지의 최단 경로
(S.S.S.P - Single Source Shortest
Path)

Floyd-Warshall

한 번에 모든 최단 경로
* 훨씬 간단
* 음의 간선도 가능

Floyd-Warshall Algorithm

- 모든 최단 경로를 구하는 알고리즘
- ‘거쳐 가는 정점’을 기준으로 알고리즘 수행
 - > 매 단계마다 거치지 않은 최단 노드를 찾을 필요 X
- 2차원 테이블에 최단 거리 정보 저장
- 점화식(DP 알고리즘)

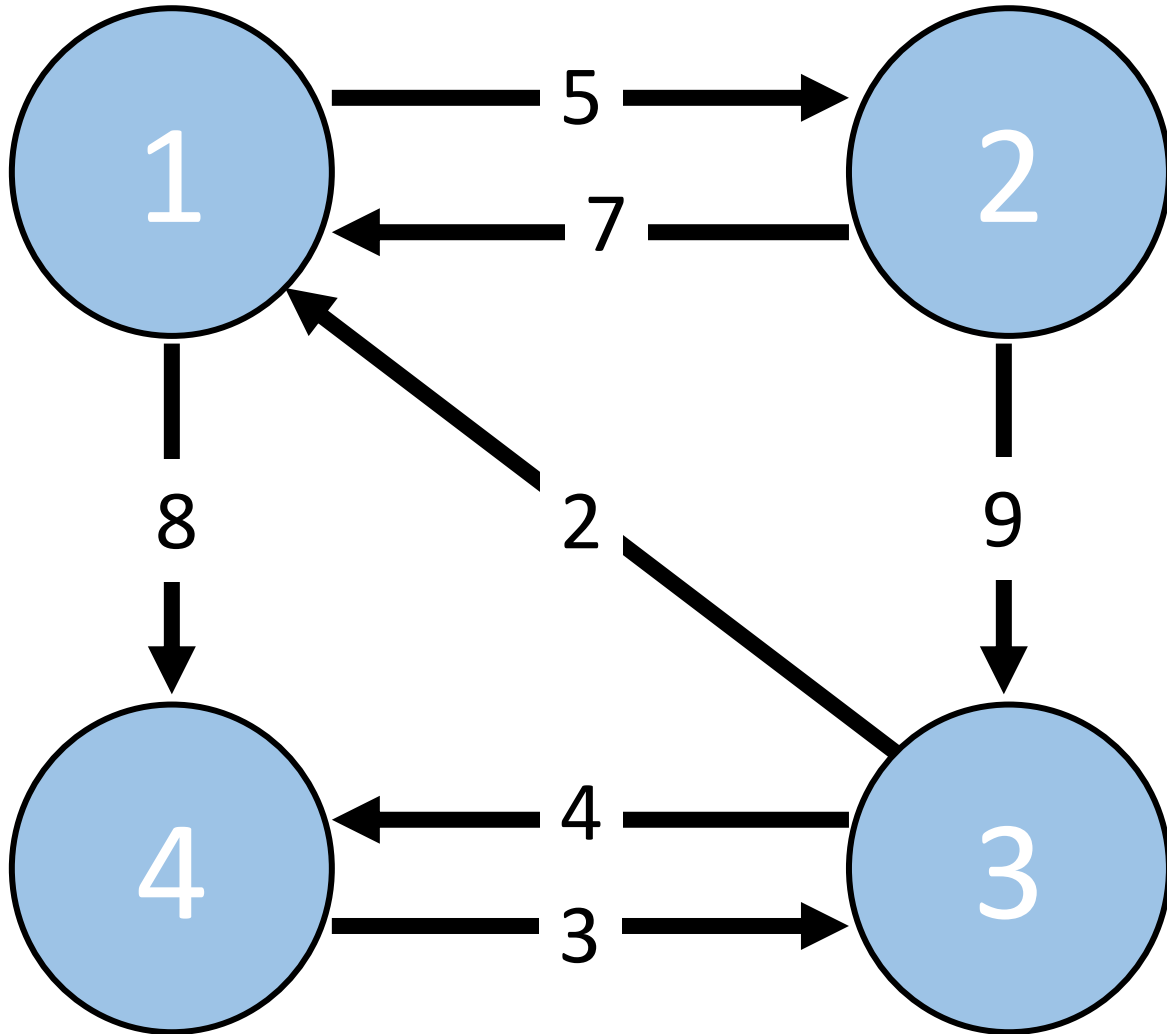
$$D_{ab} = \min (D_{ab}, D_{ak} + D_{kb})$$

- $O(N^3)$ 시간복잡도
- $O(N^2)$ 공간복잡도

Floyd-Warshall Algorithm

- 알고리즘 수행 과정
 1. 그래프를 2차원 배열로 표현
 2. 노드 Z를 거쳐가는 경우를 선택(Z는 모든 노드)
 3. $(X \rightarrow Y) > (X \rightarrow Z + Z \rightarrow Y)$ 일 때만 갱신
 4. 모든 노드에 대해 수행

예제 1



| | | | |
|-----|-----|-----|-----|
| 0 | 5 | INF | 8 |
| 7 | 0 | 9 | INF |
| 2 | INF | 0 | 4 |
| INF | INF | 3 | 0 |

노드 1을 거치는 경우

| | | | |
|-----|-----|-----|-----|
| 0 | 5 | INF | 8 |
| 7 | 0 | 9 | INF |
| 2 | INF | 0 | 4 |
| INF | INF | 3 | 0 |

| | | | |
|-----|---|-----|---|
| 0 | 5 | INF | 8 |
| 7 | 0 | | |
| 2 | | 0 | |
| INF | | | 0 |

노드 1을 거치는 경우

| | | | |
|-----|-----|-----|-----|
| 0 | 5 | INF | 8 |
| 7 | 0 | 9 | INF |
| 2 | INF | 0 | 4 |
| INF | INF | 3 | 0 |

| | | | |
|-----|------|------|------|
| 0 | 5 | INF | 8 |
| 7 | 0 | 2->3 | 2->4 |
| 2 | 3->2 | 0 | 3->4 |
| INF | 4->2 | 4->3 | 0 |

노드 1을 거치는 경우

| | | | |
|-----|-----|-----|-----|
| 0 | 5 | INF | 8 |
| 7 | 0 | 9 | INF |
| 2 | INF | 0 | 4 |
| INF | INF | 3 | 0 |

| | | | |
|-----|------------|----------|------------|
| 0 | 5 | INF | 8 |
| 7 | 0 | 2->3 : 9 | 2->4 : INF |
| 2 | 3->2 : INF | 0 | 3->4 : 4 |
| INF | 4->2 : INF | 4->3 : 3 | 0 |

노드 1을 거치는 경우

| 0 | 5 | INF | 8 |
|-----|--|--|---|
| 7 | 0 | 2->3 : 9 2->1 : 7, 1->3 : INF => 9 | 2->4 : INF 2->1 : 7, 1->4 : 8 => 15 |
| 2 | 3->2 : INF 3->1 : 2, 1->2 : 5 => 7 | 0 | 3->4 : 4 3->1 : 2, 1->4 : 8 => 4 |
| INF | 4->2 : INF 4->1 : INF, 1->2 : 5 => INF | 4->3 : 3 4->1 : INF, 1->3 : INF => 3 | 0 |

노드 1을 거치는 경우

| | | | |
|-----|-----|-----|----|
| 0 | 5 | INF | 8 |
| 7 | 0 | 9 | 15 |
| 2 | 7 | 0 | 4 |
| INF | INF | 3 | 0 |

노드 2를 거치는 경우

| | | | |
|-----|-----|-----|----|
| 0 | 5 | INF | 8 |
| 7 | 0 | 9 | 15 |
| 2 | 7 | 0 | 4 |
| INF | INF | 3 | 0 |

| | | | |
|---|-----|---|----|
| 0 | 5 | | |
| 7 | 0 | 9 | 15 |
| | 7 | 0 | |
| | INF | | 0 |

노드 2를 거치는 경우

| | | | |
|------------|-----|------------|----------|
| 0 | 5 | 1->3 : INF | 1->4 : 8 |
| 7 | 0 | 9 | 15 |
| 3->1 : 2 | 7 | 0 | 3->4 : 4 |
| 4->1 : INF | INF | 4->3 : 3 | 0 |

노드 2를 거치는 경우

| | | | |
|-----|-----|----|----|
| 0 | 5 | 14 | 8 |
| 7 | 0 | 9 | 15 |
| 2 | 7 | 0 | 4 |
| INF | INF | 3 | 0 |

노드 3을 거치는 경우

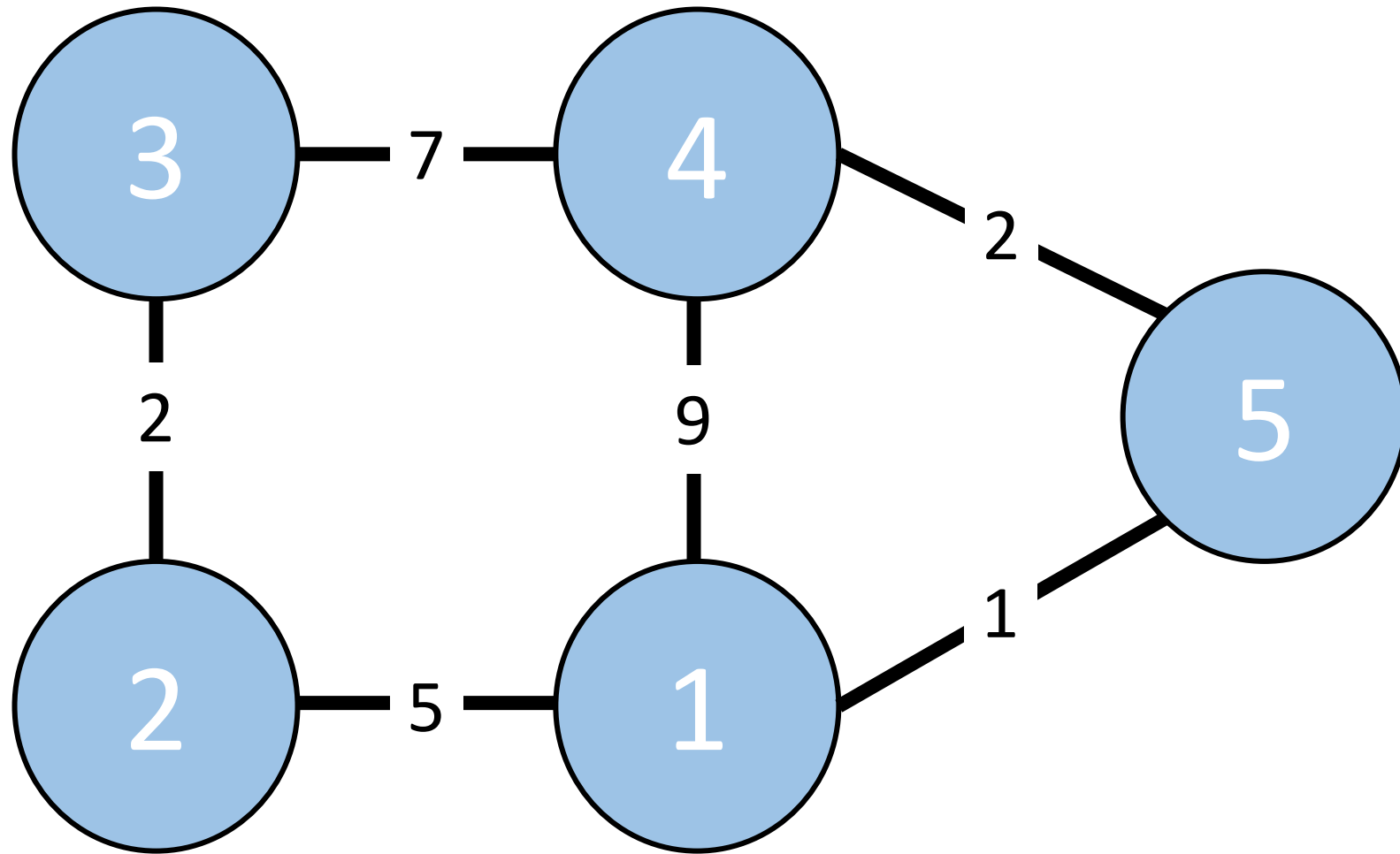
| | | | |
|-----|-----|----|----|
| 0 | 5 | 14 | 8 |
| 7 | 0 | 9 | 15 |
| 2 | 7 | 0 | 4 |
| INF | INF | 3 | 0 |

| | | | |
|---|---|----|---|
| 0 | | 14 | |
| | 0 | 9 | |
| 2 | 7 | 0 | 4 |
| | | 3 | 0 |

수행 결과

| | | | |
|---|----|----|----|
| 0 | 5 | 11 | 8 |
| 7 | 0 | 9 | 13 |
| 2 | 7 | 0 | 4 |
| 5 | 10 | 3 | 0 |

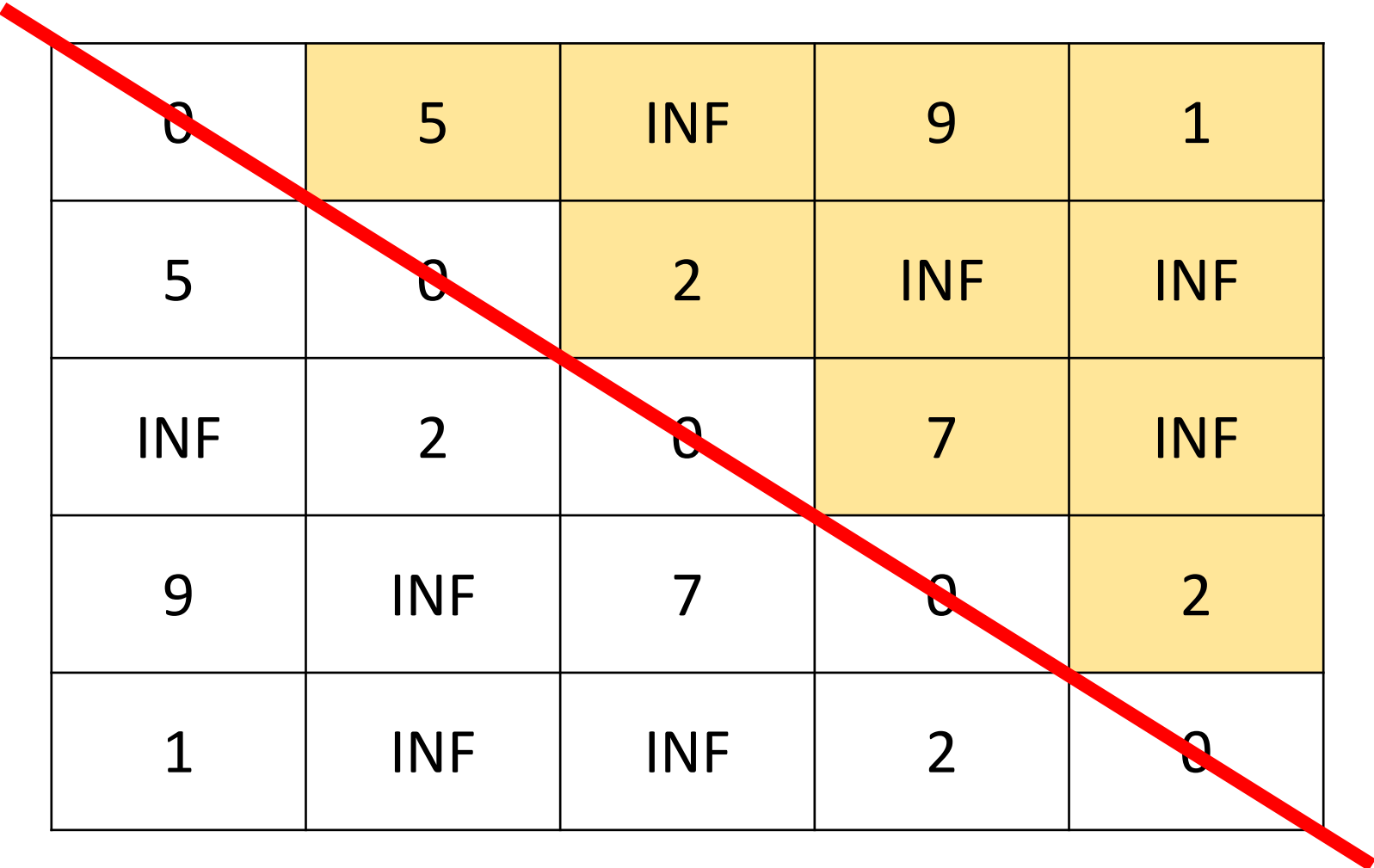
예제 2



예제 2

| | | | | |
|-----|-----|-----|-----|-----|
| 0 | 5 | INF | 9 | 1 |
| 5 | 0 | 2 | INF | INF |
| INF | 2 | 0 | 7 | INF |
| 9 | INF | 7 | 0 | 2 |
| 1 | INF | INF | 2 | 0 |

예제 2



| | | | | |
|-----|-----|-----|-----|-----|
| 0 | 5 | INF | 9 | 1 |
| 5 | 0 | 2 | INF | INF |
| INF | 2 | 0 | 7 | INF |
| 9 | INF | 7 | 0 | 2 |
| 1 | INF | INF | 2 | 0 |

플로이드 워셜 소스코드

```
void floyd(int n) {  
    // 플로이드 워셜 알고리즘  
    for(int k=1; k<=n; k++){// k = 거쳐가는 노드  
        for(int i=1; i<=n; i++){// i = 출발 노드  
            for(int j=1; j<=n; j++){// j = 도착 노드  
                if(d[i][k] + d[k][j] < d[i][j]){  
                    d[i][j] = d[i][k] + d[k][j];  
                }  
            }  
        }  
    }  
}
```

예제 1 소스코드

```
1  #include<iostream>
2  #define INF 1e9
3
4  using namespace std;
5
6  int N = 4;
7  int d[4][4]; // 결과 그래프
8  int a[4][4] = {
9      {0, 5, INF, 8},
10     {7, 0, 9, INF},
11     {2, INF, 0, 4},
12     {INF, INF, 3, 0}
13 };
14
15 void floyd(int n) {
16     // 플로이드 워셜 알고리즘
17     for(int k=1; k<=n; k++){// k = 거쳐가는 노드
18         for(int i=1; i<=n; i++){// i = 출발 노드
19             for(int j=1; j<=n; j++){// j = 도착 노드
20                 if(d[i][k] + d[k][j] < d[i][j]){
21                     d[i][j] = d[i][k] + d[k][j];
22                 }
23             }
24         }
25     }
26 }
```

```
28 int main(){
29     for(int i=0; i<N; i++){
30         for(int j=0; j<N; j++){
31             d[i][j] = a[i][j];
32         }
33     }
34
35     floyd(N);
36
37     // 결과 출력
38     for(int i=0; i<N; i++){
39         for(int j=0; j<N; j++){
40             cout << d[i][j] << " ";
41         }
42         cout << "\n";
43     }
44 }
```

Dijkstra VS Floyd-Warshall

Dijkstra

- 단계마다 최단 노드 하나씩 선택, 해당 노드 거치는 경로 **확인**하며 갱신
- 1차원 리스트(한 노드에서 다른 노드까지)
- Greedy Algorithm
- $O(N^2)$ 시간복잡도
- 긴 소스코드



Floyd-Warshall

- 단계마다 '거쳐 가는 노드' 기준, **매번** 최단 노드를 찾을 **필요 X**
- 2차원 배열
- DP Algorithm
- $O(N^3)$ 시간복잡도
- 짧은 소스코드

오늘의 문제

- #11403, '경로 찾기'
- #11404, '플로이드'
- 추가문제 #1389, '케빈 베이컨의 6단계 법칙'

경로 찾기

성공



1 실버 I

시간 제한

메모리 제한

제출

정답

맞힌 사람

정답 비율

1 초

256 MB

34432

20031

14555

57.797%

문제

가중치 없는 방향 그래프 G 가 주어졌을 때, 모든 정점 (i, j) 에 대해서, i 에서 j 로 가는 경로가 있는지 없는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정점의 개수 N ($1 \leq N \leq 100$)이 주어진다. 둘째 줄부터 N 개 줄에는 그래프의 인접 행렬이 주어진다. i 번째 줄의 j 번째 숫자가 1인 경우에는 i 에서 j 로 가는 간선이 존재한다는 뜻이고, 0인 경우는 없다는 뜻이다. i 번째 줄의 i 번째 숫자는 항상 0이다.

출력

총 N 개의 줄에 걸쳐서 문제의 정답을 인접행렬 형식으로 출력한다. 정점 i 에서 j 로 가는 경로가 있으면 i 번째 줄의 j 번째 숫자를 1로, 없으면 0으로 출력해야 한다.

플로이드

성공



4 골드 IV

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|-------|--------|-------|-------|-------|---------|
| 1 초 | 256 MB | 41864 | 17109 | 12165 | 41.801% |

문제

$n(2 \leq n \leq 100)$ 개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 $m(1 \leq m \leq 100,000)$ 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 n 이 주어지고 둘째 줄에는 버스의 개수 m 이 주어진다. 그리고 셋째 줄부터 $m+2$ 줄까지 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 c 로 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

n 개의 줄을 출력해야 한다. i 번째 줄에 출력하는 j 번째 숫자는 도시 i 에서 j 로 가는데 필요한 최소 비용이다. 만약, i 에서 j 로 갈 수 없는 경우에는 그 자리에 0을 출력한다.

케빈 베이컨의 6단계 법칙

성공



1 실버 I

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|-------|--------|-------|-------|-------|---------|
| 2 초 | 128 MB | 23568 | 13098 | 10246 | 55.946% |

문제

케빈 베이컨의 6단계 법칙에 의하면 지구에 있는 모든 사람들은 최대 6단계 이내에서 서로 아는 사람으로 연결될 수 있다. 케빈 베이컨 게임은 임의의 두 사람이 최소 몇 단계 만에 이어질 수 있는지 계산하는 게임이다.

예를 들면, 전혀 상관없을 것 같은 인하대학교의 이강호와 서강대학교의 민세희는 몇 단계만에 이어질 수 있을까?

천민호는 이강호와 같은 학교에 다니는 사이이다. 천민호와 최백준은 Baekjoon Online Judge를 통해 알게 되었다. 최백준과 김선영은 같이 Startlink를 창업했다. 김선영과 김도현은 같은 학교 동아리 소속이다. 김도현과 민세희는 같은 학교에 다니는 사이로 서로 알고 있다. 즉, 이강호-천민호-최백준-김선영-김도현-민세희 와 같이 5단계만 거치면 된다.

케빈 베이컨은 미국 할리우드 영화배우들 끼리 케빈 베이컨 게임을 했을때 나오는 단계의 총 합이 가장 적은 사람이라고 한다.

오늘은 Baekjoon Online Judge의 유저 중에서 케빈 베이컨의 수가 가장 작은 사람을 찾으려고 한다. 케빈 베이컨 수는 모든 사람과 케빈 베이컨 게임을 했을 때, 나오는 단계의 합이다.

예를 들어, BOJ의 유저가 5명이고, 1과 3, 1과 4, 2와 3, 3과 4, 4와 5가 친구인 경우를 생각해보자.

1은 2까지 3을 통해 2단계 만에, 3까지 1단계, 4까지 1단계, 5까지 4를 통해서 2단계 만에 알 수 있다. 따라서, 케빈 베이컨의 수는 $2+1+1+2 = 6$ 이다.

2는 1까지 3을 통해서 2단계 만에, 3까지 1단계 만에, 4까지 3을 통해서 2단계 만에, 5까지 3과 4를 통해서 3단계 만에 알 수 있다. 따라서, 케빈 베이컨의 수는 $2+1+2+3 = 8$ 이다.

3은 1까지 1단계, 2까지 1단계, 4까지 1단계, 5까지 4를 통해 2단계 만에 알 수 있다. 따라서, 케빈 베이컨의 수는 $1+1+1+2 = 5$ 이다.

4는 1까지 1단계, 2까지 3을 통해 2단계, 3까지 1단계, 5까지 1단계 만에 알 수 있다. 4의 케빈 베이컨의 수는 $1+2+1+1 = 5$ 가 된다.

마지막으로 5는 1까지 4를 통해 2단계, 2까지 4와 3을 통해 3단계, 3까지 4를 통해 2단계, 4까지 1단계 만에 알 수 있다. 5의 케빈 베이컨의 수는 $2+3+2+1 = 8$ 이다.

5명의 유저 중에서 케빈 베이컨의 수가 가장 작은 사람은 3과 4이다.

BOJ 유저의 수와 친구 관계가 입력으로 주어졌을 때, 케빈 베이컨의 수가 가장 작은 사람을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 유저의 수 N ($2 \leq N \leq 100$)과 친구 관계의 수 M ($1 \leq M \leq 5,000$)이 주어진다. 둘째 줄부터 M 개의 줄에는 친구 관계가 주어진다. 친구 관계는 A와 B로 이루어져 있으며, A와 B가 친구라는 뜻이다. A와 B가 친구이면, B와 A도 친구이며, A와 B가 같은 경우는 없다. 친구 관계는 중복되어 들어올 수도 있으며, 친구가 한 명도 없는 사람은 없다. 또, 모든 사람은 친구 관계로 연결되어져 있다. 사람의 번호는 1부터 N 까지이며, 두 사람이 같은 번호를 갖는 경우는 없다.

출력

첫째 줄에 BOJ의 유저 중에서 케빈 베이컨의 수가 가장 작은 사람을 출력한다. 그런 사람이 여러 명일 경우에는 번호가 가장 작은 사람을 출력한다.

#11403 C++

```
1  #include<iostream>
2  #define INF 1e9
3
4  using namespace std;
5
6  int N;
7  int d[101][101];
8
9  int main(){
10     cin >> N;
11     for(int i=1; i<=N; i++){
12         for(int j=1; j<=N; j++){
13             cin >> d[i][j];
14             if(d[i][j] == 0)
15                 d[i][j] = INF;
16         }
17     }
```

```
18     // 플로이드 워셜 알고리즘
19     for(int k=1; k<=N; k++){
20         for(int i=1; i<=N; i++){
21             for(int j=1; j<=N; j++){
22                 if(d[i][k] + d[k][j] < d[i][j]){
23                     d[i][j] = d[i][k] + d[k][j];
24                 }
25             }
26         }
27     }
28
29     for(int i=1; i<=N; i++){
30         for(int j=1; j<=N; j++){
31             if(d[i][j] == INF)
32                 cout << 0 << " ";
33             else
34                 cout << 1 << " ";
35         }
36         cout << "\n";
37     }
38     return 0;
39 }
```

#11403 Python, Java

```
#입력
N = int(input())
graph = []
for _ in range(N):
    graph.append(list(map(int, input().split())))

#플로이드-워셜 알고리즘
for k in range(N): #경로 for문이 가장 상위 단계여야 누락되지 않는다
    for i in range(N):
        for j in range(N):
            if graph[i][j] == 1 or (graph[i][k] == 1 and graph[k][j] == 1):
                graph[i][j] = 1

#출력
for row in graph:
    for col in row:
        print(col, end = " ")
    print()
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.StringTokenizer;

public class Main {

    public static void main(String[] args) throws NumberFormatException, IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
        StringTokenizer st;

        int N = Integer.parseInt(br.readLine());
        int[][] arr = new int[N][N];

        for (int i = 0; i < N; i++) {
            st = new StringTokenizer(br.readLine());
            for (int j = 0; j < N; j++) {
                arr[i][j] = Integer.parseInt(st.nextToken());
            }
        }

        // i에서 j까지 갈 수 있는가?
        // i에서 k로 가고, k에서 j로 갈 수 있는가?
        // 위에 2개의 질문은 동일함.
        for (int k = 0; k < N; k++) {
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                    // 단순히 갈 수 있는 경로가 있는지만 체크함.
                    if (arr[i][k] == 1 && arr[k][j] == 1) {
                        arr[i][j] = 1;
                    }
                }
            }
        }

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                sb.append(arr[i][j] + " ");
            }
            sb.append("\n");
        }

        bw.write(sb.toString());
        bw.flush();
        bw.close();
        br.close();
    }
}
```

#11404 C++

```
1  #include<iostream>
2  #define INF 1e9
3
4  using namespace std;
5
6  int N, M;
7  int d[101][101];
8
9  int main(){
10     int a, b, c;
11     cin >> N >> M;
12     for(int i=1; i<=N; i++){
13         for(int j=1; j<=N; j++){
14             d[i][j] = INF;
15         }
16     }
17     for(int i=0; i<M; i++){
18         cin >> a >> b >> c;
19         if(d[a][b] > c)
20             d[a][b] = c;
21     }
22
```

```
22
23     // 플로이드 워셜 알고리즘
24     for(int k=1; k<=N; k++){
25         for(int i=1; i<=N; i++){
26             for(int j=1; j<=N; j++){
27                 if(d[i][k] + d[k][j] < d[i][j]){
28                     d[i][j] = d[i][k] + d[k][j];
29                 }
30             }
31         }
32     }
33
34     for(int i=1; i<=N; i++){
35         for(int j=1; j<=N; j++){
36             if(i == j || d[i][j] == INF)
37                 cout << 0 << " ";
38             else
39                 cout << d[i][j] << " ";
40         }
41         cout << "\n";
42     }
43     return 0;
44 }
```

#1 1404 Python

```
n = int(input())
m = int(input())
inf = 100000000
s = [[inf] * n for i in range(n)]
for i in range(m):
    a, b, c = map(int, input().split())
    if s[a - 1][b - 1] > c:
        s[a - 1][b - 1] = c
for k in range(n):
    for i in range(n):
        for j in range(n):
            if i != j and s[i][j] > s[i][k] + s[k][j]:
                s[i][j] = s[i][k] + s[k][j]
for i in s:
    for j in i:
        if j == inf:
            print(0, end=' ')
        else:
            print(j, end=' ')
    print()
```

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.StringTokenizer;

public class Main {
    static final int INF = 987654321;

    public static void main(String[] args) throws NumberFormatException, IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
        StringTokenizer st;

        int N = Integer.parseInt(br.readLine());
        int M = Integer.parseInt(br.readLine());
        int[][] arr = new int[N + 1][N + 1];

        // 초기값 설정
        for (int i = 1; i <= N; i++) {
            for (int j = 1; j <= N; j++) {
                arr[i][j] = INF;

                if (i == j) {
                    arr[i][j] = 0;
                }
            }
        }

        for (int i = 0; i < M; i++) {
            st = new StringTokenizer(br.readLine());
            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());
            int c = Integer.parseInt(st.nextToken());

            // 출발 도시와 도착 도시가 같지만 시간이 다른 입력값이 들어올 수 있음.
            // 예를 들어 "1 4 1"과 "1 4 2"가 입력으로 들어왔으면,
            // "1 4 1"을 택해야 함.
            arr[a][b] = Math.min(arr[a][b], c);
        }
    }
}

```

```

// 플로이드 와샬 알고리즘
for (int k = 1; k <= N; k++) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            // 최단경로 초기화
            if (arr[i][j] > arr[i][k] + arr[k][j]) {
                arr[i][j] = arr[i][k] + arr[k][j];
            }
        }
    }
}

StringBuilder sb = new StringBuilder();
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        // 갈 수 없는 곳은 0으로 초기화
        if (arr[i][j] == INF) {
            arr[i][j] = 0;
        }

        sb.append(arr[i][j] + " ");
    }
    sb.append("\n");
}

bw.write(sb.toString());
bw.flush();
bw.close();
br.close();
}
}

```

감사합니다

