[백준]11404_플로이드

상태	진행 중
날짜	@2022년 5월 13일
공부유형	스터디
알고리즘	DP 최단경로
사이트	백준
깃허브	☐

사이트 링크 임베드

개념

최단경로

> ⭐ BFS : 가중치가 없거나 모든 가중치가 동일한 그래프에서 최단 경로 - 가장 빠름
> 다익스트라 : 음이 아닌 가중 그래프에서의 단일 쌍, 단일 출발, 단일 도착
> 벨만-포드 : 가중 그래프에서의 단일 쌍, 단일 출발, 단일 도착
> 플로이드-워셜 : 전체 쌍(정점) 최단 경로. (DP)

풀이

문제에 "***모든 도시의 쌍*** *(A, B)에 대해서 도시 A에서 B로* ***가는데 필요한 비용의 최솟값****을 구하는 프로그램을 작성하시오.* " 라고 적혀있으므로 **플로이드 워셜**을 이용하여 풀면 된다.

마침 출력도 이차원 배열로 i행→j열 최단경로 값 저장되도록 만든 것을 그대로 출력하면 되어서 굿!

초기 입력

D	1	2	3	4	5
1	0	2	3	1	10
2	INF	0	INF	2	INF
3	8	INF	0	1	1
4	INF	INF	INF	0	3
5	7	4	INF	INF	0

경유지 K = 1

D	1	2	3	4	5
1	0	2	3	1	10
2	INF	0	INF	2	INF
3	8	10	0	1	1
4	INF	INF	INF	0	3
5	7	4	10	8	0

fare[2][3] = min{fare[2][1]+fare[1][3],fare[2][3]} = INF

fare[3][2] = min{fare[3][1]+fare[1][2],fare[3][2]} = min{8+2,INF}=10

fare[5][3] = min{fare[5][1]+fare[1][3],fare[5][3]} = min{7+3,INF}=10

fare[5][4] = min{fare[5][1]+fare[1][4],fare[5][4]} = min{7+1,INF}=8

- 플로이드 워셜의 시간 복잡도는 인접행렬을 사용하면 $O(n^3)$
  - 이 문제에서 n(2 ≤ n ≤ 100)개의 도시 : 정점 n 최대 $100^3 = 1,000,000$
  - 한 도시에서 출발하여 다른 도시에 도착하는 m(1 ≤ m ≤ 100,000)개의 버스 : 간선 최대 100,000

<주의사항>

- static final int INF = 10000000;
  - Integer.MAX_VALUE로 하게되면 가중치를 더해보는 과정에서 오버플로우가 날 수 있음
  - 100,000*100 = 10,000,000
- *"시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다."*
  - 문제 잘 읽기. 입력에서부터 0 아닐 땐 최소값 받기
- INF로 초반에 직접 가는 경로가 없으면 설정해주는데 경유지를 거쳐서도 경로가 없는 경우 다시 되돌려주는 것 잊지말기. (98%에서 오류)

코드

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

public class Main_11404 { //플로이드

  static int N,M,fare[][];
```

```
  static final int INF = 10000000; //MAX_VALUE로 하게되면 가중치를 더해보는 과정에서 오버플로우가 날 수 있기 때문에!
  //100,000*100 = 10,000,000
  public static void main(String[] args) throws NumberFormatException, IOException {
    /* 입력 */
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st;
    N = Integer.parseInt(br.readLine()); // 도시의 개수
    M = Integer.parseInt(br.readLine()); // 버스의 개수
    fare = new int[N+1][N+1];

    for (int i = 0; i <M; i++) {
      st = new StringTokenizer(br.readLine());
      int a = Integer.parseInt(st.nextToken()); // 시작 도시
      int b = Integer.parseInt(st.nextToken()); // 도착 도시
      int c = Integer.parseInt(st.nextToken()); // 비용
      if(fare[a][b]==0 || fare[a][b]>c) { // 시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다.
        fare[a][b]=c;
      }
    }

    //자기자신으로의 인접 정보가 아니고 인접해있지 않다면 INF로 채우기
    //최소값을 갱신하는 것인데 0으로 되어있으면 문제가 될 수 있으니!
    for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
        if(i!=j && fare[i][j]==0) {
          fare[i][j]=INF;
        }
      }
    }

    //플로이드 워셜 수행 모든 쌍에 대해 경유지를 거치는것과 직접가는 것중에 최단 경로 저장 - 경 출 도
    for (int k = 1; k <= N; k++) {                                  // 경유지
      for (int i = 1; i <= N; i++) { // 경유지와 출발지가 같다면 다음
        if(i==k) continue;                                          // 출발지
        for (int j = 1; j <= N; j++) {                              // 도착지
          if(i==j || k==j) continue; // 목적지가 출발지와 같거나 경유지와 같다면 다음

          int indirect = fare[i][k] + fare[k][j]; // 경유지를 거칠 때 비용
          int direct = fare[i][j];                // 안거칠 때의 비용
          fare[i][j] = Math.min(indirect, direct);
        }
      }
    }

    //출력
    for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
        if(fare[i][j]==INF) fare[i][j]=0; // 경유지를 거쳐서도 i->j로의 길이 없는 경우 다시 0으로 돌려줌
        System.out.print(fare[i][j]+" ");
      }
      System.out.println();
    }
  }

}
```

참고

▼ 참고 코드