궁금한 민호

1507번 제출 맞은 사람 숏코딩 재채점 채점 현황 내 제출 난이도 기여 강의 질문 검색

궁금한 민호 성공 분류

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	4034	1972	1551	47.635%

문제

강호는 N개의 도시로 이루어진 나라에 살고 있다. 각 도시는 M개의 도로로 연결되어 있으며, 각 도로를 지날 때 필요한 시간이 존재한다. 도로는 잘 연결되어 있기 때문에, 도시 A에서 B로 이동할 수 없는 경우는 존재하지 않는다.

도시 A에서 도시 B로 바로 갈 수 있는 도로가 있거나, 다른 도시를 거쳐서 갈 수 있을 때, 도시 A에서 B를 갈 수 있다고 한다.

강호는 모든 쌍의 도시에 대해서 최소 이동 시간을 구해놓았다. 민호는 이 표를 보고 원래 도로가 몇 개 있는지를 구해보려고 한다.

예를 들어, 예제의 경우에 모든 도시 사이에 강호가 구한 값을 가지는 도로가 존재한다고 해도 된다. 하지만, 이 도로의 개수는 최솟값이 아니다. 예를 들어, 도시 1-2, 2-3, 1-4, 3-4, 4-5, 3-5를 연결하는 도로만 있다고 가정해도, 강호가 구한 모든 쌍의 최솟값을 구할 수 있다. 이 경우 도로의 개수는 6개이고, 모든 도로의 시간의 합은 55이다.

모든 쌍의 도시 사이의 최소 이동 시간이 주어졌을 때, 이 나라에 존재할 수 있는 도로의 개수의 최솟값과 그 때, 모든 도로의 시간의 합을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 N(1 ≤ N ≤ 20)이 주어진다. 둘째 줄부터 N개의 줄에 각각의 도시 사이에 이동하는데 필요한 시간 (≤ 10,000)이 주어진다. A에서 B로 가는 시간과 B에서 A로 가는 시간은 같다. 또, A와 B가 같은 경우에는 필요한 시간은 0이다.

출력

첫째 줄에 도로 개수가 최소일 때, 모든 도로의 시간의 합을 출력한다. 불가능한 경우에는 -1을 출력한다.

플로이드 와샬을 응용한 문제.

입력 값으로 모든 쌍의 도시간의 최소 이동 시간을 주어짐

`=>` 기존 값에서 플로이드 와샬을 한번 적용한 값이 라는 뜻.

플로이드 와샬의 기본 아이디어는

`i -> j` 로 가는 것보다 `i -> k -> j` 로 가는 길이 더 짧게되면 갱신하는 것이다

입력으로 주어진 값을 한번 더 최단 거리를 구했을 때,

나올 수 있는 경우의 수

1) `i -> j` 로 가는 값이 `i -> k -> j` 값 보다 크다

```
 => 이미 최단 거리를 구했는데 이런 경우가 있다는 것 자체가 오류 -> -1 출력 return


2) i -> j  로 가는 값이 i -> k -> j  값과 같다

 =>  i -> k -> j 로 가는 것이 더 많은 노드를 거쳐 갈 수 있으므로

     i -> j 로 가는 간선을 삭제해준다. ( INF )
```

코드

```
package Algo_Study_BOJ;

import java.io.*;
import java.util.StringTokenizer;

public class Main_1507_궁금한민호 {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int N = Integer.parseInt(br.readLine());

    int[][] map = new int[N + 1][N + 1];
    int[][] nextMap = new int[N + 1][N + 1];

    for(int i = 1; i <= N; i++) {
      StringTokenizer str = new StringTokenizer(br.readLine());
      for(int j = 1; j <= N; j++) {
        map[i][j] = nextMap[i][j] = Integer.parseInt(str.nextToken());
      }
    }

    for(int k = 1; k <= N; k++) {
      for(int i = 1; i <= N; i++) {
        for(int j = 1; j <= N; j++) {

          if(i == j || j == k || k == i) {
            continue;
          }

          if(map[i][j] > map[i][k] + map[k][j]) {
            System.out.println(-1);
            return;
          }
```

```
          if(map[i][j] == map[i][k] + map[k][j]) {
            nextMap[i][j] = 987654321;
          }
        }
      }
    }
    int answer = 0;
    for(int i = 1; i <= N; i++) {

      for(int j = 1; j <= N; j++) {

        if(i >= j && nextMap[i][j] != 987654321) {
          answer += nextMap[i][j];
        }
      }
    }
    System.out.println(answer);
  }
}
```