题目描述

某通信网络中有N个网络结点，用1到N进行标识。网络通过一个有向无环图表示，其中图的边的值表示结点之间的消息传递时延。

现给定相连节点之间的时延列表times[i]={u，v，w}，其中u表示源结点，v表示目的结点，w表示u和v之间的消息传递时延。

请计算给定源结点到目的结点的最小传输时延，如果目的结点不可达，返回-1。

注：

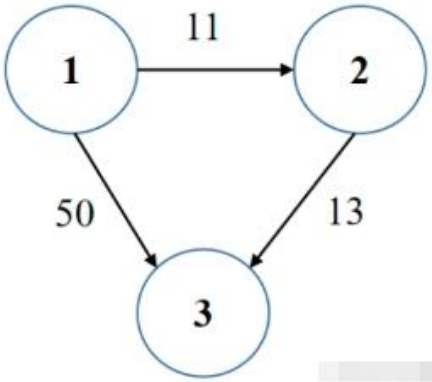
- N的取值范围为[1，100]；
- 时延列表times的长度不超过6000，且 1 <= u,v <= N，0 <= w <= 100；

输入描述

输入的第一行为两个正整数，分别表示网络结点的个数N，以及时延列表的长度M，用空格分隔；

接下来的M行为两个结点间的时延列表[u v w]；

输入的最后一行为两个正整数，分别表示源结点和目的结点。



输出描述

起点到终点得最小时延，不可达则返回-1

用例

输入	3 3 1 2 11 2 3 13 1 3 50 1 3
输出	24
说明	无

题目解析

本题是

的变种题，逻辑几乎一致，题目解析请参考链接博客。

本题采用的是dijkstra算法，即迪杰斯特拉算法求解。

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int m = sc.nextInt();

    int[][] times = new int[m][3];
    for (int i = 0; i < m; i++) {
      times[i][0] = sc.nextInt();
      times[i][1] = sc.nextInt();
      times[i][2] = sc.nextInt();
    }

    int src = sc.nextInt();
    int dist = sc.nextInt();

    System.out.println(getResult(n, times, src, dist));
  }

  public static int getResult(int n, int[][] times, int src, int tar) {
    HashMap<Integer, ArrayList<int[]>> graph = new HashMap<>();

    for (int[] time : times) {
      int u = time[0], v = time[1], w = time[2];
      graph.putIfAbsent(u, new ArrayList<>());
      graph.get(u).add(new int[] {v, w});
    }

    int[] dist = new int[n + 1];
    Arrays.fill(dist, Integer.MAX_VALUE);
    dist[src] = 0;

```

```
39      ArrayList<Integer> needCheck = new ArrayList<>();
40      while (true) {
41        boolean flag = false;
42
43        if (graph.containsKey(src)) {
44          for (int[] next : graph.get(src)) {
45            int v = next[0], w = next[1];
46            int newDist = dist[src] + w;
47
48            if (newDist >= dist[v]) break;
49            dist[v] = newDist;
50
51            if (!needCheck.contains(v)) {
52              needCheck.add(v);
53              flag = true;
54            }
55          }
56        }
57
58        if (needCheck.size() == 0) break;
59
60        if (flag) {
61          needCheck.sort((a, b) -> dist[a] - dist[b]);
62        }
63
64        src = needCheck.remove(0);
65      }
66
67      return dist[tar] == Integer.MAX_VALUE ? -1 : dist[tar];
68    }
69  }
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n, m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [n, m] = lines[0].split(" ").map(Number);
  }

  if (m && lines.length === m + 2) {
    lines.shift();
    let [src, dist] = lines.pop().split(" ").map(Number);
    let times = lines.map((line) => line.split(" ").map(Number));

    console.log(getMinTransDelay(n, times, src, dist));

    lines.length = 0;
  }
});

function getMinTransDelay(n, times, src, tar) {
  const graph = {};

  times.forEach((time) => {
    const [u, v, w] = time;
    graph[u] ? graph[u].push([v, w]) : (graph[u] = [[v, w]]);
  });

```

```
  const dist = new Array(n + 1).fill(Infinity);
  dist[src] = 0;

  const needCheck = [];

  while (true) {
    let flag = false;
    graph[src]?.forEach((next) => {
      const [v, w] = next;
      const newDist = dist[src] + w;

      if (newDist >= dist[v]) return;
      dist[v] = newDist;

      if (needCheck.indexOf(v) === -1) {
        needCheck.push(v);
        flag = true;
      }
    });

    if (!needCheck.length) break;

    if (flag) needCheck.sort((a, b) => dist[a] - dist[b]);

    src = needCheck.shift();
  }

  return dist[tar] == Infinity ? -1 : dist[tar];
}
```

Python算法源码

```
# 输入获取
import sys

n, m = map(int, input().split())
times = [list(map(int, input().split())) for _ in range(m)]
src, dist = map(int, input().split())


# 算法入口
def getResult(n, times, src, tar):
    graph = {}

    for u, v, w in times:
        if graph.get(u) is None:
            graph[u] = []
        graph[u].append([v, w])

    dist = [sys.maxsize for _ in range(n + 1)]
    dist[src] = 0

```

```
    needCheck = []
    while True:
        flag = False

        if graph.get(src) is not None:
            for v, w in graph[src]:
                newDist = dist[src] + w

                if newDist >= dist[v]:
                    break

                dist[v] = newDist

                if v not in needCheck:
                    needCheck.append(v)
                    flag = True

        if len(needCheck) == 0:
            break

        if flag:
            needCheck.sort(key=lambda x: dist[x])

        src = needCheck.pop(0)

    return -1 if dist[tar] == sys.maxsize else dist[tar]


# 测试调用
print(getResult(n, times, src, dist))
```