

# ☑ PAT 甲级题目讲解： 1003 《Emergency》

## 🔗 题目简介

这是一道带权无向图的最短路径综合题。

已知：

- 每个城市分布有若干救援队；
- 城市之间通过无向边相连，每条边有正整数距离；

我们的目标是：

- 从起点城市 `c1` 出发，以最快速度赶往目的地城市 `c2`，并沿途尽可能召集最多的救援队人员。

要求输出：

1. 从 `c1` 到 `c2` 的 **最短路径数量**；
2. 所有最短路径中，能召集到的 **最多救援队总数**。

关键建模点：

- 图中点表示城市，边表示道路；
- 要求最短路径条数 + 路径上的最大点权和 —— 典型的 **Dijkstra 拓展模型**。

## 🔧 样例分析

输入样例：

```
5 6 0 2
1 2 1 5 3
0 1 1
0 2 2
0 3 1
1 2 1
2 4 1
3 4 1
```

含义解释：

- `5` 个城市，`6` 条道路；
- 起点为城市 `0`，终点为城市 `2`；
- 每个城市救援队数量分别为：1, 2, 1, 5, 3；
- 道路 `(u, v, l)` 表示无向边  $u \leftrightarrow v$ ，长度为  $l$ 。

图结构如下：



可能路径：

- 1. 直接  $0 \rightarrow 2$ ，总长 2，救援队数量 =  $1 + 1 = 2$ ；
- 2.  $0 \rightarrow 1 \rightarrow 2$ ，总长  $1+1=2$ ，救援队数量 =  $1 + 2 + 1 = 4$ ；

共 2 条最短路径，其中最大救援队数量为 4。

输出：

2 4

## 🔍 解题思路

### 🔗 本题建模为 最短路径 + 路径统计 + 点权最大值：

基本框架为 **Dijkstra 算法**，并在基础上维护两组额外信息：

- 1. `cnt[v]`：从起点到城市 `v` 的最短路径条数；
- 2. `maxr[v]`：到城市 `v` 的最短路径中，最多可召集的救援队数。

## 🧠 变量说明

变量名	类型	含义
<code>n</code>	int	城市数
<code>m</code>	int	道路数
<code>c1, c2</code>	int	起点、终点城市编号
<code>r[i]</code>	int	第 <code>i</code> 个城市的救援队数目
<code>head[i]</code>	int	城市 <code>i</code> 的邻接链表头指针
<code>to[k]</code>	int	第 <code>k</code> 条边指向的城市编号
<code>nt[k]</code>	int	第 <code>k</code> 条边的下一条边（链式前向星）
<code>val[k]</code>	int	第 <code>k</code> 条边的长度
<code>dis[i]</code>	int	起点到城市 <code>i</code> 的最短距离
<code>cnt[i]</code>	int	起点到城市 <code>i</code> 的最短路径数量

<code>maxr[i]</code> 变量名	<code>int</code> 类型	起点到城市 <code>i</code> 的路径中最多可召集救援队 含义
<code>vis[i]</code>	<code>bool</code>	城市 <code>i</code> 是否已访问

## ☑ Step 1: 建图处理（链式前向星）

采用链式前向星建图，可高效存储无向边：

```
void adde(int x, int y, int w){ // 建立 x -> y, 权值为 w 的边
    to[++k] = y;           // 目标城市编号
    nt[k] = head[x];       // 插入链表头
    head[x] = k;           // 设置结点 x 第一条出度边为 k
    val[k] = w;            // 权重/道路长度
}
```

注意：每条无向边需调用 `adde(x,y,w)` 和 `adde(y,x,w)` 各一次。

## ☑ Step 2: Dijkstra 模板扩展

通过贪心策略，每次选出 **当前未访问的、距离起点最近的城市** `u`，然后尝试更新其所有邻接点 `v`：

```
void d(int s){ // 求从 s 出发到图中任意其他点的最短路
    dis[s] = 0; // 初始化起点最短距离为 0
    maxr[s] = r[s]; // 初始可收集的救援队
    cnt[s] = 1; // 起点到自身的路径数为 1

    for(int i = 0; i < n; i++){ // n 轮选点
        int u = -1, mind = inf; // u 记录选取最近点, mind 记录最近距离
        for(int j = 0; j < n; j++){ // 遍历 n 个点
            if(!vis[j] && dis[j] < mind){ // 找出未访问点中距离起点最近的点更新 u
                u = j; mind = dis[j];
            }
        }
        if(u == -1) break; // 上面 for() 执行完没找到合适点 -> 所有点已访问完
        vis[u] = 1; // 标记所选点被访问
```

## ☑ Step 3: 松弛操作 + 统计路径信息

对于每个 `u → v` 的边，根据新路径是否更优，进行三种情况判断：

```
for(int j = head[u]; j; j = nt[j]){ // j: 枚举 u 的所有邻接边
    int v = to[j], l = val[j]; // v: 所有与 u 邻接的点, l: u -> v 的权值
    int t = dis[u] + l; // 从起点到 v 的当前路径长度

    if(t < dis[v]){ // 若当前路径更短
        dis[v] = t; // 更新最短路径
        maxr[v] = maxr[u] + r[v]; // 更新当前路径最大救援队数量是上一步加该步的
        cnt[v] = cnt[u]; // 更新当前路径最短路径数量等于上一步最短路径数
    }
    else if(t == dis[v]){ // 路径长度相等
        cnt[v] += cnt[u]; // 多 cnt[u] 条路径
```

```

        maxr[v] = max(maxr[v], maxr[u] + r[v]); // 尝试更新最多救援队数量
    }
}
}
}

```

## ☑ 完整代码

```

#include <bits/stdc++.h>
using namespace std;

const int maxn = 250005;
const int inf = INT_MAX;
int n, m, c1, c2, r[505];
int head[505], to[maxn], nt[maxn], val[maxn], k;
int dis[505], cnt[505], maxr[505];
bool vis[505];

void adde(int x, int y, int w){ // 建立 x -> y, 权值为 w 的边
    to[++k] = y; // 目标城市编号
    nt[k] = head[x]; // 插入链表头
    head[x] = k; // 设置结点 x 第一条出度边为 k
    val[k] = w; // 权重/道路长度
}

void d(int s){ // 求从 s 出发到图中任意其他点的最短路
    dis[s] = 0; // 初始化起点最短距离为 0
    maxr[s] = r[s]; // 初始可收集的救援队
    cnt[s] = 1; // 起点到自身的路径数为 1

    for(int i = 0; i < n; i++){ // n 轮选点
        int u = -1, mind = inf; // u 记录选取最近点, mind 记录最近距离
        for(int j = 0; j < n; j++){ // 遍历 n 个点
            if(!vis[j] && dis[j] < mind){ // 找出未访问点中距离起点最近的点更新 u
                u = j; mind = dis[j];
            }
        }
        if(u == -1) break; // 上面 for() 执行完没找到合适点 -> 所有点已访问完
        vis[u] = 1; // 标记所选点被访问
        // 遍历 u 所有邻接边, 尝试松弛
        for(int j = head[u]; j; j = nt[j]){ // j: 枚举 u 的所有邻接边
            int v = to[j], l = val[j]; // v: 所有与 u 邻接的点, l: u -> v 的权值
            int t = dis[u] + l; // 从起点到 v 的当前路径长度

            if(t < dis[v]){ // 若当前路径更短
                dis[v] = t; // 更新最短路径
                maxr[v] = maxr[u] + r[v]; // 更新当前路径最大救援队数量是上一步加该步的
                // 总和
                cnt[v] = cnt[u]; // 更新当前路径最短路径数量等于上一步最短路径数
            }
            else if(t == dis[v]){ // 路径长度相等
                cnt[v] += cnt[u]; // 多 cnt[u] 条路径
                maxr[v] = max(maxr[v], maxr[u] + r[v]); // 尝试更新最多救援队数量
            }
        }
    }
}

```

```

    }
}

int main(){
    scanf("%d %d %d %d", &n, &m, &c1, &c2);
    for(int i = 0; i < n; i++){
        scanf("%d", &r[i]);
        dis[i] = inf; // 初始化为不可达
    }
    while(m--){
        int x, y, l;
        scanf("%d %d %d", &x, &y, &l);
        // 无向图，双向建边
        adde(x, y, l);
        adde(y, x, l);
    }
    // Dijkstra 求最短路径 + 路径数 + 最大点权和
    d(c1);
    // 输出答案：路径条数 + 最大救援队数
    printf("%d %d", cnt[c2], maxr[c2]);
    return 0;
}

```

## 🚩 常见错误提醒

错误类型	具体表现
没有双向建边	只调用一次 <code>adde(x,y,l)</code> ，会导致图不连通
忘记初始化 <code>dis[i]</code>	导致最短路径判断错误
忽略路径相等情况	只处理 <code>t &lt; dis[v]</code> ，忽略 <code>t == dis[v]</code> 的统计逻辑
<code>maxr</code> 更新顺序错误	忘记取 <code>max(...)</code> ，直接累加错误
没有标记访问	<code>vis[i]</code> 未标记，会导致死循环或重复访问

## ✅ 总结归纳

### 🧠 核心算法

- 本题为典型的 **单源最短路径扩展题**；
- 采用 Dijkstra 算法 + 路径计数 + 点权最大值维护；
- 图采用链式前向星高效建图，避免邻接矩阵空间浪费。

### ⚙️ 复杂度分析

- 时间复杂度： $\mathcal{O}(n^2 + m)$ ，其中  $n \leq 500$ ， $m \leq n^2$ ；
- 空间复杂度： $\mathcal{O}(n + m)$ 。

---

## 思维拓展

---

- 若将图中边权改为负数，如何处理？
  - Dijkstra 不再适用，需使用 Bellman-Ford 或 SPFA;
- 如果要求输出所有路径的具体路径信息？
  - 可通过 `pre[v]` 记录前驱链 + DFS 构建;
- 若加入“最短路径中最少经过节点数”要求，如何处理？
  - 可额外记录 `step[v]` 数组统计路径长度。