# 行车路线

> 计算学部十大打卡活动——“龙舞编程新春会”编程打卡（2024-1-28）
>
> AcWing3255.行车路线

# 一、题目

小明和小芳出去乡村玩，小明负责开车，小芳来导航。

小芳将可能的道路分为大道和小道。

大道比较好走，每走 $1$ 公里小明会增加 $1$ 的疲劳度。

小道不好走，如果连续走小道，小明的疲劳值会快速增加，连续走 $s$ 公里小明会增加 $s^2$ 的疲劳度。

例如：有 $5$ 个路口，$1$ 号路口到 $2$ 号路口为小道，$2$ 号路口到 $3$ 号路口为小道，$3$ 号路口到 $4$ 号路口为大道，$4$ 号路口到 $5$ 号路口为小道，相邻路口之间的距离都是 $2$ 公里。

如果小明从 $1$ 号路口到 $5$ 号路口，则总疲劳值为 $(2+2)^2+2+2^2=16+2+4=22$。

现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

**输入格式**

输入的第一行包含两个整数 $n$,$m$，分别表示路口的数量和道路的数量。路口由 $1$ 至 $n$ 编号，小明需要开车从 $1$ 号路口到 $n$ 号路口。

接下来 $m$ 行描述道路，每行包含四个整数 $t,a,b,c$，表示一条类型为 $t$，连接 $a$ 与 $b$ 两个路口，长度为 $c$ 公里的双向道路。其中 $t$ 为 $0$ 表示大道，$t$ 为 $1$ 表示小道。

保证 $1$ 号路口和 $n$ 号路口是连通的。

**输出格式**

输出一个整数，表示最优路线下小明的疲劳度。

**数据范围**

对于 30 的评测用例，$1 \le n \le 8$，$1 \le m \le 10$；
对于另外 20 的评测用例，不存在小道；
对于另外 20 的评测用例，所有的小道不相交；
对于所有评测用例，$1 \le n \le 500$，$1 \le m \le 10^5$，$1 \le a,b \le n$，$t$ 是 $0$ 或 $1$，$c \le 10^5$。
**保证答案不超过**$10^6$。

**时/空限制**
1s / 256MB
**输入样例：**
6 7
1 1 2 3
1 2 3 2
0 1 3 30
0 3 4 20
0 4 5 30
1 3 5 6
1 5 6 1
**输出样例：**
76
**样例解释**
从 $1$ 走小道到 $2$，再走小道到 $3$，疲劳度为 $5^2=25$；然后从 $3$ 走大道经过 $4$ 到达 $5$，疲劳度为 $20+30=50$；最后从 $5$ 走小道到 $6$，疲劳度为 $1$。

总共为 $76$。

# 二、题解

## 思路

由于题目保证答案不超过$10^6$，所以连续小路的总长度不超过$10^3$，又由于总点数不超过$500$，原有点数较少，所以可以用**拆点**的方法。
**拆点**：

- 将一个图里的边权全部化为 $1$ ，即观察图中最大边权为 $n$ ，则将每一个点拆解为 $n$ 个点，命名为 $i.1, i.2, \ldots\ldots, i.n$ 每个拆解后的点的边权为 $1$ 。
- 如何连边，使得新图和原图等价：先把 $i.1$ 与 $i.2$ 相连，$i.2$ 与 $i.3$ 相连……；再将原图相连的点连接，如果原图点 $i$ 与 $j$ 的边权为 $x$，则 $i.x$ 与 $j.1$ 相连

设置 $dist$ 二维数据，记录距离， $dist[i][j]$ 的含义：从起点走到$i$，且最后一条小道长度为 $j$ 的最短路。

更新：若 $i$ 邻接的点为 $k$ ，权重为 $w$ ：

- 若 $i-k$ 为大道：$dist[k][0]=dist[i][j]+w$
- 若 $i-k$ 为小道：$dist[k][j+w]=dist[i][j]-j^2+(j+w)^2$

此题只需要对小道上的点进行拆点，所以可以得到此题新图中的点数最多为 $50$ 万，所以使用朴素的 dijkstra 算法会超时，故采用堆优化的 dijkstra 算法。

# 代码

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>

using namespace std;

const int N = 510, M = 200010, INF = 1e6; //N为点数最大值，M为有向边数最大值，由于题目为无向边，所以

int n, m; //定义点数和边数
int h[N], e[M], f[M], w[M], ne[M], idx; //使用邻接表存图，h表示头节点，e表示其余节点，f表示类型，w表
int dist[N][1010];
bool st[N][1010]; //dijkstra中的判重数组
struct Node
{
    int x, y, v; //x表示点的编号，y表示最后一条小道长度，v表示距离
    //重载符号，实现小根堆
    bool operator< (const Node& t) const
    {
        return v > t.v;
    }
};

//加边函数
void add(int t, int a, int b, int c)
{
    e[idx] = b, f[idx] = t, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}

//dijkstra
void dijkstra()
{
    priority_queue<Node> heap; //定义堆，优化dijkstra
    heap.push({1, 0, 0});
    memset(dist, 0x3f, sizeof dist); //初始距离为正无穷
    dist[1][0] = 0;
    while (heap.size())
    {
        auto t = heap.top();
        heap.pop(); //删除堆顶元素
```

```
        if (st[t.x][t.y]) continue;
        st[t.x][t.y] = true;
        for (int i = h[t.x]; ~i; i = ne[i])
        {
            int x = e[i], y = t.y;
            //若当前边为小道
            if (f[i])
            {
                y += w[i]; //小道长度增加
                if (y <= 1000)
                {
                    if (dist[x][y] > t.v - t.y * t.y + y * y)
                    {
                        dist[x][y] = t.v - t.y * t.y + y * y;
                        if (dist[x][y] <= INF)
                            heap.push({x, y, dist[x][y]});
                    }
                }
            }
            //若当前边为大道
            else
            {
                if (dist[x][0] > t.v + w[i])
                {
                    dist[x][0] = t.v + w[i];
                    if (dist[x][0] <= INF)
                        heap.push({x, 0, dist[x][0]});
                }
            }
        }
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    //读入每条边
    while (m -- )
    {
        int t, a, b, c;
        scanf("%d%d%d%d", &t, &a, &b, &c);
        add(t, a, b, c), add(t, b, a, c); //加边
```

```
    }

    dijkstra();
    int res = INF;
    //枚举最后一个点的所有分身
    for (int i = 0; i <= 1000; i ++ ) res = min(res, dist[n][i]);
    printf("%d\n", res);

    return 0;
}
```