写在前面：我们总共有三门课讲到最短路：数据结构、离散II□算法。

（推荐任韩图论，一本跨越了数竞和计竞的高中数竞书）

为什么把这两个页面合二为一了呢？

只需要把最短路中的优先队列换成普通的队列，就变成了广度优先搜索。也就是说，如果所有的边权都为1□priority_queue也可以用queue替代。

另外还要注意□Dijkstra的实质，是将给定的源点生成最短路“场”，一下子求了**一定范围内所有点的最短路**。有的题目会用到这个场的性质。

Dijkstra的三步走

前提条件

Dijkstra算法（又称标号法），运行结果是从权重为0的起始点开始，为所有顶点标号——标权重。

要求权重全部为正。（为负的话请另寻他法）

头文件只有stdio.h□输入输出□□string.h□memset初始化）……

以及 queue 和C++里万能的语句 using namespace std;

需要两个数组。两个数组的大小都是顶点数。

数据类型小一点的数组vis□用于记录顶点是否已经编号了。

数据类型大一点的数组dis□用于记录顶点权重（即运行结果）。

您还需要一个特殊的优先队列□priority_queue<pair<int,int> > q;

介绍一下工具□priority_queue□又称**大顶堆**。每次默认弹出最大元素。

和通常的队列语法类似□pop是弹出□push是压入。

top是堆顶元素□empty是判断是否为空。

其中□pair 是特殊的结构体，两元素 first 和 second □

比较时默认先比较 first 大小□first 相同时比较 second 大小。这是在强调序关系的 priority_queue 中可以调用的原理基础。

make_pair 函数，将输入两个元素按顺序结合，成为输出的 pair 类型结构体。

第一步：初始化

利用 memset 函数，将 dis 数组全写成 0x3f □正最大值，代指距离无穷大），将 vis 数组全写为0（未访问过）。

将起始点（标号为0的点，可以不止一个）全部压入堆，同时将对应 dis 数组（之前置为最大值了）置为0。

语句为：

```
    q.push(make_pair(0,begin));
    dis[begin]=0;
```

根据堆特征□ push 和 make_pair 连用。因为要对权重（距离）排序，所以权重是第一元素。这里的 0，代表距离为 0。

第二步：找下一个标号点

```
    while(!q.empty())
    {
        int x=q.top().second;
        q.pop();
        if(vis[x])
        {
            continue;
        }
        vis[x]=1;
```

这段的意思：只要堆 q 非空——（空的话就表示图里全标完了，结束就完事了）

令 x 是要找的下一标号点（的编号，那么x是第二元素）。那么 x 一定在堆顶。

于是top □second □pop组合拳连用。

但是堆顶未必是想要的元素，没准 x 已经被访问过了。因此，检查 vis 数组。如果已经访问，就直接 continue 掉这一循环，从下一循环重新找，本循环仅仅 pop 了一个元素而已。

然后，置 vis 数组为 1，表示已经访问过。

第三步：标号

令 i 跑遍上文节点 x 的所有邻居□□ for 循环，跑遍即可□i未必是节点编号）

这里依赖于图的建构。如果用矩阵写，就跑遍矩阵的一行。如果用邻接表写，就跑遍邻接表的一行。

注意，对于无向图，建构时要将两个方向全部写入（序号、权重）。

对于每一个邻居节点，无论标过与否，都跑一遍。这里 y 是节点编号□ time 是对应权重，然后有：

```
            if(dis[y]>dis[x]+time)
            {
                dis[y]=dis[x]+time;
                q.push(make_pair(-dis[y],y));
```

```
            }
```

只有新权重（节点+权重）比老权重小的时候，才编号，其余时候并不编号。编号完了，就把这个刚编的节点压入堆。

注意！这里的堆，默认是大顶堆，而每次取的时候（见第二步），要取最小的元素。

因此**每次压入的时候，将每个权重（第一元素）取负**，变相将大顶堆**改造成小顶堆**。这个操作很巧妙。

总共只有这三个步骤。后面没有了。

完整代码

```
void Dijkstra(int begin)
{
    //步骤一
    memset(dis,0x3f,sizeof(dis));
    memset(vis,0,sizeof(vis));
    q.push(make_pair(0,begin));
    dis[begin]=0;
    //步骤二
    while(!q.empty())
    {
        int x=q.top().second;
        q.pop();
        if(vis[x])
        {
            continue;
        }
        vis[x]=1;
        //步骤三
        int i;
        for(i=0;i<top[x];i++)
        {
            int y=V[x][i].first;
            int time=V[x][i].second;
            if(dis[y]>dis[x]+time)
            {
                dis[y]=dis[x]+time;
                q.push(make_pair(-dis[y],y));
            }
        }
    }
}
```

利用队列的BFS非递归实现

```
void bfs(int v)//以v开始做广度优先搜索（非递归实现，借助队列）
```

```
{
    //步骤一
    list<int>::iterator it;
    visited[v] = true;
    cout << v << " ";
    queue<int> myque;
    myque.push(v);
    //步骤二
    while (!myque.empty())
    {
        v = myque.front();
        myque.pop();
        //步骤三
        for (it = graph[v].begin(); it != graph[v].end(); it++)
        {
            if (!visited[*it])
            {
                cout << *it << " ";
                myque.push(*it);
                visited[*it] = true;//访问过
            }
        }
    }
    cout << endl;
}
```

例题

给出了一个具有n个顶点和m条边的加权连通无向图。你知道有人在顶点1点燃了火，它会立即将当前位置烧成灰尘，并以每秒1英里的速度扩展到相邻的地方。火将在顶点处分裂到所有未点亮的边上，并在至少两个火在同一点相遇时引发爆炸。革命者喜欢爆炸。他们想让你数一数图表上发生的爆炸次数。

第一行包含整数n□m□1≤n≤3×105□0≤m≤106□-顶点数。

接下来的m行中的每一行包含3个整数ui□vi□wi□1≤ui□vi≤n□1≤wi≤9□-第i条边的端点和第i条边的权重。

保证图是连通的。

图可以包含自循环和多条边。示例1显示了处理它们的方法。

输入文件的大小可能很大。请不要读得太慢。

输出将在图形上以单行方式发生的爆炸数。

样例

输入

2 3

1 1 1

1 2 1

1 2 1

输出

2

输入

4 5

1 2 1

1 3 1

2 3 1

2 4 1

3 4 1

输出

2

题解

在点上爆炸意味着存在至少两个点可以是他最短路上的前驱；在边上爆炸意味着这条边不在最短路里。因此求出最短路即可。

边权≤ 9 并不意味着可以使用SPFA□本题的本意是使用O(nw) 的桶排序Dijkstra, 但是由于vector实现的桶排Dijkstra 和邻接表+ priority_queue 实现的O(mlogm) Dijkstra 速度在使用输入随机数生成器的方法开大数据后没法拉开差距，因此为了避免卡常数都放了过去。

总之大概的意思是，这东西恰好是最短图标记图后直接算出来。

```
#include<stdio.h>
#include<string.h>
#include<queue>

using namespace std;

struct node
{
    int to,next,val;
};

struct node e[10000050];
```

```
int head[1000005],cnt,n,m,K;

void add(int first,int second,int z)
{
    e[cnt]=(struct node){second,head[first],z};
    head[first]=cnt++;
}

priority_queue<pair<int,int> > q;

int dis[1000005],vis[1000005],ans,used[1000005];

void Dijkstra()
{
    q.push(make_pair(0,1));
    memset(dis,0x3f,sizeof(dis));
    dis[1]=0;
    while(!q.empty())
    {
        int first = q.top().second;
        q.pop();
        if(vis[first])
        {
            continue;
        }
        vis[first] = 1;
        int i;
        for(i=head[first];i!=-1;i=e[i].next)
        {
            int to1 = e[i].to;
            if(dis[to1]>dis[first]+e[i].val)
            {
                dis[to1]=dis[first]+e[i].val;
                used[to1]=1;
                q.push(make_pair(-dis[to1],to1));
            }
            else if(dis[to1]==dis[first]+e[i].val)
            {
                used[to1]++;
            }
        }
    }
}

int main()
{
    scanf("%d%d",&n,&m);
    memset(head,-1,sizeof(head));
    int i,x,y,z;
    for(i=1;i<=m;i++)
    {
```

```
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);
        add(y,x,z);
    }
    Dijkstra();
    for(i=1;i<=n;i++)
    {
        if(used[i]>=2)
        {
            used[i]--;
        }
        m-=used[i];
    }
    printf("%d\n",m);
}
```