O题解题报告：dij+堆优化（O（VE））

题目大意：在n个节点，m个有向边的图中求第s个节点到各个节点的最小距离。如果s节点无论如何也无法达到k节点，则输出-1。其中1<n<=10^5,1<=s<=2×10^5 ,1<=s<=n,1<=ui,vi<=n;0<=wi<=10^9。

首先我们应该注意到这个题目的一个特点：

数据量比较大；还有一个大坑（卡了我两三天）：权重的大小。

这道题中，单边最大权重已经可以和一个int的最大值相比了，所以初始化s节点到各个节点的最小距离时一定要初始的足够大。

```
const long long INF = 1000000000000000;
long long dis[maxn];
memset (dis,INF,sizeof dis);
```

因为个人学习路线的安排，在做这道题的时候本人使用的是Dijkstra算法。

这里稍微简单的介绍下Dijkstra算法（当作是默写复习了）。Dijkstra算法是一种贪心算法，每次松弛操作后访问累积路径最小的点。在没有负权边的情况下（如本题），Dijkstra算法所访问的节点上累积的路径长度必定是最小的。

```
//Dijkstra算法核心代码
    for(int i = 0 ; i < e[u].size() ; i++){
        int v = e[u][i].u;
        long long w = e[u][i].w;
        if(!vis[v] && dis[v] > dis[u] + w)
        {//dis[]里存的是指定节点到其他节点的位置
            dis[v] = dis[u] + w;
            //temp.u=v,temp.w=dis[v];
            //pque.push(temp);下文队列中会用到
        }
    }
```

但是，普通的Dijkstra算法在如此大的数据量面前显得有些吃力，因此我们必须对Dijstra算法进行一定的优化。

考虑到，Dijstra算法每次找最小权重边的时候事实上都是一次从1到n的遍历，因此我们可以使用优先队列优化Dijstra算法。利用优先队列，每次循环时，都会让累积路径最小的节点出列，这样就避免了大量重复无意义的循环操作，从而实现了对算法的优化。

```
    //优化队列大致结构
    priority_queue <node> pque;//定义队列
    pque.push(temp);//先把第一个放进去
    while(!pque.empty())//队列不为空，继续
    {
        node t = pque.top();//找到最优先的那个
        pque.pop();//取出来后删掉
        ……
        pque.push(variable);//把符合条件的放进队列里
        ……
    }
```

AC代码如下：

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

const int maxn = 1e6+5;
const long long INF = 1000000000000000;
bool vis[maxn];
long long dis[maxn];

struct node
{
    int u;
    long long w;
    bool operator <(const node &r)const
    {
        return w > r.w;
    }
}temp;
vector <node> e[maxn];
void dijkstra(int st)
{
    fill(vis,vis+maxn,false);
    fill(dis,dis+maxn,INF);
    priority_queue <node> pque;
    dis[st] = 0;
    temp.u=st,temp.w=0;
    pque.push(temp);
    while(!pque.empty())
    {
        node t = pque.top();
        pque.pop();
        int u = t.u;
        if(vis[u])
            continue;
        vis[u] = true;
        for(int i = 0 ; i < e[u].size() ; i++)
        {
            int v = e[u][i].u;
            long long w = e[u][i].w;
            if(!vis[v] && dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w;
                temp.u=v,temp.w=dis[v];
                pque.push(temp);
            }
        }
    }
}

int main()
{
    int n,m,s;
 scanf("%d%d%d",&n,&m,&s);
    for(int i = 1 ; i <= m ; i++)
    {
```

```
    {
        int u,v;
  long long w;
        scanf("%d%d%d",&u,&v,&w);
        temp.u=v,temp.w=w;
        e[u].push_back(temp);
    }
    dijkstra(s);
    for(int i = 1; i<=n ;i++){
 if (dis[i] == INF)
        printf("-1\n");
    else
     printf("%lld\n",dis[i]);
    }
    return 0;
}
```