迪杰斯特拉最短路

用于解决无向/ 有向图的正权边单源最短路问题，可以有环路但不能有负权边。

算法介绍

基于贪心的思路。朴素版时间复杂度$O(n^2)$，n代表点的数量；堆优化后时间复杂度$O(mlogn)$，m代表边的数量n代表点的数量

什么是单源最短路

给出一个起点S，找到图中每个点距离起点S的最小距离（根据边权来给距离）

思路

进行多轮迭代，每次找出一个距离S最近的点，**此时源点到该点的距离便是最短路**，之后根据这个点更新其他图中的点到S的距离。

可行性：新找到的距离S最近的点可以作为一个桥梁来**松弛**其他节点。比如A和C距离为10，A和B距离为2，B和C距离为3，那么A到C的距离就可以借助B这个节点来缩短。

把这个操作叫做松弛操作。可以想象成一根绷紧的绳子，长度为10；松开后自然弯折，一段为5一段为3.

```
if (源点到to的当前最短距离 > 源点到now的最短距离 + now到to的距离) {
    源点到to的当前最短距离 = 源点到now的最短距离 + now到to的距离
}
```

贪心证明

为什么源点到所有可见点的路径中长度最待人的一条一定是源点到该点的最短路径？为什么不会存在通过一些当前不可见的点间接到达该点，使得这条路径距离更短？

- 首先能保证的是，**拿到的第一个点一定是和源点直接相连且距离最短的点** 。也就是说第一轮迭代不需要进行松弛操作
- 设图G的顶点集合为V，再设一个集合S表示已求得最短路径的终点的集合（S怎么来的下面再说）。

  设下一条最短路径（终点为x），那么它只能是弧（v，x）或者通过S中的顶点到达x即（v，v_1,v_2，…，x）。我们来证明一下：

  假设（v，…,x）路径上有一个顶点不在S中，则说明存在一条终点不在S中而长度比此路径还短的路径。但这是不可能的。因为我们按长度递增的顺序来产生各最短路径，所以长度比此路径还短的所有路径均已产生，他们的终点一定在S中。

朴素迪杰斯特拉

时间复杂度为 $O(n^2)$

具体步骤

1. 初始化。设V为G中所有顶点集合，s为源点（起点）。S表示已求得最短路径的终点的集合，初始状态是S={s}。$dis[x]$表示源点到x的当前已知最短路，初始情况$dis[s]=0$，其余均为正无穷大。
2. 从源点s找可见点，并计算可见点到源点的路径长度并记录路径长度。然后对所有路径进行排序，选择最短的一条作为确定找到的最短路径，将其终点加入集合S中。
3. 利用新加入的终点 now 来更新与之相邻的点 to 的最短路距离，进行松弛操作。
4. 不断重复2和3的步骤，直至所有点已加入集合S或者搜索不到新的可见点，就终止算法。

具体的模拟可以看看这篇文章https://www.acwing.com/solution/content/38318/或者上网搜其他的博客，资料很多的。

代码

用一个bool数组vis用于标记当前点有没有加入到集合S。

用一个整数类型数组dis记录源点到该点的最短距离

```
#define INF 0x3f3f3f3f
int mp[MAXN][MAXN];//存储每条边
int dis[MAXN];//存储一号点到每个点的最短距离
int vis[MAXN];//存储每个点的最短路是否已经确定
//求一号点到n号点的最短距离，如果不存在返回-1

void init() {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (i == j) mp[i][j] = 0;
            else mp[i][j] = INF;
}

int dij(int s) {
    memset(dis, 0x3f, sizeof(dis));
    dis[s] = 0;
    for(int i = 0; i < n - 1; i++) {
        int t = -1;//在还未确定最短路的点中，寻找距离最小的点
        for(int j = 1; j <= n; j++)
            if(!vis[j] && (t == -1 || dis[t] > dis[j]))
                t=j;

        for (int j = 1; j <= n; j++)
            dis[j] = min(dis[j], dis[t] + mp[t][j]);//用t更新其他点的最短距离

        vis[t] = 1;
    }
    if (dis[n] == INF) return -1;
    return dis[n];
}
```

堆优化

时间复杂度$O(nlogm)$

可以注意到整个算法主要干了两件事

1. 从当前已知的路径中选择长度最短的路径，将其终点加入集合S
2. 利用新找到的点更新其他点的最短路路径

第一个步骤每次找一轮最小值的时间复杂度是$O(n)$，可以利用优先队列这个数据结构进行优化，可以把时间复杂度降到$O(logm)$，m代表边的个数。不知道优先队列的看这个视频 ACM程序设计竞赛暑期实训-STL*哔哩哔哩*bilibili

模板1（不调用优先队列的greater函数，将优先队列的权值设为负数即可实现小根堆）

```
#define INF 0x3f3f3f3f
typedef pair<int,int> pii;
int n;//点的数量
int dis[MAXN];//存储所有点到一号点的距离
int vis[MAXN];//存储每个点的最短距离是否已经确定

//求一号点到n号点的最短距离，如果不存在，则返回-1
int dij(int s) {
    for (int i = 1; i <= n; i++) dis[i] = INF;
    dis[s] = 0;
    priority_queue<pii> q;
    q.push({dis[s], s});
    while (!q.empty()) {
        int now = q.top().second; q.pop();
        if (vis[now]) continue;
        vis[now] = 1;
        //遍历与now相邻的所有邻点
        for(int i = head[now]; i; i = edge[i].next) {
            int to = edge[i].to;
            int val = edge[i].val;
            if(dis[to] > dis[now] + val) {
                dis[to] = dis[now] + val;
                q.push({-dis[to], to});
            }
        }
    }
    if(dis[n] == INF) return -1;
    return dis[n];
}
```

模板2，利用优先队列的greater函数将其变为小根堆

```
#define INF 0x3f3f3f3f
typedef pair<int,int> pii;
int n;//点的数量
int dis[MAXN];//存储所有点到一号点的距离
int vis[MAXN];//存储每个点的最短距离是否已经确定

//求一号点到n号点的最短距离，如果不存在，则返回-1
int dij(int s) {
    for (int i = 1; i <= n; i++) dis[i] = INF;
    dis[s] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> q;
    q.push({dis[s], s});
    while (!q.empty()) {
        int now = q.top().second; q.pop();
        if (vis[now]) continue;
        vis[now] = 1;
        //遍历与now相邻的所有邻点
        for(int i = head[now]; i; i = edge[i].next) {
            int to = edge[i].to;
```

```
            int val = edge[i].val;
            if(dis[to] > dis[now] + val) {
                dis[to] = dis[now] + val;
                q.push({dis[to], to});
            }
        }
    }
    if(dis[n] == INF) return -1;
    return dis[n];
}
```