最短路拓展

【本节概述】

本节包括最短路的一些应用：次短路，最短路计数，分层图。

一、　　次短路

次短路即算第二大的路径，一个朴素算法是：

 ①读入数据，建立图

 ②用 dijkstra 找出一条最短路径 p1。

 ③由于第二最短路径至少有一条边和 p1 不同，所以可以这样求第二短路：枚举 p1 中的每一条边 e1，将 e1 从图 G 中删除得到 G′，然后再 G′中求最短路，所有这些 G′最短路中最短的为第二短路 p2。

 时间复杂度：O(n^3)

算法二：

记最短路径长度为 d[ ]，次短路径长度为 dd[ ]，则 d[v] = min{ d[u] + cost[u][v], dd[u] + cost[u][v]}，所以我们只需要计算出最短路径和次短路径即可。这就跟最短路径不一样了，在实现 Dijkstra 算法的时候，我们既要记录最短路径，还要记录次短路径。

算法三：

正反跑两次 SPFA，然后枚举每一条边，如果起点到一个端点的最短路+另一个端点到终点的最短路+长度 ≠ 最短路，则和答案比较，保存最小值

【分析】

1 到 n 的次短路长度必然产生于：从 1 走到 x 的最短路 + g[x][y] + y 到 n 的最短路

首先预处理好 1 到每一个节点的最短路,和 n 到每一个节点的最短路　然后枚举每一条边作为中间边（x，y）或者（y，x)，如果加起来长度等于最短路长度则跳过，否则更新

从 1 走到 x 的最短路 + edge[x][y] + y 到 n 的最短路　给 dist[n] 比较 找大于 dist[n] 且是最小的那一个

实例：P3064 次短路

二、最短路计数问题

1、无向图最短路计数 P3065

问题：给出一个 **N** 个顶点 **M** 条边的无向带权图，顶点编号为 **1～N**。 问从顶点 **1** 开始，到其他每个点的最短路有几条。

【解析】

求最短路的条数只需要在 dijkstra 上面加一个数组 sumt[]记录就行，sumt[v] 表示从源点 s 出发到 v 的最短路条数，

当 dist[v] > dist[u] + d[u][v] 时，更新 sumt[v] 的值就是 sumt[u];

当 dist[v] == dist[u] + d[u][v] 时，sumt[v] += sumt[u];

判断是否存在无数条最短路，即看是否存在这样的一条边(u, v)，边权为 0，并且其中一条最短路经过这条边，也就是 源点 s 到 u 的最短距离 + v 到终点 t 的最短距离 == 最短路长度，因为边权为 0 的话就可以来回无限次地走。所以需要两次最短路分别计算出源点 s 到每个点的最短路、每个点到终点 t 的最短路，然后枚举每条边，即可判断是否存在无数条最短路

2、有向图最短路计数

方法一：Floyd 算法

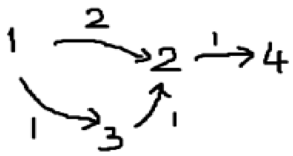
方法二：dijstra 算法

做法与无向图计数一致。参考代码：

```
void dijkstra(int S){
    int i,j,k,maxx;
    for(i=0;i<=n;i++)
        d[i]=INF;
    d[S]=0;
    g[S]=1;
    for(i=1;i<=n;i++){
        maxx=INF;k=0;
        for(j=1;j<=n;j++)
            if(!vst[j]&&d[j]<maxx){maxx=d[j];k=j;}
        if(d[k]==INF)return;
        vst[k]=1;
        for(j=1;j<=n;j++)
            if(!vst[j]&&a[k][j]!=INF){
                if(d[j]==d[k]+a[k][j]) g[j]+=g[k];
                if(d[j]>d[k]+a[k][j]) d[j]=d[k]+a[k][j],g[j]=g[k];
        }
    }
}
```

方法三：**SPFA**

Spfa 不能像 dijkstra 那样直接计算，需要做一定调整。先看一个例子



答案是：最短路=3，路径数是 2；　　如果 spfa 直接算，路径数会得到 3。

按 spfa 跑队列进队顺序是 1 2 3 4 2 4，队列中的两个 2 含义不同，靠前的 2 是 1 加入的，靠后的 2 是 3 加入的，同样也有两个 4.

但是在队列的表现完全不一样，队列中前面的 2 对于的路径条数是 1，后面 2 路径条数也应该是 1，但如果直接用 G[2]，第二个 2 相当于用 2 在计算路径数，这是错误的

所以不能笼统的 g 数组来统计。设定数组 dlt[i]表示点 i 在队列中累加的路径数

```
int dlt[2010];
void Spfa(){
    int i,j;
    for(i=1;i<=n;i++){d[i]=INF;vst[i]=0;g[i]=0;}//初始化
    d[1]=0;g[1]=1;dlt[1]=1;
    queue<int>q;
    q.push(1);
    while(!q.empty()){
        i=q.front();q.pop();
        vst[i]=0;
        for(j=1;j<=n;j++)
            if(a[i][j]!=INF&&d[i]+a[i][j]<=d[j]){
                if(d[i]+a[i][j]==d[j]){g[j]+=dlt[i];dlt[j]+=dlt[i];}
                if(d[i]+a[i][j]<d[j]){
                    d[j]=d[i]+a[i][j];
                    g[j]=dlt[i];dlt[j]=dlt[i];
                }
                if(!vst[j]){q.push(j);vst[j]=1;}
            }
        dlt[i]=0; //注意此处必须清零
    }
}
```

3、最短路计数应用

应用 1：求 S-T 经过某点 A 的路径数

算法：

①读入数据，建立图；

②分别从 S 和 A 两点求单源最短路径；

③若 S 到 T 的最短路径=S 到 A 的最短路径+A 到 T 的最短路径，则

ANS=g[S][A]*g[A][T]。

三、 分层图

分层图思想：是根据问题性质，根据干扰因素的不同类型，将原图复制成若干层并连接的更大的图，用形象的图论模型来理解抽象的模型。最后的图可以是抽象的，也可以是具体的。

这种思想是一种“**升维**”策略，通过分层放大了目标模型，使得问题分类后简化，从而找到解决途径

实例：改造路 P3067

题意概述：给定一张无向图，可以将其中 k 条边的权值改为 0，求 1 到 n 的最短路。(k<=20)

【方法一】

分析发现 k 比较小，我们可以考虑对原图进行拆点，将一个点强行拆成 k 个，这样相当于复制了 k 层，每一层的图与原图一致。然后考虑层与层之间的连边。

从第 i 层到第 i+1 层的边边权全为 0，这相当于从第 i 层到 i+1 层就是用掉了一次免费卡。最后在新图中跑一次堆优化 dijktra。

建图参考代码：

```
int main(){
    scanf("%d%d%d",&n,&m,&k);
    for (int i=1;i<=m;++i) {
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);//第0层
        add(b,a,c);
        for (int j=1;j<=k;++j) {//1~k层
            add(j*n+a,j*n+b,c);//复制原图信息
            add(j*n+b,j*n+a,c);
            add((j-1)*n+a,j*n+b,0);//第i层到第i+1层
            add((j-1)*n+b,j*n+a,0);
        }
    }
    s=1,t=n;
    dis();
    int ans=d[t];
    for (int i=0;i<=k;++i)
        ans=min(ans,d[i*n+t]);
    cout<<ans;
    return 0;
}
```

【方法二】这道题也可以考虑 DP 的思想，即将原来最短路的 dist 数组定义为二维：dist[i][j]表示到点 i 用了 j 次卡的最短路，然后最短路算法变形。这个做法也可以用分层图思想来理解，不过没有建立具体的分层图（上道题就是这样）

```
using namespace std;
const int N=10001,M=50001,K=21;
int n,m,k,i,j,next[2*M],to[2*M],head[N],w[2*M],d[N][K];
bool v[N][K];
struct node{
    int i,j,d;
}t;
bool operator<(const node&a,const node&b){
    return a.d>b.d;
}
priority_queue<node>q;
int main(){
    scanf("%d%d%d",&n,&m,&k);
    for(i=1;i<=m;i++)    {
        int u,v;
        scanf("%d%d%d",&u,&v,w+2*i);
        next[2*i-1]=head[u];head[u]=2*i-1;to[2*i-1]=v;w[2*i-1]=w[2*i];
        next[2*i]=head[v];head[v]=2*i;to[2*i]=u;
    }
    for(i=1;i<=n;i++)
        for(j=0;j<=k;j++)
            d[i][j]=1<<30;
    d[1][0]=0;
    q.push((node){1,0,0});
    while(!q.empty())   {
        t=q.top();
        q.pop();
        if(v[t.i][t.j])
            continue;
        v[t.i][t.j]=true;
        for(i=head[t.i];i;i=next[i]){
            if(t.j+1<=k&&d[to[i]][t.j+1]>t.d)    {
                d[to[i]][t.j+1]=t.d;
                q.push((node){to[i],t.j+1,t.d});
            }
            if(d[to[i]][t.j]>t.d+w[i]){
                d[to[i]][t.j]=t.d+w[i];
                q.push((node){to[i],t.j,d[to[i]][t.j]});
            }
        }
    }
    printf("%d\n",d[n][k]);
    return 0;
}
```

实例 2：孤岛营救问题

题意：一个 M*N 各图，有墙有门，从起点到终点最短路

【解析】本题有两个做法：1、状压+bfs 完成；2、分层图完成

【分层分析】

考虑有门和钥匙因素，主要是钥匙对门的影响，将图分为 2^p 层对应钥匙的 2^p 个状态

本层：根据钥匙状态改造每层节点，相邻连通点有长度 1 的边
层与层之间：对于存在钥匙的点，应该向得到钥匙后钥匙状态层 对应的点连一条长 0 的边（即层与层之间通过钥匙连接）

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int ,int>pii;
#define mp make_pair
const int inf=0x3f3f3f3f;
const int maxn = 1e5+10;
const int maxm = 2e6+10;
#define ll long long
#define in read()
inline int read(){
    int x=0,f=1;
    char ch=getchar();
    for(;ch<'0'||ch>'9';ch=getchar())if(ch=='-')f=-1;
    for(;ch>='0'&&ch<='9';ch=getchar())x=(x<<1)+(x<<3)+ch-'0';
    return x*f;
}
struct Edge{
    int v,w,nxt;
}e[maxm];
int first[maxn],cnt=0;
inline void add(int u,int v ,int w){
    e[++cnt].v=v;e[cnt].w=w;e[cnt].nxt=first[u];first[u]=cnt;
}
vector <pii >key[20];
int n,m,layer,M,N,keyn,r,g[1001][1001],dis[maxn],vis[maxn];
inline int num(int i,int j){
    return (i-1)*n+j;
}

void build(){
    int vis[15]={};
    for(int k=0;k<layer;k++){
        for(int p=1;p<=keyn;p++){
            vis[p]=k&(1<<p-1);//记录当前层是否有钥匙 p
            for(int i=1;i<=m;i++)// 当前层建图
                for(int j=1;j<=n;j++){
                    int x=num(i,j),y=num(i,j+1);//向右连边
                    if(j<n && g[x][y]!=-1)
                        if(g[x][y]==0 || vis[g[x][y]])
                            add(k*M+x,k*M+y,1),add(k*M+y,k*M+x,1);
                    y=num(i+1,j);//向下连边
```

```
                if(i<m && g[x][y]!=-1)
                    if(g[x][y]==0 || vis[g[x][y]])
                            add(k*M+x,k*M+y,1),add(k*M+y,k*M+x,1);
            }
        }
        for(int i=1;i<=keyn;i++)
            if(!vis[i]){//当前层没有这把钥匙才转移
                int t=k+(1<<i-1);//向有有钥匙层转移
                for(int j=0;j<key[i].size();j++){
                    int x=num(key[i][j].first,key[i][j].second);
                        add(k*M+x,t*M+x,0);

                }
            }
    }
}
void spfa(){
    memset(vis,0,sizeof(vis));memset(dis,0x3f,sizeof(dis));
    queue<int>q;
    q.push(1);dis[1]=0;
    while(!q.empty()){
        int u=q.front();q.pop();
        vis[u]=0;
        for(int i=first[u];i;i=e[i].nxt){
            int v=e[i].v,w=e[i].w;
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                if(!vis[v])q.push(v),vis[v]=1;
            }
        }
    }
}
signed main(){
    m=in;n=in;keyn=in;r=in;
    M=m*n;layer=(1<<keyn);N=M*layer;//M 每一层的数量,layer 有多少层,N 总节点数
    for(int i=1;i<=r;i++){
        int x1=in,y1=in,x2=in,y2=in,t=in;
        int u=num(x1,y1),v=num(x2,y2);
        if(t==0)t=-1;
        g[u][v]=g[v][u]=t;//u->v =-1 表墙，=0 无门>0 有门
    }
    r=in;
    for(int i=1;i<=r;i++){
        int u=in,v=in,p=in;
```

```
        key[p].push_back(mp(u,v));
    }

    build();
    int ans=inf;
    spfa();
    int T=num(m,n);
    for(int i=0;i<layer;i++)ans=min(ans,dis[i*M+T]);
    if(ans<inf)cout<<ans<<"\n";else cout<<-1<<"\n";
    return 0;
}
```

四、　差分约束

引例：给定 n 个变量和 m 个不等式，求 x[n-1] - x[0] 的最大值

$$x1 - x0 <= 2 \qquad (1)$$

$$x2 - x0 <= 7 \qquad (2)$$

$$x3 - x0 <= 8 \qquad (3)$$

$$x2 - x1 <= 3 \qquad (4)$$

$$x3 - x2 <= 2 \qquad (5)$$

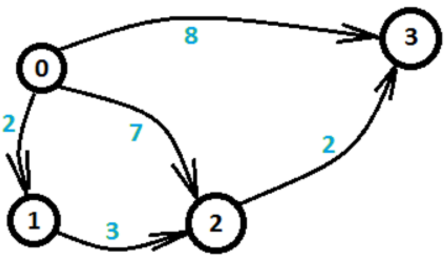
【分析】观察 x3 - x0 的性质，我们如果可以通过不等式的两两加和得到 c 个形如 x3 - x0 <= Ti 的不等式，那么 min{ Ti | 0 <= i < c } 就是我们要求的 x3 - x0 的最大值。

于是费尽千辛万苦，终于整理出以下三个不等式：

1.　(3)　　　　　　　　x3 - x0 <= 8
2.　(2) + (5)　　　　　x3 - x0 <= 9
3.　(1) + (4) + (5)　　x3 - x0 <= 7

这里的 T 等于{8, 9, 7}，所以 min{ T } = 7，答案就是 7。的确是 7 吗？我们再仔细看看，发现的确没有其它情况了。那么问题就是这种方法即使做出来了还是带有问号的，不能确定正确与否，如何**系统地解决**这类问题呢？

我们解析来看另外一个问题：下面图中，第 0 个岛到第 3 个岛的最短距离？



分析有三条路线，如下：

1.　0 -> 3　　　　　　长度为 8
2.　0 -> 2 -> 3　　　 长度为 7+2 = 9
3.　0 -> 1 -> 2 -> 3　长度为 2 + 3 + 2 = 7

是不是和上面的不等式约数问题一样？

这就是今天要说的差分约束类问题。

差分约束系统：如若一个系统由 n 个变量和 m 个不等式组成，并且这 m 个不等式对应的系数矩阵

中每一行有且仅有一个 1 和-1，其它的都为 0，这样的系统称为差分约束(difference constraints)系统。引例中的不等式组可以表示成如图的系数矩阵。

$$\begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{pmatrix} x0 \\ x1 \\ x2 \\ x3 \end{pmatrix} \leqslant \begin{pmatrix} 2 \\ 7 \\ 8 \\ 3 \\ 2 \end{pmatrix}$$

对于这样的系统如何高效求解呢？

根据刚才的两个例子对比，再观察不等式。观察 x[i] - x[j] <= a[k]， 将这个不等式稍稍变形，将 x[j]移到不等式右边，则有 x[i] <= x[j] + a[k]，然后我们令 a[k] = w(j, i)，再将不等式中的 i 和 j 变量替换掉，i = v， j = u，将 x 数组的名字改成 d（以上都是等价变换，不会改变原有不等式的性质），则原先的不等式变成了以下形式：

d[u] + w(u, v) >= d[v]。

这时候联想到 SPFA 中的一个松弛操作：

```
if(d[u] + w(u, v) < d[v]) {
        d[v] = d[u] + w(u, v);
 }
```

于是可以得出一个**结论**：

对于每个不等式 x[i] - x[j] <= a[k]，对结点 j 和 i 建立一条 j -> i 的有向边，边权为 a[k]，求 x[n-1] - x[0] 的最大值就是求 0 到 n-1 的最短路。

当然，问题有可能不一定有解，这个可以根据建的图来判定：

情况 1：图存在最短路则有解。

情况 2：图存在负环，则无解。路径中出现负权圈，则表示最短路无限小，即不存在最短路，那么在不等式上的表现即 X[t] - X[s] <= T 中的 T 无限小，得出的结论就是 X[t] - X[s]的最大值 不存在。

情况 3：即从起点 s 无法到达 t 则有无数解。 表明 X[t]和 X[s]之间并没有约束关系，这种情况下 X[t] - X[s]的最大值是无限大，这就表明了 X[t]和 X[s]的取值有无限多种。

实例 1：奶牛的站位 P3043

【解析】

1、建图

X,y 是朋友距离不超过 c，则:add(x,y,c)

X,y 是敌人距离不少于 c,则：add(x,y,-c)

2、判无解：是否有负环

3、无穷解：没有通路

4、最大解：最短路

【参考代码】

```
#include<bits/stdc++.h>
#define N 20009
using namespace std;
int n,f,e,tot=0;
int nxt[N],head[N],to[N],w[N],dis[N];
void add(int x,int y,int z){
```

```
    nxt[++tot]=head[x];head[x]=tot;to[tot]=y;w[tot]=z;
}
int cnt[N];
bool vis[N],flag=0;
queue<int > q;
void spfa(int s){
    for(int i=1;i<=n;++i)
        vis[i]=0,dis[i]=0x3f3f3f3f;
    dis[s]=0;
    q.push(s);
    vis[s]=1;cnt[s]++;
    while(!q.empty()){
        int u=q.front();q.pop();
        vis[u]=0;
        for(int i=head[u];i;i=nxt[i]){
            int v=to[i];
            if(dis[v]>dis[u]+w[i]){
                dis[v]=dis[u]+w[i];
                if(++cnt[v]>=n) {
                    flag=1; return;
                }
                if(!vis[v]){
                    q.push(v);
                    vis[v]=1;
                }
            }
        }
    }
}
int main(){
    int a,b,c,x,y,z,i,j;
    scanf("%d%d%d",&n,&f,&e);
    for(i=1;i<=f;++i){
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
    }
    for(i=1;i<=e;++i){
        scanf("%d%d%d",&a,&b,&c);
        add(b,a,-c);
    }
    spfa(1);
    if(flag)
        printf("-1\n");
    else{
```

```
        if(dis[n]<0x3f3f3f3f) printf("%d",dis[n]);
        else printf("-2");
    }
    return 0;
}
```

拓展 1：求差值最小值。

我们将问题进行一个简单的转化，将原先的"<="变成">="，转化后的不等式如下：

B - A >= c (1)

C - B >= a (2)

C - A >= b (3)

然后求 C - A 的最小值，类比之前的方法，需要求的其实是 max{b, c+a}，于是对应的是图三从 A 到 C 的最长路。

同样可以推广到 n 个变量 m 个不等式的情况。

拓展 2：不等式标准化

如果给出的不等式有"<="也有">="，又该如何解决呢？很明显，首先需要关注最后的问题是什么，如果需要求的是两个变量差的最大值，那么需要将所有不等式转变成"<="的形式，建图后求最短路；相反，如果需要求的是两个变量差的最小值，那么需要将所有不等式转化成">="，建图后求最长路。

如果有形如：A - B = c 这样的等式呢？我们可以将它转化成以下两个不等式：

A - B >= c (1)

A - B <= c (2)

再通过上面的方法将其中一种不等号反向，建图即可。

最后，如果这些变量都是整数域上的，那么遇到 A - B < c 这样的不带等号的不等式，我们需要将它转化成"<="或者">="的形式，即 A - B <= c - 1。