最短路入门

1.Dijkstra

单源最短路算法，最常用时间复杂度（n^2）优化后可以达到（nlogn），不能解决负边问题，稀疏图（点的范围很大但是边不多，边的条数|E|远小于|V|²）需要耗费比较多的空间。

```
#include <bits/stdc++.h>

#define INF 0x3f3f3f3f
#define maxn 105
typedef long long ll;
using namespace std;

int n,m;
int maps[maxn][maxn];//储存路径花费

void dijkstra(int sta,int ed)
{
        int v,vis[maxn];//v为距离当前点最近的下一个点，vis为标记数组
        int d[maxn];//储存起点到每一点的最少路径花费
        for(int i=1;i<=n;i++)   {
                d[i]=maps[sta][i];//将路径花费初始化
        }
        memset(vis,0,sizeof(vis));//标记数组清零
        vis[sta]=1;//标记起点
        for(int i=1;i<=n;i++)   {
                if(vis[ed]) break;//已到达起点退出循环
                int Min=INF;//INF为最大值
                for(int j=1;j<=n;j++)   {
                        if(d[j]<Min&&!vis[j]) Min=d[v=j];//寻找下一个可到达的最小花费点
                }
                vis[v]=1;//标记已访问;
                for(int j=1;j<=n;j++)   {
                        if(!vis[j]&&d[j]>d[v]+maps[v][j])//更新最少花费
                                d[j]=d[v]+maps[v][j];
                }
        }
        printf("%d\n",d[ed]);
}

int main()
{
        while(scanf("%d%d",&n,&m)!=EOF&&n&&m)
        {
                for(int i=1;i<=n;i++) {
                        for(int j=1;j<=n;j++)
                        {
                                maps[i][j]= i==j?0:INF;//初始化路径花费，INF为不可到达即无边
                        }
```

```
                }
                for(int i=1;i<=m;i++)   {
                        int v,u,c;
                        scanf("%d%d%d",&u,&v,&c);
                        if(maps[u][v]>c) maps[v][u]=maps[u][v]=c;//实时更新，只保留最小花费;
                }
                dijkstra(1,n);
        }
        return 0;
}
```

2.Floyd

复杂度最高（n^3），通常用在点比较少的起点不固定的问题中。能解决负边（负权）但不能解决负环。

```
#include <bits/stdc++.h>

#define INF 0x3f3f3f3f
#define maxn 105
typedef long long ll;
using namespace std;

int n,m;
int maps[maxn][maxn];//储存路径花费

int main()
{
        while(scanf("%d%d",&n,&m)!=EOF&&n&&m)
        {
                for(int i=1;i<=n;i++)   {
                        for(int j=1;j<=n;j++)   {
                                maps[i][j]= i==j?0:INF;//初始化路径花费，INF表示无边
                        }
                }
                for(int i=1;i<=m;i++)   {
                        int u,v,c;
                        scanf("%d%d%d",&u,&v,&c);
                        if(maps[u][v]>c) maps[u][v]=maps[v][u]=c;//实时更新路径最小花费
                }
                for(int k=1;k<=n;k++)   {
                        for(int i=1;i<=n;i++)   {
                                for(int j=1;j<=n;j++)   {
                                        maps[i][j]=min(maps[i][j],maps[i][k]+maps[k][j]);//
                                }
                        }
                }
                printf("%d\n",maps[1][n]);//输出起点至终点的路线花费
```

```
        }
        return 0;
}
```

3.Bellman_Ford
适合稀疏图，可以解决带有负权边，负环的问题，但是在稠密图中效率比Dijkstra要低。

```
#include <bits/stdc++.h>

#define INF 0x3f3f3f3f
#define maxn 105
typedef long long ll;
using namespace std;

int n,m;
struct node
{
        int u,v;//边的起点和终点
        int cost;//路径花费
}edge[maxn*maxn/2];
int dis[maxn];//记录起点到各点的最小花费

int Bellman_Ford(int sta,int ed)
{
        for(int i=1;i<=n;i++)   {
                dis[i]=INF;
        }
        dis[sta]=0;
        for(int i=0;i<n-1;i++)  {
                for(int j=0;j<m*2;j++)  {
                        if(dis[edge[j].v]>dis[edge[j].u]+edge[j].cost)
                                dis[edge[j].v]=dis[edge[j].u]+edge[j].cost;
                }
        }
        return dis[ed];
}

int main()
{
        int start,ends,cost;
        int answer;
        while(scanf("%d%d",&n,&m)!=EOF&&n&&m)
        {
                for(int i=0;i<m;i++)    {
                        scanf("%d%d%d",&start,&ends,&cost);
                        edge[i*2].u=start,edge[i*2].v=ends,edge[i*2].cost=cost;
```

```
                edge[i*2+1].u=ends,edge[i*2+1].v=start,edge[i*2+1].cost=cost;//建边
            }
            answer=Bellman_Ford(1,n);
            printf("%d\n",answer);
        }
        return 0;
}
```