最短路：

1、最短路

1.1 Dijkstra单源最短路,邻接矩阵形式,权值必须是非负

*/\**

*\** 单源最短路径，*Dijkstra*算法，邻接矩阵形式，复杂度为*O(n^2)*

*\** 求出源*s*到所有点的最短路径，传入图的顶点数，和邻接矩阵*cost[][]*

*\** 返回各点的最短路径*d[],* 路径*pre[].pre[i]*记录*s*到*i*路径上的父结点，*pre[s]=-1*

*\** 可更改路径权类型，但是权值必须为非负

*\*/*

**const int** N=1010;

**const int** INF=0x3f3f3f3f;*//*防止后面溢出，这个不能太大

**bool** vis[N];

**int** pre[N];

**void** Dijkstra(**int** cost[][N],**int** d[],**int** n,**int** s)

{

**for**(**int** i=0;i<n;i++)

{

d[i]=INF;vis[i]=0;pre[i]=-1;

}

d[s]=0;

**for**(**int** j=0;j<n;j++)

{

**int** k=-1;

**int** Min=INF;

**for**(**int** i=0;i<n;i++)

**if**(!vis[i]&&d[i]<Min)

{

Min=d[i];

k=i;

}

**if**(k==-1)**break**;

vis[k]=1;

**for**(**int** i=0;i<n;i++)

**if**(!vis[i]&&d[k]+cost[k][i]<d[i])

{

d[i]=d[k]+cost[k][i];

pre[i]=k;

}

}

}

1.2 Dijkstar算法+堆优化

使用优先队列优化,复杂度O (E\*logE)

*/\**

*\** 使用优先队列优化*Dijkstra*算法

*\** 复杂度*O(ElogE)*

*\** 注意对*vector<e>E[N]*进行初始化后加边

*\*/*

**const int** INF=0x3f3f3f3f;

**const int** N=1000010;

**struct** P

{

**int** v;

**int** c;

P(**int** \_v=0,**int** \_c=0):v(\_v),c(\_c){}

**bool operator** <(**const** P &r)**const**

{

**return** c>r.c;

}

};

**struct** e

{

**int** v,cost;

e(**int** \_v=0,**int** \_cost=0):v(\_v),cost(\_cost){}

};

vector<e>E[N];

**bool** vis[N];

**int** d[N];

**void** Dijkstra(**int** n,**int** s)*//*点的编号从*1*开始

{

memset(vis,0,**sizeof**(vis));

**for**(**int** i=1;i<=n;i++)d[i]=INF;

priority\_queue<P>que;

**while**(!que.empty())que.pop();

d[s]=0;

que.push(P(s,0));

P tmp;

**while**(!que.empty())

{

tmp=que.top();

que.pop();

**int** u=tmp.v;

**if**(vis[u])**continue**;

vis[u]=1;

**for**(**int** i=0;i<E[u].size();i++)

{

**int** v=E[tmp.v][i].v;

**int** cost=E[u][i].cost;

**if**(!vis[v]&&d[v]>d[u]+cost)

{

d[v]=d[u]+cost;

que.push(P(v,d[v]));

}

}

}

}

**void** adde(**int** u,**int** v,**int** w)

{

E[u].push\_back(e(v,w));

}

1.3 单源最短路 bellman\_ford算法

*/\**

*\** 单源最短路*bellman\_ford*算法，复杂度*O(VE)*

*\** 可以处理负边权图。

*\** 可以判断是否存在负环回路。返回*1,*当且仅当图中不包含从源点可达的负权回路

*\* vector<e>E;*先*E.clear()*初始化，然后加入所有边

*\** 点的编号从*1*开始*(*从*0*开始简单修改就可以了*)*

*\*/*

**const int** INF=0x3f3f3f3f;

**const int** N=550;

**int** d[N];

**struct** e

{

**int** u,v;

**int** cost;

e(**int** \_u=0,**int** \_v=0,**int** \_cost=0):u(\_u),v(\_v),cost(\_cost){}

};

vector<e>E;

**bool** bellman\_ford(**int** s,**int** n)*//*点的编号从*1*开始

{

**for**(**int** i=1;i<=n;i++)d[i]=INF;

d[s]=0;

**for**(**int** i=1;i<n;i++)*//*最多做*n-1*次

{

**bool** ok=0;

**for**(**int** j=0;j<E.size();j++)

{

**int** u=E[j].u;

**int** v=E[j].v;

**int** cost=E[j].cost;

**if**(d[v]>d[u]+cost)

{

d[v]=d[u]+cost;

ok=1;

}

}

**if**(!ok) **return** 1;*//*没有负环回路

}

**for**(**int** j=0;j<E.size();j++)

**if**(d[E[j].v]>d[E[j].u]+E[j].cost)

**return** 0;*//*有负环回路

**return** 1;*//*没有负环回路

}

1.4 单源最短路SPFA

*/\**

*\** 单源最短路*SPFA*

*\** 时间复杂度 *0(kE)*

*\** 这个是队列实现，有时候改成栈实现会更加快，很容易修改

*\** 这个复杂度是不定的

*\*/*

**const int** N=1010;

**const int** INF=0x3f3f3f3f;

**struct** e

{

**int** v;

**int** cost;

e(**int** \_v=0,**int** \_cost=0):v(\_v),cost(\_cost){}

};

vector<e>E[N];

**void** adde(**int** u,**int** v,**int** w)

{

E[u].push\_back(e(v,w));

}

**bool** vis[N];*//*在队列标志

**int** cnt[N];*//*每个点的入队列次数

**int** d[N];

**bool** SPFA(**int** s,**int** n)

{

memset(vis,0,**sizeof**(vis));

**for**(**int** i=1;i<=n;i++)d[i]=INF;

vis[s]=1;

d[s]=0;

queue<**int**>que;

**while**(!que.empty())que.pop();

que.push(s);

memset(cnt,0,**sizeof**(cnt));

cnt[s]=1;

**while**(!que.empty())

{

**int** u=que.front();

que.pop();

vis[u]=0;

**for**(**int** i=0;i<E[u].size();i++)

{

**int** v=E[u][i].v;

**if**(d[v]>d[u]+E[u][i].cost)

{

d[v]=d[u]+E[u][i].cost;

**if**(!vis[v])

{

vis[v]=1;

que.push(v);

**if**(++cnt[v]>n)**return** 0;

*//cnt[i]*为入队列次数，用来判定是否存在负环回路

}

}

}

}

**return** 1;

}

HDU 2544

每组数据第一行是两个整数n、m（n<=100，m<=10000），n表示大街上有几个地点，标号为1的为起始位置，标号为n的路口是目的地，m则表示共有m条路。每条路包括3个参数a，b，c（1<=a,b<=n,1<=c<=1000）,表明a与b的距离为c，求1到n的最短距离。

Dijkstra：

#include<stdio.h>

#include<algorithm>

#include<string.h>

#include<math.h>

#include<queue>

#include<map>

#include<set>

#include<vector>

#define ll long long

#define db double

const int INF=100000000;

const int N=1001;

using namespace std;

bool vis[N];

int Map[N][N],d[N],pre[N];//pre[]记录前驱

int n, m;

void dijkstra(int v)

{

int i, j, u=v , min;

for(i=0;i<=n;i++)

{

d[i]=Map[v][i];

vis[i]=0;

//if(i!=v&&adj[v][i]!=INF) pre[i] = v;

// else pre[i] = -1;

}

vis[v]=1;d[v]=0;

for(i=1;i<n;i++)

{

min = INF;

for(j=1;j<=n;j++)

{

if(!vis[j]&&min > d[j])

{

min = d[j];

u = j;

}

}

if(min == INF) break;

vis[u]=1;

for(j=1;j<=n;j++)

{

if(!vis[j]&&Map[u][j]!=INF&&d[u]+Map[u][j]<d[j])

{

d[j] = Map[u][j] + d[u];

// pre[j] = u;

}

}

}

}

int main()

{

int i, j, x, y, w;

while(~scanf("%d%d",&n,&m)&&n)

{

for(i=0;i<=n;i++)

{

for(j=0;j<=n;j++)

if(i==j)Map[i][j]=0;

else Map[i][j] = INF;

}

while(m--)

{

scanf("%d%d%d",&x,&y,&w);

Map[x][y] = Map[y][x] = w;

}

dijkstra(0);

printf(“%d\n",d[n]);

//以下为输出路径

/\*———————————————

int p, len=0, ans[N];

p = n-1;

while(p!=0)

{

ans[len++] = p;

p = pre[p];

}

for(i=len-1;i>=0;i--)

printf(“%d->",ans[i]);

————————————————-\*/

}

return 0;

}

Floyd：

void F(){

int i, j, k;

for(k = 1; k <= n; k ++)

for(i = 1; i <= n; i ++)

for(j = 1; j <= n; j ++)

d[i][j] = min(d[i][j], d[i][k]+d[k][j]);

printf("%d\n", d[1][n]);

}