/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Dijkstra(heap)

O(E \* logV)

无负权边时可用

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <algorithm>

#include <string.h>

#include <queue>

using namespace std;

const int MAX = 1e5 + 5;

const int infi = 0x3f3f3f3f;

//所谓链式前向星（？）

struct edge{

int v, w; // v为指向点， w为边权

int next;

edge(int v\_ = 0, int w\_ = 0, int n\_ = -1) : v(v\_), w(w\_), next(n\_) {}

};

int n, cnt;

edge e[MAX]; // 无向图翻倍

int head[MAX];

int dis[MAX]; // 最短距离

int pre[MAX]; //最短路的上一个点

int vis[MAX]; //visited?

void init()

{

cnt = 0;

memset(head, -1, sizeof head);

}

void addedge(int u, int v, int w) // 有向图: form u to v, cost w

{

e[cnt] = edge(v, w, head[u]); head[u] = cnt++;

/\* 无向图只需要添加注释部分 \*/

/\*

e[cnt] = edge(u, w, head[v]); head[v] = cnt++;

\*/

}

struct node{

int p, w;

node(int p\_ = 0, int w\_ = 0) : p(p\_), w(w\_) {}

bool operator< (const node b)const

{

return w > b.w; ~~//鬼畜的优先队列~~

}

};

bool dijkstra(int s, int t) // s为起点， t为终点

{

priority\_queue<node> q;

memset(dis, infi, sizeof dis);

memset(pre, -1, sizeof pre);

memset(vis, 0, sizeof vis);

dis[s] = 0;

q.push(node(s, 0));

while (!q.empty())

{

int u = q.top().p;

q.pop();

if (vis[u])

continue;

vis[u] = 1;

for (int i = head[u]; i != -1; i = e[i].next)// 遍历u点所连边~~(鬼知道在说什么)~~

{

int v = e[i].v, w = e[i].w; // 非必需，习惯。。

if (!vis[v] && dis[v] > dis[u] + w)

{

dis[v] = dis[u] + w;

pre[v] = u;

q.push(node(v, dis[v]));

}

}

}

return pre[t] != -1; //是否能到达终点t

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPFA~~(Super Party Festival Algorithm)~~

O(kE) //常数成迷，易被卡

记录每个点进队次数，超过|V|时存在负环

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

#include <algorithm>

#include <queue>

using namespace std;

const int MAX = 1e5 + 5;

const int infi = 0x3f3f3f3f;

//链式前向星。

struct edge{

int v, w;

int next;

edge(int v\_ = 0, int w\_ = 0, int n\_ = -1) : v(v\_), w(w\_), next(n\_) {}

};

int n, cnt;

edge e[MAX]; //无向图需翻倍

int head[MAX];

int pre[MAX];

int inq[MAX]; // in queue or not

int dis[MAX]; //最短距离

void init()

{

cnt = 0;

memset(head, -1, sizeof head);

}

void addedge(int u, int v, int w) // Directed: from u to v, cost w

{

e[cnt] = edge(v, w, head[u]); head[u] = cnt++;

/\* if undirected \*/

/\*

e[cnt] = edge(u, w, head[v]); head[v] = cnt++;

\*/

}

bool spfa(int s, int t) // s 起点， t 终点

{

queue<int> q;

memset(dis, infi, sizeof dis);

memset(inq, 0, sizeof inq);

memset(pre, -1, sizeof pre);

q.push(s); inq[s] = 1; dis[s] = 0;

while (!q.empty())

{

int u = q.front();

q.pop(); inq[u] = 0;

for (int i = head[u]; i != -1; i = e[i].next)

{

int v = e[i].v, w = e[i].w;

if (dis[v] > dis[u] + w)

{

dis[v] = dis[u] + w;

pre[v] = u;

if (!inq[v])

q.push(v), inq[v] = 1;

}

}

}

return pre[t] != -1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

floyd

O(n3)

多源最短路

dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]) (1 <= k <= n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define min(a, b) ((a) < (b) ? (a) : (b))

const int MAX = 1005;

int dis[MAX][MAX]; //初始邻接矩阵

void floyd()

{

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

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

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

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

}