学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 王一博 | **学号** | 2021902655 |
| **学院** | 信息工程学院 | **专业** | 计算机科学与技术 |

（请在下面表格本周学习情况）

|  |
| --- |
| 学习情况简述 |
| 根据网站ACWing和书籍《大话数据结构》进行学习，学习内容包括：深度优先搜索（DFS），宽度优先搜索（BFS），拓扑排序，Dijkstra算法，Bellman-ford算法，SPFA算法等 |
| 本周练习过的代码（例） |
| （本周练习写过的代码，如实填写，疑似抄袭会单独测试）  *代码1：*  *#include<stdio.h>*  *int main()*  *{*  *printf("HelloWorld\n");*  *return 0;*  *}*  *代码2：*  …….. |
| 本周练习过的代码 |

代码1：

//Dijkstra算法中处理的图一般比较稠密，图用邻接矩阵存比较好(二维数组)

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int N = 510;

int g[N][N]; //g[a][b]中存的是边a，b之间的距离

int dist[N]; //存起点(1号点)到该点的距离

bool st[N]; //记录是否找到了起点到该点的当前最短距离

int n, m;

int dijkstra() // 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1

{

memset(dist, 0x3f, sizeof dist); //将所有到起点的距离设为无穷大

dist[1] = 0;

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

{

int t = -1; //t存尚未确定的点中，离起点距离最小的点

for (int j = 1; j <= n; j ++ ) //遍历所有点，寻找未确定的点中离起点最近的(t)

{

if (!st[j] && (t == -1 || dist[t] > dist[j] ))

t = j;

}

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

dist[j] = min(dist[j], dist[t] + g[t][j]);

st[t] = true;

}

if (dist[n] == 0x3f3f3f3f) return -1;

return dist[n];

}

int main()

{

memset(g, 0x3f, sizeof g);

scanf("%d%d", &n, &m);

while (m -- )

{

int a, b, c;

scanf("%d%d%d", &a, &b, &c);

g[a][b] = min(c, g[a][b]);

}

cout << dijkstra();

}

代码2：

#include <iostream>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

typedef pair<int, int> PII; // <距离起点的长度，节点编号>

const int N = 1.5e5 + 10;

int n, m;

int e[N], w[N], ne[N], h[N], idx;

int dist[N];

bool st[N];

void add(int a, int b, int c)

{

e[idx] = b;

w[idx] = c;

ne[idx] = h[a];

h[a] = idx ++;

}

int dijkstra()

{

memset(dist, 0x3f, sizeof dist);

dist[1] = 0;

priority\_queue <PII, vector<PII>, greater<PII>> heap; //用STL构建小根堆（根是最小的）

heap.push({0,1});

while (heap.size())

{

auto t = heap.top();

heap.pop();

int dis = t.first, ver = t.second; //分别取出堆顶元素的距离和编号

if(st[ver]) continue; //若该点已经访问过则跳过

st[ver] = true;

for(int i = h[ver]; i != -1; i = ne[i])

{

int j = e[i];

if(dist[j] > dist[ver] + w[i])

{

dist[j] = dist[ver] + w[i]; //dist[ver]也可换成dis

heap.push({dist[j], j});

}

}

}

if(dist[n] == 0x3f3f3f3f) return -1;

return dist[n];

}

int main()

{

scanf("%d%d", &n, &m);

memset(h, -1, sizeof h);

while(m --)

{

int a, b, c;

scanf("%d%d%d", &a, &b, &c);

add(a, b, c);

}

cout << dijkstra();

}

代码3：

#include <iostream>

#include <cstring>

using namespace std;

const int N = 510, M = 1e5 + 10;

struct Edge

{

int a, b, w;

}edges[M];

int n, m, k;

int dist[N];

int back[N];

void bellman\_ford()

{

memset(dist, 0x3f, sizeof dist);

dist[1] = 0;

while(k --)

{

memcpy(back, dist, sizeof dist); //把dist数组copy到back数组中

for(int j = 0; j < m; j ++)

{

auto e = edges[j];

dist[e.b] = min(dist[e.b], back[e.a] + e.w);

}

}

}

int main()

{

scanf("%d%d%d", &n, &m, &k);

for (int i = 0; i < m; i ++ )

{

int a, b, w;

scanf("%d%d%d", &a, &b, &w);

edges[i] = {a, b, w};

}

bellman\_ford();

if (dist[n] > 0x3f3f3f3f / 2) puts("impossible");

else cout << dist[n];

}

代码4：

#include <iostream>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

const int N = 1e5 + 10;

int e[N], ne[N], h[N], w[N], idx;

int dist[N];

bool st[N]; //判断当前的点是否已经在队列中

int n, m;

void add(int a, int b, int c)

{

e[idx] = b;

w[idx] = c;

ne[idx] = h[a];

h[a] = idx ++;

}

int spfa()

{

memset(dist, 0x3f, sizeof dist);

dist[1] = 0;

queue <int> q;

q.push(1);

st[1] = true;

while(q.size())

{

int t = q.front();

q.pop();

st[t] = false;

for(int i = h[t]; i != -1; i = ne[i])

{

int j = e[i];

if(dist[j] > dist[t] + w[i])

{

dist[j] = dist[t] + w[i];

if(!st[j]) //若队列中没有则入队

{

q.push(j);

st[j] = true;

}

}

}

}

return dist[n];

}

int main()

{

scanf("%d%d", &n, &m);

memset(h, -1, sizeof h);

while (m -- )

{

int a, b, w;

scanf("%d%d%d", &a, &b, &w);

add(a, b, w);

}

int t = spfa();

if(t == 0x3f3f3f3f) puts("impossible");

else printf("%d", t);

}

代码5：

#include <iostream>

using namespace std;

const int N = 10;

int n, path[N], state[N]; //state[i] ==1表示该《数字》已用过，==0表示没用过

void dfs(int u) //u表示的是《位置》

{

if (u == n)

{

for (int i = 0; i < n; i ++ ) printf("%d ", path[i]);

puts("");

return;

}

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

{

if (!state[i]) //当前数i尚未被用过

{

path[u] = i;

state[i] = 1;

dfs(u + 1); //递归，填下一个位置

state[i] = 0; //回溯，重置i使其变成没用过（递归返回后，i从大数开始往小数回，将对应state置为0）

}

}

}

int main()

{

scanf("%d", &n);

dfs(0);

}

代码6：

#include <iostream>

using namespace std;

const int N = 10;

int n;

char g[N][N];

bool col[N], dg[2 \* N], udg[2 \* N];

void dfs(int u)

{

if (u == n)

{

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

puts( g[i] );

puts("");

return;

}

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

{

if (!col[i] && !dg[n + u - i] && !udg[u + i]) //此三项都是0，即三条线上没有Queen

{

g[u][i] = 'Q';

col[i] = dg[n + u - i] = udg[u + i] = true;

dfs (u + 1);

col[i] = dg[n + u - i] = udg[u + i] = false; //恢复现场，回溯后将原先用过的列、正反对角线置为0

g[u][i] = '.';

}

}

}

int main()

{

scanf("%d", &n);

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

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

g[i][j] = '.' ;

dfs(0);

}

代码7：

//如果所有点都进过队列（入度为0的点），则可以拓扑排序

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int N = 1e5 + 10;

int h[N], e[N], ne[N], idx;

int q[N], d[N]; //q[]存储入度为0的点，即可以输出的点；d[]存储的是某点的入度

int n, m;

void add(int a, int b)

{

e[idx] = b;

ne[idx] = h[a];

h[a] = idx ++;

}

bool tuopu() //是否存在拓扑结构

{

int hh = 0, tt = -1;

for (int i = 1; i <= n; i ++ ) //寻找入度为0的点

{

if(!d[i]) //若该点入度为0则入队

q[++ tt] = i;

}

while (hh <= tt)

{

int tmp = q[hh ++]; //取出队头元素

for(int i = h[tmp]; i != -1; i = ne[i]) //遍历队头元素所连接的边

{

int j = e[i];

if (-- d[j] == 0) //删除队头元素后，若连接的下一点入度为0，则入队

q[++ tt] = j;

}

}

return tt == n - 1; //是否所有元素都入过队

}

int main()

{

scanf("%d%d", &n, &m);

memset (h, -1, sizeof h);

while(m --)

{

int a, b;

scanf("%d%d", &a, &b);

add(a, b);

d[b] ++; //每对b点add一次，b点入度加一

}

if (!tuopu()) puts("-1");

else

{

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

cout << q[i] << " ";

}

}

代码8：

#include <iostream>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

const int N = 1e5 + 10;

int h[N], e[N], ne[N], idx; //构成有向图的要素

int d[N]; //存储1到该点的距离

int n, m;

void add(int a, int b)

{

e[idx] = b;

ne[idx] = h[a];

h[a] = idx ++;

}

int bfs()

{

memset(d, -1, sizeof d);

d[1] = 0; //从第一个点开始

queue <int> q;

q.push(1);

while (q.size())

{

int t = q.front();

q.pop();

for(int i = h[t]; i != -1; i = ne[i])

{

int j = e[i];

if(d[j] == -1) //当前点还未走过

{

d[j] = d[t] + 1;

q.push(j);

}

}

}

return d[n];

}

int main()

{

scanf("%d%d", &n, &m);

memset (h, -1, sizeof h);

while (m -- )

{

int a, b;

scanf("%d%d", &a, &b);

add(a,b);

}

cout << bfs();

}

代码9：

#include <cstring>

#include <iostream>

using namespace std;

const int N = 1e5 + 10;

const int M = 2 \* N; //以有向图的方式存储无向图，所以最多有2n-2条边

int h[N]; //存所有的头结点

int e[M]; //存储元素（每个链表中至多会有M个数据）

int ne[M]; //存储链表的next值

int idx, n;

int ans = N;

bool st[N]; //记录结点是否被访问过

void add(int a, int b)

{

e[idx] = b;

ne[idx] = h[a];

h[a] = idx ++;

}

int dfs(int u) //返回以u为根的子树中，结点的个数（包含u）

{

st[u] = true;

int size = 0; //存删掉u结点后，剩余连通块中最大的节点数

int sum = 1; //存 以u为根的子树 的结点数，u为第一个

for(int i = h[u]; i != -1; i = ne[i]) //访问u的子节点

{

int j = e[i];

if(!st[j]) //若结点没被访问过

{

int s = dfs(j); //递归处理其子结点，并用s存储子树的结点数

size = max(size, s); //记录最大连通子图的节点数

sum += s;

}

}

size = max(size, n - sum);

ans = min(size, ans);

return sum ;

}

int main()

{

cin >> n;

memset(h, -1, sizeof h);

for (int i = 0; i < n - 1; i ++)

{

int a, b;

cin >> a >> b;

add(a, b), add(b, a); //无向图

}

dfs(1);

cout << ans;

}