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：Bellman-ford算法，SPFA算法，floyd算法，prim算法，kruskal算法，染色法，匈牙利算法等 |
| 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：

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

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

int n1, n2, m;

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

int match[N]; //存放右边点对应的左边点

bool st[N];

void add(int a, int b)

{

e[idx] = b;

ne[idx] = h[a];

h[a] = idx ++;

}

bool find(int x) //为男生x找对象

{

for(int i = h[x]; i != -1; i = ne[i]) //遍历x的所有暧昧对象

{

int j = e[i];

if(!st[j])

{

st[j] = true;

if(match[j] == 0 || find(match[j])) //若当前女孩没有对象，或其对象可以换个备胎

{

match[j] = x;

return true;

}

}

}

return false;

}

int main()

{

scanf("%d%d%d", &n1, &n2, &m);

memset(h, -1, sizeof h);

while (m -- )

{

int a, b;

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

add(a, b);

}

int res = 0;

for (int i = 1; i <= n1; i ++ )

{

memset(st, false, sizeof st); //使该男生有机会向所有他心仪的女生询问

if(find(i)) res ++;

}

printf("%d", res);

}

代码2：

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int N = 1e5 + 10, M = 2e5 + 10;

int n, m;

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

int color[N]; //存每个点的颜色，0表示未上色，1表示一个色，2表示另一个色

void add(int a, int b)

{

e[idx] = b;

ne[idx] = h[a];

h[a] = idx ++;

}

bool dfs(int u, int c)

{

color[u] = c; //将点u染为c色

for(int i = h[u]; i != -1; i = ne[i]) //为所有与点u相连的点染色

{

int j = e[i];

if(!color[j]) //若该点未被染色

{

if(!dfs(j, 3 - c)) return false; //为该点染色为另一个颜色是否成功

}

else if(color[j] == c) return false; //若与该点相连的点颜色相同，则无法构成二分图

}

return true;

}

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), add(b, a);

}

bool flag = true;

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

{

if(!color[i])

{

if(!dfs(i, 1) )

{

flag = false; //如果染色过程中有矛盾，则flag被设置为false

break;

}

}

}

if(flag) puts("Yes");

else puts("No");

}

代码3：

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int N = 1e5 + 10, M = 2e5 + 10, INF = 0x3f3f3f3f; //两个未连通的点之间距离为无穷大

int n, m;

int p[N]; //存某点的父亲结点

struct Edge

{

int a, b, w;

bool operator < (const Edge &E) const //重载小于号（因为sort函数中是用小于号进行排序的）

{

return w < E.w;

}

}edges[M];

int find(int x) //返回（寻找）x的祖宗结点

{

if(p[x] != x) p[x] = find (p[x]);

return p[x];

}

int kruskal()

{

sort(edges, edges + m); //将每条边按权重排序

for (int i = 1; i <= n; i ++ ) p[i] = i; //初始化并查集

int res = 0, cnt = 0; //res为生成树中的边权之和，cnt为树的边之和

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

{

int a = edges[i].a, b = edges[i].b, w = edges[i].w;

a = find(a), b = find(b);

if(a != b)

{

p[a] = b;

res += w;

cnt ++;

}

}

if(cnt < n - 1) return INF;

return res;

}

int main()

{

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

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

{

int a, b, w;

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

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

}

int t = kruskal();

if(t == INF) puts("impossible");

else cout << t << endl;

}

代码4：

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int N = 510, INF = 0x3f3f3f3f;

int n, m;

int dist[N]; //存该点到已确定的生成树的距离

int g[N][N]; //邻接矩阵存图，存边的权值

bool st[N];

int prim()

{

memset(dist, 0x3f, sizeof dist);

int res = 0;

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

{

int t = -1;

for (int j = 1; j <= n; j ++ ) //遍历所有的点

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

t = j; //找到当前离生成树最近的点，存入t中

if(i && dist[t] == INF) return INF; //当t与已生成的树之间间断，即无法生成一棵树时

if(i) res += dist[t];

st[t] = true;

for(int j = 1; j <= n; j ++) dist[j] = min(dist[j], g[t][j]);

}

return res;

}

int main()

{

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

memset(g, 0x3f, sizeof g);

while (m --)

{

int a, b, w;

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

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

}

int t = prim();

if(t == INF) puts("impossible");

else printf("%d", t);

}

代码5：

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int N = 210, INF = 1e9; //最长的路：10000 \* 20000

int n, m, q;

int d[N][N];

void floyd()

{

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

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

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

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

}

int main()

{

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

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

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

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

else d[i][j] = INF;

while (m -- )

{

int a, b, c;

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

d[a][b] = min(d[a][b], c); //用min是为了取重边中最小的

}

floyd();

while (q --)

{

int a, b;

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

int t = d[a][b];

if (t > INF / 2) puts("impossible"); //因为存在负权边（无负权回路），所以要 >INF/2

else printf("%d\n", t);

}

}

代码6：

#include <iostream>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

const int N = 2010, M = 1e4 + 10;

int n, m;

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

int dist[N], cnt[N]; //cnt[]表示当前最短路的边的数量

bool st[N];

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

{

e[idx] = b;

w[idx] = c;

ne[idx] = h[a];

h[a] = idx;

idx ++;

}

bool spfa()

{

queue <int> q;

for (int i = 1; i <= n; i ++ ) //因为无法保证负环一定可以从1开始到达，因此将所有点都入队

{

st[i] = true;

q.push(i);

}

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];

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

if(cnt[j] >= n) return true;

if(!st[j])

{

q.push(j);

st[j] = true;

}

}

}

}

return false;

}

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);

}

if(spfa()) cout << "Yes";

else cout << "No";

}

代码7：

#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;

}