# 算法导论-22.2-7-树的直径

一、题目

树T=（V,E）的直径（diameter）定义为max(u,v)，亦即，树的直径是树中所有最短路径长度中的最大值。试写出计算树的直径的有效算法，并分析算法的运行时间。

## 二、思考

step1：以树中任意一个结点为源点，进行一次广度优先遍历，找出离源点距离最远的点d
step2：以d为源点，进行一次广度优先遍历，找出离d最远的点，并记录其长度

## 三、代码

```cpp
//用邻接表实现图的转置
#include <iostream>
#include <queue>
using namespace std;

#define N 8
#define WHITE 0
#define GRAY 1


//边结点结构
struct Edge
{
    int start;//有向图的起点
    int end;//有向图的终点
    Edge *next;//指向同一个起点的下一条边
    Edge(int s, int e):start(s),end(e),next(NULL){}
};
//顶点结点结构
struct Vertex
{
    Edge *head;//指向以该顶点为起点的下一条边
    bool color;//颜色，我觉得两种颜色就够了
    Vertex *p;//指向遍历结果的父结点
    int d;//与源点之间的距离
    Vertex():head(NULL),color(WHITE),p(NULL),d(0x7fffffff){}
};
//图结构
struct Graph
{
    Vertex *V[N+1];//N个顶点
    Graph()
```

```
    {
        int i;
        for(i = 1; i <= N; i++)
            V[i] = new Vertex;
    }
    ~Graph()
    {
        int i;
        for(i = 1; i <= N; i++)
            delete V[i];
    }
};

//广搜的先进先出用STL中的队列来实现
queue<Vertex*> Q;

//插入边
void InsertEdge(Graph *G, Edge *E)
{
    //如果没有相同起点的边
    if(G->V[E->start]->head == NULL)
        G->V[E->start]->head =E;
    //如果有，加入到链表中，递增顺序排列，便于查重
    else
    {
        //链表的插入，不解释
        Edge *e1 = G->V[E->start]->head, *e2 = e1;
        while(e1 && e1->end < E->end)
        {
            e2 = e1;
            e1 = e1->next;
        }
        if(e1 && e1->end == E->end)
            return;
        if(e1 == e2)
        {
            E->next = e1;
            G->V[E->start]->head =E;
        }
        else
        {
            e2->next = E;
            E->next = e1;
        }
    }
}
//在广度优先搜索的同时，记录从离源点最远的点及其长度
void BFS(Graph *G, Vertex *s, int &ls, int &lv)
{
    int i;
    //虽然所有结点在构造函数中初始化过了，但是如果对同一图多次搜索，每次都要重新初始化
    for(i = 1; i <= N; i++)
    {
        G->V[i]->color = WHITE;
```

```
87.         G->V[i]->d = 0x7fffffff;
88.         G->V[i]->p = NULL;
89.     }
90.     //对s进行特殊的初始化
91.     s->color = GRAY;
92.     s->d = 0;
93.     s->p = NULL;
94.     //初始化队列，使其仅含源顶点s
95.     while(!Q.empty())
96.         Q.pop();
97.     Q.push(s);
98.     //只要队列中还有灰色顶点，循环将一直进行下去
99.     while(!Q.empty())
100.    {
101.        //确定队列头部的灰色顶点u，将其从队列中去掉
102.        Vertex *u = Q.front();
103.        Q.pop();
104.        //考察u的邻接表中每条边的终点v
105.        Edge *e = u->head;
106.        while(e)
107.        {
108.            Vertex *v = G->V[e->end];
109.            //如果v是白色的，表明该顶点尚未被发现
110.            if(v->color == WHITE)
111.            {
112.                //置为灰色
113.                v->color = GRAY;
114.                //计算距离
115.                v->d = u->d + 1;
116.                if(v->d > ls)
117.                {
118.                    ls = v->d;
119.                    lv = v->id;
120.                }
121.                //u被标记为该顶点的父母
122.                v->p = u;
123.                //将它置于队列的尾部
124.                Q.push(v);
125.            }
126.            e = e->next;
127.        }
128.        //当u的邻接表中的所有顶点被检查完后，u被置为黑色（仅仅是便于理解，没有实际意义）
129.        u->color = GRAY;
130.    }
131. }
132. //输出
133. void Print(Graph *G)
134. {
135.     int i;
136.     //遍历每个顶点
137.     for(i = 1; i <= N; i++)
138.     {
139.         cout<<i<<':';
140.         //输出以i为起点的边的终点
```

```
        Edge *e = G->V[i]->head;
        while(e)
        {
            cout<<e->end<<' ';
            e = e->next;
        }
        cout<<endl;
    }
    cout<<endl;
}

int main()
{
    //构造一个空的图
    Graph *G = new Graph;
    Edge *E;
    Print(G);
    //输入边
    int i, start, end;
    for(i = 1; i <= 10; i++)
    {
        cin>>start>>end;
        E = new Edge(start, end);
        InsertEdge(G, E);
        //无向图，要加两条边
        E = new Edge(end, start);
        InsertEdge(G, E);
    }
    Print(G);
    int ls = -1, lv;
    //第一次搜索，从任意结点出发，找出最远的点lv，lv一定是所求路径上的一个端点
    BFS(G, G->V[1], ls, lv);
    ls = -1;
    //第二次搜索，从lv出发，找出最长的路径
    BFS(G, G->V[lv], ls, lv);
    cout<<ls<<endl;
    return 0;
}
```