中国科学技术大学计算机学院
《数据结构》报告

实验题目：图及其应用

学生姓名：高楚晴

学生学号：PB18111688

完成日期：2019.12.19

实验目的

熟练掌握图的存储表示特征，各类图的创建、遍历方法以及基于遍历的算法应用。

设计思路

DFS的应用

本实验目的为基于 邻接矩阵的存储结构，使用 非递归的深度优先搜索算法，求无向连通图中的全部关节点，并按照顶点编号升序输出。

由于书中给出了递归形式的基于邻接表存储结构的求无向图中全部关节点的搜索算法，初步思路为修改其结构为邻接矩阵存储，并进一步修改为非递归的栈形式的实现。

但在初步分析后，发现书中给的算法并不完全正确，其默认根节点有几个子树，去掉后连通图变为森林，因此为关节点，但在实际情况中，我们随机选取的根节点0不一定为关节点，其子节点之间可以相互联通。因此在判断时对0单独判断，即在对其余顶点按照书中算法判断后用基本的DFS方式对0以外遍历，判断其是否为连通图，若是连通图，则0不是关节点，反之则是。

此时测试算法正确。进一步改变存储形式为邻接矩阵，同时将递归算法修改为使用栈的非递归算法，对于邻接矩阵，其基本信息要求栈中存放w,v值，而在递归过程中min不断改变，因此也存放于栈节点中进行保存。同时编写对应的pop与push算法即可。

此外，由于要求关节点以按编号升序输出，因此引入一个标记数组isart，若查找中发现其为关节点即将数组对应位置置1，最终根据数组存放情况输出即可。

BFS的应用

实验目的为基于 邻接表的存储结构，依次输出从顶点0到顶点1、2、……、n-1的最短路径和各路径中的顶点信息。

使用一个pre数组存放最短路径上每个节点的前继节点，按照类层序遍历的方式使用BFS原理查找，即不断通过已找到路径的节点，对其邻接点搜索，若还未找到最短路径，则其邻接点的最短路径为前继节点的最短路径加上前继节点本身。

最终使用回溯的方式存放路径并记录路径长度，倒序输出其路径。

关键代码讲解

DFS的应用

寻找关节点

由于图搜索时将其类比树的方式，而其根节点是随机选择的，相较于其他节点需进行单独处理，因此先对根节点搜索关节点获得第一层信息，再对其子树搜索。

```
void Findart() {              //寻找关节点
    int i, j, k = 0, t = 0;
    int v = 0;
    count = 0;
    visited[0] = 1;
    for (i = 0; i < G.vexnum; ++i) {       //初始值清零
        visited[i] = 0;
        isart[i] = 0;
```

```
    }
    DFSarti(v);        //从顶点w开始DFS关节点
    if (count < G.vexnum) {
        while (1) {
            while (!G.arcs[v][0].adj&&v < G.vexnum)    //找该顶点全部邻接点
                v++;
            if (v >= G.vexnum)           //该节点可能的邻接点全部访问完
                break;
            if (visited[v] == 0)
                DFSarti(v);
        }
    }
    for (i = 1; i < G.vexnum; i++)
        flag[i] = 1;
    while (G.arcs[0][t].adj == 0)
        t++;
    DFS(t);                    //从根的第一个邻接点开始搜索，检测根之外是否为连通片
    for (i = 1; i < G.vexnum; i++)
        if (flag[i] == 1)
            isart[0] = 1;       //根节点的叶子结点不完全联通
    //******************输出*****************
    for (i = 0; i < G.vexnum; i++)
        if (isart[i])
            printf("%d ", i);
}
```

检测根以外连通性

flag相当于visited标记，为全局函数，初始值为1，访问后清0

```
void DFS(int v) {
    int w;
    flag[v] = 0;
    for (w = 1; w < G.vexnum; w++)
        if (G.arcs[v][w].adj&&flag[w])
            DFS(w);
}
```

非递归实现搜索

原函数功能为从某个节点开始搜索关节点，修改为非递归算法后实际上为从某个节点开始搜索全部关节点，实际只调用一次。

```
void DFSarti(int v0) {
    snode *p;
    int min;
    visited[v0] = min = ++count;
    int v1 = 0;
    Push(v0, v1,min);
    while (!Isempty()) {
        p = Pop();
        v0 = p->v;              //恢复原值
        v1 = p->w + 1;
```

```
        min = p->min;
        while (G.arcs[v0][v1].adj == 0 && v1 < G.vexnum)    //找该顶点全部邻接点
            v1++;
        if (v1 >= G.vexnum)     //该节点可能的邻接点全部访问完
            continue;
        if (visited[v1] == 0) {
            Push(v0, v1,min);
            v0 = v1;
            v1 = 0;
            visited[v0] = min = ++count;
            low[v0] = min;
            Push(v0, v1,min);
            continue;
        }
        else {
            if (low[v1] < min)
                min = low[v1];
            if (low[v1] >= visited[v0])
                isart[v0] = 1;
            if (visited[v1] < min)
                min = visited[v1];
        }
    }
}
```

BFS的应用

搜索最短路径上的前继节点

used数组用于检测该节点是否已经被用作寻找下个顶点，以此缩短搜索时间。搜索时先搜索根节点的全部邻接点，按层序遍历的思路按照最短路径的长度不断对其他节点的前继节点进行判断。当count变量显示全部顶点都已经找到前继节点，搜索结束。

```
void find(Algraph *G) {
    int used[MAX_NUM];
    int i, v0 = 0;
    int count = 1;
    Arcnode *p;
    pre[0] = 0;
    used[0] = 1;
    for (i = 1; i < G->vexnum; i++) {
        pre[i] = -1;
        used[i] = 0;
    }
    for (p = G->vertices[0].firstarc; p; p = p->nextarc) {
        pre[p->adjvex] = 0;
        count++;
    }
    while (count != G->vexnum) {
        for (i = 1; i < G->vexnum; i++) {
            if (pre[i] != -1 && used[i] == 0) {    //还没有用当前节点找过下一个点
                used[i] = 1;
                for (p = G->vertices[i].firstarc; p; p = p->nextarc)
                    if (pre[p->adjvex] == -1) {
                        pre[p->adjvex] = i;
```

```
                        count++;
                    }
                }
            }
        }
    }
```

输出最短路径

即用回溯的方式将路径上的点倒序加入路径存放数组中并长度不断增加，直至回溯到起始点0时结束，倒序输出路径并输出路径长即可。

```
ivoid output(Algraph *G) {
    int i, j, len, path[MAX_NUM];
    for (i = 1; i < G->vexnum; i++) {
        len = 0;
        j = i;
        path[len++] = j;
        while (pre[j] != 0) {           //回到起始顶点前不断将前继节点添加到路径上
            j = pre[j];
            path[len++] = j;
        }
        printf("%d 0", len);
        for (j = len - 1; j >= 0; j--)
            printf("->%d", path[j]);
        printf("\n");
    }
}
```

调试分析

初次完成代码后进行调试，发现邻接表结构初始化失败，经检查调试后发现是将`if (G0->vertices[c2].firstarc == NULL)`写成了`if (G0->vertices[c2].firstarc = NULL)`。

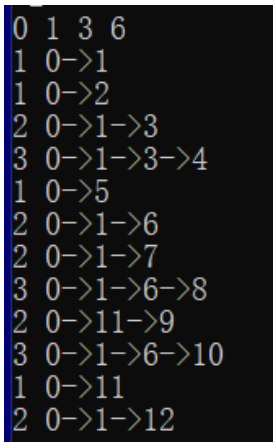
再次调试发现最短路径输出与预计不同，路径中出现重复值，经检查发现为路径记录模块赋值与回溯两语句顺序颠倒，修改后程序运行正常。

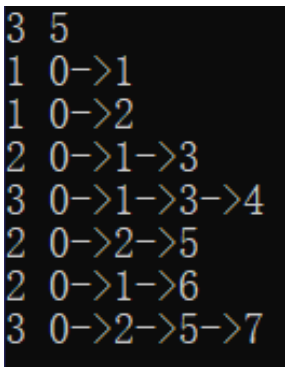
对于DFS应用于寻找关节点，由于其算法实际上为一个遍历的过程，因此时间复杂度为$O(n+e)$，其中n为顶点数，e为边数。应用非递归算法时，邻接矩阵的空间复杂度为$O(n^2)$，递归栈的空间复杂度为$O(n)$, 而其余各标记变量引入的空间复杂度也都在$O(n)$之内，综上，空间复杂度为$O(n^2)$。

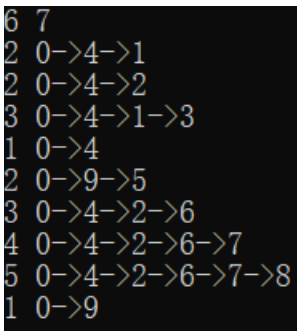
对于BFS应用，由于为两重循环（对当前层次每个点寻找其邻接点的情况），其时间复杂度为$O(n^2)$， 其输出模块的时间复杂度为$O(n)$，因此总时间复杂度为$O(n^2)$。邻接表存储的空间复杂度为$O(n^2)$，各类标记变量和记录变量复杂度均在$O(n^2)$之内，综上，总空间复杂度为$O(n^2)$。

代码测试

对课程主页给出的三组数据执行结果如下。







与理论值一致，显示运行正常，代码无误。

附录

PB18111688_高楚晴_4.cpp