算法设计与分析第一次作业

曲宇勋 201928014628016 人工智能学院

1.证明下列结论：

2）在一个有向图D中，如果每个顶点的出度都大于等于1，则该图一定含有一个有向圈。

Soluiton:

在有向图D中构建最长有向路$v_0e_0v_1e_1...v_n$。

由题设可知节点v_n出度大于等于1，由于其是最长有向路的终点，以其为起点的边e_n不可能指向最长路以外的节点，所以假设其指向$v_k, n > k > 1$，则$v_ke_kv_{k+1}e_{k+1}...v_ne_nv_k$就可构成一个有向圈，所以图中一定存在一个有向圈。

3．设G是具有n个顶点和m条边的无向图，如果G是连通的，而且满足$m = n - 1$,证明G是树。

Soluiton:

假设G不是树，那么G中至少包含一个圈。

利用归纳法证明该定理

(1)当$n = 1, m = 2$时，易知图内没有圈，G一定是树.

(2)假设$n = k, m = k + 1$时的任意图G均为树，那么对于$n = k + 1, m = k + 2$的图G_1分析.

首先，假设G中所有顶点度都大于等于2，那么边的数量$m = \frac{1}{2}\sum d_n \geq n$，与$m = n - 1$相违背，所以至少存在一个顶点$v_k$的度为1。

删除v_k以及连接其的边，则边数和顶点数同时减一，该操作不影响图G_1中圈的数量，图G_1转化为$n = k, m = k + 1$的一张图G_2。

由归纳假设可知G_2中不包含圈，所以G_1中也不包含圈。所以G_1是树。

综上所述，若G是具有n个顶点和m条边的无向图，如果G是连通的，而且满足$m = n - 1$,则G是树

5.下面的无向图以邻接链表存储，而且在关于每个顶点的链表中与该顶点相邻的顶点是按照字母顺序排列的。试以此图为例描述讲义中算法DFNL的执行过程。

Soluiton:

首先经历DFNL中的深度遍历过程，具体而言过程如下

经过A点，$DFN = L = 1$.

搜索A的邻接点，经过B点，$DFN = L = 2$.
搜索B的邻接点，经过C点，$DFN = L = 3$.
搜索C的邻接点，经过D点，$DFN = L = 4$.
回溯回C点，继续搜索C的邻接点，经过E点，$DFN = L = 5$.
搜索E的邻接点，经过A点，A点已经被访问，说明连接E与A的边是余边，更新E点的$DFN = 5, L = 1$
回溯回E点，继续搜索E的邻接点，经过F点，$DFN = L = 6$
搜索F的邻接点，经过G点，$DFN = L = 7$
可以形成图2左侧图，DFN与L更新均如左侧图所示。
搜索G的邻接点，经过E点，E点已经被访问，说明连接G与E的边是余边，更新G点的$L = 5$
回溯回F点，F点的L更新为$min\{u, v\}$，u为F，v为G，$L = 5$.
回溯回E点，E点的L更新为$min\{u, v\}$，u为E，v为F，$L = 5$.
回溯回C点，C点的L更新为$min\{u, v\}$，u为C，v为E，$L = 1$.
回溯回B点，B点的L更新为$min\{u, v\}$，u为B，v为C，$L = 1$.
回溯回A点，A的L不变.
遍历结束.

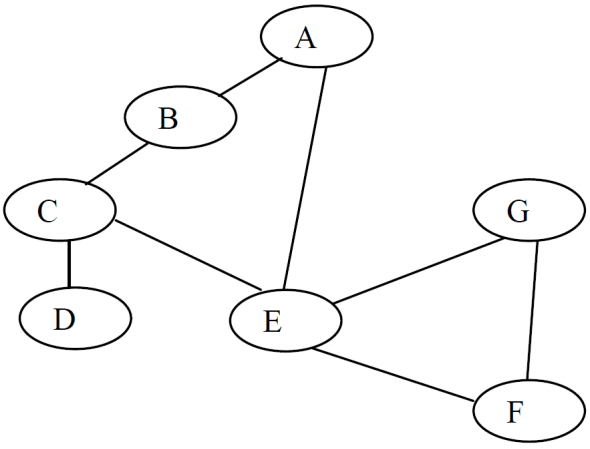

图 1: 无向图

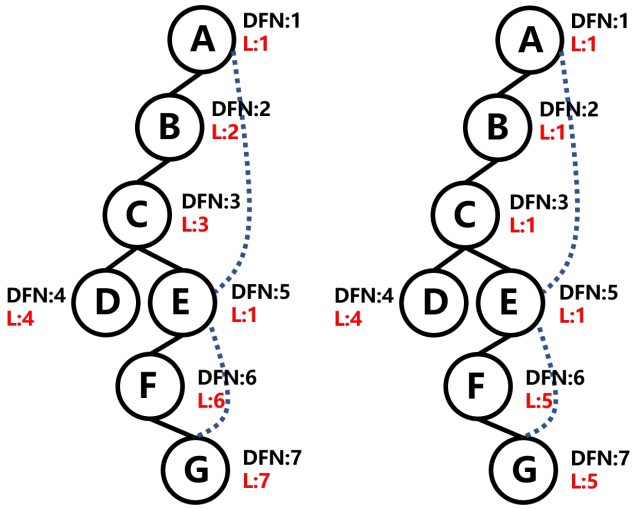

图 2: 树的DFN与L

6.对图的另一种检索方法是D-Search。该方法与BFS的不同之处在于将队列换成栈，即下一个要检测的节点是最新加到未检测节点表的那个节点。

1）写一个D-Search算法；

2）证明由节点v开始的D-Search能够访问v可到达的所有节点；

3）你的算法的时、空复杂度是什么？

Soluiton:

(1)首先构建从v节点开始的D-Search算法

```
1  void D–Search(v)
2  /*
3  visited 标示各顶点被访问的序数，为一维数组;
4  s 将用来标示各顶点是否被检索过，是则标记为1 ，否则标记为0 ，为一
       维数组;
5  count 计数到目前为止已经被访问的顶点个数，其初始化为在v 之前已
       经被访问的顶点个数;
6  L 初始化为空栈;
7  Push(L,v)是将压栈v;
8  Pop(L)是取出栈顶的节点;
9  */
10             Push(L,v)
```

```
        while L 非空 do
                u:=Pop(L);
                count:=count+1;
                visited[u]:=count;
                for 邻接于u 的所有顶点w do
                        if s[w]=0 then
                                Push(L,w);
                                s[w]:=1;
                        end{if}
                end{for}
        end{while}
end{D-Search}
```

接着构建全图的D-Search算法

```
void D-Search-T(G,n)
/*
G 为输入的图，为全局变量;
n 为顶点数目;
s 将用来标示各顶点是否被检索过，是则标记为1 ，否则标记为0 ，为一
    维数组，为全局变量;
count 计数到目前为止已经被访问的顶点个数，其初始化为在v 之前已
    经被访问的顶点个数，为全局变量;
branch 表示连通的分支数目;
*/
        count:=0;
        branch:=0;
        for i=0 to n-1 do
                s[i]:=0;
        end{for}
        for i=0 to n-1 do
                if s[i] != 0 then
                        D-Search(的第顶点Gi);
                        branch = branch+1;
                end{if}
```

```
19              end{for}
20  end{D-Search-T}
```

(2)由于对于图G中在栈中的每一个顶点的相邻顶点均进行了遍历，事实上，遍历过程包含了图G中存在的v可达点构成子图的所有边，只是对于重复遍历的顶点不再遍历其相邻边。所以，若u至v是可达的，连接两点的路一定在遍历过程中出现过，所以可达点一定能被该算法访问.

(3)除了顶点v，仅有当节点w满足s[w]=0时，其才能被压栈，所以栈中每个顶点都至多出现一次，所以栈所占空间与$n-1$成正比，同理visited所占空间也与n成正比，所以空间复杂度为O(n).

如果采用邻接链表，则for语句的执行次数为$d(u)$，for语句内部的时间复杂度为O(1)，所以经过while后执行次数为$\sum d(u) = m$，复杂度为O(m)，而while共执行n次，所以count和visited的赋值复杂度为O(m)，所以时间复杂度为O(m+n).

如果采用邻接矩阵，则for语句执行次数为n，while执行次数也为n，时间复杂度为O(n^2).