深度优先遍历知识点复习

2022.4.16

图遍历：DFS主框架

```
DFS(v):
v.color = GRAY;
<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w);
        <Backtrack processing of edge vw>;
    else
        <check edge vw>;
<Postorder processing of node v>
v.color = BLACK;
```

```
DFS-WRAPPER(G):
foreach node of G do:
    if node.color == WHITE them
        DFS(node);
```

有向图的深度优先遍历树

遍历过程中的四种边

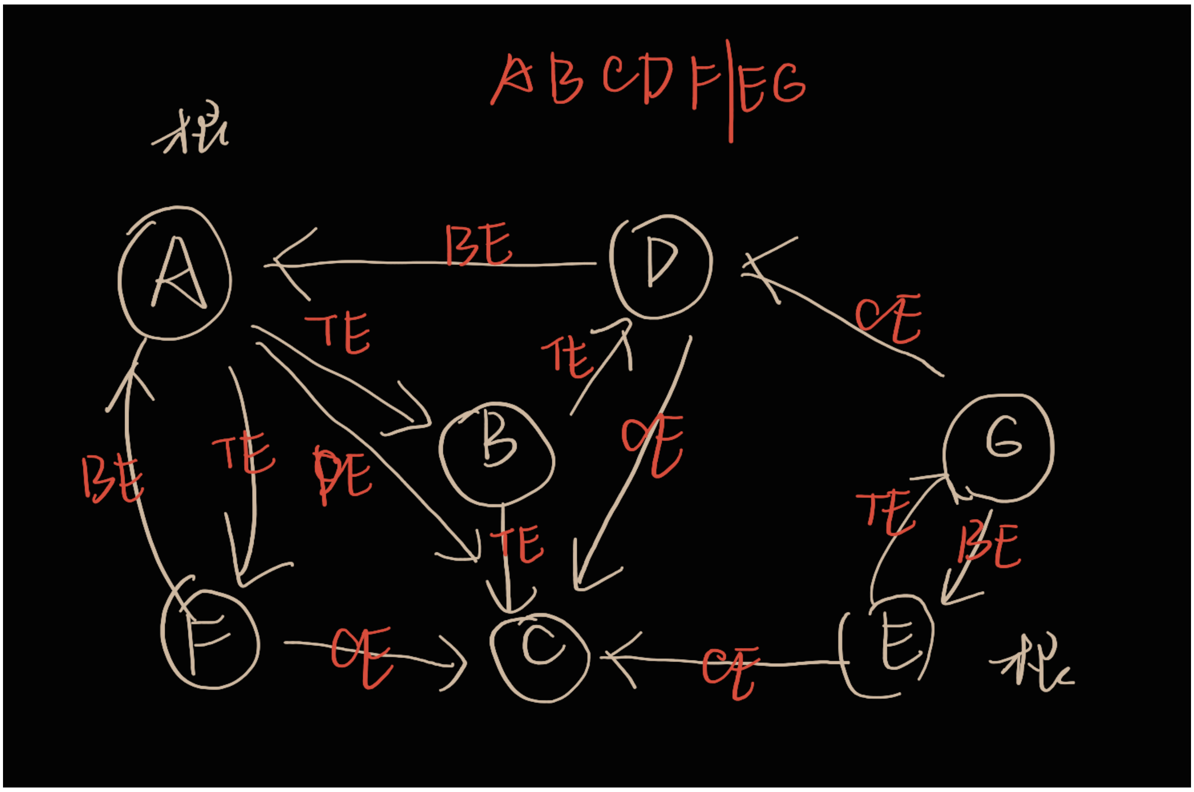
TE:遍历树中的父亲-孩子的边

BE：后向边back edge ，访问某节点的邻居时候发现它的邻居已经被访问过了，而且在遍历树中这条边还是自己的祖先，那么这条边就是后向边。

DE：前向边也叫fe。访问邻居的时候发现邻居已经被访问过了，而且邻居还是自己的子孙节点，那这条边就是前向边。

CE:不是TE BE DE的边就是CE

举一个栗子：



活动区间

discovertime finishtime

每一次有颜色发生变化时间加一

```
DFS(v):
v.color = GRAY;
time+=1;
v.discovertime = time;

<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w);
        <Backtrack processing of edge vw>;
    else
        <check edge vw>;
<Postorder processing of node v>
time+=1;
v.finishtime = time;
v.color = BLACK;
```

活动区间相关定理及证明：

1.w是v在DFS树中的后继节点，当且仅当active(w)被active(v)包含。

2.w和v没有祖先后继关系，当且仅当active(w)和active(v)没有重叠

3.如果vg是G中的边

①vw是CE（v->w），iff active(w)在active(v)前面

> 由2可知不重叠，所以访问w之前一定没有访问v,所以……

②vw是DE（v->w）当且仅当存在第三个节点x a(w)被a(x)包含，a(x)被……

> w已经被访问过了，还是v的子孙

③vw是BE 当且仅当active(v)被active(w)包含

> w被访问过了，w是v的祖先

④vw是TE 当且仅当active(w)被active(v)包含且不存在第三个节点使得……

证明1和证明2是相互推理的。

证明1 ->

定义一种偏序关系：<：表示在dfs tree中w是v的子孙

Ⅰ使用数学归纳法证明，对v进行归纳：

base:v的子孙只有自己，显然正确。

假设 如果对于v的子孙x都满足1，active(w)被active(x)包含

那么对于v来说，其真正的儿子结点由dfs树性质可知满足I，其孙子结点由归纳假设……

<-

如果active(w)被active(v)包含，但w不是v在DFS树中的后继节点

那么w和v有两种情况：

①不在一棵树上，此时是不包含的，所以不符合

②在一棵树上，有证明2可知也没有重叠

所以说是错误的。

证明2：

->

①w和v没在一棵树上，那么两者不会有任何重叠

②w和v在一棵树上，没有祖先后继关系，说明是CE。

<u>那么至少存在一个顶点x->w,x->v且不重合（即公共祖先）</u>

所以考虑x->w,x->v，由于x->w路径上的第一个点y,x->v路径上的第一个点z

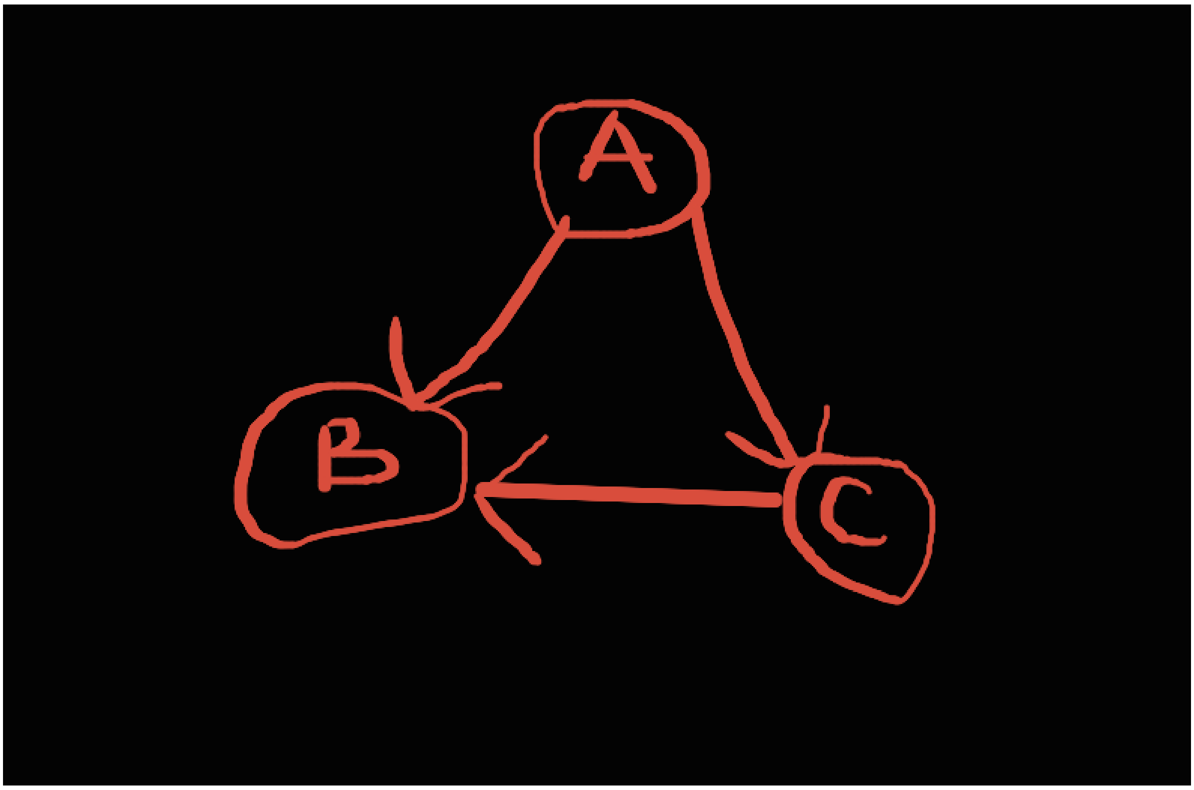
由dfs性质可知y，z的路径是不重合的，而active(w)一定被active(y)包含……可证。

<-

反证法：假设active(w)和active(v)没有重叠，但有祖先后继关系，则一定有包含关系，但是因为两者没有重叠，所以错误。

如果dfs树上两个节点之间有祖先后继关系，则它们在图中也一定有路径相连，反之则不一定成立

容易构造反例：



B,C就没有祖先后继关系。

总结一下，在dfs图中没有祖先关系的两个节点可能的情况：

①不相连

②不在一棵树上

③在一棵树上但是关系是CE，一定有共同祖先。

判断遍历树中的祖先后继关系：白色路径定理（考过）

> 在深度优先遍历树中，节点v是w的祖先，当且仅当在遍历过程中发现点v的时刻，存在一条从v到w的全部由白色节点组成的路径。

=>若v是w的祖先，则首先发现的是v，由dfs性质可知发现w的这一路上一定都是经过一堆白色节点。

<=若存在v到w的白色路径的话

对**白色路径的长度**进行归纳：

k=0时，v就是自己的祖先。

假设所有小于k的白色路径都成立，考虑一条比k长的白色路径，可以划分成v->xi,xi->w，后面那条路径是小于k的，那么xi是w的祖先，又因为v的discovertime<xi.discovertime,同时finishtime也>，因为如果finishtime比它小的话，xi仍为白色节点但是v已经遍历完了，这与dfs遍历相违背，所以说可证。

有向图上深度优先遍历的应用

拓扑排序

全序关系，只要存在边就可以比较序号。

v1,v2,v3……任意vi->vj 有i<j

无拓扑排序的图意味着图中有循环依赖，也就是说vi->vj，不一定有i<j

如何检测G有环，**BE出现就说明有环**

BE出现的条件：在遍历过程中遍历到了一个颜色为灰色的节点

引理：

> 如果有向图G有环，G就不存在拓扑排序
>
> 如果没有环，就一定存在拓扑排序

定理

> 图G是有向无环图，当且仅当图G有拓扑排序。

globalnum=n+1

```
DFS(v):
v.color = GRAY;
<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w);
        <Backtrack processing of edge vw>;
    else
        <check edge vw>;
<Postorder processing of node v>
globalNum--;
v.topoNum=globlanum;//分配拓扑序号
v.color = BLACK;
```

关键路径

最早开始时间est

最早结束时间eft

定义：

```
如果ai不依赖于其它任务，esti=0;
如果ai的est知道了，efti = esti+l
如果一个任务依赖于其它任务，esti = max(eftj)
```

关键路径定义：v0,v1……vk

```
v0不依赖于其它任务
esti=efti-1
vk的最早结束时间是所有任务的最早结束时间中最大的。
```

```
CRITICAL_PATH(v):
v.color = GRAY;

+v.est=-∞;v.CritDep = -1;

<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w);
        <Backtrack processing of edge vw>;
        if w.eft > v.est then
            v.est = w.eft;
            v.CritDep = w;
    else
        <check edge vw>;
        if w.eft > v.est then
            v.est = w.eft;
            v.CritDep = w;
<Postorder processing of node v>
v.eft = v.est + v.l;
globalNum--;
v.topoNum=globlanum;//分配拓扑序号
v.color = BLACK;
```

数据结构课上的算法是每次都找入度为0的节点，而且还可以判断是不是这真的是有向无环图，上面的算法是在已知有向无环的情况下求出一个拓扑排序。

有向图中的强连通片和收缩图

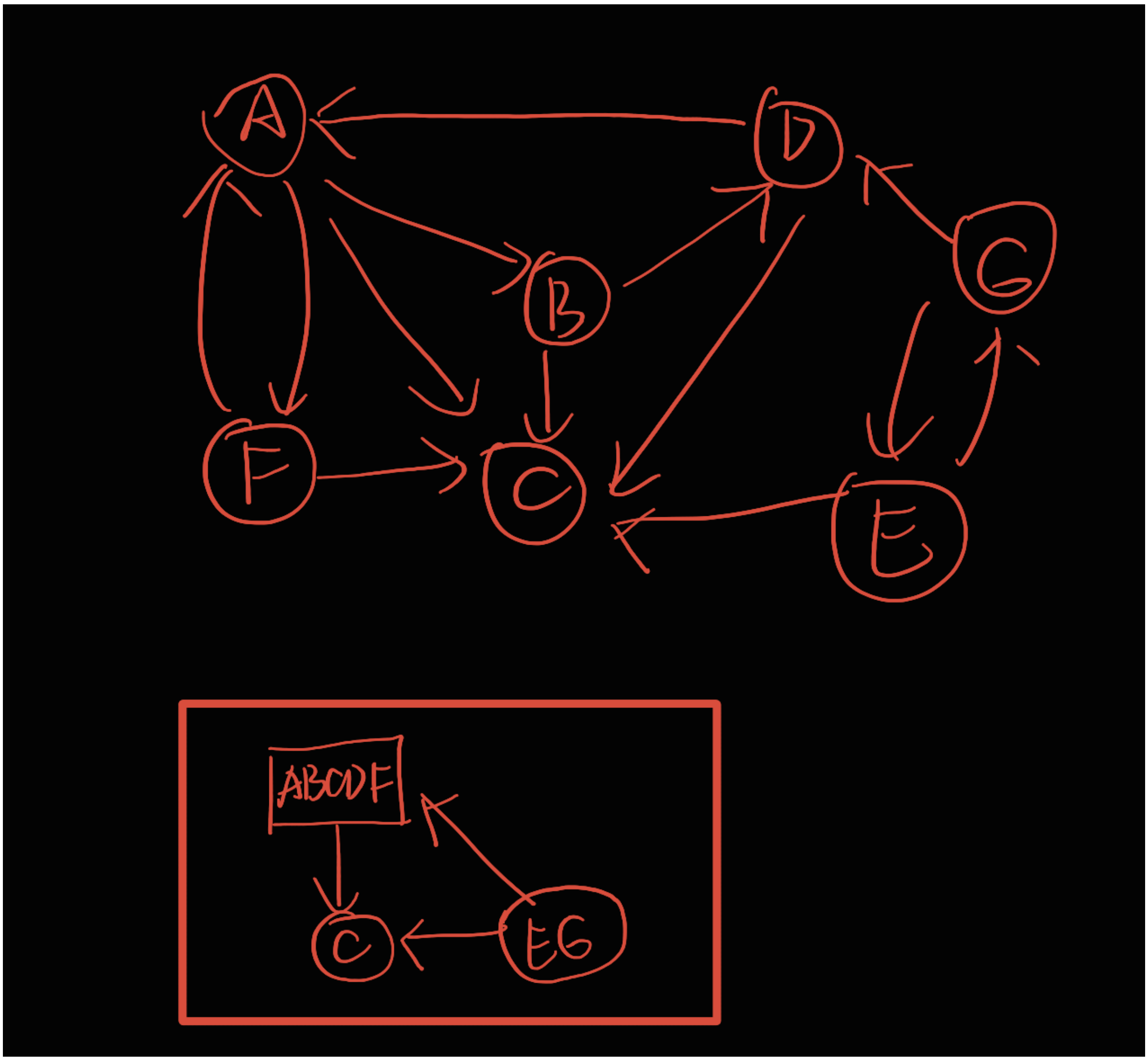
强连通片：有向图中的两个节点任意可达

收缩图，以强连通片为一个单位收缩成一个点，强连通片之间的边收缩成一条有向边

收缩图的性质

> 收缩图一定是有向无环图（反证法易证

从收缩图中的一个无出度的强连通片中的一个点出发，就可以遍历该连通片的所有节点。

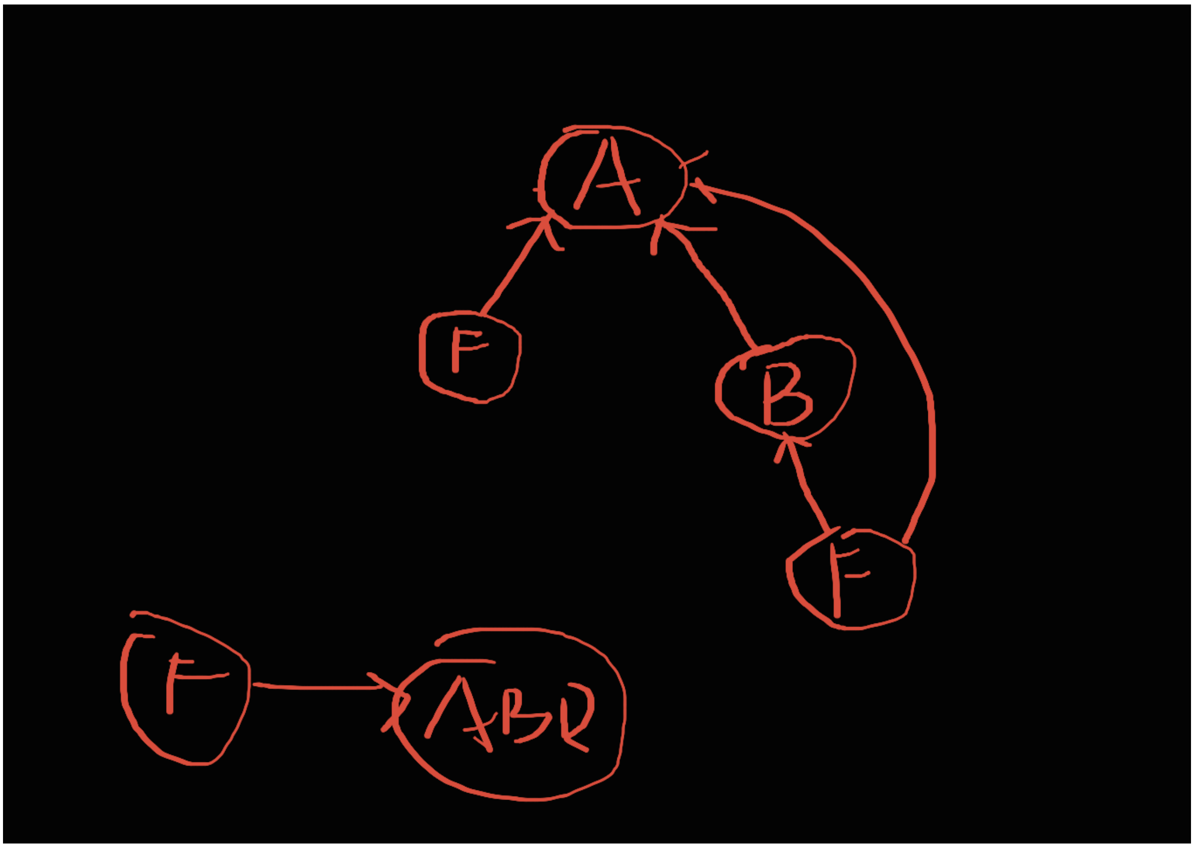


先从c出发，遍历第一个强连通片，然后从abcdf中的一个点出发，就能遍历第二个强连通片……

sink SCC是指收缩图里面没有出度的节点。

遍历得到所有强连通片的关键是寻找到第一个节点：这个节点是转置的G中finishtime最大的节点。（c就是这样的点，对应在原图里面没有出度)

Q：为什么不找G中finishtime最小的节点？这样找出来的不一定是想要的点，比如说下面这种情况：



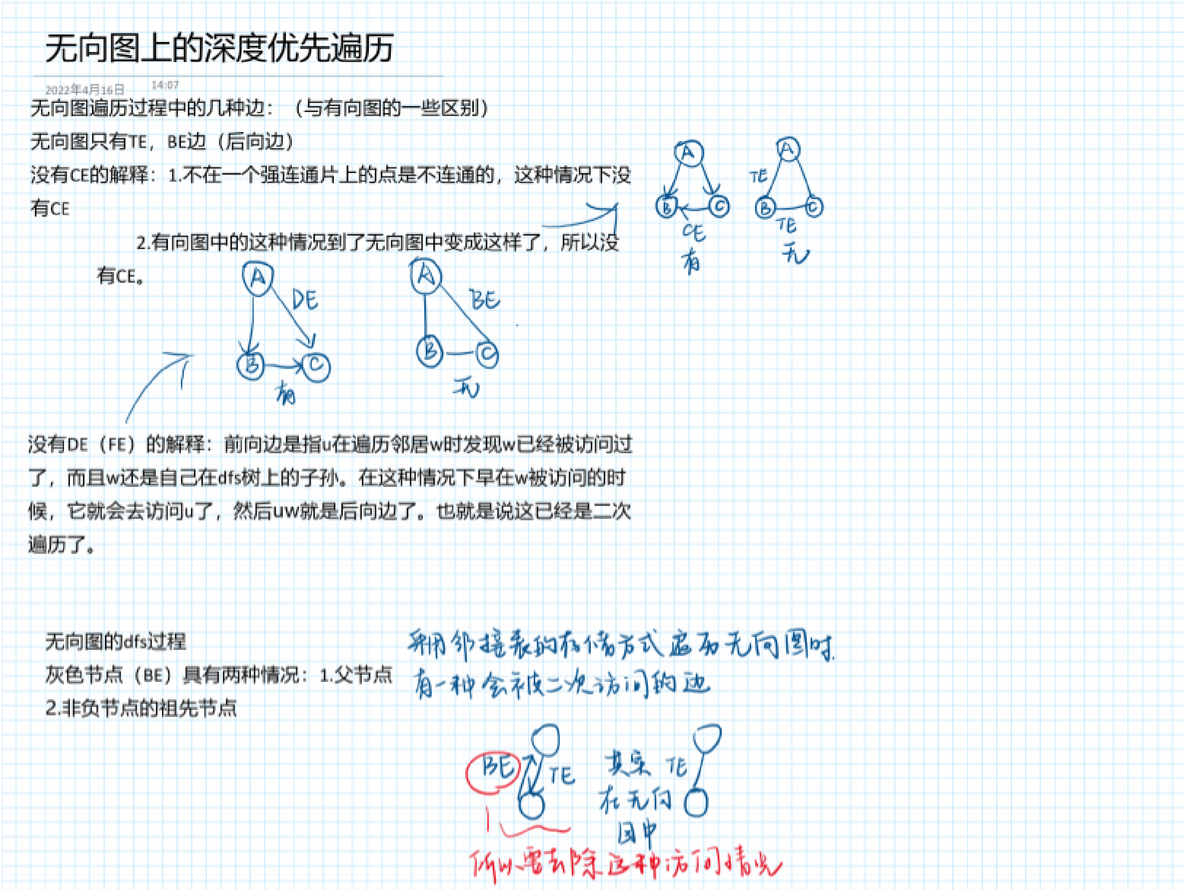
在这样的情况下，最大finishtime的是F，并不是预期中的ABD中的某一个

有向图的转置：把边的关系都转置一下

```
找到转置G中的finishtime最大的顶点。按finishtime排一个序（用堆实现比较好）
从该顶点开始在G中做一次DFS，找出其连通片上的点，每找到一个就删掉一个节点，然后找到finishtime最大的那一个，继续dfs，直到存finishtime的地方为空
```

无向图的深度优先遍历

无向图的遍历树上的边


无向图上的深度优先遍历

2022年4月16日 14:07

无向图遍历过程中的几种边：（与有向图的一些区别）

无向图只有TE，BE边（后向边）

没有CE的解释：1.不在一个强连通片上的点是不连通的，这种情况下没有CE

2.有向图中的这种情况到了无向图中变成这样了，所以没有CE。

没有DE（FE）的解释：前向边是指u在遍历邻居w时发现w已经被访问过了，而且w还是自己在dfs树上的子孙。在这种情况下早在w被访问的时候，它就会去访问u了，然后uw就是后向边了。也就是说这已经是二次遍历了。

无向图的dfs过程

灰色节点（BE）具有两种情况：1.父节点 2.非负节点的祖先节点

采用邻接表的存储方式遍历无向图时 有一种会被二次访问的边

所以要去除这种访问情况

无向图的边：TE边以及非父子的BE边

```
DFS(v,parent):
v.color = GRAY;
<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w,v);
        <Backtrack processing of edge vw>;
    else
        if w.color == GRAY && w!=parent then
        <check edge vw>;
<Postorder processing of node v>
v.color = BLACK;
```

无向图上dfs的应用

2-点连通：任意去掉图中某一点之后，剩下的图依然保持连通。

定义：割点、桥 （删去之后不再连通）

割点基于路径的定义：点v为割点当且仅当存在w和x满足w到x的所有路径一定结果v。
证明：

<=:易证没了v之后w就没有到x的一条连通的路径了。

=>:反证法

割点基于dfs的定义：v不是遍历的根节点，v为割点当且仅当在遍历树中**存在v的一棵子树，没有任何BE指向v的祖先**。

寻找割点的算法：

```
DFS(v,parent):
v.color = GRAY;
time = time+1;
v.discovertime = time;
v.back = v.discovertime;
<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w,v);
        if w.back >= v.discovertime then
            output v as an articulation point;
        <Backtrack processing of edge vw>;
    else
        if w.color == GRAY && w!=parent then
            v.back = min(v.back,w.discovertime)
        <check edge vw>;
<Postorder processing of node v>
v.color = BLACK;
```

```
if w.back >= v.discovertime then
            output v as an articulation point;
```

也就是说存在节点回溯上去时跟v的祖先没有BE边，所以说v是割点。

寻找桥的算法

桥基于dfs的定义：给定遍历树中的TE边的uv(u是父节点)，uv是桥，当且仅当以v为根的所有遍历树子树中没有BE指向v的祖先。

```
DFS(v,parent):
v.color = GRAY;
time = time+1;
v.discovertime = time;
v.back = v.discovertime;
<Preorder processing of node v>
foreach neighbor w of v do:
    if w.color == WHITE then
        <Exploratory processing of edge vw>;
        DFS(w,v);
        v.back = min(v.back,w.back);
        if w.back > v.discovertime then
            output uv as a bridge;
        <Backtrack processing of edge vw>;
    else
        if w.color == GRAY && w!=parent then
            v.back = min(v.back,w.discovertime)
        <check edge vw>;
<Postorder processing of node v>
v.color = BLACK;
```