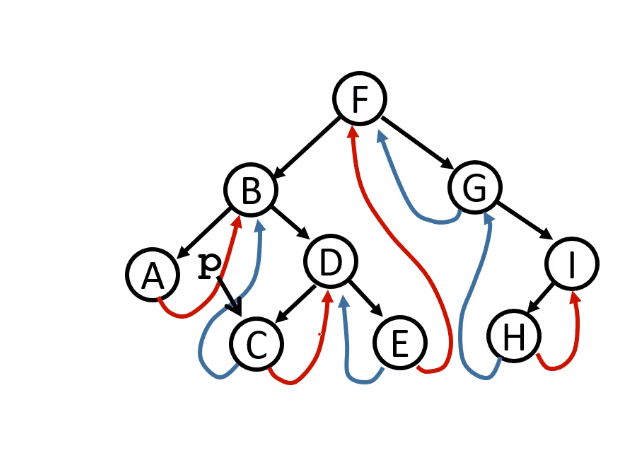
**二叉树线索-中序**

一、

·我们先上个图，来看一下，我们该怎么一步一步实现



·构造二叉树思路

1. 由于我们需要遍历所有的节点，所以我们只需在中序比哪里的基础上进行，把原来的visit操作改成我们相应的操作
2. 我们此时还要在原来的基础上添加两个标记（LTag、RTag），pre=NULL
3. 我们第一次进行操作的是D节点，此时它有LTag、RTag
4. 如果我们当前左孩子是空闲的话，我们就利用起来，如果pre的有孩子是空的话，我们也利用起来
5. 我们应该让D的左孩子指向NULL=pre，且LTag=1；因为我们有指向
6. 4操作结束后，我们就判断pre是否为空&&pre->right\_child是否为空，如果为空，我们就利用其的RTag
7. Pre=p 保证pre永远在p之前的一个元素

·遍历

1. 我们需要一直往左孩子走，走到D节点
2. Visit（）
3. 如果Tag=0说明有指针，如果Tag=1说明有线索，所以直接p=p->right\_child
4. 那么如果我们的RTag=0，说明有右指针，那么我们指向它的右孩子，再一路遍历到它的左孩子
5. 重复上面的操作即可

·全部代码

// 线索二叉树.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <iostream>

using namespace std;

typedef struct BitTree {

char data;

BitTree\* right\_child, \* left\_child;

int RTage, LTage;

}BitNode;

//初始化一个空节点

BitNode\* newNode() {

BitNode\* p = (BitNode \*)malloc(sizeof(BitNode)); //开辟一个空间

p->left\_child = p->right\_child = 0;

p->RTage = p->LTage = 0;

return p;

}

//构建线索中序

void creat\_ind\_threading(BitNode\* p, BitNode\* &pre) {

//为空我们就return回去就可以了

if (!p) return;

//如果不为空，我们来进行下面的操作，就是整体结构是按照中序遍历的

creat\_ind\_threading(p->left\_child, pre);

//具体操作

if (!p->left\_child) { //如果左孩子是闲置的话，我们就进行操作

p->left\_child = pre;

p->LTage = 1;

}

if (pre && !pre->right\_child) { //若pre的右孩子闲置的话

pre->right\_child = p;

pre->RTage = 1;

}

pre = p;

//具体操作结束

creat\_ind\_threading(p->right\_child, pre);

}

// 现在我们来遍历输出一下

void print\_tree(BitNode\* T, void(\*visit)(const BitNode\* cur\_p)) {

if (!T) return;

BitNode\* p = T;

while (p->left\_child) {

p = p->left\_child;

}

while (p) {

visit(p);

if (!p->RTage) { //我们看p的右还是是不是闲置的，如果是我们进去到下面

p = p->right\_child;

while (p&&!p->LTage) {

p = p->left\_child;

}

}

else { //如果不是闲置的，说明，我们可以利用该指针访问到下一个要访问的节点

p = p->right\_child;

}

}

}

void print\_e(const BitNode \*T) {

cout << T->data << " ";

}

int main()

{

/\*现在我们先来构造以可ABCDE的树

A

/ \

B C

/ \

D E\*/

BitNode\* T = newNode(); T->data = 'A';

T->left\_child = newNode(); T->left\_child->data = 'B';

T->right\_child = newNode(); T->right\_child->data = 'C';

BitNode\* p = T->left\_child;

p->left\_child = newNode(); p->left\_child->data = 'D';

p->right\_child = newNode(); p->right\_child->data = 'E';

//现在书写我们的构建线索二叉树的代码

BitNode\* pre = 0;

creat\_ind\_threading(T, pre);

cout << "线索二叉树构造结束" << endl;

print\_tree(T, print\_e);

cout << "结束" << endl;

}