資料結構 F74109016_hw11 葉惟欣

```
1  //finsihed in 2021/11/06 葉惟欣 F74109016
2  #include<stdio.h>
3  #include<stdlib.h>
4  typedef int bool;
5  enum { false, true };
6  typedef struct threadedTree *threadedPointer;
7
8  struct threadedTree{
9      threadedPointer leftChild;
10     int data;
11     threadedPointer rightChild;
12     bool rightThread;
13     bool leftThread;
14 };
15
16 threadedPointer insucc(threadedPointer tree){
17     threadedPointer temp;
18     temp = tree->rightChild;
19     if(tree->rightThread==false){
20         while(temp->leftThread==false){
21             temp = temp->leftChild;
22         }
23     }
24     return temp;
25 }
26 threadedPointer inasce(threadedPointer tree){
27     threadedPointer temp;
28     temp = tree->leftChild;
29     if(tree->leftThread==false){
30         while(temp->rightThread==false){
31             temp = temp->rightChild;
32         }
33     }
34     return temp;
35 }
```

Inasce 函數是用來找中序的錢個節點的函數

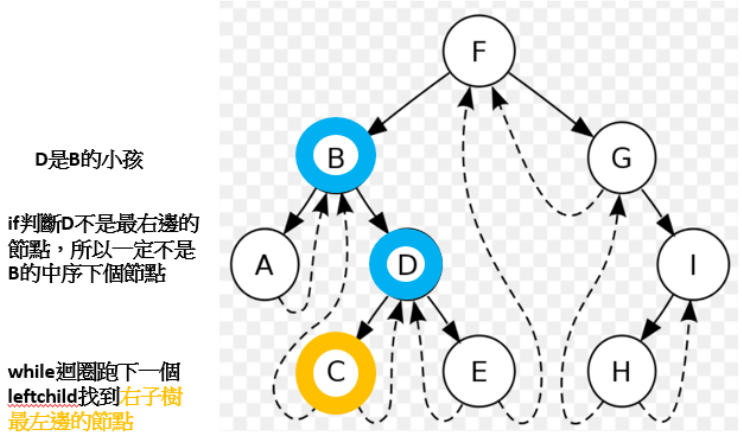
insucc 函數是用來找中序的下個節點的函數 (以找下個節點來詳細說明)

此為 threaded binary tree 最大的優點➔中序採訪不用用遞迴又或是 stack

temp 是用來裝節點 tree 的右小孩 (第 18 行)

接下來的 19~23 行 的 “判斷子句” 是用來看 “右小孩” 是否是整個樹最右邊的節點。

當不是最右邊的節點時就跑下面的 While 迴圈找到右小孩所屬子樹的最左邊的小孩，這樣才是中序的下個節點。



```
void inorder(threadedPointer tree){
    threadedPointer temp = tree;
    for(;;){
        if(temp != NULL){
            printf("%d ",temp->data);
            temp = insucc(temp);
        }
        else{
            break;
        }
    }
}

threadedPointer tinorder(threadedPointer tree,int _id){
    threadedPointer temp = tree;
    for(;;){
        if(temp!= NULL){
            if(temp->data == _id){
                return temp;
            }
            temp = insucc(temp);
        }
        else{
            return NULL;
        }
    }
}
```

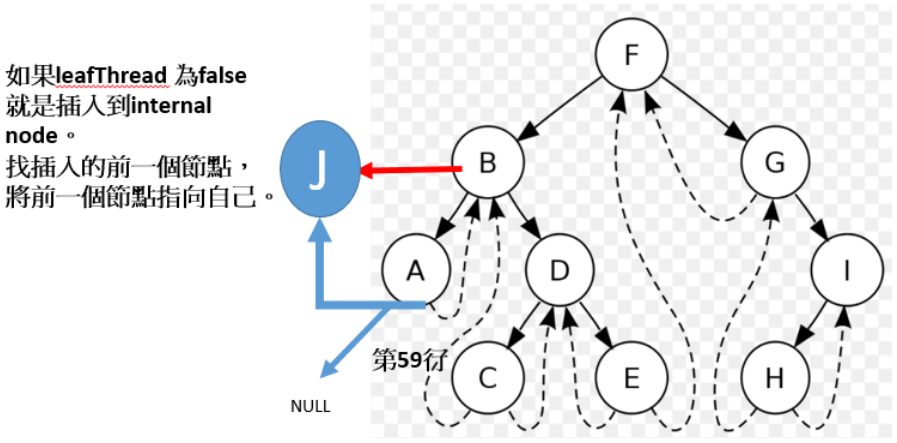
第一個函數 inorder 是最後要印出的中序採訪順序。當還找的到下一個右小孩就繼續印出來。

第二個函數是透過中序採訪(insucc 函數)找節點的位置，並且回傳。

➔任何一個節點都容易找出它的中序後繼者與中序先行者。

```
void insertLeft(threadedPointer s,threadedPointer l){
    threadedPointer temp;
    l->leftChild = s->leftChild;
    l->leftThread = s->leftThread;
    l->rightChild = s;
    l->rightThread = true;
    s->leftChild = l;
    s->leftThread = false;
    if(l->leftThread==false){
        temp = inasce(l);
        temp->rightChild = l;
    }
}

void insertRight(threadedPointer s,threadedPointer r){
    threadedPointer temp;
    //狀況一
    r->rightChild = s->rightChild;
    r->rightThread = s->rightThread;
    r->leftChild = s;
    r->leftThread = true;
    s->rightChild = r;
    s->rightThread = false;
    //狀況二
    if(r->rightThread==false){
        temp = insucc(r);
        temp->leftChild = r;
    }
}
```




個節點。

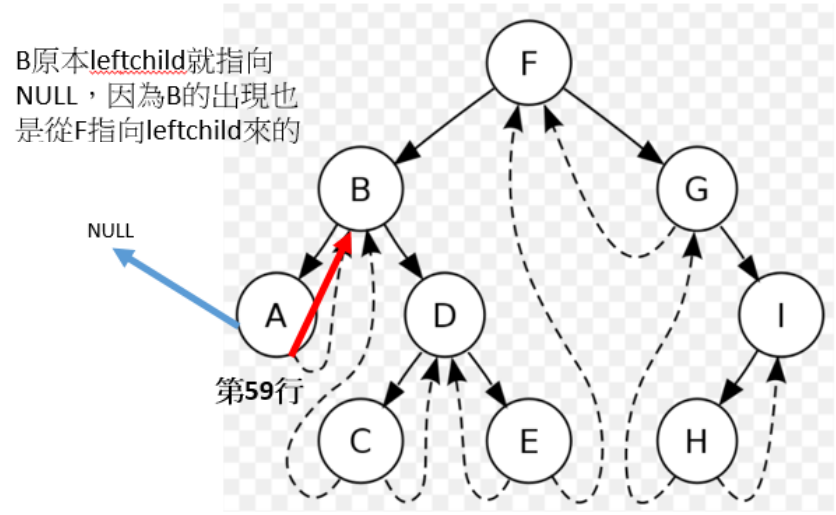
一個節點。

當為 false 時，則代表 rightChild 為指標，指向在下個的指標。

| Left Thread | Lchild | Data | Rchild | Right Thread |
|---|---|---|---|---|

下面以插入左邊做說明的兩種情況

插入到左邊節點的 leaf




插入到左邊節點的中間

leftMost 函數

```
93  threadedPointer leftMost(threadedPointer n){
94      if(n == NULL){
95          return NULL;
96      }
97      while(n->leftChild != NULL){
98          n = n->leftChild;
99      }
100     return n;
101 }
```

此函數一直在找最左邊的函數，這樣之後找要插入節點的位置時才能從最左邊開始找，中序採訪才不會漏掉節點。

Root 的節點所指向的左小孩與右小孩都預設為 NULL，這樣找最左小孩與中序採訪才不會繞進無群迴圈。

```
int main(){
    int n,r;

    scanf("%d %d",&n,&r);
    //n represents the number of threaded binary tree insertion operation.
    //r represents the node id of the root node.

    int i,parent_id,direction,new_node_id;          //direction 0:左 1:右
    char string [20];
    int direction_array[n+1];
    int parent_array[n+1];
    int new_node_array[n+1];

    threadedPointer root = (threadedPointer) malloc (sizeof(struct threadedTree));
    root->leftThread = true;
    root->rightThread = true;
    root->rightChild = NULL;
    root->leftChild = NULL;
    root->data = r;

    for(i=1;i<n+1;i++){
        scanf("%d",&parent_array[i]);
        scanf("%s",string);
        if(strcmp(string,"left")==0){
            direction_array[i] = 0; //0:左
        }
        if(strcmp(string,"right")==0){
            direction_array[i] = 1;
        }
        scanf("%d",&new_node_array[i]);

        threadedPointer leftmost = leftMost(root);
```

先找到最左小孩，再由最左小孩找到想插入位置的 parent。在 insert 進去即可。

```
        if(direction_array[i]==0){
            threadedPointer l = (threadedPointer) malloc (sizeof(struct threadedTree));
            l->leftThread = false;
            l->rightThread = true;
            l->rightChild = NULL;
            l->leftChild = NULL;
            l->data = new_node_array[i];
            threadedPointer s = tinorder(leftmost,parent_array[i]);
            insertLeft(s,l);
        }

        if(direction_array[i]==1){
            threadedPointer r = (threadedPointer) malloc (sizeof(struct threadedTree));
            r->leftThread = true;
            r->rightThread = false;
            r->rightChild = NULL;
            r->leftChild = NULL;
            r->data = new_node_array[i];
            threadedPointer s = tinorder(leftmost,parent_array[i]);
            insertRight(s,r);
        }
    }
    threadedPointer leftmost1 = leftMost(root);
    inorder(leftmost1);
}
```