Normal Tree Traversals

Author：secret

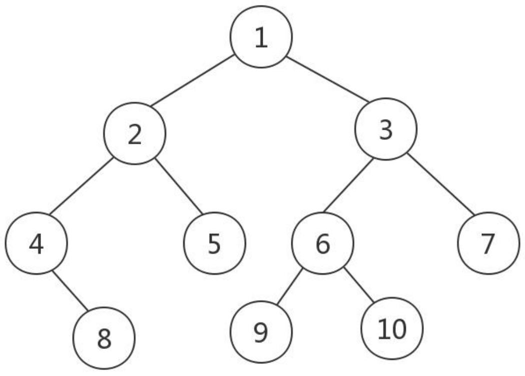
Date: 2021-10-26

Chapter 1: Introduction

Given the partial results of a binary tree's traversals in in-order, pre -order, and post-order. You are supposed to output the complete results and the level order traversal sequence of the corresponding tree.

Each input file contains one test case. For each case, a positive integer N (≤100) is given in the first line. Then three lines follow, containing the incomplete in-order, pre-order and post-order traversal sequences, respectively. It is assumed that the tree nodes are numbered from 1 to N and no number is given out of the range. A - represents a missing number.

For each case, print in four lines the complete in-order, pre-order and post-order traversal sequences, together with the level order traversal sequence of the corresponding tree. The numbers must be separated by a space, and there must be no extra space at the beginning or the end of each line. If it is impossible to reconstruct the unique tree from the given information, simply print Impossible.ground of the algorithms.



Chapter 2: Algorithm Specification

```
Algorithm 1:int check(int h1, int h2, int h3, int l){
    If the length of section=0
    Then this tree is traversaled without failure, return 1;
    For( i from the head of preorder to length, i++){
        If the head of preorder != the end of postorder
            Then this tree can’t be created correctly, return 0;
        Enumerate every data of inorder to see if it can become root;
        If the possible root appears three times
            Then this data can be a root;
        If the possible root appears once or twice
            Then to see if we can complete the missing data ;
        If the possible root never appear
            Then we can complete the order using missing number;
        Then check the left subtree;
        Then check the right subtree;
        If the left subtree and right subtree are right
            Then return 1;
    }
      Else
            It can't create the unique tree, Return 0;
    }
```

```
Algorithm 2: int main(){
        Input the total n;
        Input inorder, preorder, postorder, record missing_cnt;
        If missing_cnt is more than1
                Then we can't create the unique tree because missing data can replace each other;
        Put the inorder, preorder, postorder array into function check;
        If  the return of check is 1
                Then we can create unique tree;
                Then output the four kind of complete order;
        Else
                Output "Impossible";
}
```

```
Algorithm 3: void LevelTraversal(BinTree BT){
        Create an array a[] to store the data like a queue;
        Put the root of tree into a[0];
        Then put the left child and right child into the array;
        Create temp to store the output data;
        Every time output the data, change root into next data;
        When the array is empty, finish traversal;
}
```

Chapter 3:   Testing Results

Sample Input 1: //there is no missing data, it can create correct tree

9

3 - 2 1 7 9 - 4 6

9 - 5 3 2 1 - 6 4

3 1 - - 7 - 6 8 -

Sample Output 1: //pass

3 5 2 1 7 9 8 4 6

9 7 5 3 2 1 8 6 4

3 1 2 5 7 4 6 8 9

9 7 8 5 6 3 2 4 1

Sample Input 2:

//missing data is more than one, it can’t create tree

3

- - -

- 1 -

1 - -

Sample Output 2://pass

Impossible

Sample Input 3: //the number of missing data is 1,it can create correct tree

9

3 - 2 1 7 9 - 4 6

9 - - 3 2 1 - 6 4

3 1 - - 7 - 6 8 -

Sample Output 3://pass

3 5 2 1 7 9 8 4 6

9 7 5 3 2 1 8 6 4

3 1 2 5 7 4 6 8 9

9 7 8 5 6 3 2 4 1

Sample Input 4: //there is only 1 data, the Bin_Tree is unique

1

-

-

-

Sample Output 4://pass

1

1

1

1

Sample Input 5: //left_single tree, it can create correct tree

9

1 2 3 4 5 6 7 8 -

- 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 -

Sample Output 5://pass

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

Sample Input 6: //appear two possible root when checking,it can’t create tree

9

1 2 - 6 7 8 - 5

- 7 6 8 2 1 - 5

7 4 6 8 2 5 3 1

Sample Output 6://pass

Impossible

Sample Input 7: //there is double-digit in input,it can create tree

10

1 2 3 4 5 6 7 8 9 10

- 9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9 -

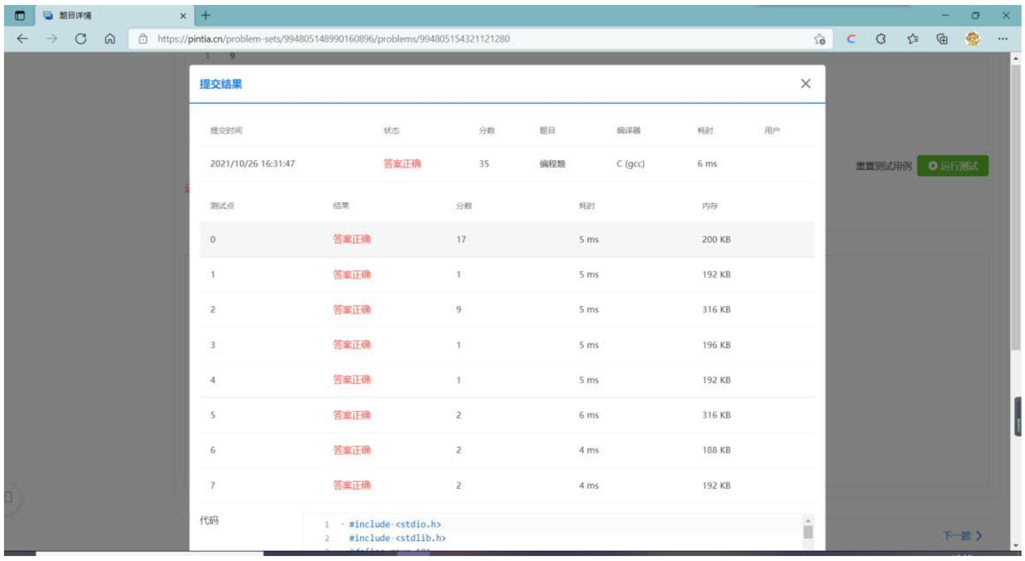
Sample Output 5://pass

1 2 3 4 5 6 7 8 9 10

10 9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9 10

10 9 8 7 6 5 4 3 2 1


题目详情
https://pintia.cn/problem-sets/994805148990160896/problems/994805154321121280
重置测试用例
运行测试
下一题

提交结果

提交时间	状态	分数	题目	编译器	耗时	用户
2021/10/26 16:31:47	答案正确	35	编程题	C (gcc)	6 ms	

测试点	结果	分数	耗时	内存
0	答案正确	17	5 ms	200 KB
1	答案正确	1	5 ms	192 KB
2	答案正确	9	5 ms	316 KB
3	答案正确	1	5 ms	196 KB
4	答案正确	1	5 ms	192 KB
5	答案正确	2	6 ms	316 KB
6	答案正确	2	4 ms	188 KB
7	答案正确	2	4 ms	192 KB

代码

```
#include <stdio.h>
#include <stdlib.h>
```

图 1- testing result in PTA

Chapter 4: Analysis and Comments

In the Algorithm :

```
int check(int h1, int h2, int h3, int l){
……
        int b1 = check(h1,h2+1,h3,l1); //check the l_child tree
        int b2 = check(h1+l1+1,h2+1+l1,h3+l1,l2);
……}
```

We can see clearly that T(N) = T(N/2)+T(N/2), then T(N/2) = T(N/4)+T(N/4)……， finally, the time complexities is O(logN). because we must traversal N data in three order, we should take 3N memory, so the space complexities is O(N)

In the Algorithm :

```
void LevelTraversal(BinTree BT){
   BinTree a[101];
   int i=0,j=0;
   a[0] = BT;
   while(BT){
      if(BT->lchild!=NULL){
         a[++i] = BT->lchild;}
      if(BT->rchild!=NULL){
         a[++i] = BT->rchild;}
      BinTree temp = a[j];
```

```
        printf("%d",temp->data);
        if(i==j) break;
        else printf(" ");
        j++;
        BT = a[j];  }
    printf("\n");}
```

I use array to replace the queue so that I don’t need to write the code of queue, and it can make the code shorter. This algorithm’s time complexities is O(logN). This array should use N memory, so the space complexities is O(N). But this array has maxn data, so when N is very big, this algorithm may Consume too much memory. If we use queue, we can solve this problem better.

Appendix: Source Code (in C)

```c
#include <stdio.h>
#include <stdlib.h>
#define maxn 101


typedef struct TNode{ //creates struct tree
    int data;
    struct TNode *lchild,*rchild;
}TNode,*BinTree;

BinTree CreateTree(int A[],int B[],int n){ //creates binary tree
    BinTree BT = (BinTree)malloc(sizeof(struct TNode)); //malloc the memory
    if(n<=0) return NULL; // if tree is empty,creating fail
    else{ //if tree is not empty
        int temp = A[n-1];
        int i;
        for(i=0;i<n;i++){   //find the index of root node in inorder
            if(temp== B[i]){
                break;
            }
        }
        BT->data = temp;  //creates the tree node
        BT->lchild = CreateTree(A,B,i);//travels left son_tree
        BT->rchild = CreateTree(A+i,B+i+1,n-i-1);//travels right son_tree
        return BT;
    }
}

void LevelTraversal(BinTree BT){ //level_travelsal
    BinTree a[101];  //create an array to memory tree nodes
    int i=0,j=0;     //define variables这个数组作为队列来用
    a[0] = BT;       //memory tree nodes into array a[0]
    while(BT){
        if(BT->lchild!=NULL){
            a[++i] = BT->lchild;
        }            //every time when you visit root node,and put l_child_node into array
        if(BT->rchild!=NULL){
            a[++i] = BT->rchild;
        }            //every time when you visit root node,and put l_child_node into array
        BinTree temp = a[j];
        printf("%d",temp->data);
        if(i==j) break;//this level completes, don't need to add spacing
        else printf(" "); //add spacing
        j++;
        BT = a[j];   //travelsal all the tree
    }
    printf("\n");
}

int inorder[maxn], preorder[maxn], postorder[maxn], appear_time[maxn]={0}; //record initial data, appear_time records appear times
int reinorder[maxn], repreorder[maxn], repostorder[maxn]; //record the complete array
int miss_cnt , miss_number; //miss_cent records the number of missing_umber, miss_number records missing data
int n,flag; //n is the number of nodes, flag is judging variable
int check(int h1, int h2, int h3, int l); //check if the section can create the only BinTree

int get(void);//change "-" into "0" and transfer data into int
int get(void) {
    int tmp, i;
    char str[10]; //define array of char
    scanf("%s", str);
    if (str[0] == '-') {
        return 0; //change "-" into "0"
    }
    else {
        tmp = 0;
        for (i = 0; str[i]; i++) {
            tmp = tmp * 10 + str[i] - '0'; //in case that input double-digit
        }
        return tmp; //return the final int
    }
}

int main(void) {
    int i;
    scanf("%d", &n);

    for (i = 0; i < n; i++) {inorder[i] = get(); appear_time[inorder[i]]++; //put the input data into array inorder[]
    if(inorder[i] != 0) reinorder[i] = inorder[i];}//if the input is not 0,then put it into complete array
    for (i = 0; i < n; i++) {preorder[i] = get(); appear_time[preorder[i]]++; //put the input data into array preorder[]
    if(preorder[i] != 0) repreorder[i] = preorder[i];}//if the input is not 0,then put it into complete array
    for (i = 0; i < n; i++) {postorder[i] = get(); appear_time[postorder[i]]++; //put the input data into array postorder[]
    if(postorder[i] != 0) repostorder[i] = postorder[i];}//if the input is not 0,then put it into complete array

    miss_cnt = 0;
    for (i = 1; i <= n; i++) {
      if (appear_time[i]==0){
          miss_number = i;
          miss_cnt++;} //if i never appear, the add the miss_cnt
      }

    if(n==1&&inorder[0]==0&&preorder[0]==0&&postorder[0]==0){
        for(i=0;i<4;i++){
            printf("1\n"); //if the tree has only one node, we can create the tree
        }
        return 0;
    } //even if there is no element, still can create the only Bin_Tree
    if(miss_cnt > 1 ) {
        printf("Impossible");
        return 0;
    } //if the number of missing is more than one, then fail

    flag = check(0, 0, 0, n); //input flag

    if(n==1&&flag==1){
        int a=reinorder[0];
        for(i=0;i<4;i++)
        printf("%d\n",a);
    }
    if(flag == 1&&n>1){ //output the complete inorder queue
        for(i=0;i<n;i++){
            if(i==0)
            printf("%d",reinorder[0]); //output the first data
            if(i>0&&i<n-1)
            printf(" %d",reinorder[i]); //output the other data
            if(i>0&&i==n-1)
            printf(" %d\n",reinorder[i]); //when finish, newline
        }
        for(i=0;i<n;i++){ //output the complete inorder queue
            if(i==0)
            printf("%d",repreorder[0]); //output the first data
            if(i>0&&i<n-1)
            printf(" %d",repreorder[i]);  //output the other data
            if(i>0&&i==n-1)
            printf(" %d\n",repreorder[i]); //when finish, newline
        }
        for(i=0;i<n;i++){ //output the complete inorder queue
            if(i==0)
            printf("%d",repostorder[0]); //output the first data
            if(i>0&&i<n-1)
            printf(" %d",repostorder[i]); //output the other data
            if(i>0&&i==n-1)
            printf(" %d\n",repostorder[i]); //when finish, newline
        }
        BinTree BT = CreateTree(repostorder,reinorder,n); //creat the BinTree according the repostorder and reinorder
        LevelTraversal(BT); //level traversal the tree
    }
    if(flag ==0){
        printf("Impossible");
        return 0;
    }
    return 0;
}

int check(int h1, int h2, int h3, int l){//h1 is the head inorder,h2 is head of preorder,h3 is head of postorder,l is length
    if(l == 0) return 1;//l=0,then finish the traversal,can create correct tree
    int l1,k=0;
    for(l1 =0; l1<l; l1++){
        int l2 = l-l1-1;
        int size = 0; //record the number of possible root
        int cnt = 0; //record the appear times of possible root
        int root_number;
        if(preorder[h2] > 0 && postorder[h3+l-1] > 0 && preorder[h2] != postorder[h3+l-1]){
            return 0;  //if the head of preorder!=the tail of postorder,it is impossible to create the tree
        }
        if(inorder[h1+l1] > 0) {
            size++;      //traversal the inorder to find the possible root
            cnt++;
            root_number=inorder[h1+l1];
        }
        if(preorder[h2] > 0){
            cnt++;       //traversal the preorder to find the possible root
            if(inorder[h1+l1] > 0&&preorder[h2] != inorder[h1+l1])
            size++;
            if(inorder[h1+l1] == 0)
            root_number = preorder[h2];
        }
        if(postorder[h3+l-1] > 0){
            cnt++;       //traversal the postorder to find the possible root
            if((inorder[h1+l1]>0&&postorder[h3+l-1]!=inorder[h1+l1])||(preorder[h2] > 0&&postorder[h3+l-1]!=preorder[h2]))
            size++;
            if(inorder[h1+l1] == 0&&preorder[h2]==0)
            root_number = postorder[h3+l-1];
        }
        if(size > 1) //if size>1, then appear more than one possible root,then fail
        continue;
        if(cnt == 0){ //if size = 0,then three order missing
            if(miss_cnt == 0) continue; //if there is no missing,then fail
            reinorder[h1+l1] = miss_number; //complete the order
            repreorder[h2] = miss_number; //complete the order
            repostorder[h3+l-1] = miss_number; //complete the order
            miss_cnt--; //if complete,then miss_cnt decline
        }
        else{ //if size=1,this level is correct

            if(cnt != appear_time[root_number]) continue; //check if the number equal with appear_time
            reinorder[h1+l1] = root_number; //complete missing number
            repreorder[h2] = root_number; //complete missing number
            repostorder[h3+l-1] = root_number; //complete missing number
        }
        int b1 = check(h1,h2+1,h3,l1); //check the l_child tree
        int b2 = check(h1+l1+1,h2+1+l1,h3+l1,l2); //check the r_child tree
        if(!cnt)
            miss_cnt = 1;
        if(b1==1&&b2==1) return 1;
    }
    return 0;
}
```

Declaration

I hereby declare that all the work done in this project titled "Normal Tree Traversals" is of my independent effort