Fundamentals of Data Structures

Laboratory Projects 2

Tree Traversals

Chapter 1: Introduction

1.1 The content of the question

Given the partial results of a binary tree's traversals in in-order, pre-order, post-order.In this task, we are supposed to build a tree. After having solved the problem, we need to output the complete traversals of in-order, pre-order, post-order, and level order.

In the input, the first line of the input is the number of nodes you need to deal with in the tree. A `-` represents the current number is missing, you are supposed to find it using your program. Finally, output your answer.

1.2 The input and output sample(given by the question)

Sample Input 1

```
9
3 - 2 1 7 9 - 4 6
9 - 5 3 2 1 - 6 4
3 1 - - 7 - 6 8 -
```

Sample Output 1

```
3 5 2 1 7 9 8 4 6   /*in-order*/
9 7 5 3 2 1 8 6 4   /*pre-order*/
3 1 2 5 7 4 6 8 9   /*post-order*/
9 7 8 5 6 3 2 4 1   /*lever-order*/
```

Sample Input 2

```
3
- - -
- 1 -
1 - -
```

Sample Output2

```
Impossible
```

Chapter 2: Algorithm Specification

2.1 Searching Algorithm(searching for the node)

Pseudo Code

```
int Search(int L1,int R1,int L2,int R2,int L3,int R3)
{
  int root=pre[L2]||post[R2];
  /*The last node of post-order or the first node of pre-order is the root number.*/
  for(int i=L1;i<=R1;i++)
    if(in[i]==root) p=i; /*find the root*/
  while(cannot find the root)
  {
    if(in[i]==0) p=i;  /*traverse all the i to find the position that have chance to be
the root*/
    for(int j=L2+1;j<=L2+length;j++)
      for(int k=L3;k<=L3+length;k++)
      {
        judge(root);/*judge whether it's the root*/
        if(root)  break;
      }
    i++;
    /*using enumeratio to judge the root in in-order.*/
  }
  lc[root]=Search(newL1,newR1,newL2,newR2,newL3,newR3);//searching in the left subtree
  rc[root]=Search(newwL1,newwR1,newwL2,newwR2,newwL3,newwR3);//searching in the right
subtree
  return root;
}
```

2.2 BFS Algorithm

Pseudo Code

```
void bfs(int root)
{
  int level[105]; //It's used to store the traversal of level order.
  int head=-1,tail=0;
  level[0]=root;
  while(head!=tail)
  {
    head++;
    if(lc[level[head]]) level[++tail]=lc[level[head]]//have a left child
    if(rc[level[head]]) level[++tail]=rc[level[head]]//have a right child
  }
}
```

2.3 Main Function

```
int main()
{
  scanf("%d",&n);
  input(in[]);
  input(pre[]);
  input(post[]);
  int root = Search(1,n,1,n,1,n);
  bfs(root);
  output(in[]);
  output(pre[]);
  output(post[]);
  output(level[]);
}
```

Chapter 3: Test cases

3.1 Case 1: The Binary tree is a Queue

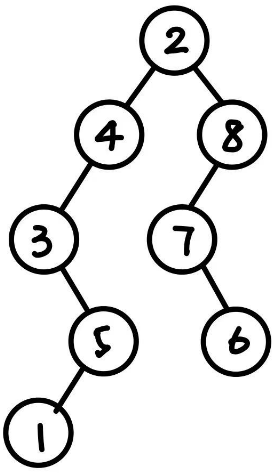
Input Sample

```
9
3 5 9 - 4 2 7 - 6
2 4 - 3 9 1 8 - 6
3 - 9 5 4 7 - 8 2
```

Output Sample

```
3 5 9 1 4 2 7 8 6
2 4 5 3 9 1 8 7 6
3 1 9 5 4 7 6 8 2
2 4 8 5 7 6 3 9 1
```

The binary tree



3.2 Case 2: The tree have 2 possibilities

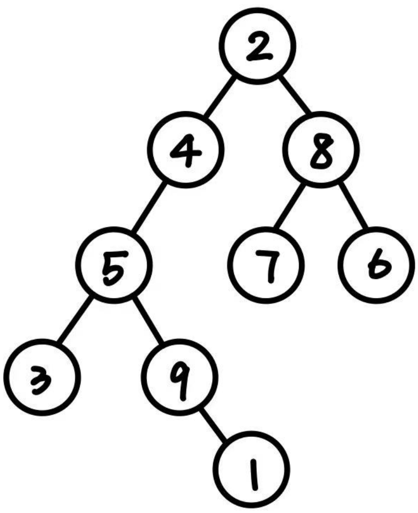
Input Sample

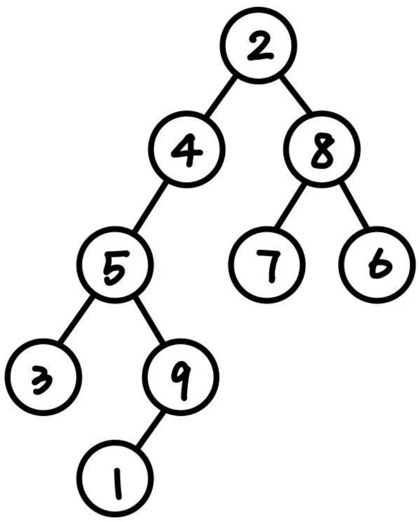
```
9
- 5 9 - - - 8 - 6
2 4 5 - 9 - 8 - 7
3 1 - - 4 - 6 - 2
```

Output Sample

```
3 5 9 1 4 2 8 7 6
2 4 5 3 9 1 8 6 7
3 1 9 5 4 7 6 8 2
2 4 8 5 6 3 9 7 1
```

The binary tree





3.3 Case 3: The input is incorrect(we cannot build such a tree)

Input Sample

```
9
2 - 9 5 4 - - 8 1
1 - 4 3 - 9 - 7 6
- - 3 4 1 - - 8 2
```

Output Sample

```
Impossible
```

3.5 Case 5: The tree has the minumum n.

Input Sample

```
1
1
1
1
```

Output Sample

```
1
1
1
1
```

Chapter 4: Analysis and Comments

4.1 Complexity of Brute-Force Algorithm

4.1.1 The main idea of BFA

In the in-order traversal, BFA will enumerate all the possible root.

4.1.2 Time Complexity of the Algorithm

Time Complexity = O(N!)

The first root have n cases, the second n-1 cases......

4.1.3 Space Complexity of the Algorithm

Space Complexity = O(N)

We use an array to store the searching result.

4.2 Complexity of BFS Algorithm

4.2.1Time Complexity of the Algorithm

Time Complexity =O(N)

The worst case is that this binary tree is a queue, therefore we need O(N) to traverse all the node.

4.2.2 Space Complexity of the Algorithm

Space Complexity =O(N)

That's because we use an array to store the searching result.

Chapter 5: Declaration

I hereby declare that all the work done in this project is of my independent effort.

Appendix：Source Code in C

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
int pre[105], in[105], post[105],level[105];
int lc[105], rc[105], root, n;
int f[105];//0-not visited,1-visited
void quit();
void bfs(int root);
int solve(int L1, int R1, int L2, int R2, int L3, int R3);
int main()
{
  int i;
    scanf("%d",&n);
    for (i = 1; i <= n; i++)
    {
      scanf("%d",&in[i]);
    f[in[i]] = 1;
  }
    for (i = 1; i <= n; i++)
    {
      scanf("%d",&pre[i]);
    f[pre[i]] = 1;
  }
    for (i = 1; i <= n; i++)
    {
      scanf("%d",&post[i]);
    f[post[i]] = 1;
  }

    root = solve(1, n, 1, n, 1, n);
    bfs(root);
```

```
    //print in-order
    printf("%d",in[1]);
    for(i=2;i<=n;i++)
      printf(" %d",in[i]);
    printf("\n");
    //pre-order
    printf("%d",pre[1]);
    for(i=2;i<=n;i++)
      printf(" %d",pre[i]);
    printf("\n");
    //post-order
    printf("%d",post[1]);
    for(i=2;i<=n;i++)
      printf(" %d",post[i]);
    printf("\n");
    //lever-order
  printf("%d",level[0]);
    for(i=1;i<n;i++)
      printf(" %d",level[i]);
    return 0;
}
void quit()
{
    printf("Impossible");
    exit(0);
}
void bfs(int root)
{
    int head = -1, tail = 0;
  //head represent the number of nodes we have asserted,tail represent the present one.
    level[0]=root;
  //the first node when we traverse in level-order is the root.
    while(head!=tail)
    {
        head++;
        //has a left child
    if (lc[level[head]])
    {
            tail++;
      level[tail]=lc[level[head]];
    }
    //has a right child
    if (rc[level[head]])
    {
            tail++;
      level[tail]=rc[level[head]];
    }
    }
```

```
}
int solve(int L1, int R1, int L2, int R2, int L3, int R3)
{
  int i,j,k,tmp,size,root,p;
  int cnt=0, cnt1 = 0, cnt2 = 0;
  int fl = 0, flag = 0;
    if (L1 > R1) return 0;
    //have traversaled in-order
    if (L1 == R1)
  {
        tmp = in[L1] | pre[L2] | post[L3];
        if (!tmp)
    {
            for (i = n; i >= 1; i--)
                if (!f[i])
        {
                    tmp = i;
                    f[i] = 1;
                    break;
                }
        }
        in[L1] = pre[L2] = post[L3] = tmp;
        return in[L1];
    }
    //impossible case: the last node of post-order doesn't equals to the first node of
pre-order
    if (pre[L2] && post[R3] && pre[L2]!=post[R3])
    quit();
    //not both of the first node of pre-order and the last node of post-order are 0
    if (!pre[L2]||!post[R3])
        pre[L2] = post[R3] = pre[L2] + post[R3];
        //let pre[L2] and post[R3] contains the same node
    //we can't find the node
    if (!pre[L2])
    quit();
    root = pre[L2];
  p = L1;
  cnt = 0;
    for (i = L1; i <= R1; i++)
    {
      if(in[i] == 0)
        cnt++;
      //count the nodes missing in in-order[L1,R1].
    if (root == in[i])
    {
            flag = 1;
        p=i;
        }
    }
```

```
    //cant find the root and only one node missing, then it's the root.
    if(!flag && cnt==1)
      for(i=L1;i<=R1;i++)
        if(!in[i])
      {
        in[i]=root;
        p = i;
        flag=1;
      }
    if(flag)  size = p - L1;
    else
  {
        for (p = L1 + 1; p <= R1; p++)
          if (in[p] == 0)
      {
               size = p - L1;
               //we need to judge which one is the root.
                //pre[L2 + 1 ~ L2 + size]   root,L2+1,L2+2°≠R2
                //post[L3 ~ L3 + size - 1]  L3,L3+1,L3+2°≠R3
                int cnt1 = 0, cnt2 = 0;
                for (i = L2 + 1; i <= L2 + size; i++)
                    if (!pre[i]) cnt1++;//the number of 0 in pre-order
                for (i = L3; i <= L3 + size - 1; i++)
                    if (!post[i]) cnt2++;//the number of 0 in post-order
                for (i = L2 + 1; i <= L2 + size; i++)
                    if (pre[i])
          {
                        fl = 0;
                        for (j = L3; j <= L3 + size - 1; j++)
                            if (pre[i] == post[j]) fl = 1;
                        if (!fl) cnt2--;
                    }
                for (i = L3; i <= L3 + size - 1; i++)
                    if (post[i])
          {
                        fl = 0;
                        for (j = L2 + 1; j <= L2 + size; j++)
                            if (post[i] == pre[j]) fl = 1;
                        if (!fl) cnt1--;
                    }
                if (cnt1 >= 0&&cnt2 >= 0) break;
                //the number is all missing in pre-order and post-order -> it's the
root.
          }
    }
    pre[L2] = root;
    post[R3] = root;
    in[p] = pre[L2];
    size = p - L1;
```

```
    if(p > R1) quit();
    lc[root] = solve(L1, p - 1, L2 + 1, L2 + size, L3, L3 + size - 1);//searching the left tree
    rc[root] = solve(p + 1, R1, L2 + size + 1, R2, L3 + size, R3 - 1);//searching the right tree
    return root;
}
```