

VARUN BHARGAVA - 241010282

DATA STRUCTURES TASK-9

Task 01: Restore File Directory from Creation and Listing Logs:

(<https://github.com/varunnnb/dsa-sem3-iiitnr/blob/main/lab9/lab9-1.c>)

A computer system stores the folder structure of files as a binary tree. Unfortunately, the directory system crashed, and you only have two logs left:

- preorder → the order in which directories and files were created.
- inorder → the order in which files/directories appear when listed alphabetically.

Your task is to rebuild the original folder structure tree and return it. You may assume there are no duplicate directory/file names.

Input:

```
preorder = ["root", "docs", "assignments", "photos", "music"]
inorder = ["assignments", "docs", "root", "photos", "music"]
```

Output:

```
["root", "docs", "music", "assignments", null, "photos", null]
```

The screenshot shows a terminal window with the following session:

```
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9> cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9"
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9> if ($?) { gcc lab9-1.c -o lab9-1 }
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9> ./lab9-1
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9> [ "root", "docs", "photos", "assignments", null, "music" ]
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9>
```

The terminal output shows the execution of the C program `lab9-1` and its successful compilation and execution. The command `./lab9-1` is run, followed by the expected output: `["root", "docs", "photos", "assignments", null, "music"]`.

lab9 > C lab9-1.c > newnode(char)

```

38
39 TreeNode *buildTreeUtil(char *preorder[], char *inorder[], int inStart, int inEnd, int *preIndex)
40 {
41     int inIndex = findIndex(inorder, inStart, inEnd, root->val);
42
43     root->left = buildTreeUtil(preorder, inorder, inStart, inIndex - 1, preIndex);
44     root->right = buildTreeUtil(preorder, inorder, inIndex + 1, inEnd, preIndex);
45
46     return root;
47 }
48
49 TreeNode *buildTree(char *preorder[], char *inorder[], int n)
50 {
51     int preIndex = 0;
52     return buildTreeUtil(preorder, inorder, 0, n - 1, &preIndex);
53 }
54
55 typedef struct Queue
56 {
57     TreeNode **data;
58     int front, rear, size;
59 } Queue;
60
61 Queue *createQueue(int size)
62 {
63     Queue *q = (Queue *)malloc(sizeof(Queue));
64     q->data = (TreeNode **)malloc(sizeof(TreeNode *) * size);
65     q->front = q->rear = 0;
66     q->size = size;
67     return q;
68 }
69
70 int isEmpty(Queue *q)
71 {
72     return q->front == q->rear;
73 }
74
75 void enqueue(Queue *q, TreeNode *node)
76 {
77     if (q->rear < q->size)
78         q->data[q->rear++] = node;
79 }
80

```

PS main*+ 41 0! Run Testcases 0 0 △ 0

lab9 > C lab9-1.c > newnode(char)

```

81 TreeNode *dequeue(Queue *q)
82 {
83     if (isEmpty(q))
84         return NULL;
85     return q->data[q->front++];
86 }
87
88 void printLevelOrder(TreeNode *root)
89 {
90     if (!root)
91     {
92         printf("[]\n");
93         return;
94     }
95
96     Queue *q = createQueue(100);
97     enqueue(q, root);
98
99     TreeNode *levelOrder[200];
100    int idx = 0;
101
102    while (!isEmpty(q))
103    {
104        TreeNode *node = dequeue(q);
105        levelOrder[idx++] = node;
106        if (node)
107        {
108            enqueue(q, node->left);
109            enqueue(q, node->right);
110        }
111    }
112
113    int lastNonNull = idx - 1;
114    while (lastNonNull >= 0 && levelOrder[lastNonNull] == NULL)
115        lastNonNull--;
116
117    printf("\n");
118    for (int i = 0; i <= lastNonNull; i++)
119    {
120        if (levelOrder[i])
121            printf("\n%5s", levelOrder[i]->val);
122    }

```

PS main*+ 41 0! Run Testcases 0 0 △ 0

C lab9-1.c U C lab9-2.c U C lab9-3.c U

lab9 > lab9-1.c > newNode(char)

```
88 void printLevelOrder(TreeNode *root)
116
117     printf("[" );
118     for (int i = 0; i <= lastNonNull; i++)
119     {
120         if (levelOrder[i])
121             printf("%s", levelOrder[i]->val);
122         else
123             printf("null");
124         if (i < lastNonNull)
125             printf(",");
126     }
127     printf("]\n");
128 }
129
130 int main()
131 {
132     char *preorder[] = {"root", "docs", "assignments", "photos", "music"};
133     char *inorder[] = {"assignments", "docs", "root", "photos", "music"};
134     int n = sizeof(preorder) / sizeof(preorder[0]);
135
136     TreeNode *root = buildTree(preorder, inorder, n);
137
138     printLevelOrder(root);
139
140     return 0;
141 }
```

CHAT

TERMINAL

```
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsalab9> cd ":";Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsalab9\ ; if ($?) { gcc lab9-1.c -o lab9-1 ; if (?) { .\lab9-1 } }
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsalab9> [ root , docs , photos , assignments , null , null , music ]
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsalab9>
```

Task 02: Restore Family Tree from Birth Records:

(<https://github.com/varunnnb/dsa-sem3-iiitnr/blob/main/lab9/lab9-2.c>)

A genealogy company stores family data in two forms:

- **inorder** → listing of people in chronological order of birth within families.
 - **postorder** → listing of children before their parents.

The database was corrupted, and you must reconstruct the original family tree.

You may assume all names are unique.

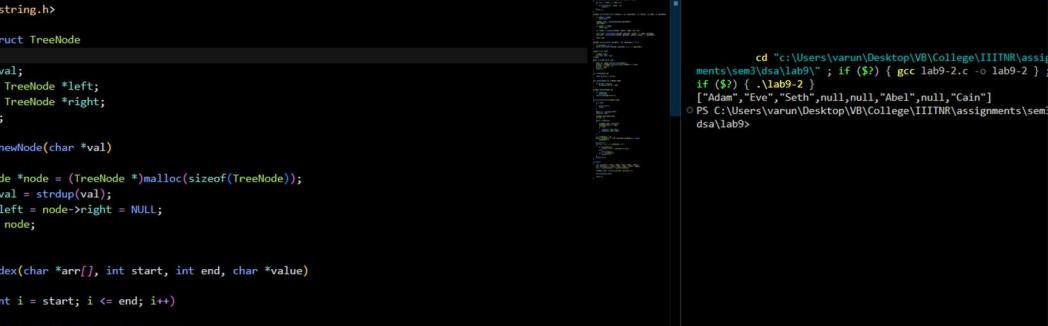
Input:

```
inorder = ["Eve","Adam","Cain","Abel","Seth"]
```

```
postorder = ["Eve","Cain","Abel","Seth","Adam"]
```

Output:

```
["Adam","Eve","Seth",null,null,"Cain","Abel"]
```



```
lab9 > C lab9-2.c U x lab9-3.c U
lab9 > C lab9-2.c > TreeNode
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct TreeNode
6 {
7     char *val;
8     struct TreeNode *left;
9     struct TreeNode *right;
10 } TreeNode;
11
12 TreeNode *newNode(char *val)
13 {
14     TreeNode *node = (TreeNode *)malloc(sizeof(TreeNode));
15     node->val = strdup(val);
16     node->left = node->right = NULL;
17     return node;
18 }
19
20 int findIndex(char *arr[], int start, int end, char *value)
21 {
22     for (int i = start; i <= end; i++)
23     {
24         if (strcmp(arr[i], value) == 0)
25             return i;
26     }
27     return -1;
28 }
29
30 TreeNode *buildTreeUtil(char *inorder[], char *postorder[], int inStart, int inEnd, int *postIndex)
31 {
32     if (inStart > inEnd)
33         return NULL;
34
35     TreeNode *root = newNode(postorder[*postIndex]);
36     (*postIndex)++;
37
38     if (inStart == inEnd)
39         return root;
40
41     int inIndex = findIndex(inorder, inStart, inEnd, root->val);
42
43     root->left = buildTreeUtil(inorder, postorder, inStart, inIndex - 1, postIndex);
44     root->right = buildTreeUtil(inorder, postorder, inIndex + 1, inEnd, postIndex);
45
46     return root;
47 }
48
49 int main()
50 {
51     char inorder[] = "B D E C A F G";
52     char postorder[] = "D E B C F G A";
53
54     TreeNode *root = buildTreeUtil(inorder, postorder, 0, 7, 0);
55
56     // Print the tree to verify
57     printTree(root);
58
59     return 0;
60 }
```

lab9-1.c U lab9-2.c U lab9-3.c U

```
lab9 > C lab9-2.c > TreeNode
30 TreeNode *buildTreeUtil(char *inorder[], char *postorder[], int inStart, int inEnd, int *postIndex)
31 {
32     if (inStart > inEnd)
33         return NULL;
34
35     int root = postIndex[0];
36     TreeNode *rootNode = (TreeNode *)malloc(sizeof(TreeNode));
37     rootNode->val = inorder[root];
38
39     int index = search(inorder, inStart, inEnd, rootNode->val);
40
41     if (index == -1)
42         return NULL;
43
44     rootNode->right = buildTreeUtil(inorder, postorder, index + 1, inEnd, postIndex);
45     rootNode->left = buildTreeUtil(inorder, postorder, inStart, index - 1, postIndex);
46
47     return rootNode;
48 }
49
50 TreeNode *buildTree(char *inorder[], char *postorder[], int n)
51 {
52     int postIndex = n - 1;
53     return buildTreeUtil(inorder, postorder, 0, n - 1, &postIndex);
54 }
55
56 typedef struct Queue
57 {
58     TreeNode **data;
59     int front, rear, size;
60 } Queue;
61
62 Queue *createQueue(int size)
63 {
64     Queue *q = (Queue *)malloc(sizeof(Queue));
65     q->data = (TreeNode **)malloc(sizeof(TreeNode *) * size);
66     q->front = q->rear = 0;
67     q->size = size;
68     return q;
69 }
70
71 int isEmpty(Queue *q)
72 {
73     return q->front == q->rear;
74 }
75
76 void enqueue(Queue *q, TreeNode *node)
77 {
78     if (q->rear < q->size)
79     {
80         q->data[q->rear++] = node;
81     }
82 }
83
84 TreeNode *dequeue(Queue *q)
85 {
86 }
```

main.c Dif Off Run Testcases ⌂ 0 △ 0

lab9-1.c U lab9-2.c U lab9-3.c U

```
lab9 > C lab9-2.c > TreeNode
81 TreeNode *dequeue(Queue *q)
82 {
83     if (isEmpty(q))
84     {
85         return NULL;
86     }
87
88     void printLevelOrder(TreeNode *root)
89 {
90     if (!root)
91     {
92         printf("[]\n");
93         return;
94     }
95
96     Queue *q = createQueue(100);
97     enqueue(q, root);
98
99     TreeNode *levelOrder[200];
100    int idx = 0;
101
102    while (!isEmpty(q))
103    {
104        TreeNode *node = dequeue(q);
105        levelOrder[idx++] = node;
106        if (node)
107        {
108            enqueue(q, node->left);
109            enqueue(q, node->right);
110        }
111    }
112
113    int lastNonNull = idx - 1;
114    while (lastNonNull >= 0 && levelOrder[lastNonNull] == NULL)
115        lastNonNull--;
116
117    printf("[");
118    for (int i = 0; i <= lastNonNull; i++)
119    {
120        if (levelOrder[i])
121            printf("\\%s\\", levelOrder[i]->val);
122    }
123 }
```

main.c Dif Off Run Testcases ⌂ 0 △ 0

```

lab9 > C lab9-2.c > 65 TreeNode
88 void printlevelOrder(TreeNode *root)
112
113     int lastNonNull = idx - 1;
114     while (lastNonNull >= 0 && levelOrder[lastNonNull] == NULL)
115         lastNonNull--;
116
117     printf("[");
118     for (int i = 0; i <= lastNonNull; i++)
119     {
120         if (levelOrder[i])
121             printf("\%s", levelOrder[i]->val);
122         else
123             printf("null");
124         if (i < lastNonNull)
125             printf(",");
126     }
127     printf("]\n");
128 }
129
130 int main()
131 {
132     char *inorder[] = {"Eve", "Adam", "Cain", "Abel", "Seth"};
133     char *postorder[] = {"Eve", "Cain", "Abel", "Seth", "Adam"};
134     int n = sizeof(inorder) / sizeof(inorder[0]);
135
136     TreeNode *root = buildTree(inorder, postorder, n);
137
138     printlevelOrder(root);
139
140     return 0;
141 }

```

Task 03: Reconstruct Network Topology from Connection Logs:

(<https://github.com/varunnnb/dsa-sem3-iiitnr/blob/main/lab9/lab9-3.c>)

In a distributed network, servers and routers form a tree topology.

- preorder → the order in which connections were established from the main server outward.
- postorder → the order in which messages were acknowledged back to the main server.

Given these two logs, reconstruct the network topology.

If the network is not a full binary tree (where every router has either 0 or 2 connections), the topology cannot be uniquely reconstructed.

Input:

preorder = [1,2,4,5,3,6,7]

postorder = [4,5,2,6,7,3,1]

Output:

[1,2,3,4,5,6,7]

Input:

preorder = [1,2,3]

postorder = [2,3,1]

Output:

"Network topology cannot be uniquely reconstructed"

The screenshot shows a terminal window with two panes. The left pane displays the source code for `lab9-3.c`, which contains C code for building a full binary tree and printing its level-order traversal. The right pane shows the terminal command `cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9"> cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9>` and the resulting output of the program's execution.

```
lab9 > C lab9-3.c U C lab9-3.c U C lab9-3.c U
lab9 > C lab9-3.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct TreeNode
5 {
6     int val;
7     struct TreeNode *left;
8     struct TreeNode *right;
9 } TreeNode;
10
11 TreeNode *newNode(int val)
12 {
13     TreeNode *node = (TreeNode *)malloc(sizeof(TreeNode));
14     node->val = val;
15     node->left = node->right = NULL;
16     return node;
17 }
18
19 TreeNode *buildFullBinaryTree(int *pre, int *post, int *preIndex, int postStart, int postEnd, int n)
20 {
21     if (*preIndex >= n || postStart > postEnd)
22         return NULL;
23
24     TreeNode *root = newNode(pre[*preIndex]);
25     (*preIndex)++;
26
27     if (postStart == postEnd || *preIndex >= n)
28         return root;
29
30     int leftChild = pre[*preIndex];
31     int i;
32     for (i = postStart; i <= postEnd; i++)
33     {
34         if (post[i] == leftChild)
35             break;
36     }
37     if (i > postEnd)
38         return NULL;
39
40     if (i < postEnd - 1)
41     {
42         root->left = buildFullBinaryTree(pre, post, preIndex, postStart, i, n);
43         root->right = buildFullBinaryTree(pre, post, preIndex, i + 1, postEnd - 1, n);
44     }
45     else
46     {
47         return NULL;
48     }
49     return root;
50 }
51
52 void printLevelOrder(TreeNode *root)
53 {
54     if (!root)
55     {
56         printf("[]\n");
57         return;
58     }
59     TreeNode *queue[100];
60     int front = 0, rear = 0;
61     queue[rear++] = root;
62     int idx = 0;
63     int vals[100];
64     while (front < rear)
65     {
66         TreeNode *node = queue[front++];
67         vals[idx++] = node->val;
68         if (node->left)
69             queue[rear++] = node->left;
70         if (node->right)
71             queue[rear++] = node->right;
72     }
73     printf("[" );
74     for (int i = 0; i < idx; i++)
75     {
76         printf("%d", vals[i]);
77         if (i < idx - 1)
78             printf(",");
79     }
80 }
81
82 main() { int pre[] = {1,2,3,4,5,6,7}; int post[] = {1,2,3,4,5,6,7}; printLevelOrder(newNode(1)); }
```

CHAT

TERMINAL

```
cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9"> cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9>
cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9"> if ($?) { gcc lab9-3.c -o lab9-3 }
if ($?) { ./lab9-3 }
[1,2,3,4,5,6,7]
[1,2,3]
PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9>
```

The screenshot shows a code editor with three tabs open: lab9-1.c, lab9-2.c, and lab9-3.c. The lab9-3.c tab is active, displaying the following C code:

```
lab9 > C lab9-3.c > main()
52 void printLevelOrder(TreeNode *root)
53 {
54     for (int i = 0; i < idx; i++)
55     {
56         printf("%d", vals[i]);
57         if (i < idx - 1)
58             printf(",");
59     }
60     printf("\n");
61 }
62
63 int main()
64 {
65     int preorder1[] = {1, 2, 4, 5, 3, 6, 7};
66     int postorder1[] = {4, 5, 2, 6, 7, 3, 1};
67     int n1 = sizeof(preorder1) / sizeof(preorder1[0]);
68     int preIndex1 = 0;
69     TreeNode *root1 = buildFullBinaryTree(preorder1, postorder1, &preIndex1, 0, n1 - 1, n1);
70     if (root1)
71         printLevelOrder(root1);
72     else
73         printf("\\"Network topology cannot be uniquely reconstructed\"\n");
74
75     int preorder2[] = {1, 2, 3};
76     int postorder2[] = {2, 3, 1};
77     int n2 = sizeof(preorder2) / sizeof(preorder2[0]);
78     int preIndex2 = 0;
79     TreeNode *root2 = buildFullBinaryTree(preorder2, postorder2, &preIndex2, 0, n2 - 1, n2);
80     if (root2)
81         printLevelOrder(root2);
82     else
83         printf("\\"Network topology cannot be uniquely reconstructed\"\n");
84
85     return 0;
86 }
```

The terminal window on the right shows the command to compile the code and the resulting output:

```
cd "c:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9"; if ($?) { gcc lab9-3.c ; if ($?){ ./lab9-3 } [1,2,3,4,5,6,7] [1,2,3] PS C:\Users\varun\Desktop\VB\College\IIITNR\assignments\sem3\dsa\lab9>
```