

# Fundamentals Of Data structures

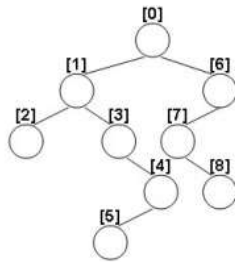
## Laboratory Project 2 Build A Binary Search Tree

Date: 2018-11-18

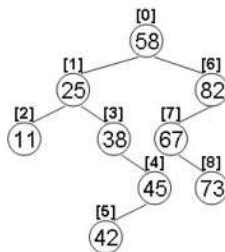
## Chapter1 Introduction

### 1.1 Description of the problem

It is widely acknowledged that there is a unique possible binary search tree (BST)<sup>1</sup> which can be built given a known structure of a binary tree and a sequence of distinct integer. For example, the structure is illustrated as the following picture:



Given a sequence of "73 45 11 58 82 25 67 38 42", a unique binary search tree can be built:



The goal of the program is to build that unique binary search tree with its given structure the sequence. The index of each node's child is given and the root's index is always "0". After building the tree, the program will print the level order traversal sequence of the tree.

### 1.2 Way to solve the problem

The plan is to use an array to simulate the structure of a binary search tree. Noticing that the sequence is unnecessarily given in order, the sequence ought to be sorted at first. Then insert the numbers respectively to the nodes which they are belonged to. The inserting sequence is from the smallest number to the biggest one. Eventually, use an array as a queue to traverse the BST in level order and output the results.

## Chapter2 Algorithm Specification

**An important note:** The BST is represented by an array whose type of elements are struct nodes. As the following pictures illustrated, a tree node has three objects: the key of the node, the indexes in the array of its left and right child. This is the foundation of this program.

---

<sup>1</sup> A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties: The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than or equal to the node's key. Both the left and right subtrees must also be binary search trees.

```
typedef struct Tree{
    int Element;
    int Left;
    int Right;
}BTree;
```

```
for(int i=0; i<N; i++)
{
    scanf("%d %d",&left,&right);
    BST1[i].Left=left;
    BST1[i].Right=right;
} //bulid the Binary Search Tree
```

## 2.1 Sort

Use bubble sort to sort the given sequence. Use two cycles and exchange each adjacent elements' value as long as the former one is bigger than the latter one. Here omits other facts about the principle of bubble sorting since it's a simple fundamental method of sorting.

```
void ArraySort(int *a,int n)//define a function to sort the element and get the result of Inorder Traversal
{
    int temp = 0;
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n-1; j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
```

## 2.2 Insert

Making use of the properties of BST, it's possible to find the position of the smallest node. Additionally, the position of each distinct number can be found. Because the in-order traversal sequence of the BST must be a nondecreasing sequence. As a result, the elements of the sorted array may be inserted into the BST in in-order traversal order. In this way, the program succeeds in building a BST according to the problem's demand.

```
void insert(BTree *s,int n)//define a function to insert values into Binary Search Tree
{
    int b[n];
    int Flag=0;
    int top=0;
    int Temp=0;
    while(top || Flag!=Null)
    {
        while(Flag!=Null)
        {
            b[top++]=Flag;
            Flag=s[Flag].Left;
        }
        Flag=b[--top];
        s[Flag].Element=a[Temp++];
        if(Flag!=Null)
        {
            Flag=s[Flag].Right;
        }
    }
}
```

Use an array b as a queue and Flag as a temporary tag. If the current node has a left child then push it into the queue, whenever there's no left child, give the existing smallest number to it. Then push it's right child into the queue. Repeat the process until flag turns to -1 or all

numbers have been inserted.

## 2.3 Traverse

The nodes should be printed in level-order traverse. Using an array as a queue to process the BST. Push the root into the queue at first. Push every child of the current node into the queue. If the current node has no child or all its children have been popped, then pop it.

```
void levelOrder(BTree *T1,BTree *T2,int n)//define a function to output the level order traversal sequence of the Binary Search Tree
{
    int Flag=0;
    T2[0]=T1[0];
    int Front=0;
    int Rear=1;
    while(Front!=Rear)
    {
        Flag=T2[Front].Left;
        if(Flag!=Null)
        {
            T2[Rear++]=T1[Flag];
        }
        Flag=T2[Front++].Right;
        if(Flag!=Null)
        {
            T2[Rear++]=T1[Flag];
        }
    }
    for(int i=0; i<n; i++)
    {
        if(i!=n-1)
        {
            printf("%d ",T2[i].Element);
        }
        else
        {
            printf("%d",T2[i].Element);
        }
    }
}
```

In this way, the program output the sequence in level-order traverse.

# Chapter3 Testing Results

## 3.1 Test cases

### 3.1.1 Minimum test cases

Cases that have only one node are listed as follows:

#### test\_data\_1

###Group 1:

1

-1 -1

8

answer:

8

###Group 2:

1

-1 -1

94

answer:

### 3.1.2 simple test cases

Cases that have relatively simple structure are listed as follows:

#### test\_data\_5

###Group 1:

5  
1 2  
3 -1  
-1 -1  
-1 4  
-1 -1  
1 3 4 2 5

answer:

4 3 5 1 2

###Group 2:

5  
1 2  
-1 -1  
3 4  
-1 -1  
-1 -1  
5 3 1 4 2

answer:

2 1 4 3 5

#### test\_data\_10

###Group 1:

10  
1 2  
3 -1  
-1 5  
4 -1  
-1 6  
7 8

-1 -1  
-1 -1  
9 -1  
-1 -1  
1 3 9 5 8 2 7 6 4 10

answer:  
5 4 6 3 8 1 7 10 2 9

###Group 2:

10  
1 2  
3 5  
-1 6  
4 -1  
-1 -1  
-1 7  
8 9  
-1 -1  
-1 -1  
-1 -1  
9 5 8 2 7 6 4 3 1 10

answer:  
6 3 7 2 4 9 1 5 8 10

### 3.1.3 Special test cases

Unusual and abnormal cases are listed as follows:

**test\_data\_30(With skewed tree)**

###Group 1(Skewed tree):

30  
-1 1  
-1 2  
-1 3  
-1 4  
-1 5  
-1 6  
-1 7  
-1 8  
-1 9

-1 10  
-1 11  
-1 12  
-1 13  
-1 14  
-1 15  
-1 16  
-1 17  
-1 18  
-1 19  
-1 20  
-1 21  
-1 22  
-1 23  
-1 24  
-1 25  
-1 26  
-1 27  
-1 28  
-1 29  
-1 -1

3 9 13 27 18 16 20 11 1 26 5 10 8 25 15 6 24 23 7 2 14 22 29 19 21 17 28 30 4 12

answer:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

###Group 2:

30  
1 2  
3 4  
5 6  
7 -1  
8 9  
-1 -1  
-1 10  
11 12  
-1 -1  
-1 13  
14 15  
16 -1  
-1 17  
18 19

-1 -1  
-1 20  
-1 21  
22 23  
-1 -1  
24 -1  
-1 25  
-1 26  
-1 -1  
-1 -1  
-1 -1  
27 -1  
28 29  
-1 -1  
-1 -1  
-1 -1  
3 9 13 18 27 16 20 11 1 26 5 10 25 8 15 6 24 23 7 2 14 22 29 19 17 21 28 30 4 12

answer:

21 13 23 12 15 22 24 7 14 16 26 6 8 18 25 27 1 10 17 20 28 2 9 11 19 30 4 29 3 5

### 3.1.4 Complicated test cases

Cases that are quite complicated are listed as follows:

#### test\_data\_50

50  
1 2  
3 4  
5 6  
7 8  
9 10  
11 12  
13 14  
15 -1  
16 -1  
-1 17  
-1 -1  
-1 18  
-1 -1  
-1 -1  
-1 19  
20 -1  
-1 21



-1 -1  
22 23  
24 25  
26 -1  
-1 27  
-1 -1  
28 29  
-1 -1  
-1 30  
31 32  
33 34  
-1 -1  
-1 -1  
35 36  
37 -1  
-1 -1  
38 39  
40 -1  
-1 -1  
-1 41  
-1 42  
-1 -1  
-1 -1  
-1 43  
44 45  
46 47  
-1 48  
49 -1  
-1 -1  
-1 -1  
-1 -1  
-1 -1  
-1 -1  
8 25 50 35 30 37 4 34 27 38 10 49 7 33 31 3 48 20 22 11 28 41 43 44 47 14 46 9 6 5 18 16 17 45 42 12 29  
40 39 15 26 23 36 13 32 19 24 21 2 1

answer:

28 23 37 11 26 35 39 10 22 24 27 29 36 38 40 9 12 25 31 42 8 13 30 33 41 43 6 17 32 34 45 5 7 15 21 44  
46 1 14 16 18 49 3 19 48 50 2 4 20 47

### 3.1.5 Comprehensive and complicated test cases with maximum scale

test\_data\_100

100  
1 2  
3 4  
5 6  
7 8  
9 10  
11 12  
13 14  
15 16  
-1 17  
18 19  
-1 25  
-1 -1  
20 21  
29 -1  
-1 30  
-1 31  
-1 32  
33 34  
22 23  
-1 24  
26 -1  
27 28  
-1 -1  
35 -1  
-1 36  
-1 37  
-1 -1  
38 39  
-1 -1  
40 41  
-1 42  
43 -1  
44 45  
-1 -1  
46 47  
-1 -1  
-1 51  
-1 52  
-1 -1  
-1 -1  
48 49  
-1 -1  
50 -1

-1 -1  
53 54  
-1 55  
-1 -1  
-1 56  
59 60  
-1 61  
62 63  
57 -1  
-1 58  
64 65  
-1 66  
-1 67  
68 69  
-1 -1  
-1 -1  
-1 70  
-1 71  
72 -1  
-1 73  
74 -1  
75 76  
-1 -1  
77 -1  
78 -1  
79 -1  
-1 80  
81 82  
-1 83  
-1 -1  
84 -1  
-1 85  
-1 -1  
91 -1  
-1 92  
-1 -1  
-1 93  
94 95  
86 -1  
-1 87  
-1 88  
-1 89  
-1 90  
-1 96

```

-1 97
-1 -1
98 -1
99 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
29 16 5 96 88 82 78 76 77 81 87 95 6 19 34 50 68 97 23 51 84 22 59 7 52 4 58 21 83 49 28 8 90 73 69 70
79 98 17 42 74 26 67 37 10 89 71 72 93 18 48 11 64 53 47 63 12 57 39 40 62 25 3 24 61 45 80 41 56 32 55
44 14 54 92 13 43 15 27 94 36 9 35 65 31 30 38 86 1 60 99 85 66 75 2 91 20 33 46 100

```

answer:

```

54 37 65 22 48 56 86 4 23 42 49 55 59 85 87 1 5 25 39 43 50 58 63 83 88 3 17 24 27 38 41 44 51 57 61 64
79 84 100 2 12 18 26 28 40 45 52 60 62 74 80 94 10 13 19 32 47 53 66 75 82 89 99 7 11 16 21 31 33 46 70
76 81 93 95 6 9 14 20 29 35 69 71 77 90 96 8 15 30 34 36 67 72 78 92 98 68 73 91 97

```

## 3.2 Check

The program is logically sensible and the results of all the tests prove to be correct. All test cases reinforce the fact that the program solves the problem properly.

## Chapter4 Analysis and Comments

### 4.1 Space complexity

The space required grows as the scale of the problem grows. And they have a linear relationship. Because the program has to be allocated space several times as big as the scale of the problem which are used as arrays to serve as queues and trees. Consequently, the space complexity of the program is  $O(n)$ .

### 4.2 Time complexity

All the nodes are only traversed once, so the time complexity of traversal is  $O(n)$ . The function "insert" is in-order traversal and function "LevelOrder" is level-order traversal. Then it can be known that the time complexity of the two functions is  $O(n)$ . however, the time complexity of the function "ArraySort" is  $O(n^2)$  because the function uses bubble sorting which requires two cycles. In conclusion, the time complexity of the program is  $O(n^2)$ .

### **4.3 Comments**

On one hand, the program solves the problem properly and perfectly meets the demand of the program. On the other hand, the time complexity is relatively higher because bubble sorting is used. Though bubble sorting is easy to write and understand, it indeed takes relatively longer time to sort the array compared to other methods of sorting.

*Declaration: We hereby declare that all the work done in the project titled “Performance Measurement” is of our independent effort as a group.*

*Group assignment:*

*Programmer:*任启岚

*Tester:*彭子帆

*Document writer:*陈宇威