Project 2 - Build A Binary Search Tree

Group 23

November 2, 2018

Code - Jiawei Peng, **Document** - Jianglai Dai, **Test** - Shuting Guo

Contents

Chapter 1. Introduction

1.1 What is Binary Search Tree

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than or equal to the node's key.
3. Both the left and right subtrees must also be binary search trees.

Here is an example following (Figure 1):

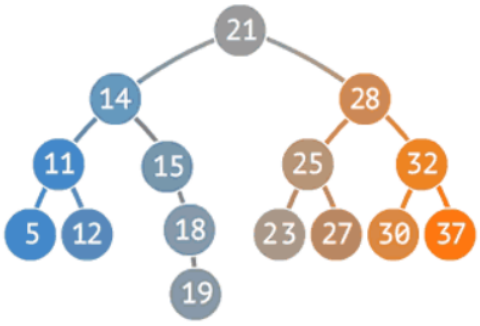


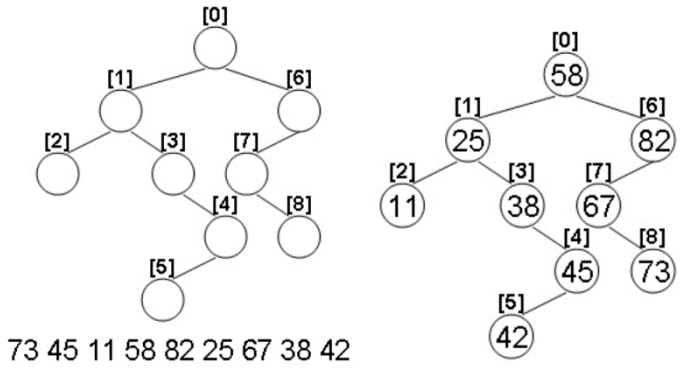
Figure 1: BST example

1.2 Problem Introduction

Given the structure of a binary tree and a sequence of distinct integer keys, there is only one way to fill these keys into the tree so that the resulting tree satisfies the definition of a BST. You are supposed to output the level order traversal sequence of that tree. The sample is illustrated by figures below.

1.3 Functions of programs

To solve this problem, we give three codes(see Appendix), whose functions are as follows:



1.3.1 *project2.cpp*

Input Includes $n + 2$ lines.

a) First line contains a single integer n.

b) In the following n lines, numbered $i = 0, 1, \ldots n - 1$, each line contains a pair of integers l and r, seperated by spaces, which represents the left and right child of node i.

c) Last line contains n intergers seperated by spaces, the elements in the BST.

Output One line, n integers seperated by spaces, the breadth-first-search order of tree nodes. **Note that, in the code, add `#define Debug` command, the code will output some helpful info for you to draw the tree you've input.**

1.3.2 *gen.cpp*

Input n and lim, the number of nodes and max element limit.

Output The program will out put a random input for *project2.cpp*, and the standard answer will be output to *ans.txt*, then you can check if *project2.cpp* works well.

1.3.3 *checker.cpp*

Input n and lim, the number of nodes and max element limit.

Output The program will check *project2.cpp* with *gen.cpp* automatically. It use *gen.cpp* to generate a random input for *project.cpp*, and then run project2.cpp, finally compare the output of *project2.cpp* (redirected to *out.txt*) and *ans.txt*. If the output is correct, *checher.cpp* will output ***Accepted!***

Chapter 2. Algorithm Specification

2.1 Detailed explanation of Algorithm

Because we have already know that the root is 0, therefore, we just link all edges of the tree in $O(n)$ time, and start the program from root node 0. (If root is not 0, we can find the root in this way. When we link the edges of the tree, use an array $Child[i]$ to represent whether i is a child of certain node. When we add an edge to the tree, for example, from x to y, meaning y is a child of x. Then, we mark $Child[y] =$ **true**, that means y cannot be a root, for it is a child of x. In the end, There will be the only one node *Root* whose $Child[Root] =$ **false**, and this node *Root* is the root of the tree.)

First of all, we need the elements to be sorted, so that we can insert the numbers into the tree of definite shape. So, we use STL sort function to sort the elements in $O(nlogn)$ time.

Then, run Depth-First-Search (DFS) and fill the numbers into tree nodes by **inorder traversing**. This will cost $O(n)$ time.

Algorithm 1 DFS procedure

Require: TreeNode x, sorted interger array a
Ensure: Match the nodes and numbers

```
1: function DFS(x)
2:     if x has left child then
3:         DFS(x.LeftChild)
4:     end if
5:     x.Number ← a[cnt]
6:     cnt ← cnt + 1
7:     if x has right child then
8:         DFS(x.RightChild)
9:     end if
10: end function
11: cnt ← 0
12: Input(Tree, a)
13: Srot(a)
14: DFS(Root)
```

Finally, we have done every thing on the tree, what need to do is printing. For printing function, we use a Breadth-First-Search (BFS) algorithm to print layer by layer.

BFS runs as follows:

First we use a queue which pushes elements to the back, and pops out elememts from front. Root of the tree is in the queue at the beginning. For each node poped from the queue(including the root), we push its children to the back of the queue, so that we can deal with them later. After we are finished with a node, we pop it out and go to deal with the next node.

We can easily find out that, in this way, we can access nodes in the order of their depth.

Algorithm 2 BFS procedure

Require: Tree T, TreeNode $Root$, Queue Q
Ensure: The Bfs order of nodes

```
1: function BFS(T,Root,Q)
2:     Q.push(Root)
3:     while Q is not empty do
4:         x ← Q.front
5:         Print(x)
6:         Q.pop()
7:         if x has left child then
8:             Q.push(x.LeftChild)
9:         end if
10:        if x has right child then
11:            Q.push(x.RightChild)
12:        end if
13:    end while
14: end function
```

2.2 Complexity Analyze

2.2.1 Time Complexity

In the algorithm described above in 2.1, we can know that the total Complexity is,

$$O(n + nlogn + n) = O(nlogn)$$

2.2.2 Space Complexity

We can easily find that, the space complexity is linear. Because every container in the algorithm needs only $O(n)$ space. In a word, the total space complexity is

$$O(n)$$

.

Chapter 3. Correctness Checking

3.1 Auto-match program

Test Use *checker.cpp* match *project2.cpp* with *gen.cpp* (the standard answer). In these cases, we ran *checker.exe* for ***10minutes***, no difference is found.

Conclusion The correctness of the code and the performance of time and space are reliable.

n	5	10	20	100	200	1000
lim	20	100	100	1000	1000	5000

3.2 Sample test cases

We hereby paste some test cases in case of being penalized.

Input1
10 8 1 5 2 3 9 -1 4 -1 -1 6 -1 -1 7 -1 -1 -1 -1 -1 -1 50 10 89 11 42 25 40 98 47 29
Output1
0 –LEFT—> 8 0 –RIGHT–> 1 1 –LEFT—> 5 5 –LEFT—> 6 6 –RIGHT–> 7 1 –RIGHT–> 2 2 –LEFT—> 3 3 –RIGHT–> 4 2 –RIGHT–> 9 11 10 42 40 89 25 47 98 29 50

Input2
10 1 6 2 3 -1 5 4 -1 -1 -1 7 -1 -1 -1 9 8 -1 -1 -1 -1 804 738 40 120 538 301 612 970 500 40
Output2
0 –LEFT—> 1 1 –LEFT—> 2 2 –RIGHT–> 5 5 –LEFT—> 7 7 –LEFT—> 9 7 –RIGHT–> 8 1 –RIGHT–> 3 3 –LEFT—> 4 0 –RIGHT–> 6 804 538 970 40 738 500 612 120 40 301

Input3(line wrapped)
100 1 4 3 2 6 7 -1 5 15 50 14 10 28 32 8 9 23 12 11 17 19 16 36 -1 13 18 -1 25 98 -1 37 33 24 22 58 20 69 26 53 -1 47 21 31 41 86 -1 35 -1 29 -1 -1 60 27 -1 42 93 80 34 30 46 57 -1 39 40 38 -1 -1 56 43 49 84 -1 -1 51 -1 52 59 44 79 68 -1 91 62 -1 -1 -1 -1 45 63 -1 72 48 -1 73 -1 95 54 -1 -1 75 -1 67 -1 65 -1 -1 77 55 74 66 -1 -1 76 -1 64 83 -1 70 -1 87 -1 61 -1 -1 90 -1 94 -1 -1 -1 -1 -1 -1 82 81 71 -1 -1 -1 -1 78 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 85 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 89 96 -1 -1 -1 -1 92 88 -1 -1 -1 -1 -1 -1 -1 97 -1 -1 -1 -1 -1 -1 99 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 258 752 712 71 930 936 490 920 80 342 472 240 990 829 980 232 49 0 732 701

573 527 772 601 216 328 636 276 778 352
350 120 146 362 324 560 504 436 418 122
228 815 124 250 2 246 140 224 890 762
274 540 840 969 556 670 473 480 160 571
828 962 768 872 24 301 294 84 492 100
400 604 468 159 292 284 502 808 320 504
762 232 930 44 455 448 563 570 500 44
789 75 780 244 275 145 324 280 902 600

Output3(line wrapped)

0 –LEFT—> 1 # 1 –LEFT—> 3 # 3 –RIGHT–> 5 #
5 –LEFT—> 14 # 14 –LEFT—> 98 # 5 –RIGHT–> 10 #
10 –LEFT—> 19 # 19 –LEFT—> 53 # 53 –LEFT—> 77 #
53 –RIGHT–> 55 # 10 –RIGHT–> 16 # 16 –LEFT—> 24 #
24 –LEFT—> 29 # 29 –LEFT—> 30 # 30 –LEFT—> 57 #
57 –LEFT—> 64 # 57 –RIGHT–> 83 # 29 –RIGHT–> 46 #
46 –RIGHT–> 73 # 16 –RIGHT–> 22 # 22 –LEFT—> 86 #
1 –RIGHT–> 2 # 2 –LEFT—> 6 # 6 –LEFT—> 28 #
28 –LEFT—> 80 # 28 –RIGHT–> 34 # 34 –LEFT—> 43 #
43 –RIGHT–> 45 # 45 –LEFT—> 72 # 45 –RIGHT–> 48 #
48 –LEFT—> 54 # 54 –LEFT—> 74 # 54 –RIGHT–> 66 #
66 –RIGHT–> 82 # 34 –RIGHT–> 49 # 49 –RIGHT–> 75 #
75 –RIGHT–> 85 # 6 –RIGHT–> 32 # 32 –LEFT—> 38 #
38 –LEFT—> 59 # 59 –RIGHT–> 87 # 38 –RIGHT–> 44 #
44 –LEFT—> 63 # 63 –LEFT—> 94 # 2 –RIGHT–> 7 #
7 –LEFT—> 8 # 8 –LEFT—> 23 # 23 –LEFT—> 35 #
35 –LEFT—> 84 # 84 –LEFT—> 92 # 92 –LEFT—> 99 #
84 –RIGHT–> 88 # 88 –RIGHT–> 97 # 8 –RIGHT–> 12 #
12 –LEFT—> 13 # 13 –RIGHT–> 25 # 25 –RIGHT–> 60 #
60 –RIGHT–> 61 # 12 –RIGHT–> 18 # 18 –LEFT—> 69 #
18 –RIGHT–> 26 # 26 –LEFT—> 27 # 27 –LEFT—> 42 #
27 –RIGHT–> 93 # 7 –RIGHT–> 9 # 9 –LEFT—> 11 #
11 –LEFT—> 36 # 36 –RIGHT–> 51 # 51 –RIGHT–> 65 #
9 –RIGHT–> 17 # 17 –LEFT—> 58 # 58 –RIGHT–> 70 #
70 –LEFT—> 78 # 17 –RIGHT–> 20 # 20 –LEFT—> 47 #
47 –RIGHT–> 95 # 20 –RIGHT–> 21 # 21 –LEFT—> 31 #
31 –LEFT—> 39 # 39 –LEFT—> 79 # 39 –RIGHT–> 68 #
31 –RIGHT–> 40 # 40 –RIGHT–> 91 # 21 –RIGHT–> 41 #
41 –LEFT—> 62 # 62 –LEFT—> 90 # 0 –RIGHT–> 4 #
4 –LEFT—> 15 # 15 –LEFT—> 37 # 37 –RIGHT–> 52 #
15 –RIGHT–> 33 # 33 –RIGHT–> 56 # 56 –LEFT—> 76 #
4 –RIGHT–> 50 # 50 –RIGHT–> 67 # 67 –LEFT—> 81 #
81 –LEFT—> 89 # 81 –RIGHT–> 96 # 67 –RIGHT–> 71 #
828 224 930 0 352 872 930 44 294 560
829 890 980 24 80 232 350 472 600 840

920 962 990 2 75 159 228 276 324 468
500 573 670 902 936 969 49 146 216 232
280 301 342 455 473 504 563 601 732 44
71 124 160 244 284 320 328 418 480 502
556 570 636 701 780 122 140 240 275 292
324 400 436 490 527 571 604 712 768 815
100 145 250 362 448 492 504 540 762 772
808 84 120 246 258 752 762 778 789 274

Chapter 4. Declaration

We hereby declare that all the work done in this project is of our independent effort as a group.

Chapter 5. Appendix

5.1 project2.cpp

```cpp
#include <bits/stdc++.h>

//#define Debug

//  Add this line if 0 is not root
//#define RootNot0

using namespace std;

//  Node of tree
//    'data' is the value stored on the node
//    'l' is the index of left child
//    'r' is the index of right child
//    global variable will be initialized by 0
struct Node { int data,l,r; }S[1100];

//  'a' is the array to sort elements
//  'cnt' is a counter to fill sorted elements
//    into the tree one by one
int a[1100],cnt;

//  If Child[x] is
//    true:  x is a child of certain node
//    false: x is root
//  There will be only one node whose Child[x] is false,
//    and this node is root of the tree.
```

```
27 bool Child[1100];
28
29 //  Fill sorted numbers into nodes
30 void Dfs(const int x)
31 {
32     //  Note: 'if(~x)' is equal to 'if(x!=-1)'
33     //    for only ~(-1)==0
34
35 #ifdef Debug
36     if(~S[x].l)printf("%3d --LEFT---> %3d\n",x,S[x].l);
37 #endif
38     //  Fill left sub-tree first.
39     if(~S[x].l) Dfs(S[x].l);
40     //  Fill node x itself.
41     S[x].data=a[cnt++];
42 #ifdef Debug
43     if(~S[x].r)printf("%3d --RIGHT--> %3d\n",x,S[x].r);
44 #endif
45     //  Fill right sub-tree in the end.
46     if(~S[x].r) Dfs(S[x].r);
47 }
48
49 int main()
50 {
51     int n,x,y,Root=0;
52
53     //  Read 'n'
54     scanf("%d",&n);
55     for(int i=0;i<n;++i)
56     {
57         //  Read n pairs of numbers
58         scanf("%d%d",&x,&y);
59         //  Link the tree by index
60         S[i].l=x; S[i].r=y;
61
62         //  Note: 'if(~x)' is equal to 'if(x!=-1)'
63         //    for only ~(-1)==0
64
65 #ifdef RootNot0
66         //  Mark x,y as children
67         if(~x)Child[x]=true;
68         if(~y)Child[y]=true;
69 #endif
70     }
71
72 #ifdef RootNot0
```

```
73      //  Find out root
74      //  If 'i' is not root then Child[i]==true
75      for(int i=0;i<n;++i)
76          if(!Child[i]) { Root=i; break; }
77  #endif
78
79      //  Input n numbers
80      for(int i=0;i<n;++i) scanf("%d",&a[i]);
81      //  Sort by increasing order
82      sort(a,a+n);
83
84      //  Deep-First-Search
85      //  Fill the numbers by Inorder traversal
86      Dfs(Root);
87
88      //  Bfs, Breadth-First-Search
89      //  Print the tree by the order of node depth
90
91      //  STL queue
92      queue<int> Q; Q.push(Root);
93      while(!Q.empty())
94      {
95          //  Pop first element from queue
96          int temp=Q.front(); Q.pop();
97          //  Print the node
98          printf("%d ",S[temp].data);
99
100         //  Note: 'if(~x)' is equal to 'if(x!=-1)'
101         //    for only ~(-1)==0
102
103         //  Push left child
104         if(~S[temp].l) Q.push(S[temp].l);
105         //  Push right child
106         if(~S[temp].r) Q.push(S[temp].r);
107     }
108
109     //  Printing a new line in the end is a good habit.
110     printf("\n");
111
112     //  End
113     return 0;
114 }
```

5.2 gen.cpp

```
1   #include <bits/stdc++.h>
```

```
2
3  using namespace std;
4
5  // Data array
6  int a[1100];
7  // Counter for node allocate
8  int cnt;
9
10 // Tree node struct
11 struct node { int n,l,r; }T[1100];
12
13 // Insert x into sub-tree S
14 void Insert(const int S,const int x)
15 {
16     // If x < current data goto left
17     if(x<T[S].n)
18     {
19         // If there is no left child, allocate a new one
20         if(!T[S].l) T[S].l=++cnt,T[cnt]=(node){x,0,0};
21         // Insert recursively
22         else Insert(T[S].l,x);
23     }
24     // If x >= current data goto Right
25     else
26     {
27         // If there is no right child, allocate a new one
28         if(!T[S].r) T[S].r=++cnt,T[cnt]=(node){x,0,0};
29         // Insert recursively
30         else Insert(T[S].r,x);
31     }
32 }
33
34 int main(int argv,char ** argc)
35 {
36     FILE* ans = fopen("ans.txt","w");
37     // Input n and number-max-limit
38     int n,lim,seed=0;
39     if(argv>=4)
40     {
41         n=atoi(argc[1]);
42         lim=atoi(argc[2]);
43         seed=atoi(argc[3]);
44     }
45     else scanf("%d %d",&n,&lim);
46
47     if(!seed)srand(time(0));
```

```
    else srand(seed);

    // Get random array
    for(int i=1;i<=n;++i) a[i]=rand()*rand()%lim;

    // Creat a BST
    T[++cnt]=(node){a[1],0,0};
    for(int i=2;i<=n;++i) Insert(1,a[i]);

    // Print Answer, BFS output node data
    queue<int> Q;
    Q.push(1);
    while(!Q.empty())
    {
        int temp=Q.front(); Q.pop();
        if(T[temp].l) Q.push(T[temp].l);
        if(T[temp].r) Q.push(T[temp].r);
        fprintf(ans,"%d ",T[temp].n);
    }
    fprintf(ans,"\n");
    fclose(ans);


    // Output sample input//
    printf("%d\n",n);
    for(int i=1;i<=n;++i)
        printf("%d %d\n",T[i].l-1,T[i].r-1);

    for(int i=1;i<=n;++i)
        swap(a[rand()%n+1],a[rand()%n+1]);

    for(int i=1;i<=n;++i) printf("%d ",a[i]);
    printf("\n");

    return 0;
}
```

5.3 checker.cpp

```
#include <bits/stdc++.h>

using namespace std;

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

```
    scanf("%d%d",&n,&lim);
    srand(time(0));
    while(true)
    {
        char str[110];
        sprintf(str,"gen %d %d %d >in.txt",n,lim,rand());
        system(str);
        system("project2 <in.txt >out.txt");
        if(system("fc out.txt ans.txt")) break;
        else printf("Accepted!\n");
    }
    return 0;
}
```