Build A Binary Search Tree

Fundamental of Data Structure

Project 2

Jie Feng **Yizhou Chen** **Jinze Wu**

2018-11-01

Chapter 1 Introduction

1.1 Background Information

As we all now, binary search tree is a particular kind of binary tree, whose nodes are arranged such that for every node n, all of the nodes' key in n's left subtree less than n' key, and all nodes' key in n's right subtree greater than n' s key. Besides, in the best case(AVLTree), the time complexity of BST is o(logn), in the worst case(Skewed Binary Tree), the time complexity of BST is o(n).

1.2 Problem Description

General Discription

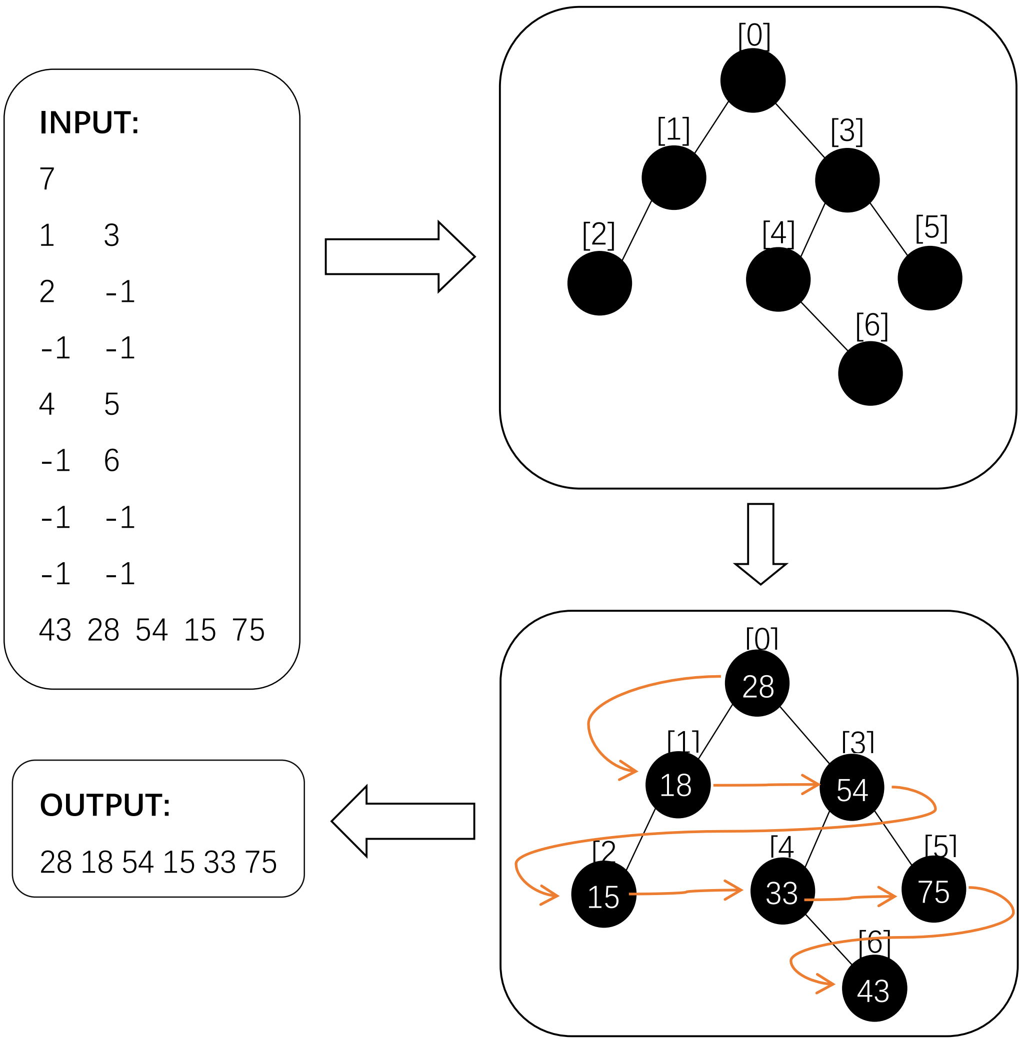
The problem is to complete a binary search tree according to the given structure with key value and output the tree by level-order traversal.

INPUT / OUTPUT Specification

- **INPUT**
  - a positive integer N to define the number of nodes.
  - N lines each contains the left and right children' s index, -1 representing a missing child. All the nodes are numbered in the range [0,N-1].
  - N distinct integers as the key of the nodes.

- **OUTPUT**
  - The lever order traversal sequence of the completed tree.

Sample



1.3 project specification

Program Part

- Build a BST with a given structure.
- Fill the tree with a sequence of distinct integer keys.
    1) Sorting the sequence in ascending order by **Selection Sort**
    2) Add the sorted sequence into the tree by **Inorder Traversal**
- Output the tree by **level order Traversal**.

Test part

- Croctness Test

  We generate random trees with N nodes with the required input and answer, and compare the output of our program and answer for M cases.
- Time Consumption Test

  We use C' s standard library time.h to test the program' s running time when N takes distinct value.

Chapter 2 Algorithm Specification

2.1 Structure Description

This is the Structure of the tree’ s node. The “Element” represents the value, “Left” and “Right” is point to the left and right child node of the parent node.

Struct TreeNode

```
ElementType Element
Pointer Left
Pointer Right
```

2.2 A Sketch of Main Program

2.2.1 Specification

The whole program can be seperated into three parts.

1) Input the data to create binary search tree in **main function**.
2) Sorting the key by Selectsort in **main function**.
3) Fill the tree by using inorder traversal in **completethetree()**
4) Using Queue ADT to output the tree in level order traversal in the function of **output( )**

2.2.2 Pseudo code:

Main Function

```
function main( )
    Input : N ← total number of the nodes
    For i = 0 to N-1 do
        Input : leftnode rightnode
    For i = 0 to N-1 do
        Input : keys[i]
```

```
        Selectsort keys[ ]
        Call completethetree( tree , keys , N )
        Call output( tree, N )
End function
```

2.3 Fucntion completethetree()

2.3.1 Specification

After building the tree's structure and sorting the key, we use this function to assign the TreeNode with values. In this function, we fill the binary search tree by using stack to do inorder traversal, as the binary seach tree must possess a non-decreasing sequence as infix-order traversal sequence.

2.3.2 Pseudo code:

Void completethetree( )

```
Function completethetree( )
        create stack
        top ← -1
        i ← 0
        t← root
        while the stack is not empty
                while t is not NULL
                        Push(key[i])
                        t←t’ s left child
                End while
                If top ≠ -1 then
                        pop( )
                        t->element ← keys[i]
                        i ← i + 1
                        t ← t->right
```

```
        End if
    End while
End function
```

2.4 Fucntion output()

2.4.1 Specification

Up to now, we have already created the completed tree and the only thing we need to do now is to output the tree in level order traversal. In this step, we use Queue ADT to output the Tree. The property of a tree’ s level order is that if the order of each group of the child node follows the order of their parent, so every time we output a parent node, we deque this node and enque its child from left to right, in this way we ensured the level-order.

2.4.2 Pseudo code:

Void output( )

```
Function output( )
    Enqueue(t)
    While t is not empty do
        Visit ( T ← Dequeue(t) )
        For each child C of T do
            Enqueue(t)
    End While
```

Chapter 3 Testing Results

3.1 Purpose

Our purpose is to test the correctness and efficiency of the algorithm.

3.2 Specification

3.2.1 Correctness Test

Special Data Test

Firstly, we test the algorithm with common sample and extreme data, including a common sample, tree with 1 node and skewed trees to see if program works well.

Massive Data Test

Secondly, we test the algorithm with massive random data, to further ensure its correctness. We implement a program **GenerateTree.c** to randomly generate a binary search tree with the required input data and output data. We output both of them into a file **test.in**. Then two programs, **testfile.c (the test form of our program)** and **answer.c (directly receive the answer provided by generatetree.c and output it)**, receive the data from test.in and output their answer to **test1.out** and **test2.out**. Then we implement a program **compare.c**, to compare **test1.out** and **test2.out** to see if the output of our program matches the correct answer.

3.2.2 Efficiency Test

Time Consumption Test

We use **GenerateTree.c** randomly generate a binary search tree with nodes= 10,30,50,70 90,100 and use **TestTime.c** to find the average time consumption of each cases, using C’ s standard library time.h.

3.3 Test Data

3.3.1 Special data test

Sample1 (sample)

purpose : To test the common case of small-scale comprehensive test.

input

```
9
1 6
2 3
-1 -1
-1 4
5 -1
-1 -1
7 -1
-1
-1 -1
73 45 11 58 82 25 67 38 42
```

output

```
58 25 82 11 38 67 45 73 42
```

Sample2 (only 1 node)

purpose : To test the special case of smallest size tree test.

input

```
1
-1 -1
25
```

output

```
25
```

Sample3 (chain)

purpose : To test the special case of a skewed tree with only left child.

input

```
4
1 -1
2 -1
3 -1
-1 -1
12 45 98 25
```

output

```
98 45 25 12
```

Sample4 (zigzag tree)

purpose : To test the special case of a skewed zigzagging tree.

input

```
5
-1 1
2 -1
-1 3
4 -1
-1 -1
12 32 45 98 25
```

output

```
12 98 25 45 32
```

3.3.2 Mass data test

sample (nodes=100) with purpose: This is a comprehensive test, an example of our random tests.

input

```
100
1 8
2 -1
3 6
4 -1
-1 5
-1 -1
7 -1
-1 -1
9 13
10 12
-1 11
-1 -1
-1 -1
14 25
15 20
16 19
17 -1
-1 18
-1 -1
-1 -1
21 22
-1 -1
23 -1
24 -1
-1 -1
26 64
27 37
```

```
28 32
29 -1
30 -1
31 -1
-1 -1
-1 33
-1 34
35 36
-1 -1
-1 -1
38 49
39 42
40 -1
41 -1
-1 -1
43 44
-1 -1
45 -1
46 47
-1 -1
48 -1
-1 -1
50 57
51 -1
52 55
-1 53
54 -1
-1 -1
```

```
56 -1
-1 -1
58 62
59 -1
-1 60
-1 61
-1 -1
-1 63
-1 -1
-1 65
66 98
67 93
68 69
-1 -1
70 80
-1 71
72 75
73 74
-1 -1
-1 -1
76 79
-1 77
-1 78
-1 -1
-1 -1
81 84
82 83
-1 -1
```

```
-1 -1
85 86
-1 -1
87 92
88 91
89 90
```

```
-1 -1
-1 -1
-1 -1
-1 -1
94 -1
95 -1
```

```
-1 96
-1 97
-1 -1
-1 99
-1 -1
```

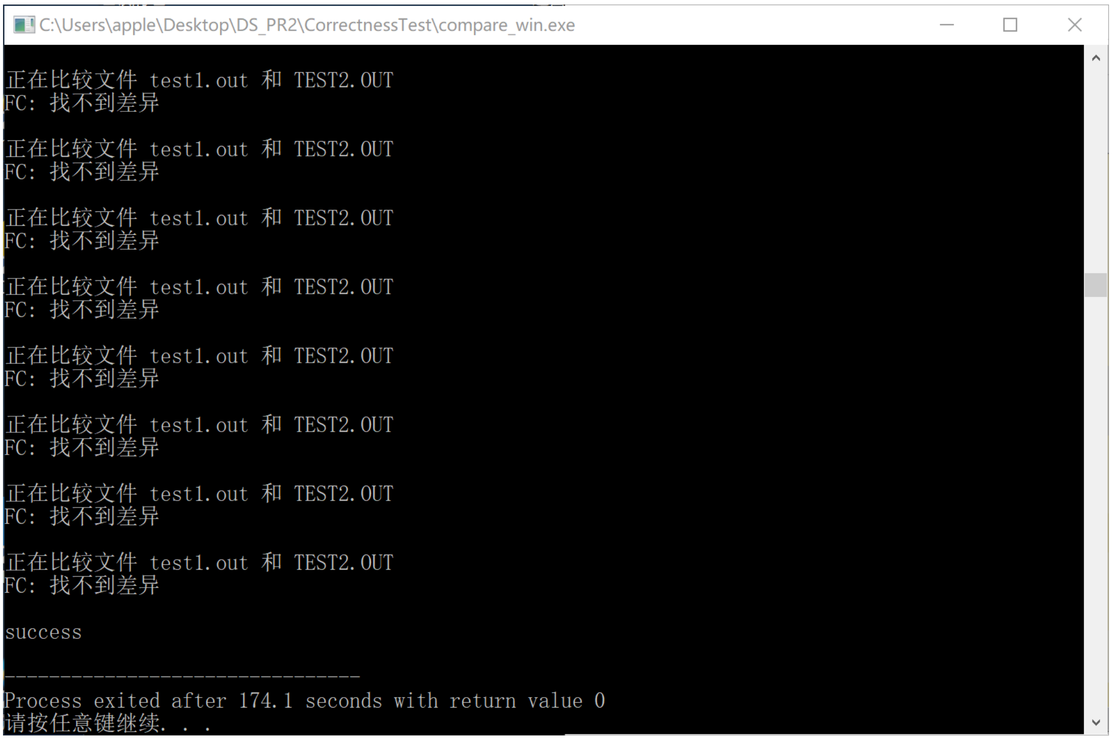
```
985 951 980 979 992 957 939 984 940 978 943 931
956 935 981 965 944 912 869 989 952 909 926 923
962 977 885 973 930 910 983 925 921 958 967 982
991 974 922 945 955 886 946 924 972 997 895 933
968 914 990 897 998 994 963 970 995 913 917 934
966 964 927 993 906 988 919 975 986 969 915 959
942 938 996 916 954 960 937 907 953 932 949 903
941 987 976 918 971 803 928 920 950 948 999 947
793 936 961 929
```

output

```
903 897 912 885 909 924 869 895 906 910 918 963
793 886 907 916 920 935 964 803 915 917 919 923
929 947 997 913 922 928 930 939 955 991 998 914
921 927 931 938 941 954 960 966 996 999 926 933
937 940 946 951 959 961 965 977 995 925 932 934
936 943 948 953 956 962 967 981 992 942 945 950
952 957 971 979 983 993 944 949 958 969 975 978
980 982 989 994 968 970 972 976 987 990 973 985
988 974 984 986
```

Batch test (nodes=100 cases=1000)

purpose: To confirm the correctness through a great number of test cases.



we tested 1000 cases of random trees whose nodes=100, and there existed no error. All the outputs matched the correct answer.

3.2 Time Consumption test

N	5	20	40	60	80	100
ITERATIONS	100000	10000	1000	1000	1000	1000
TOTALTIME	0.0402	0.0258	0.007800	0.014400	0.02500	0.03900
DURATION	0.000000402	0.00000258	0.0000078	0.0000144	0.000025	0.000039

According to our pre-analysis Exponent=2

Number	1	2	3	4	5
Theory= $(\frac{N_{latter}}{N_{former}})^{Exponent}$	15.0000	4.0000	2.2500	1.7778	1.5625
Data= $(\frac{Duration_{latter}}{Duration_{former}})^{Exponent}$	6.4179	3.0233	1.8462	1.7361	1.5600

Chapter 4 Analysis and Comments

4.1 Analysis

4.1.1 Correctness

In the first part, we choose some special cases for testing, and we can see that all the outputs are correct. In the second part, we generate a great amount of random data, and all the outputs are the same to the standard answer. So, we can basically confirm that our program is correct.

4.1.2 Time Complexity

The program can be mainly divided into three parts: constructing the binary searching tree, completing the tree, and output the level-order of the tree.

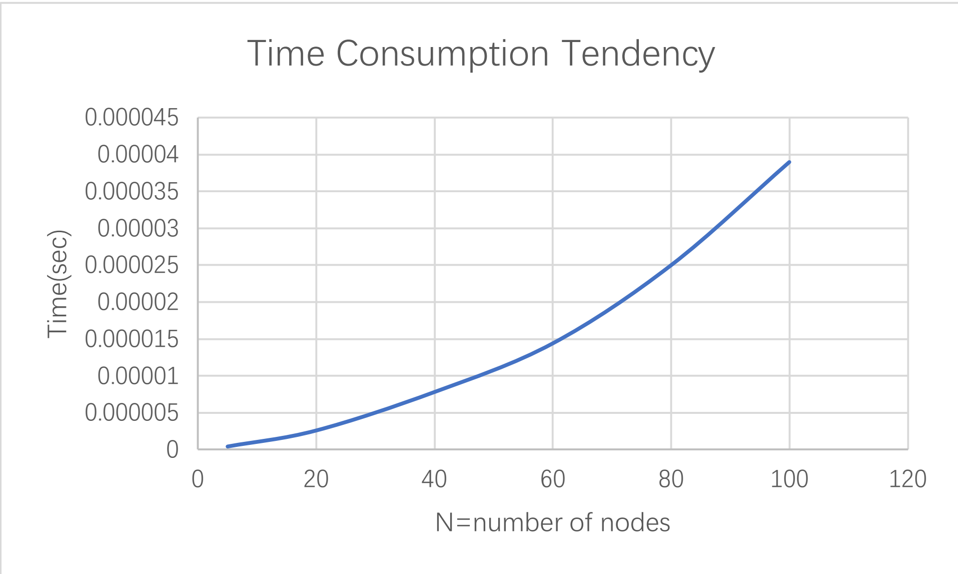
Construct the Tree: only need 1 loop to construct every node ->O(N)

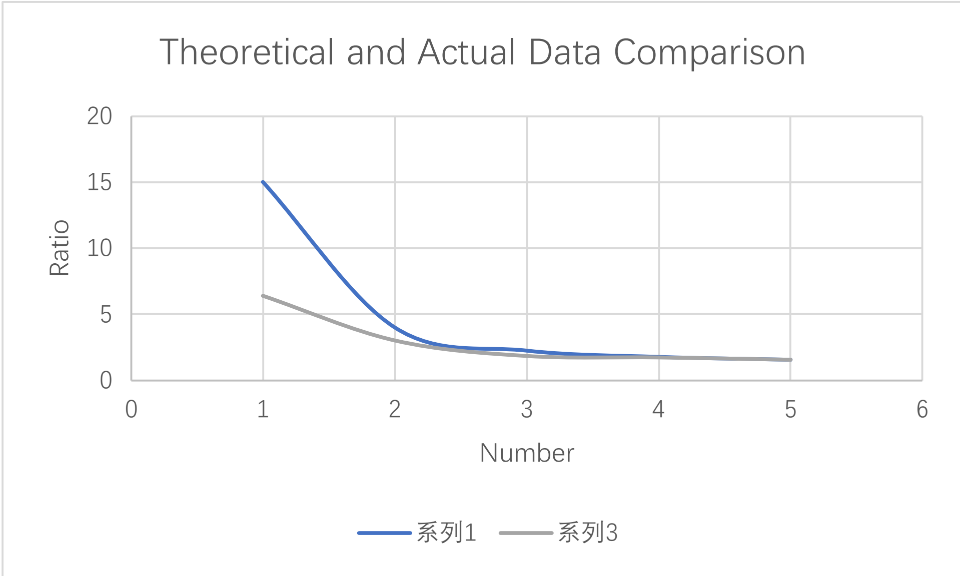
Sort: selection sort O(N^2)

Complete the tree: implement an in-order traversal O(n)

Output Level-order: each node will enqueue and dequeue for 1 time O(n)

To sum it up, the time complexity of the program should be O(n^2).





From the first graph, we can see the tendency approximately matches O(n^2). From the second graph, we can see that when n grows, the fitting degree of the theoretical and actual data also gets higher. So we confirm that the time complexity of our program is O(n^2).

4.1.3 Space Complexity

We implement a stack to implement the in-order traversal instead of directly recursion, using the system’ s stack. This is an optimization of extra space complexity, help to avoid stack overflow at the same time. As we use malloc() function to allocate the memory for each nodes every time, so the space complexity will be O(n).

4.2 Comments

1) It’ s the selection sort mainly contributes to the O(n^2) time complexity, change it to quick sort or heap sort and improve the time complexity from O(n^2) to O(nlogn).

2) Implement a stack instead of recursion is a good optimization of space complexity.

Chapter5 Appendix

A.1 Program

A.1.1 Header

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

struct node;
typedef struct node* ptrtoNode;

struct node{
 int element;     /*the key*/
 ptrtoNode left;  /* point to the left child*/
 ptrtoNode right;  /* point to the right child*/
};
void completethetree(ptrtoNode tree,int* a,int N);
/* After build the tree,use this function to assign the nodes with values*/
void output(ptrtoNode tree, int N);
/* This function is used to output the result in level order*/
```

A.1.2 main

```
int main(void)
{
    int N;
    int i,j,k,tmp;
    int leftnode,rightnode;
     /*the leftnode and rightnode are the left child_index and right child_index*/
    int* keys;
    /*the keys is a array to store the integer keys*/
    scanf("%d",&N);/* N nodes contained*/

    ptrtoNode tree = (ptrtoNode)malloc(sizeof(struct node)*N);
    /* this pointer is an array of structs ,they are numbered from 0 to N-1,and the number is
        exactly its index*/
    keys=(int *)malloc(sizeof(int)*N);

    if(tree==NULL||keys==NULL)
    exit(0);
    /* if malloc fails,quit*/
    for(i=0;i<N;i++)
    {
        scanf("%d %d",&leftnode,&rightnode);

        if(leftnode==-1)
        tree[i].left = NULL;
        else
        tree[i].left = &tree[leftnode]; /* build the left child*/

        if(rightnode==-1)
        tree[i].right = NULL;
        else
        tree[i].right = &tree[rightnode];/* build the right child*/
    }
    /* the input folowed the order of the tree array's index,so use one single*/
    /*'for' to traverse the tree array and build up the binary tree*/
```

```c
    for(i=0;i<N;i++)
    {
        scanf("%d",&keys[i]);
    }

    /*get the N interger keys*/
    for(i=0;i<N-1;i++)
    {
        k=i;
        for(j=i;j<N;j++)
        {
            if(keys[j]<keys[k])
            k=j;               /*find the smallest one*/
        }
        tmp=keys[i];
        keys[i]=keys[k];
        keys[k]=tmp;      /* exchange the number*/
    }/*get the keys sorted,using an old algorithm called selectsort*/
    completethetree(tree,keys,N); /*fill the empty tree with numbers in order*/
    output(tree, N);
    return 0;
}
```

A.1.3 completethetree()

```c
void completethetree(ptrtoNode tree,int* a,int N)
{
  ptrtoNode* stack;
  ptrtoNode t;

  stack = (ptrtoNode*)malloc(sizeof(ptrtoNode)*N);/*build up the stack*/
  if(stack==NULL)
  return;
  /* if empty,quit*/

  int top=-1;/* the top of stack,if top=-1,the stack is empty*/
  int i=0;
  t=tree;   /* tree equals to the address of the root*/
  while(t||top!=-1)/* the loop stops after the tree is all traversed and the stack is empty*/
  {
    while(t)
    {
      top++;
      stack[top]=t;   /* push in*/
      t=t->left;      /*in this loop,we find the far left node*/
    }
    if(top!=-1)
    {
      t=stack[top];     /* pop the node*/
      stack[top]=NULL;  /* clean it in the stack*/
      top--;
      t->element=a[i];  /*assign the poped node*/
      i++;              /*move to the bigger key*/
      t=t->right;    /*turn to the right child*/
    }
  }
  free(stack);  /*free the space*/
}
/* for binary search tree,we can output the keys in order by InOrderTraversal*/
/* as we have built the tree,we can assign values to the nodes exactly in the same way*/
/* by just changing printing to assiging(the a is the sorted keys)*/
```

A.1.4 output()

```c
 void output(ptrtoNode tree, int N)
{
  ptrtoNode* queue;/* use queue to output in level order*/
  ptrtoNode t;

  queue=(ptrtoNode*)malloc(sizeof(ptrtoNode)*N);
  if(queue==NULL)
  return;
  /*if empty , quit*/
  int front,rear;/* the front and rear end of the queue*/
  int flag=0;  /*the flag is used to show whether it's the first time to print or not,'0' menas no number was printed before*/
  front=0;rear=0;

  queue[0]=tree; /* enqueue the root first*/
  rear++;        /* the rear end move */
  while(front!=rear)/*while the queue is not empty*/
  {
    t=queue[front]; /*pop out*/
    if(flag==0)
    {
      printf("%d",queue[front]->element);
      flag=1;         /*if the first one,printf the single number*/
    }
    else
    printf(" %d",queue[front]->element);/* else with a blank in front of the number */
    queue[front++]=NULL;    /*deque from the front end*/
    if(t->left)
    queue[rear++]=t->left; /* if it has left child,enque*/
    if(t->right)
    queue[rear++]=t->right; /*if it has right child,enque*/
  }
  printf("\n");  /*line break*/
  free(queue);  /*free the space*/
}
```

A.2 Test Program

A.2.1 GenerateTree

- **CreateBTree()**

```
/*
 *Description: This procedure is to generate a binary search tree using the method of generating tree
     with infix-order and level-order
 *Parameter[in] il the left bound of infix-order in recursion
 *Parameter[in] ir the right bound of infix-order in recursion
 *Parameter[in] ll the left bound of pseudolevel-order in recursion
 *Parameter[in] ir the right bound of pseudolevel-order in recursion
 *return tree  the root of the generated tree in recursion
 */

Tree CreateBtree(int il,int ir,int ll,int lr) {
    Tree r;
    int i,j,key=0;
    if (il>ir) return NULL;
    /* the two loops are to find the current root which must be the first key value shown both in the
        infix-order and pseudolevel-order */
    for (i=ll;i<=lr;i++) {
        for (j=il;j<=ir;j++)
            if (pseudolevel[i]==infix[j]) {
                key=pseudolevel[i];
                break;
            }
        if (key!=0) break;
    };
    r=(Tree)malloc(sizeof(struct treenode));
    r->Data=key;  //key if the key of each nodes
    r->Number=N;  //N is the index of each nodes
    Btree[N++]=r; //store the nodes in the Btree array according to the number
    r->Left=CreateBtree(il,j-1,ll,lr); //recurs to create the left-child-tree
    r->Right=CreateBtree(j+1,ir,ll,lr); //recurs to creat the right-child-tree
    return r;
}
```

- **GenerateLevelOrder()**

```
/*
 *Description: This procedure is find the level-order of the binary search tree we generated
 *Parameter[in] root the root of the binary tree
 */

void GenerateLevelOrder(Tree root) {
    Tree q[110];
    int h=0,t=0;
    q[h]=root;
    while (h<=t) {
        level[M++]=q[h]->Data; //fetch the value of the front of the queue as the next value in level-
            order
        if (q[h]->Left!=NULL) q[++t]=q[h]->Left;  //enque its left-child
        if (q[h]->Right!=NULL) q[++t]=q[h]->Right; //enque its right-child
        h++; //deque the front element
    }
}
```

- # OutputInputData()

```c
/*
 *Description: This procedure is to output the structure of the generated tree in the required form using as the input of our program
 *Parameter[in] n the number of nodes of the tree
 */

void OutputInputData(int n) {
    int i,k,temp;
    printf("%d\n",n);

    for (i=0;i<n;i++) {
        if (Btree[i]->Left==NULL) printf("-1 ");  //if no left-child output -1
        else printf("%d ",Btree[i]->Left->Number);  //output the left-child's index
        if (Btree[i]->Right==NULL) printf("-1 ");  //if no right-childe output -1
        else printf("%d ",Btree[i]->Right->Number);  //output the right-child's index
        printf("\n");
    }

    /* this part uses shuffle algorithm to disorder the infix-order into a random sequence*/
    for (i=1;i<=n;i++)
        input[i]=infix[i];
    for (i=n;i>=2;i--) {
        k=rand()%i+1;  // choose a radom element before the current element
        /* swap the two element */
        temp=input[i];
        input[i]=input[k];
        input[k]=temp;
    }
    /* this part is to output the random squence as the required input keys sequence */
    for (i=1;i<=n;i++) {
        printf("%d",input[i]);
        if (i!=n) printf(" ");
        else printf("\n");
    }
}
```

- # OutputOutputData()

```c
/*
 *Description: This procedure is to output the level-order of the generated tree for answer.c to output the correct answer
 *Parameter[in] n the number of nodes of the tree
 */

void OutputOutputData(int n) {
    int i;
    for (i=1;i<=n;i++) {
        printf("%d",level[i]);
        if (i!=n) printf(" ");
        else printf("\n");
    }
}
```

- # Main For CorrectnessTest()

```c
int main(void) {
    freopen("input.in","r",stdin);
    freopen("test.in","w",stdout); //this is to write the output of this program into a file test.in

    int n,i,k,temp;
    int NUMBER_OF_NODES,ITERATIONS;

    /* initialize */
    memset(infix,0,sizeof(infix));
    memset(pseudolevel,0,sizeof(pseudolevel));
    memset(level,0,sizeof(level));
    memset(input,0,sizeof(input));

```

```c
    scanf("%d%d",&NUMBER_OF_NODES,&ITERATIONS) ;
    n=NUMBER_OF_NODES;  //choose the size of the tree
    srand((unsigned)time(NULL));
    /* randomly create an increasing squence as the infix-order of the binary tree*/
    for (i=1;i<=n;i++)
        infix[i]=rand()%(1000-(n-i)-infix[i-1]-1)+infix[i-1]+1;
    /* shuffle the infix-order to generate a constraint */
    for (i=1;i<=n;i++)
        pseudolevel[i]=infix[i];
    for (i=n;i>=2;i--) {
        k=rand()%i+1;
        temp=pseudolevel[i];
        pseudolevel[i]=pseudolevel[k];
        pseudolevel[k]=temp;
    }

    /* use the infix-order and the constraint to generate a binary search tree*/
    root=CreateBtree(1,n,1,n);

    /* generate the level-order of the tree */
    GenerateLevelOrder(root);


    /* output the input data of the test program*/
    OutputInputData(n);

    /* output the answer */
    OutputOutputData(n);

    return 0;
}
```

- # Main For TimeConsumptionTest()

```c
int main(void) {
    freopen("input.in","r",stdin);  //read the user's input
    freopen("test.in","w",stdout);  //this is to write the output of this program into a file test.in

    int n,i,k,temp;
    int ITERATIONS,NUMBER_OF_NODES;;

    /* initialize */
    memset(infix,0,sizeof(infix));
    memset(pseudolevel,0,sizeof(pseudolevel));
    memset(level,0,sizeof(level));
    memset(input,0,sizeof(input));

    scanf("%d%d",&NUMBER_OF_NODES,&ITERATIONS);
    n=NUMBER_OF_NODES;  //choose the size of the tree
    srand((unsigned)time(NULL));
    /* randomly create an increasing squence as the infix-order of the binary tree*/
    for (i=1;i<=n;i++)
        infix[i]=rand()%(1000-(n-i)-infix[i-1]-1)+infix[i-1]+1;
    /* shuffle the infix-order to generate a constraint */
    for (i=1;i<=n;i++)
        pseudolevel[i]=infix[i];
    for (i=n;i>=2;i--) {
        k=rand()%i+1;
        temp=pseudolevel[i];
        pseudolevel[i]=pseudolevel[k];
        pseudolevel[k]=temp;
    }

    /* use the infix-order and the constraint to generate a binary search tree*/
    root=CreateBtree(1,n,1,n);

    OutputInputData(n,ITERATIONS);

    return 0;
}
```

- # A.2.2 answer()

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

int main(void) {
    freopen("test.in","r",stdin);  //read the data from the file test.in
    freopen("test2.out","w",stdout);

    int n,x,y,i;
    /* this part is to read the requred input*/
    scanf("%d",&n);
    for (i=1;i<=n;i++)
        scanf("%d%d",&x,&y);
    for (i=1;i<=n;i++)
        scanf("%d",&x);

    /* this part is to read the answer and directly output it */
    for (i=1;i<=n;i++) {
        scanf("%d",&x);
        if (i!=n) printf("%d ",x);
        else printf("%d\n",x);
    }
    return 0;
}
```

- # A.2.2 compare()

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

int main(void) {
    int i,nodes,TEST_TIMES;
    freopen("input.in","r",stdin);
    scanf("%d%d",&nodes,&TEST_TIMES);
    /* test for TEST_TIMES times*/
    for (i=1;i<=TEST_TIMES;i++) {
        /*run the three executable fils*/
        system("GenerateTree");
        system("testfile");
        system("answer");
        /* compare the output of our program and the answer*/
        if (system("fc test1.out test2.out")) {
            printf("wrong in --> %d",i); // if they are different output "wrong"
            system("pause");
            return 0;
        }
    }
    printf("success\n"); // if the program passed all the test output "success"
    system("pause");
    return 0;
}

```

A.2.3 TestTime()

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

int main(void) {
    system("GenerateTree");  //run the generating program
    system("testfile");  //run the testing program
    system("pause");
}

```

Appendix B Declaration

We hereby declare that all the work done in the project titled "Building A Binary Search Tree" is of our independent effort as a group.

Duty Assignments

Programmer: Jie Feng

Tester: Yizhou Chen

Reporter: Jinze Wu