FDS Project

A + B with Binary Search Trees

Date 2024-3-29

1 Introduction

This task involves finding pairs of numbers from two Binary Search Trees (BST) that sum up to a specified target integer.

What to be done:

The goal is to find at least one pair of numbers exists, where one number is from T1 and the other from T2, such that their sum equals N.

And I will use $\{a_n\}$,$\{b_n\}$ to store the input and then use my function to solve this problem.

Why:

This task is important because it demonstrates the use of binary search trees to solve the searching problem in an more efficient way.

2 Algorithm Specification

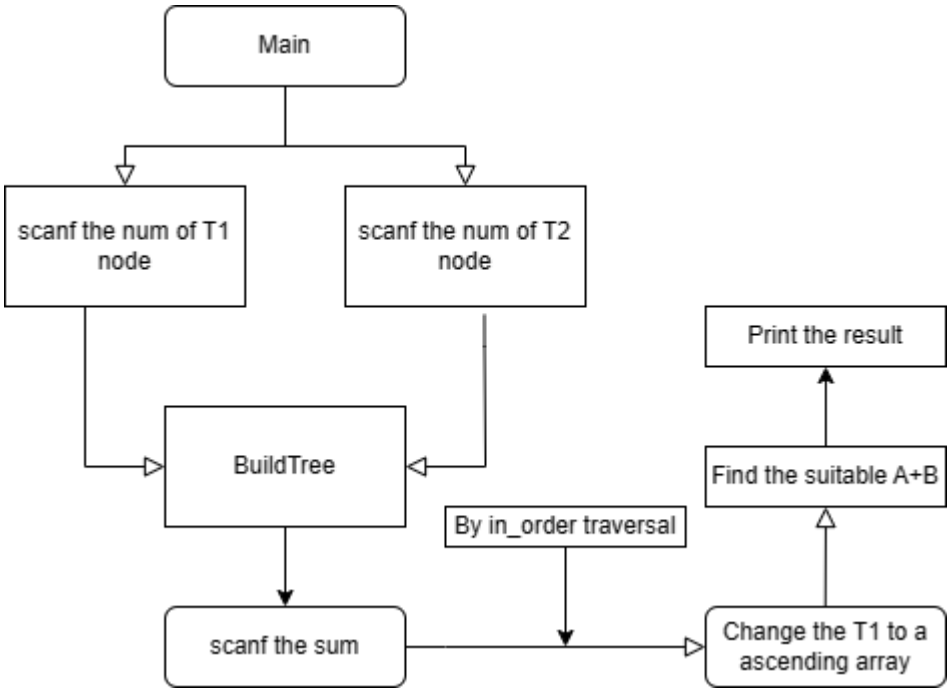
2.1 Data structure

The TreeNode structure is used to represent a node in the binary search tree. It contains the following fields:

- val: an integer value representing the key of the node.
- left: a pointer to the left child of the node.
- right: a pointer to the right child of the node.

2.2 Flowchart

If you want a faster understanding,you can also look at this flowchart.



2.3 pseudo-code

The pseudo-code is as follows:
(Tip:each piece of code is annotated in detail in Appendix.So if you can't understand the pseudo-code,you could find the corresponding function in Appendix which has a better comments.)

2.3.1 buildTree

buildTree is one key algorithm!

```
function buildTree(n):
    T = array of TreeNode of size n
    root = NULL
    // read input and find the root
    for i from 0 to n-1:
        read a[i], b[i]
        T[i] = newTreeNode(a[i])
        if b[i] is -1:
            root = T[i]
    // build the tree according to the index
    for i from 0 to n-1:
```

```
        if b[i] is not -1:
            if T[i].val < T[b[i]].val:
                T[b[i]].left = T[i]
            else:
                T[b[i]].right = T[i]
    return root
```

2.3.2 printTree

```
function printTree(T, isroot):
    if T is NULL:
        return
    if isroot:
        print T.val
    else:
        print " ", T.val //" " is a blank space
    printTree(T.left, 0)
    printTree(T.right, 0)
```

2.3.3 in_order traversal

in_order traversal is one key algorithm!

```
function in_order(T, size, arr):
    // save the value in ascending order
    if T is NULL:
        return
    in_order(T.left, size, arr)
    arr[size++] = T.val
    in_order(T.right, size, arr)
```

2.3.4 main

```
main:
    number_T1, number_T2, n = read from input
    // build the tree
    T1 = buildTree(number_T1)
    T2 = buildTree(number_T2)

    // change T1 to array whose value is in ascending order
    array_T1 = array of size 20005
    size_T1 = 0
    in_order(T1, size_T1, array_T1)

    // find the A+B=N
    if_find = 0
    for i from 0 to number_T1-1:
        // avoid duplicate data
        if i < number_T1 and array_T1[i] == array_T1[i+1]:
            continue
        tmp_T2 = T2
        while tmp_T2 is not NULL:
            if array_T1[i] + tmp_T2.val == n:
                if if_find is 0:
                    print "true"
                if_find = 1
                print n, "=", array_T1[i], "+", tmp_T2.val
                break
            else if array_T1[i] + tmp_T2.val < n:
                tmp_T2 = tmp_T2.right
            else:
                tmp_T2 = tmp_T2.left
    if if_find is 0:
        print "false"

    // print the tree
```

```
    printTree(T1, 1)
    printTree(T2, 1)
```

3 Testing Results

3.1 Input Specification

Each input begin an integer n_1 which is the number of nodes in T1.
Then n_1 lines follow, where the ith line contains the key value k and the parent node index of the ith node.After T1, T2 is given in the same format of T1.
Finally the last line gives the target N which is the sum of A+B

3.2 Test cases and results

These test cases may cover almost every situation.

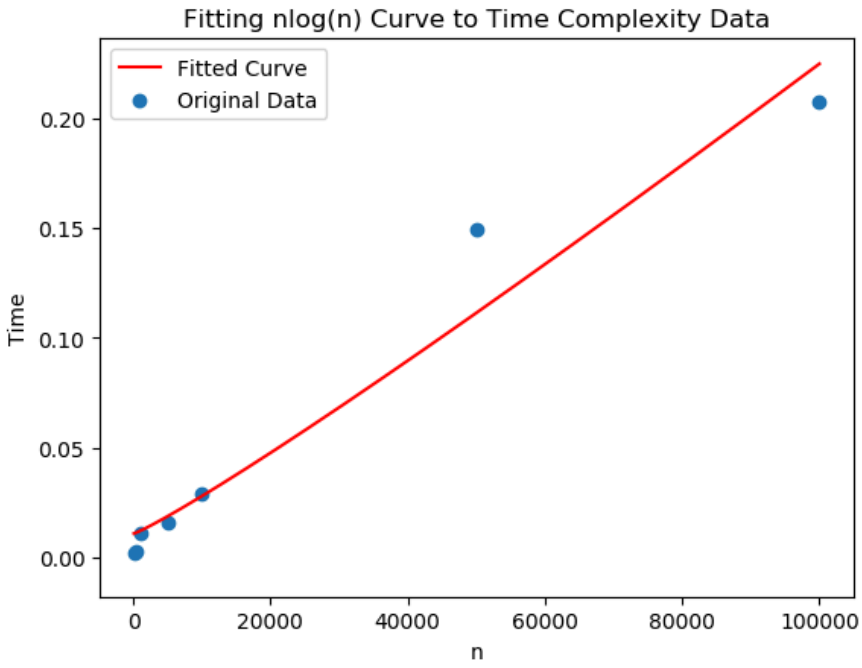
Index	Specification	status
1	Sample Input 1 from PTA	pass
2	Sample Input 2 from PTA	pass
3	Two Empty Trees	pass
4	No Solution Exits	pass
5	Trees with Repeated nodes	pass
6	Extreme Cases with $n_1 = n_2 = 10^6$	pass
7	Extreme Cases with $n_1 = n_2 = 10^6$	pass
8	Worst Case(Degenerate into list)	pass

> Tips:
> The worst case may be too large to be tested,so you need to be patient to wait the input.And the output may be also very slow and large,so would you mind waiting a little longer.

3.3 Time

Here we set T1 and T2 have same number of nodes.

n	100	500	1000	5000	10000	50000	100000
Time	0.002	0.003	0.011	0.016	0.029	0.149	0.207



4 Analysis and Comments

- **TimeComplexity:O(nlogm)**
  The main part of the code that contributes to the time complexity is the nested loop that iterates through the elements of T1 and T2 trees. The outer loop iterates through the elements of T1, which has n elements. The inner loop iterates through the elements of T2, which has m elements.
  Since the trees are binary search trees, the time complexity of searching for an element in a tree is O(logn) in the average case. Therefore, the inner loop has a time complexity of O(logm).
  Since the inner loop is nested inside the outer loop, the total time complexity is O(nlogm).
  However, in the worst case scenario, where the trees are skewed and have a height of n or m, the time complexity of searching for an element in a tree becomes O(n) or O(m) respectively. In this case, the total time complexity would be O(nm).
  Therefore, the overall time complexity of the given code is O(nlogm) in the average case and O(nm) in the worst case.

- **SpaceComplexity:O(n)**
  The space complexity of the code is O(n), where n is the number of nodes in the tree T1. This is because the space is primarily used by the array_T1 array (size n) and the recursive call stacks of the in_order and printTree functions (size proportional to the height of the trees).

- **Further possible comments**:
    - We can use the AVL tree(balanced binary search tree) to avoid the worst case scenario.Then the time complexity will be O(nlogm).
    - We can also use the hash table to store the value of T1,and then find the suitable value in T2 to meet the requirement of A+B=N.Then the time complexity will be O(n).

5 Appendix: Source Code

```c
#include<stdio.h>
#include<stdlib.h>
// declare the fp,fpout
FILE *fp;
FILE *fpout;
// Declare the global variables
int size = 0;
int array_T1[20000005];
// Define the structure of the tree node
typedef struct TreeNode{
    int val;
    struct TreeNode *left,*right;
}TreeNode;
// Define the function to create a new tree node
TreeNode *newTreeNode(int val){
    TreeNode *new = malloc(sizeof(TreeNode));
    new->val = val;
    // Initialize the left and right children to NULL
    new->left=new->right=NULL;
    return new;
}
// Define the function to build a tree
TreeNode *buildTree(int n){
    //used to store the first line
    int *a = malloc(n*sizeof(int));
    //used to store the second line
    int *b = malloc(n*sizeof(int));
    //used to store each tree node
    TreeNode **T = malloc(n*sizeof(TreeNode*));
```

```
    //used to store the root of the tree
    TreeNode *root = NULL;
    for(int i=0;i<n;i++){
        // Read the values from the input
        fscanf(fp,"%d %d",&a[i],&b[i]);
        T[i] = newTreeNode(a[i]);
        // if b[i] is -1 ,then it's root
        if(b[i] == -1) root = T[i];
    }
    // Connect the nodes
    // according to it's parent node index
    for(int i = 0;i<n;i++){
        if(b[i] != -1){
            /* If the value of the node is less than
            the value of its parent node, then it is
            the left child of its parent node*/
            if(T[i]->val < T[b[i]]->val) T[b[i]]->left = T[i];
            // Otherwise, it is the right child of its parent node
            else T[b[i]]->right = T[i];
        }
    }
    // free to avoid error
    free(a); free(b);free(T);
    return root;
}
// printf the tree
// is preOrder
void printTree(TreeNode *T,int isroot){
    if(T == NULL) return;
    if(isroot) fprintf(fpout,"%d",T->val);
    else fprintf(fpout," %d",T->val);
    printTree(T->left,0);
    printTree(T->right,0);
}
// Store data in increasing order
void in_order(TreeNode *T,int arr[]){
    if(T == NULL) return;
    in_order(T->left,arr);
```

```
    //store the value of each node in the array
    arr[size++] = T->val;
    in_order(T->right,arr);
}
int main(int argc,char *argv[]){
    // The basic operation according to the question
    char in[20] = ".\\data\\8.txt";
    char out[20] = ".\\data\\8out.txt";
    // chaning dirs according to the system
     if(argv[1][1] == 'l') in[1]=in[6]=out[1]=out[6]='/';
     in[7] = out[7] = argv[2][1];
    // represent the input and output file
    printf("%s\n",in);
    printf("%s\n",out);
    fp = fopen(in,"r"); //open the input file
    fpout = fopen(out,"w"); //open the output file
    // The main function
    int number_T1,number_T2,n;
    fscanf(fp,"%d",&number_T1);
    // build the tree according to the input
    TreeNode *T1 = buildTree(number_T1);
    fscanf(fp,"%d",&number_T2);
    TreeNode *T2 = buildTree(number_T2);
    fscanf(fp,"%d",&n);
    // Sort the values in T1
    int size_T1 = 0;
    in_order(T1,array_T1);
    // Find the number A from T1 and B from T2 such that A+B=N
    int if_find = 0; // used as a flag
    for(int i = 0;i<number_T1;i++){
        // avoid repetition
        if(i < number_T1 && array_T1[i] == array_T1[i+1]) continue;
        TreeNode *tmp_T2 = T2;
        while(tmp_T2!=NULL){
            if(array_T1[i]+tmp_T2->val == n){
                // Print the result first
                if(if_find == 0) fprintf(fpout,"true\n");
                // never print ture again
```

```
                if_find = 1;
                // output
                fprintf(fpout,"%d = %d + %d\n",
                n,array_T1[i],tmp_T2->val);
                break;
            }
            // A+B < n imply B is small,so try to find in right leaf
            else if(array_T1[i]+tmp_T2->val < n){
                tmp_T2 = tmp_T2->right;
            }
            // A+B > n imply B is too big,so try to find in left leaf
            else tmp_T2 = tmp_T2->left;
        }
    }
    if(if_find == 0) fprintf(fpout,"false\n");
    // Print the T1 tree
    printTree(T1,1);
    fprintf(fpout,"\n");
    // Print the T2 tree
    printTree(T2,1);
    return 0;
}
```

6 Declaration

I hereby declare that all the work done in this project titled "A+B with Binary Search Trees" is of my independent effort