Fundamentals of Data Structures

Project 2: A+B with Binary Search Trees(Normal)

Date: 2025.3.29

2024-2025 Spring & Summer Semester

Contents

Chapter 1： Introduction

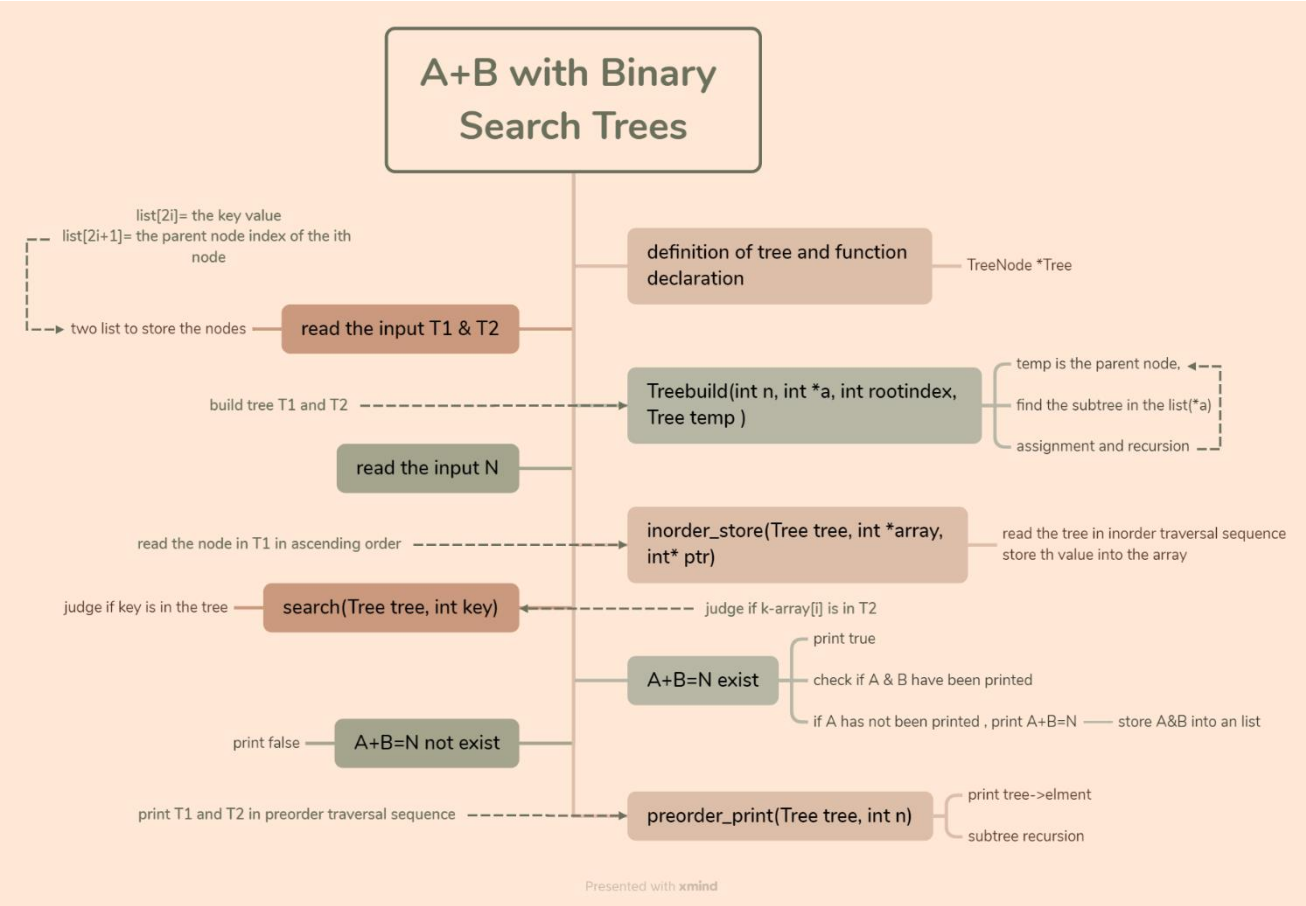
A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

- 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.

Given two binary search trees T1 and T2, and an integer N, we need to judge if an integer A from T1 and B from T2 such That A+B=N. Then we should print the preorder traversal sequences of T1 and T2.

Chapter 2: Algorithm Specification

The specifications of different functions are as follows.



Chapter 3: Testing Results

I used the given example from PTA website. The results are as follows.

Sample Input 1:

```
8
12 2
16 5
13 4
18 5
15 -1
17 4
14 2
18 3
7
20 -1
16 0
25 0
13 1
18 1
21 2
28 2
36
```

Sample Output 1:

```
true
36 = 15 + 21
36 = 16 + 20
36 = 18 + 18
15 13 12 14 17 16 18 18
20 16 13 18 25 21 28
```

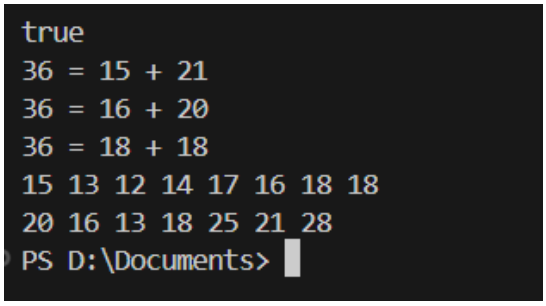


Figure 1: The output 1

Sample Input 2:

```
5
10 -1
5 0
15 0
2 1
7 1
3
15 -1
10 0
20 0
40
```

Sample Output 2:

```
false
10 5 2 7 15
15 10 20
```

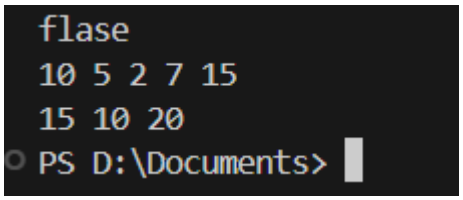


Figure 2: The output 2

I generated several testing points with Deepseek, you can find them in "examples.txt" and judge the program.

Chapter 4: Analysis and Comments

The Treebuild function constructs the tree by iterating through all nodes for each parent, leading to $O(n^2)$ time complexity. Use a hash map to store children by parent index, reducing construction to O(n) time.

The code does not free allocated memory for trees at the same time. While not critical for passing test cases, proper deallocation is

recommended for robustness.

If the input has not been checked, the Treebuild function will not return any value if encountered building errors. Ensure all paths return an appropriate value.

The code used lots of custom functions. If needed we can build a .h file including custom functions, which reduces the length of main .c file.

Appendix: Source Code (in C)

```c
#include<stdio.h>
#include<stdlib.h>
typedef struct TreeNode *Tree; //define Tree structure
struct TreeNode {
    int element;
    Tree left;
    Tree right;
};
//global factors
int n1,n2;  //record the nodes of each tree
int k;      //the aimed sum result
int rootindex1, rootindex2;
//function declaration
int Treebuild(int n, int a[], int rootindex,Tree temp);  //build the binary
 search tree from input
int search(Tree tree, int key);   //judge if an element is in the tree
void preoder_print(Tree tree, int n);  //print the tree in preoder sequence
void inorder_store(Tree tree, int array[], int* ptr);  //store the element
into a list
int main(){
    //read the input and build tree1
    scanf("%d", &n1);
    int a[2*n1];  //creat a temporary list
    for(int i=0;i<n1;i++){
        scanf("%d %d", &a[2*i], &a[2*i+1]); //read all the input, and save
them into a temporary list
        if (a[2*i+1]==1) rootindex1=i;  //rootindex is used to record the p
lace off the root node
```

```
    }
    Tree tree1=(Tree)malloc(sizeof(struct TreeNode)); //initialize the root
 node
    //read the input and build tree2
    scanf("%d", &n2);
    int b[2*n2];  //creat a temporary list
    for(int i=0;i<n2;i++){
        scanf("%d %d", &b[2*i], &b[2*i+1]); //read all the input, and save
them into a temporary list
        if (b[2*i+1]==-1) rootindex2=i;
         //rootindex is used to record the place off the root node
    }
    Tree tree2=(Tree)malloc(sizeof(struct TreeNode)); //initialize the root
 node
    Treebuild(n1,a,rootindex1,tree1);  //build tree1
    Treebuild(n2,b,rootindex2,tree2);  //build tree2
    //searching and matching
    scanf("%d",&k);  //read the aimed k
    int array[n1];   //array is used to store the inorder seq.
    int output[2*n1];int out_index=0,i; //output and out_index is used to c
heck if the output is repetitive
    int index=0; //index refers to the last element in the array;
    inorder_store(tree1, array, &index);
    int flag=0;//flag is used to judge if A+B exist
    for(int j=0;j<n1;j++){
        if (search(tree2,k-array[j])){ //A and B exist
            if(!flag){
                printf("true\n");
                flag=1;
            }
            for(i=0;i<out_index;i++){
                if(output[i]==array[j]) i=out_index; //found the same equat
ion and stop circulation
            }
            if(i==out_index){ //the element has not been printed yet
                printf("%d = %d + %d\n", k, array[j], k-array[j]);
```

```
                output[out_index]=array[j];
                out_index++;
                output[out_index]=k-array[j];
                out_index++; //store the printed element into the array;
            }
        }
    }
    if(!flag) printf("flase\n");
    preoder_print(tree1, 0);
    printf("\n");
    preoder_print(tree2, 0);
    return 0;
}
int Treebuild(int n, int *a, int rootindex, Tree temp){ //temp is a new tre
e node(null)
    if(temp==NULL) return 0; //travelled to the next node of leaf node; sto
p resursion
    temp->element=a[2*rootindex];
    temp->left=NULL;
    temp->right=NULL;
    //initialize the new node, with value and left/right subtree
    for(int i=0;i<n;i++){
        if(a[2*i+1]==rootindex&&a[2*i]<temp->element){  //find the left sub
tree from the list(input)
            Tree Left=(Tree)malloc(sizeof(struct TreeNode));
               //create a new node
            temp->left=Left;
            Treebuild(n,a,i,Left); //link the parent node and recurse
        }
        if(a[2*i+1]==rootindex&&a[2*i]>=temp->element){
           //find the right subtree from the list(input)
            Tree Right=(Tree)malloc(sizeof(struct TreeNode));
            temp->right=Right;
            Treebuild(n,a,i,Right);
        }
    }
```

```
}
int search(Tree tree, int key){
    if(tree==NULL) return 0; //the element does not exist
    if(key<tree->element){
        search(tree->left, key);
//if the key is less, turn to the left subtree
    }else if(key>tree->element){
        search(tree->right, key);
//if the key is greater, turn to the right subtree
    }else{
        return 1;    //found the element
    }
}
void preoder_print(Tree tree, int n){
    if(tree!=NULL){
        if(!n){
            printf("%d", tree->element); //tht first element
        }else{
            printf(" %d", tree->element);
//the following element, including a blank
        }
        preoder_print(tree->left, n+1); //the left subtree
        preoder_print(tree->right, n+1); //the right subtree
    }
}
void inorder_store(Tree tree, int* array, int* ptr){
    if(tree==NULL) return;
    inorder_store(tree->left, array, ptr);
    //found the deepest node(left)
    array[*ptr]=tree->element; //store the value into the array
    *ptr=*ptr+1; //index move forward
    inorder_store(tree->right, array, ptr);
}
```

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