A+B with Binary Search Trees

2024-3-27

Chapter 1 Introduction

Binary search tree(BST) is a tree which has the following properties: 1.Binary search tree is a binary tree 2.The left subtree contains nodes whose element is smaller than that of the node 3.The right subtree contains nodes whose element is equal to or bigger than that of the node

Our task is to find an integer N by taking advantage of the properties of BST above.The main target is to see whether N can be written as A+B where A is an element in Tree1 and B is an element in Tree2. We input two trees,T1 and T2 and an integer N which to be found. If N can be written as A+B,where A is an element in T1 and B is an element in T2,then we print true and all the equations N = A + B,else we simply print false. After that,we print the preorder traversal of T1 and T2.

Chapter 2 Algorithm Specification

- BuildTree First find the node whose parent_index is -1,defining it as the head of the tree. Then find the nodes whose parent_index points to the head of the tree,defining them as the left child or the right child of the head. After that,defining them as the head of the left subtree or the right subtree and repeat the operations above using recursive method. The following is the pseudo code of the function BuildTree and Build_subtree:

```
function Build_subtree(parent_index, n, a):
    if parent_index == -1:
        return NULL

    Node = create_new_node(a[parent_index][0])
    Node.Left = NULL
    Node.Right = NULL

    left_child_index = -1
    right_child_index = -1

    for i in range(n):
        if a[i][1] == parent_index:
            if a[i][0] < Node.Element:
                left_child_index = i
            else:
                right_child_index = i

    Node.Left = Build_subtree(left_child_index, n, a)
    Node.Right = Build_subtree(right_child_index, n, a)

    return Node

function BuildTree(n, a):
    root_index = -1

    for i in range(n):
        if a[i][1] == -1:
            root_index = i
            break

    return Build_subtree(root_index, n, a)
```

- FindNumber First find the maximum number and the minimum number of T2,then use function Findbyinorder to find N. In function Findbyinorder,to avoid unnecessary recursions we can simply return if N-T1->Element > maxT2||N-T1->Element < minT2,because it is impossible to find an equation under such circumstances. if N-T1->Element<=maxT2&&N-T1->Element>=minT2 then first inorder traverse T1 and for each node of T1,use the function Compare to find equations. In function Compare,we judge whether X+T->Element is larger,equal to or smaller than N,and use recursive method to attempt to find the proper element in T2 such that X+T2->Element == N.During this process,we take advantage of the property of THE binary search tree that the worst time complexity is only logN when judging whether a

proper element is exist for a certain X such that X+T->Element == N. The following is the pseudo code of the function FindNumber,Findbyinorder and Compare:

```
function Compare(N, X, T):
    if T != null:
        if X + T.Element == N:
            mark ← mark + 1
            if mark == 1:
                print("true")
            print(N, " = ", X, " + ", T.Element)
        else if X + T.Element < N:
            Compare(N, X, T.Right)
        else:
            Compare(N, X, T.Left)

function Findbyinorder(T1, T2, N, maxT2, minT2):
    if T1 != null:
        if T1.Left != null and T1.Left.Element == T1.Element:
            T1 ← T1.Left
        Findbyinorder(T1.Left, T2, N, maxT2, minT2)
        if N - T1.Element <=> maxT2 and N - T1.Element >= minT2:
            Compare(N, T1.Element, T2)
        if T1.Right != null and T1.Right.Element == T1.Element:
            T1 ← T1.Right
        Findbyinorder(T1.Right, T2, N, maxT2, minT2)

function FindNumber(N, T1, T2):
    maxT2 ← Findmax(T2)
    minT2 ← Findmin(T2)
    Findbyinorder(T1, T2, N, maxT2, minT2)
    if mark == 0:
        print("false")
```

- Preorder print the elements of T1 and T2 in preorder

```
function Preorder(T,n)
    if T != NULL then
        if cnt != n then
            print(T.Element, " ")
            cnt ← cnt + 1
            Preorder(T.Left, n)
            Preorder(T.Right, n)
        else
            print(T.Element)
```

Data structure analysis:

- 2D array

Two 2D arrays each have n1 and n2 rows and both have two columns are used for building the Tree T1 and T2,where the first column is used for storing the elements of the nodes and the second column is used for storing the head_index which points to the parent of the node.

- Binary search tree(BST):the main data structure of the programme

The definition of a binary search tree: 1.Every node has an element which is an integer, and the elements are distinct. 2.The elements in a nonempty left subtree must be smaller than the element in the root of the subtree. 3.The elements in a nonempty right subtree must be larger than the element in the root of the subtree. 4.The left and right subtrees are also binary search trees. Note that the binary search tree we use in the programme is a bit different from the definition above,because in the programme,nodes having the same element is allowed(if two nodes share the same element,then one node is the right child of the other,if more nodes share the same element,then each one is the right child of another) The advantage of a binary search tree is that it can quickly perform search(the main task of the programme), insertion, and deletion operations, with a time complexity of O(logN), where N is the number of nodes in the tree.

Chapter 3 Testing Results

The programme can correctly make sure whether N can be found by first creating two binary search trees and then judging whether N can be written as A+B by inorder traversing T1 and then comparing N-T1->Element with T2->Element. Through the programme,we know that binary search tree is an effective data structure for lower the worst time complexity for a certain task. By taking advantage of the property of the binary search tree,when finding the number N,we only need to compare N-T1->Element with T2->Element,if N-T1->Element < T2->Element,then compare N-T1->Element with T2->Left>Element,if N-T1->Element > T2->Element,then compare N-T1->Element with T2->Right->Element,if T2->Element+T1->Element==N,print the equation N = A + B,note that the same equation only appears once. In the worst cases,for certain T1->Element,we should do the comparisons as many times as the level of T2 namely logN times where N is the number of nodes of T2. Meanwhile,if N-T1->Element > maxT2 or N-T1->Element < minT2,then there's no need for other comparison for it is impossible to find N under such T1->Element.

Chapter 4: Analysis and Comments

- Time complexity:

The worst time complexity of the functions is as follows: BuildTree:O(N^2) The function first iterates through the array a (of size N) to find the root index. This takes O(N) time. Then it calls Build_subtree.The function Build_subtree recursively builds the left and right subtrees of the tree. For each node, it iterates through the entire array a (of size N) to find the left and right child indices. Therefore, the time complexity is O(N^2). The overall time complexity is O(N^2).

FindNumber:O(NlogN) Function Findmax and Findmin share the time complexity of both O(logN) Function Findbyinorder uses the inorder traversal method to traverse T1;therefore,the time complexity of this function is O(N*the time complexity of function Compare) Function Compare compares N-T1->Element with the elements in T2 by taking advantage of the properties of binary search tree,whose worst time complexity is O(logN) Therefore,the total time complexity of the function FindNumber is O(NlogN)

Preorder:O(N) The function performs a preorder traversal of the tree. For each node, it prints the element and recursively visits its left and right children. Since each node is visited once, the time complexity is O(N).

Total time complexity of the programme is O(N^2)

- Space complexity:

The worst space complexity of the functions is as follows: BuildTree:O(N) The recursive calls create a new stack frame for each node. Since the maximum depth of the recursion is the height of the tree (which can be at most N for a skewed tree), the space complexity is O(N).

FindNumber:O(NlogN) Function Findmax and Findmin share the space complexity of both O(logN) Function Findbyinorder uses the inorder traversal method to traverse T1;therefore,the space complexity of this function is O(N*the space complexity of function Compare) unction Compare compares N-T1->Element with the elements in T2 by taking advantage of the properties of binary search tree,whose worst space complexity is O(logN) Therefore,the total time complexity of the function FindNumber is O(NlogN)

Preorder:O(N) Each call of the function prints one element of the tree,since the tree has n nodes,the total time complexity of the function if O(N).

Total space complexity of the programme is O(NlogN)

Through the project,we could more detailedly understand the advantages of the binary search tree. Hope the project can give us a rough but more intuitive understanding of the binary search tree.

Declaration

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