

【华为OD机考 统一考试机试C卷】计算三叉搜索树的高度 (C++ Java JavaScript Python C语言)

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

真题目录：华为OD机考机试 真题目录 (C卷 + D卷 + B卷 + A卷) + 考点说明

专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)

华为OD面试真题精选：华为OD面试真题精选

在线OJ：点击立即刷题，模拟真实机考环境

题目描述

定义构造三叉搜索树规则如下：

每个节点都存有一个数，当插入一个新的数时，从根节点向下寻找，直到找到一个合适的空节点插入。查找的规则是：

- 如果数小于节点的数减去500，则将数插入节点的左子树
- 如果数大于节点的数加上500，则将数插入节点的右子树
- 否则，将数插入节点的中子树

给你一系列数，请按以上规则，按顺序将数插入树中，构建出一棵三叉搜索树，最后输出树的高度。

输入描述

第一行为一个数 N ，表示有 N 个数， $1 \leq N \leq 10000$

第二行为 N 个空格分隔的整数，每个数的范围为 $[1,10000]$

输出描述

输出树的高度（根节点的高度为1）

用例1

输入

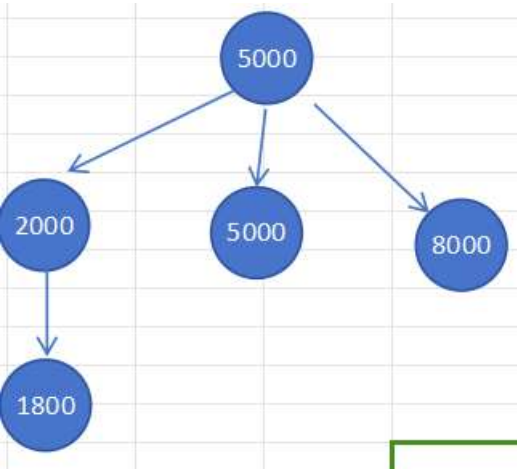
1	5
2	5000 2000 5000 8000 1800

输出

1	3
---	---

说明

最终构造出的树如下，高度为3：



用例2

输入

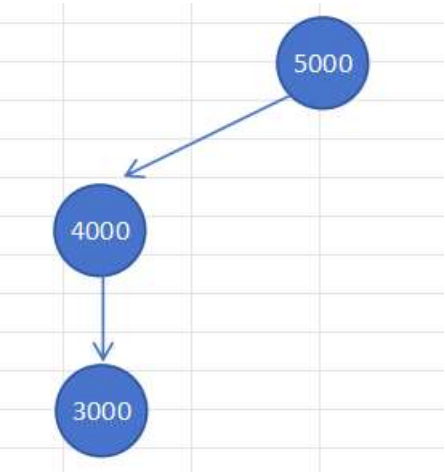
1	3
2	5000 4000 3000

输出

1 | 3

说明

最终构造出的树如下，高度为3：



用例3

输入

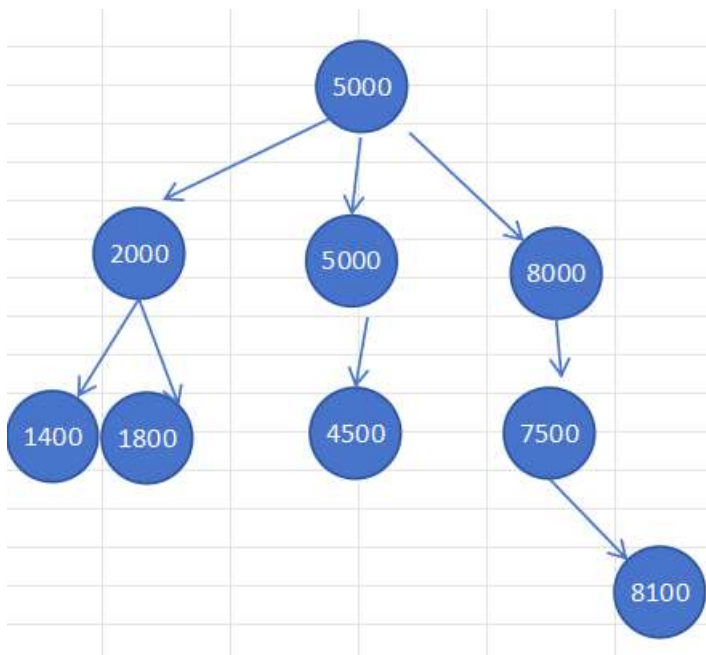
1 | 9
2 | 5000 2000 5000 8000 1800 7500 4500 1400 8100

输出

1 | 4

说明

最终构造出的树如下，高度为4：



解题思路

定义树类 `Tree`，包含两个方法：`insert` 和 `getHeight`。

1. `insert` 方法用于向树中插入新值。它接受当前树的根节点和要插入的值作为参数。
 - 如果当前节点为空，创建一个新的 `TreeNode` 实例，并返回它作为新的根节点。
 - 如果要插入的值小于当前节点值减去500，递归地在左子树中插入该值。
 - 如果要插入的值大于当前节点值加上500，递归地在右子树中插入该值。
 - 如果要插入的值在当前节点值加减500的范围内，递归地在中间子树中插入该值。
 - 每次递归插入后，返回当前节点作为该子树的新根节点。
2. `getHeight` 方法用于计算树的高度。它接受树的根节点作为参数。
 - 如果当前节点为空，返回0，表示高度为0。
 - 否则，递归地计算左子树、中间子树和右子树的高度。
 - 取三者中的最大值，然后加1（当前节点的高度），作为整棵树的高度。

C++

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  // 树节点结构体
5  struct TreeNode {
6      int val; // 节点值
7      TreeNode *left, *mid, *right; // 左、中、右子节点指针
8      TreeNode(int x) : val(x), left(nullptr), mid(nullptr), right(nullptr) {} // 构造函数
9  };
10
11 // 树类
12 class Tree {
13 public:
14     // 插入方法: 向树中插入值
15     TreeNode* insert(TreeNode* root, int val) {
16         if (root == nullptr) {
17             return new TreeNode(val); // 如果根节点为空, 创建新节点作为根节点
18         }
19         if (val < root->val - 500) {
20             root->left = insert(root->left, val); // 如果值小于根节点值减500, 插入到左子树
21         } else if (val > root->val + 500) {
22             root->right = insert(root->right, val); // 如果值大于根节点值加500, 插入到右子树
23         } else {
24             root->mid = insert(root->mid, val); // 如果值在根节点值加减500范围内, 插入到中间子树
25         }
26         return root; // 返回根节点
27     }
28
29     // 获取树的高度
30     int getHeight(TreeNode* root) {
31         if (root == nullptr) {
32             return 0; // 如果根节点为空, 高度为0
33         }
34         int leftHeight = getHeight(root->left); // 计算左子树的高度
35         int midHeight = getHeight(root->mid); // 计算中间子树的高度
36         int rightHeight = getHeight(root->right); // 计算右子树的高度
37         return max({leftHeight, midHeight, rightHeight}) + 1; // 返回三者中最大的高度加1
38     }
39 }
```

```

40 };
41
42 int main() {
43     Tree tree; // 创建树对象
44     int N;
45     cin >> N; // 读取节点数量
46     TreeNode* root = nullptr; // 初始化根节点为null
47     for (int i = 0; i < N; ++i) {
48         int num;
49         cin >> num; // 循环读取节点值
50         root = tree.insert(root, num); // 将每个整数插入树中
51     }
52     int height = tree.getHeight(root); // 获取树的高度
53     cout << height << endl; // 输出树的高度
54     return 0;
55 }

```

Java

```

1  import java.util.Scanner;
2
3  // 主类
4  public class Main {
5
6      // 静态内部类: 树
7      static class Tree {
8          // 插入方法: 向树中插入值
9          public TreeNode insert(TreeNode root, int val) {
10             if (root == null) {
11                 return new TreeNode(val); // 如果根节点为空, 创建新节点作为根节点
12             }
13             if (val < root.val - 500) {
14                 root.left = insert(root.left, val); // 如果值小于根节点值减500, 插入到左子树
15             } else if (val > root.val + 500) {
16                 root.right = insert(root.right, val); // 如果值大于根节点值加500, 插入到右子树
17             } else {
18                 root.mid = insert(root.mid, val); // 如果值在根节点值加减500范围内, 插入到中间子树
19             }
20             return root; // 返回根节点
21         }
22     }
23 }

```

```

22
23 // 获取树的高度
24 public int getHeight(TreeNode root) {
25     if (root == null) {
26         return 0; // 如果根节点为空, 高度为0
27     }
28     int leftHeight = getHeight(root.left); // 计算左子树的高度
29     int midHeight = getHeight(root.mid); // 计算中间子树的高度
30     int rightHeight = getHeight(root.right); // 计算右子树的高度
31     return Math.max(Math.max(leftHeight, midHeight), rightHeight) + 1; // 返回三者中最大的高度加1
32 }
33 }
34
35 // 静态内部类: 树节点
36 static class TreeNode {
37     int val; // 节点值
38     TreeNode left, mid, right; // 左子节点、中间子节点、右子节点
39     TreeNode(int x) { val = x; } // 构造方法, 初始化节点值
40 }
41
42
43 public static void main(String[] args) {
44     Tree tree = new Tree(); // 创建树对象
45     Scanner scanner = new Scanner(System.in); // 创建扫描器读取输入
46     int N = scanner.nextInt(); // 读取第一个整数作为后续要输入的节点数量
47     TreeNode root = null; // 初始化根节点为null
48     for (int i = 0; i < N; i++) {
49         int num = scanner.nextInt(); // 循环读取N个整数作为节点值
50         root = tree.insert(root, num); // 将每个整数插入树中
51     }
52     scanner.close(); // 关闭扫描器
53     int height = tree.getHeight(root); // 获取树的高度
54     System.out.println(height); // 输出树的高度
55 }
56 }

```

javaScript

```

1 class TreeNode {
2     // 构造函数: 创建树节点
3

```

```

3     constructor(val) {
4         this.val = val; // 节点值
5         this.left = this.mid = this.right = null; // 初始化左、中、右子节点为null
6     }
7 }
8
9 class Tree {
10    // 插入方法: 向树中插入值
11    insert(root, val) {
12        if (root === null) {
13            return new TreeNode(val); // 如果根节点为空, 创建新节点作为根节点
14        }
15        if (val < root.val - 500) {
16            root.left = this.insert(root.left, val); // 如果值小于根节点值减500, 插入到左子树
17        } else if (val > root.val + 500) {
18            root.right = this.insert(root.right, val); // 如果值大于根节点值加500, 插入到右子树
19        } else {
20            root.mid = this.insert(root.mid, val); // 如果值在根节点值加减500范围内, 插入到中间子树
21        }
22        return root; // 返回根节点
23    }
24
25    // 获取树的高度
26    getHeight(root) {
27        if (root === null) {
28            return 0; // 如果根节点为空, 高度为0
29        }
30        let leftHeight = this.getHeight(root.left); // 计算左子树的高度
31        let midHeight = this.getHeight(root.mid); // 计算中间子树的高度
32        let rightHeight = this.getHeight(root.right); // 计算右子树的高度
33        return Math.max(leftHeight, midHeight, rightHeight) + 1; // 返回三者中最大的高度加1
34    }
35 }
36
37 // 主程序
38 const readline = require('readline').createInterface({
39     input: process.stdin,
40     output: process.stdout
41 });
42
43

```



```

44 const tree = new Tree();
45 let root = null;
46
47 readline.on('line', N => {
48     N = parseInt(N);
49     readline.on('line', nums => {
50         nums.split(' ').forEach(num => {
51             root = tree.insert(root, parseInt(num)); // 将每个整数插入树中
52         });
53         const height = tree.getHeight(root); // 获取树的高度
54         console.log(height); // 输出树的高度
55         readline.close();
56     });
});

```

Python

```

1 class TreeNode:
2     def __init__(self, val):
3         self.val = val # 节点值
4         self.left = self.mid = self.right = None # 左、中、右子节点
5
6 class Tree:
7     # 插入方法: 向树中插入值
8     def insert(self, root, val):
9         if root is None:
10             return TreeNode(val) # 如果根节点为空, 创建新节点作为根节点
11         if val < root.val - 500:
12             root.left = self.insert(root.left, val) # 如果值小于根节点值减500, 插入到左子树
13         elif val > root.val + 500:
14             root.right = self.insert(root.right, val) # 如果值大于根节点值加500, 插入到右子树
15         else:
16             root.mid = self.insert(root.mid, val) # 如果值在根节点值加减500范围内, 插入到中间子树
17         return root # 返回根节点
18
19 # 获取树的高度
20 def get_height(self, root):
21     if root is None:
22         return 0 # 如果根节点为空, 高度为0
23     left_height = self.get_height(root.left) # 计算左子树的高度
24

```

```

24         mid_height = self.get_height(root.mid) # 计算中间子树的高度
25         right_height = self.get_height(root.right) # 计算右子树的高度
26         return max(left_height, mid_height, right_height) + 1 # 返回三者中最大的高度加1
27
28 if __name__ == '__main__':
29     tree = Tree() # 创建树对象
30     N = int(input()) # 读取节点数量
31     root = None # 初始化根节点为None
32     nums = list(map(int, input().split()))
33     for num in nums:
34         root = tree.insert(root, num) # 将每个整数插入树中
35     height = tree.get_height(root) # 获取树的高度
36     print(height) # 输出树的高度

```

C语言

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // 树节点结构体
5  typedef struct TreeNode {
6      int val; // 节点值
7      struct TreeNode *left, *mid, *right; // 左、中、右子节点指针
8  } TreeNode;
9
10 // 创建新节点
11 TreeNode* createNode(int val) {
12     TreeNode* node = (TreeNode*)malloc(sizeof(TreeNode)); // 分配内存
13     node->val = val; // 设置节点值
14     node->left = node->mid = node->right = NULL; // 初始化子节点为NULL
15     return node; // 返回新创建的节点
16 }
17
18 // 向树中插入值
19 TreeNode* insert(TreeNode* root, int val) {
20     if (root == NULL) {
21         return createNode(val); // 如果根节点为空, 创建新节点作为根节点
22     }
23     if (val < root->val - 500) {
24         root->left = insert(root->left, val); // 如果值小于根节点值减500, 插入到左子树
25     }

```

```

25     } else if (val > root->val + 500) {
26         root->right = insert(root->right, val); // 如果值大于根节点值加500, 插入到右子树
27     } else {
28         root->mid = insert(root->mid, val); // 如果值在根节点值加减500范围内, 插入到中间子树
29     }
30     return root; // 返回根节点
31 }
32
33 // 获取树的高度
34 int getHeight(TreeNode* root) {
35     if (root == NULL) {
36         return 0; // 如果根节点为空, 高度为0
37     }
38     int leftHeight = getHeight(root->left); // 计算左子树的高度
39     int midHeight = getHeight(root->mid); // 计算中间子树的高度
40     int rightHeight = getHeight(root->right); // 计算右子树的高度
41     int maxHeight = leftHeight > midHeight ? leftHeight : midHeight; // 计算左子树和中间子树的最大高度
42     maxHeight = maxHeight > rightHeight ? maxHeight : rightHeight; // 计算最大高度
43     return maxHeight + 1; // 返回最大高度加1
44 }
45
46 int main() {
47     int N, num;
48     scanf("%d", &N); // 读取节点数量
49     TreeNode* root = NULL; // 初始化根节点为NULL
50     for (int i = 0; i < N; ++i) {
51         scanf("%d", &num); // 循环读取节点值
52         root = insert(root, num); // 将每个整数插入树中
53     }
54     int height = getHeight(root); // 获取树的高度
55     printf("%d\n", height); // 输出树的高度
56     return 0;
57 }

```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

[题目描述](#)

[输入描述](#)

输出描述

用例1

用例2

用例3

解题思路

C++

Java

javaScript

Python

C语言

机考真题 华为OD



CSDN @算法大师