【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 ≤ N ≤ 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

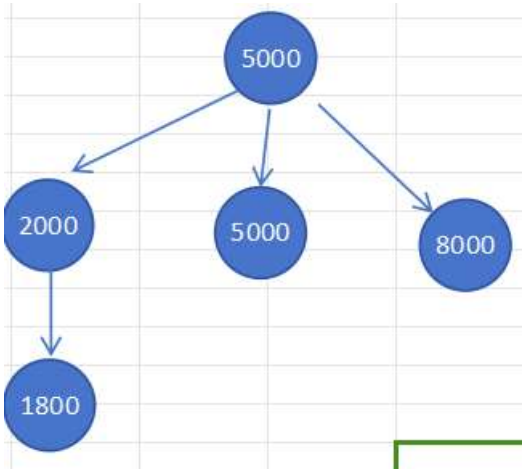
```
5
5000 2000 5000 8000 1800
```

输出

```
3
```

说明

最终构造出的树如下，高度为3：



用例2

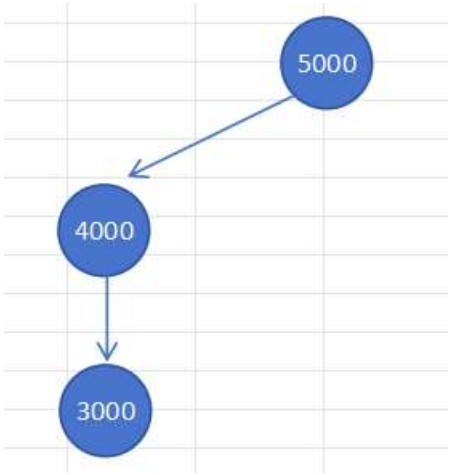
输入

```
3
5000 4000 3000
```

输出

```
3
```

说明
最终构造出的树如下，高度为3：



用例3

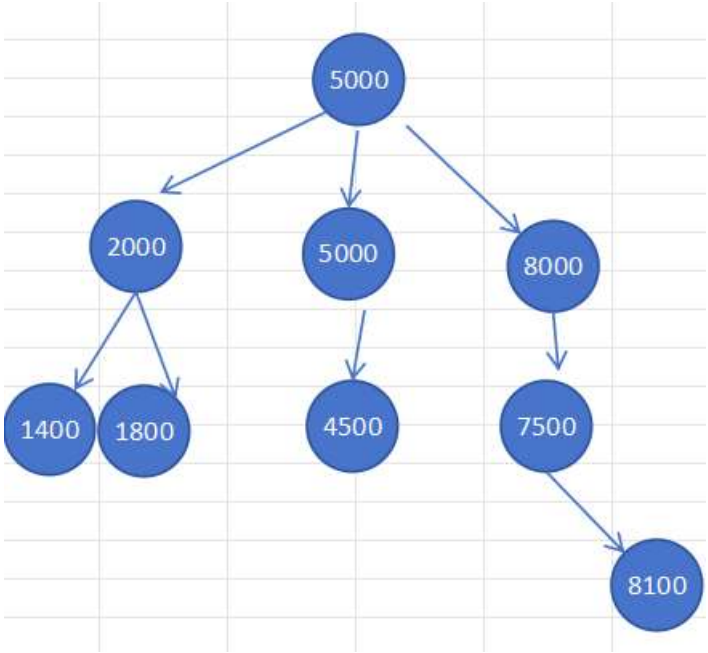
输入

```
9
5000 2000 5000 8000 1800 7500 4500 1400 8100
```

输出

```
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++

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
// 树节点结构体
struct TreeNode {
    int val; // 节点值
    TreeNode *left, *mid, *right; // 左、中、右子节点指针
    TreeNode(int x) : val(x), left(nullptr), mid(nullptr), right(nullptr) {} // 构造函数
};

// 树类
class Tree {
public:
    // 插入方法：向树中插入值
    TreeNode* insert(TreeNode* root, int val) {
        if (root == nullptr) {
            return new TreeNode(val); // 如果根节点为空，创建新节点作为根节点
        }
        if (val < root->val - 500) {
            root->left = insert(root->left, val); // 如果值小于根节点值减500，插入到左子树
        } else if (val > root->val + 500) {
            root->right = insert(root->right, val); // 如果值大于根节点值加500，插入到右子树
        } else {
            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
        }
        return root; // 返回根节点
    }

    // 获取树的高度
    int getHeight(TreeNode* root) {
        if (root == nullptr) {
            return 0; // 如果根节点为空，高度为0
        }
        int leftHeight = getHeight(root->left); // 计算左子树的高度
        int midHeight = getHeight(root->mid); // 计算中间子树的高度
        int rightHeight = getHeight(root->right); // 计算右子树的高度
        return max({leftHeight, midHeight, rightHeight}) + 1; // 返回三者中最大的高度加1
    }

```

```
};

int main() {
    Tree tree; // 创建树对象
    int N;
    cin >> N; // 读取节点数量
    TreeNode* root = nullptr; // 初始化根节点为null
    for (int i = 0; i < N; ++i) {
        int num;
        cin >> num; // 循环读取节点值
        root = tree.insert(root, num); // 将每个整数插入树中
    }
    int height = tree.getHeight(root); // 获取树的高度
    cout << height << endl; // 输出树的高度
    return 0;
}
```

Java

```
import java.util.Scanner;

// 主类
public class Main {

    // 静态内部类：树
    static class Tree {
        // 插入方法：向树中插入值
        public TreeNode insert(TreeNode root, int val) {
            if (root == null) {
                return new TreeNode(val); // 如果根节点为空，创建新节点作为根节点
            }
            if (val < root.val - 500) {
                root.left = insert(root.left, val); // 如果值小于根节点值减500，插入到左子树
            } else if (val > root.val + 500) {
                root.right = insert(root.right, val); // 如果值大于根节点值加500，插入到右子树
            } else {
                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
            }
            return root; // 返回根节点
        }
```

```java

        // 获取树的高度
        public int getHeight(TreeNode root) {
            if (root == null) {
                return 0; // 如果根节点为空，高度为0
            }
            int leftHeight = getHeight(root.left); // 计算左子树的高度
            int midHeight = getHeight(root.mid); // 计算中间子树的高度
            int rightHeight = getHeight(root.right); // 计算右子树的高度
            return Math.max(Math.max(leftHeight, midHeight), rightHeight) + 1; // 返回三者中最大的高度加1
        }
    }

    // 静态内部类：树节点
    static class TreeNode {
        int val; // 节点值
        TreeNode left, mid, right; // 左子节点、中间子节点、右子节点
        TreeNode(int x) { val = x; } // 构造方法，初始化节点值
    }


    public static void main(String[] args) {
        Tree tree = new Tree(); // 创建树对象
        Scanner scanner = new Scanner(System.in); // 创建扫描器读取输入
        int N = scanner.nextInt();  // 读取第一个整数作为后续要输入的节点数量
        TreeNode root = null; // 初始化根节点为null
        for (int i = 0; i < N; i++) {
            int num = scanner.nextInt();  // 循环读取N个整数作为节点值
            root = tree.insert(root, num);  // 将每个整数插入树中
        }
        scanner.close(); // 关闭扫描器
        int height = tree.getHeight(root);  // 获取树的高度
        System.out.println(height);  // 输出树的高度
    }
}
```

javaScript

```javascript
class TreeNode {
    // 构造函数：创建树节点
```

```
    constructor(val) {
        this.val = val; // 节点值
        this.left = this.mid = this.right = null; // 初始化左、中、右子节点为null
    }
}

class Tree {
    // 插入方法：向树中插入值
    insert(root, val) {
        if (root === null) {
            return new TreeNode(val); // 如果根节点为空，创建新节点作为根节点
        }
        if (val < root.val - 500) {
            root.left = this.insert(root.left, val); // 如果值小于根节点值减500，插入到左子树
        } else if (val > root.val + 500) {
            root.right = this.insert(root.right, val); // 如果值大于根节点值加500，插入到右子树
        } else {
            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
        }
        return root; // 返回根节点
    }

    // 获取树的高度
    getHeight(root) {
        if (root === null) {
            return 0; // 如果根节点为空，高度为0
        }
        let leftHeight = this.getHeight(root.left); // 计算左子树的高度
        let midHeight = this.getHeight(root.mid); // 计算中间子树的高度
        let rightHeight = this.getHeight(root.right); // 计算右子树的高度
        return Math.max(leftHeight, midHeight, rightHeight) + 1; // 返回三者中最大的高度加1
    }
}

// 主程序
const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});


```

```
const tree = new Tree();
let root = null;

readline.on('line', N => {
    N = parseInt(N);
    readline.on('line', nums => {
        nums.split(' ').forEach(num => {
            root = tree.insert(root, parseInt(num)); // 将每个整数插入树中
        });
        const height = tree.getHeight(root); // 获取树的高度
        console.log(height); // 输出树的高度
        readline.close();
    });
});
```

Python

```
class TreeNode:
    def __init__(self, val):
        self.val = val  # 节点值
        self.left = self.mid = self.right = None  # 左、中、右子节点

class Tree:
    # 插入方法：向树中插入值
    def insert(self, root, val):
        if root is None:
            return TreeNode(val)  # 如果根节点为空，创建新节点作为根节点
        if val < root.val - 500:
            root.left = self.insert(root.left, val)  # 如果值小于根节点值减500，插入到左子树
        elif val > root.val + 500:
            root.right = self.insert(root.right, val)  # 如果值大于根节点值加500，插入到右子树
        else:
            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
        return root  # 返回根节点

    # 获取树的高度
    def get_height(self, root):
        if root is None:
            return 0  # 如果根节点为空，高度为0
        left_height = self.get_height(root.left)  # 计算左子树的高度
```

```
        mid_height = self.get_height(root.mid)  # 计算中间子树的高度
        right_height = self.get_height(root.right)  # 计算右子树的高度
        return max(left_height, mid_height, right_height) + 1  # 返回三者中最大的高度加1

if __name__ == '__main__':
    tree = Tree()  # 创建树对象
    N = int(input())  # 读取节点数量
    root = None  # 初始化根节点为None
    nums = list(map(int, input().split()))
    for num in nums:
        root = tree.insert(root, num)  # 将每个整数插入树中
    height = tree.get_height(root)  # 获取树的高度
    print(height)  # 输出树的高度
```

C语言

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

// 树节点结构体
typedef struct TreeNode {
    int val; // 节点值
    struct TreeNode *left, *mid, *right; // 左、中、右子节点指针
} TreeNode;

// 创建新节点
TreeNode* createNode(int val) {
    TreeNode* node = (TreeNode*)malloc(sizeof(TreeNode)); // 分配内存
    node->val = val; // 设置节点值
    node->left = node->mid = node->right = NULL; // 初始化子节点为NULL
    return node; // 返回新创建的节点
}

// 向树中插入值
TreeNode* insert(TreeNode* root, int val) {
    if (root == NULL) {
        return createNode(val); // 如果根节点为空，创建新节点作为根节点
    }
    if (val < root->val - 500) {
        root->left = insert(root->left, val); // 如果值小于根节点值减500，插入到左子树
```

```
    } else if (val > root->val + 500) {
        root->right = insert(root->right, val); // 如果值大于根节点值加500，插入到右子树
    } else {
        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
    }
    return root; // 返回根节点
}

// 获取树的高度
int getHeight(TreeNode* root) {
    if (root == NULL) {
        return 0; // 如果根节点为空，高度为0
    }
    int leftHeight = getHeight(root->left); // 计算左子树的高度
    int midHeight = getHeight(root->mid); // 计算中间子树的高度
    int rightHeight = getHeight(root->right); // 计算右子树的高度
    int maxHeight = leftHeight > midHeight ? leftHeight : midHeight; // 计算左子树和中间子树的最大高度
    maxHeight = maxHeight > rightHeight ? maxHeight : rightHeight; // 计算最大高度
    return maxHeight + 1; // 返回最大高度加1
}

int main() {
    int N, num;
    scanf("%d", &N); // 读取节点数量
    TreeNode* root = NULL; // 初始化根节点为NULL
    for (int i = 0; i < N; ++i) {
        scanf("%d", &num); // 循环读取节点值
        root = insert(root, num); // 将每个整数插入树中
    }
    int height = getHeight(root); // 获取树的高度
    printf("%d\n", height); // 输出树的高度
    return 0;
}
```

文章目录