【华为OD机考 统一考试机试C卷】悄悄话（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

给定一个二叉树，每个节点上站一个人，节点数字表示父节点到该节点传递悄悄话需要花费的时间。

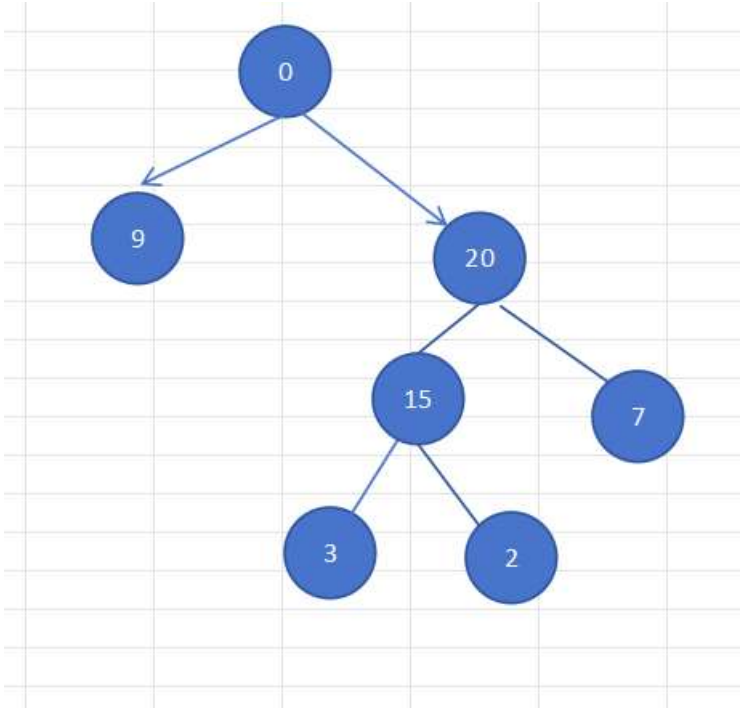
初始时，根节点所在位置的人有一个悄悄话想要传递给其他人，求二叉树所有节点上的人都接收到悄悄话花费的时间。

输入描述

给定二叉树

> 0 9 20 -1 -1 15 7 -1 -1 -1 -1 3 2

注：-1表示空节点



输出描述

返回所有节点都接收到悄悄话花费的时间

> 38

用例

输入	0 9 20 -1 -1 15 7 -1 -1 -1 -1 3 2
输出	38
说明	无

解题思路

2. **读取输入**:

- 读取一行输入，这行输入包含了一系列的整数，每个整数代表从父节点到子节点的悄悄话传递时间。

3. **处理根节点**：
   - 将根节点（索引为0）加入队列，并设置其悄悄话接收时间为0。
4. **层次遍历**：
   - 当队列不为空时，循环执行以下步骤：
     - 从队列中取出一个节点（包括节点索引和该节点的悄悄话接收时间）。
     - 计算左右子节点的索引。
     - 检查左右子节点是否存在（索引有效且不为-1）。
5. **更新子节点时间**：
   - 如果子节点存在，将当前节点的悄悄话接收时间加上从当前节点到子节点的悄悄话传递时间，得到子节点的悄悄话接收时间。
   - 将子节点及其悄悄话接收时间加入队列。
6. **更新最大时间**：
   - 每次子节点的悄悄话接收时间被计算后，更新最大时间为当前子节点时间和已记录的最大时间中的较大值。

模拟计算

给定的输入数组 `0 9 20 -1 -1 15 7 -1 -1 -1 -1 3 2` 代表一棵二叉树，其中每个值代表从父节点到子节点的悄悄话传递时间。数组中的 `-1` 表示没有子节点。数组索引代表节点的顺序，按照完全二叉树的顺序排列。

模拟计算过程如下：

1. **初始化队列**：
   - 将根节点索引 `0` 加入队列，此时队列为 `[0]`。
2. **开始层次遍历**：
   - 队列非空，继续遍历。
3. **处理根节点**：
   - 取出队列头部元素（根节点索引 `0`），队列变为 `[]`。

- 计算左子节点索引 `1` （`2*0+1`），右子节点索引 `2` （`2*0+2`）。
- 左子节点值为 `9`，更新为 `0+9=9`，加入队列，队列变为 `[1]`。
- 右子节点值为 `20`，更新为 `0+20=20`，加入队列，队列变为 `[1, 2]`。
- 更新 `maxTime` 为 `20`。

4. **处理索引为1的节点**：

- 取出队列头部元素 `1`，队列变为 `[2]`。
- 计算左子节点索引 `3` （`2*1+1`），右子节点索引 `4` （`2*1+2`）。
- 左右子节点值均为 `-1`，没有子节点，不做操作。

5. **处理索引为2的节点**：

- 取出队列头部元素 `2`，队列变为 `[]`。
- 计算左子节点索引 `5` （`2*2+1`），右子节点索引 `6` （`2*2+2`）。
- 左子节点值为 `15`，更新为 `20+15=35`，加入队列，队列变为 `[5]`。
- 右子节点值为 `7`，更新为 `20+7=27`，加入队列，队列变为 `[5, 6]`。
- 更新 `maxTime` 为 `35`。

6. **处理索引为5的节点**：

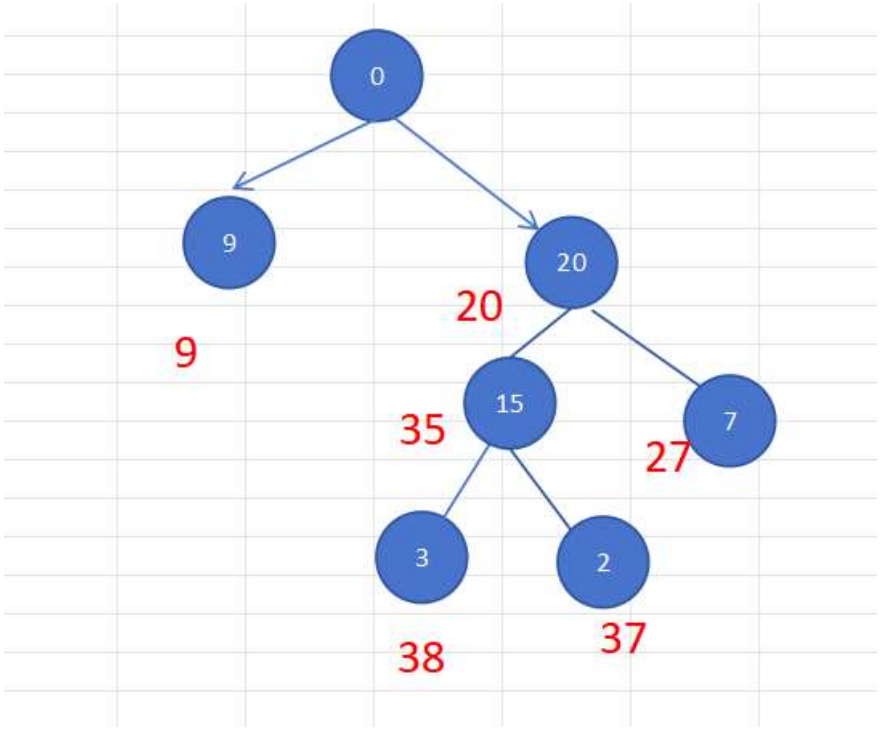
- 取出队列头部元素 `5`，队列变为 `[6]`。
- 计算左子节点索引 `11` （`2*5+1`），右子节点索引 `12` （`2*5+2`）。
- 左子节点值为 `3`，更新为 `35+3=38`，加入队列，队列变为 `[6, 11]`。
- 右子节点值为 `2`，更新为 `35+2=37`，加入队列，队列变为 `[6, 11, 12]`。
- 更新 `maxTime` 为 `38`。

7. **处理索引为6的节点**：

- 取出队列头部元素 `6`，队列变为 `[11, 12]`。
- 计算左子节点索引 `13` （`2*6+1`），右子节点索引 `14` （`2*6+2`）。
- 由于索引超出数组长度，没有子节点，不做操作。

8. **处理索引为11和12的节点**：
    - 取出队列头部元素 `11` 和 `12`，队列变为 `[]`。
    - 由于索引超出数组长度，没有子节点，不做操作。
9. **结束遍历**：
    - 队列为空，遍历结束。
10. **输出结果**：
    - 最大时间 `maxTime` 为 `38`，这是最后一个节点接收悄悄话的时间。

因此，所有节点接收悄悄话的总时间为 `38`。



C++

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <queue>

using namespace std;
int main() {
    // 读取一行输入并将其转换为整数数组
    // 数组中的每个元素代表从父节点到当前节点的时间
    string line;
    getline(cin, line);
    istringstream iss(line);
    vector<int> whisperTimes;
    int time;
    while (iss >> time) {
        whisperTimes.push_back(time);
    }

    // 记录最后一个节点接收悄悄话的时间
    int maxTime = 0;

    // 使用队列来进行二叉树的层次遍历
    queue<int> nodeQueue;
    // 将根节点索引0加入队列
    nodeQueue.push(0);

    // 当队列不为空时，继续遍历
    while (!nodeQueue.empty()) {
        // 从队列中取出一个节点索引
        int parentNodeIndex = nodeQueue.front();
        nodeQueue.pop();

        // 计算左子节点索引
        int leftChildIndex = 2 * parentNodeIndex + 1;
        // 计算右子节点索引
        int rightChildIndex = 2 * parentNodeIndex + 2;

        // 如果左子节点存在，处理左子节点
        if (leftChildIndex < whisperTimes.size() && whisperTimes[leftChildIndex] != -1) {
```

```
            // 更新左子节点的时间（父节点时间 + 当前节点时间）
            whisperTimes[leftChildIndex] += whisperTimes[parentNodeIndex];
            // 将左子节点加入队列
            nodeQueue.push(leftChildIndex);
            // 更新最大时间
            maxTime = max(maxTime, whisperTimes[leftChildIndex]);
        }

        // 如果右子节点存在，处理右子节点
        if (rightChildIndex < whisperTimes.size() && whisperTimes[rightChildIndex] != -1) {
            // 更新右子节点的时间（父节点时间 + 当前节点时间）
            whisperTimes[rightChildIndex] += whisperTimes[parentNodeIndex];
            // 将右子节点加入队列
            nodeQueue.push(rightChildIndex);
            // 更新最大时间
            maxTime = max(maxTime, whisperTimes[rightChildIndex]);
        }
    }

    // 所有节点都接收到悄悄话后，打印最大时间
    cout << maxTime << endl;
    return 0;
}
```

Java

```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 创建扫描器读取输入
        Scanner scanner = new Scanner(System.in);
        // 读取一行输入并将其转换为整数数组，数组中的每个元素代表从父节点到当前节点的时间
        int[] whisperTimes = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        // 关闭扫描器
        scanner.close();

```

```java
        // 记录最后一个节点接收悄悄话的时间
        int maxTime = 0;

        // 使用队列来进行二叉树的层次遍历
        Queue<Integer> nodeQueue = new LinkedList<>();
        // 将根节点索引0加入队列
        nodeQueue.add(0);

        // 当队列不为空时，继续遍历
        while (!nodeQueue.isEmpty()) {
            // 从队列中取出一个节点索引
            int parentNodeIndex = nodeQueue.poll();

            // 计算左子节点索引
            int leftChildIndex = 2 * parentNodeIndex + 1;
            // 计算右子节点索引
            int rightChildIndex = 2 * parentNodeIndex + 2;

            // 如果左子节点存在，处理左子节点
            if (leftChildIndex < whisperTimes.length && whisperTimes[leftChildIndex] != -1) {
                // 更新左子节点的时间（父节点时间 + 当前节点时间）
                whisperTimes[leftChildIndex] += whisperTimes[parentNodeIndex];
                // 将左子节点加入队列
                nodeQueue.add(leftChildIndex);
                // 更新最大时间
                maxTime = Math.max(maxTime, whisperTimes[leftChildIndex]);
            }

            // 如果右子节点存在，处理右子节点
            if (rightChildIndex < whisperTimes.length && whisperTimes[rightChildIndex] != -1) {
                // 更新右子节点的时间（父节点时间 + 当前节点时间）
                whisperTimes[rightChildIndex] += whisperTimes[parentNodeIndex];
                // 将右子节点加入队列
                nodeQueue.add(rightChildIndex);
                // 更新最大时间
                maxTime = Math.max(maxTime, whisperTimes[rightChildIndex]);
            }
        }

        // 所有节点都接收到悄悄话后，打印最大时间
```

```
        System.out.println(maxTime);
    }
}
```

javaScript

```javascript
const readline = require('readline');

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 提示用户输入数据
rl.on('line', (input) => {
  // 将输入的字符串按空格分隔，转换为整数数组
  const whisperTimes = input.split(' ').map(Number);

  // 记录最后一个节点接收悄悄话的时间
  let maxTime = 0;

  // 使用队列来进行二叉树的层次遍历
  const nodeQueue = []; // 初始化队列
  nodeQueue.push(0); // 将根节点索引0加入队列

  // 当队列不为空时，继续遍历
  while (nodeQueue.length > 0) {
    // 从队列中取出一个节点索引
    const parentNodeIndex = nodeQueue.shift();

    // 计算左子节点索引
    const leftChildIndex = 2 * parentNodeIndex + 1;
    // 计算右子节点索引
    const rightChildIndex = 2 * parentNodeIndex + 2;

    // 如果左子节点存在，处理左子节点
    if (leftChildIndex < whisperTimes.length && whisperTimes[leftChildIndex] !== -1) {
      // 更新左子节点的时间（父节点时间 + 当前节点时间）
      whisperTimes[leftChildIndex] += whisperTimes[parentNodeIndex];
```

```
      // 将左子节点加入队列
      nodeQueue.push(leftChildIndex);
      // 更新最大时间
      maxTime = Math.max(maxTime, whisperTimes[leftChildIndex]);
    }

    // 如果右子节点存在，处理右子节点
    if (rightChildIndex < whisperTimes.length && whisperTimes[rightChildIndex] !== -1) {
      // 更新右子节点的时间（父节点时间 + 当前节点时间）
      whisperTimes[rightChildIndex] += whisperTimes[parentNodeIndex];
      // 将右子节点加入队列
      nodeQueue.push(rightChildIndex);
      // 更新最大时间
      maxTime = Math.max(maxTime, whisperTimes[rightChildIndex]);
    }
  }

  // 所有节点都接收到悄悄话后，打印最大时间
  console.log( maxTime);

  // 关闭readline接口实例
  rl.close();
});
```

Python

```
from collections import deque

# 读取一行输入并将其转换为整数列表
# 列表中的每个元素代表从父节点到当前节点的时间
whisper_times = list(map(int, input().split()))

# 记录最后一个节点接收悄悄话的时间
max_time = 0

# 使用队列来进行二叉树的层次遍历
node_queue = deque([0])  # 将根节点索引0加入队列

# 当队列不为空时，继续遍历
while node_queue:
```

```
    # 从队列中取出一个节点索引
    parent_node_index = node_queue.popleft()

    # 计算左子节点索引
    left_child_index = 2 * parent_node_index + 1
    # 计算右子节点索引
    right_child_index = 2 * parent_node_index + 2

    # 如果左子节点存在，处理左子节点
    if left_child_index < len(whisper_times) and whisper_times[left_child_index] != -1:
        # 更新左子节点的时间（父节点时间 + 当前节点时间）
        whisper_times[left_child_index] += whisper_times[parent_node_index]
        # 将左子节点加入队列
        node_queue.append(left_child_index)
        # 更新最大时间
        max_time = max(max_time, whisper_times[left_child_index])

    # 如果右子节点存在，处理右子节点
    if right_child_index < len(whisper_times) and whisper_times[right_child_index] != -1:
        # 更新右子节点的时间（父节点时间 + 当前节点时间）
        whisper_times[right_child_index] += whisper_times[parent_node_index]
        # 将右子节点加入队列
        node_queue.append(right_child_index)
        # 更新最大时间
        max_time = max(max_time, whisper_times[right_child_index])

# 所有节点都接收到悄悄话后，打印最大时间
print(max_time)
```

C语言

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

#define MAX_SIZE 10000 // 假设二叉树节点数不超过10000

int main() {
    // 读取一行输入并将其转换为整数数组
    char input[MAX_SIZE];
```

```
    fgets(input, sizeof(input), stdin);

    int whisperTimes[MAX_SIZE];
    int i = 0, time;
    char *token = strtok(input, " ");
    while (token != NULL) {
        sscanf(token, "%d", &time);
        whisperTimes[i++] = time;
        token = strtok(NULL, " ");
    }
    int length = i; // 数组长度

    // 记录最后一个节点接收悄悄话的时间
    int maxTime = 0;

    // 使用数组模拟队列进行二叉树的层次遍历
    int queue[MAX_SIZE];
    int front = 0, rear = 0; // 队列的头和尾索引

    // 将根节点索引0加入队列
    queue[rear++] = 0;

    // 当队列不为空时，继续遍历
    while (front < rear) {
        // 从队列中取出一个节点索引
        int parentNodeIndex = queue[front++];

        // 计算左子节点索引
        int leftChildIndex = 2 * parentNodeIndex + 1;
        // 计算右子节点索引
        int rightChildIndex = 2 * parentNodeIndex + 2;

        // 如果左子节点存在，处理左子节点
        if (leftChildIndex < length && whisperTimes[leftChildIndex] != -1) {
            // 更新左子节点的时间（父节点时间 + 当前节点时间）
            whisperTimes[leftChildIndex] += whisperTimes[parentNodeIndex];
            // 将左子节点加入队列
            queue[rear++] = leftChildIndex;
            // 更新最大时间
            if (whisperTimes[leftChildIndex] > maxTime) {
```

```
                maxTime = whisperTimes[leftChildIndex];
            }
        }

        // 如果右子节点存在，处理右子节点
        if (rightChildIndex < length && whisperTimes[rightChildIndex] != -1) {
            // 更新右子节点的时间（父节点时间 + 当前节点时间）
            whisperTimes[rightChildIndex] += whisperTimes[parentNodeIndex];
            // 将右子节点加入队列
            queue[rear++] = rightChildIndex;
            // 更新最大时间
            if (whisperTimes[rightChildIndex] > maxTime) {
                maxTime = whisperTimes[rightChildIndex];
            }
        }
    }

    // 所有节点都接收到悄悄话后，打印最大时间
    printf("%d\n", maxTime);
    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师