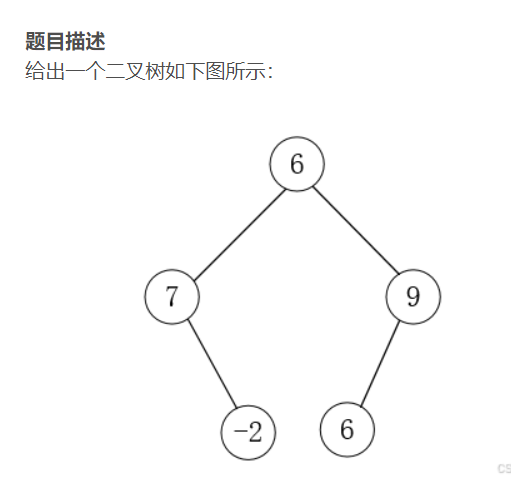
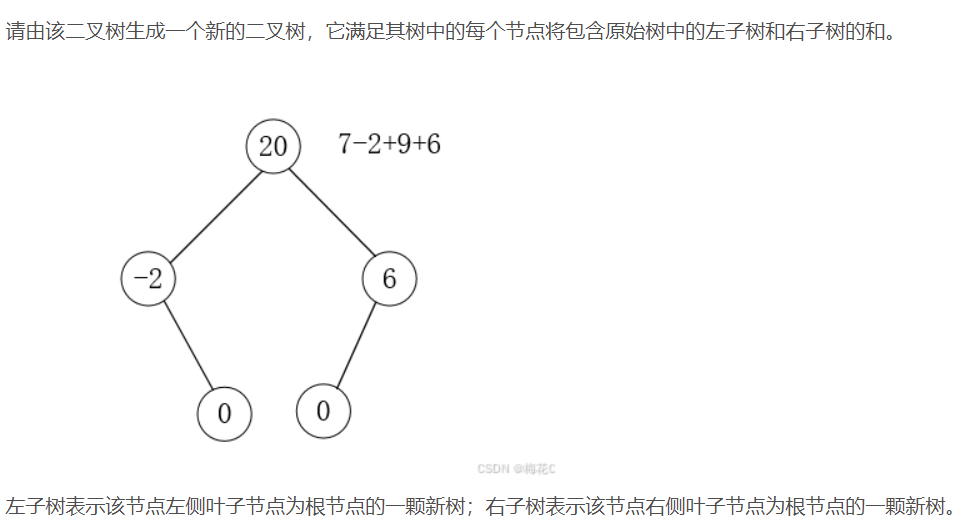
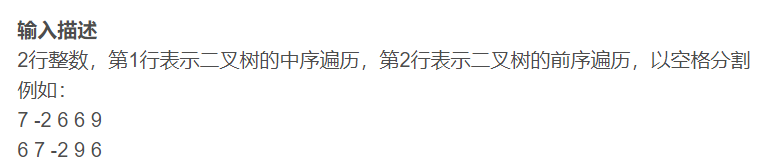
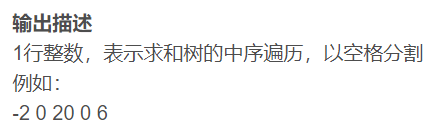
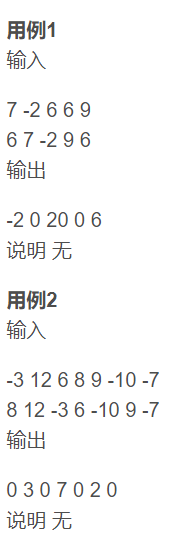
# **E卷-二叉树计算[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**













import java.util.\*;

class Node {

int val;

Node left, right;

int last;

Node(int v) {

val = v;

left = right = null;

last = 0;

}

}

public class Main {

// 构建树的递归函数

public static Node buildTree(int[] pre, int[] mid) {

Map<Integer, List<Integer>> mp = new HashMap<>();

Map<Integer, Integer> cnt = new HashMap<>();

for (int i = 0; i < mid.length; ++i) {

int val = mid[i];

mp.putIfAbsent(val, new ArrayList<>());

mp.get(val).add(i);

}

int i[] = {0};

Map<Integer, Integer> c = new HashMap<>();

int index[] = new int[pre.length];

for (int k = 0; k < pre.length; ++k) {

int val = pre[k];

c.put(val, c.getOrDefault(val, 0) + 1);

index[k] = c.get(val) - 1;

}

return build(0, mid.length - 1, pre, mp, index, i);

}

private static Node build(int lc, int rc, int[] pre,

Map<Integer, List<Integer>> mp, int[] index, int[] i) {

if (i[0] >= pre.length) return null;

int val = pre[i[0]];

int idx = mp.get(val).get(index[i[0]]);

Node now = new Node(val);

if (lc <= idx - 1) {

i[0]++;

now.left = build(lc, idx - 1, pre, mp, index, i);

}

if (rc >= idx + 1) {

i[0]++;

now.right = build(idx + 1, rc, pre, mp, index, i);

}

return now;

}

// 深度优先遍历，为每个节点计算 last 值

public static void dfs(Node now) {

if (now.left != null) {

dfs(now.left);

now.last += now.left.last + now.left.val;

}

if (now.right != null) {

dfs(now.right);

now.last += now.right.last + now.right.val;

}

}

// 中序遍历，获取每个节点的 last 值

public static void zhong(Node now, List<Integer> ans) {

if (now == null) return;

zhong(now.left, ans);

ans.add(now.last);

zhong(now.right, ans);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String[] tmp = sc.nextLine().split(" ");

int n=tmp.length;

int[] mid = new int[n];

int[] pre = new int[n];

for (int i = 0; i < n; ++i) mid[i] = Integer.parseInt(tmp[i]);

tmp = sc.nextLine().split(" ");

for (int i = 0; i < n; ++i) pre[i] = Integer.parseInt(tmp[i]);

Node root = buildTree(pre, mid);

dfs(root);

List<Integer> ans = new ArrayList<>();

zhong(root, ans);

for (int i = 0; i < ans.size(); ++i) {

if (i > 0) System.out.print(" ");

System.out.print(ans.get(i));

}

System.out.println();

}

}



from collections import defaultdict

class node:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val # 节点值

self.left = left # 左子节点

self.right = right # 右子节点

self.last = 0 # 存储该节点以及其子节点的值总和

def buildTree(pre, mid):

# 构建哈希映射，用于保存中序遍历节点的索引

mp = defaultdict(lambda: defaultdict(lambda: defaultdict(int)))

cnt = defaultdict(int)

# 记录中序遍历中每个节点的出现位置

for k, v in enumerate(mid):

cnt[v] += 1

mp[v][cnt[v] - 1] = k

i = 0

c = defaultdict(int)

index = [0] \* len(pre)

# 记录前序遍历中每个节点的出现位置

for k, v in enumerate(pre):

c[v] += 1

index[k] = c[v] - 1

# 构建树的递归函数

def build(lc, rc):

nonlocal i

if i >= len(pre):

return

val = pre[i]

idx = mp[val][index[i]]

now = node(val)

if lc <= idx - 1:

i += 1

now.left = build(lc, idx - 1)

if rc >= idx + 1:

i += 1

now.right = build(idx + 1, rc)

return now

return build(0, len(mid) - 1)

# 读取中序遍历和前序遍历序列

mid = list(map(int, input().split()))

pre = list(map(int, input().split()))

root = buildTree(pre, mid)

def dfs(now):

if now.left != None:

dfs(now.left)

now.last += now.left.last + now.left.val

if now.right != None:

dfs(now.right)

now.last += now.right.last + now.right.val

dfs(root)

ans = []

def zhong(now):

if now == None:

return

global ans

zhong(now.left)

ans.append(now.last)

zhong(now.right)

zhong(root)

print(" ".join(str(x) for x in ans))



#include <bits/stdc++.h>

using namespace std;

// 定义节点类

class Node {

public:

int val;

Node\* left;

Node\* right;

int last;

Node(int v = 0) : val(v), left(nullptr), right(nullptr), last(0) {}

};

// 构建树的递归函数

Node\* buildTree(vector<int>& pre, vector<int>& mid) {

unordered\_map<int, vector<int>> mp;

unordered\_map<int, int> cnt;

for (int i = 0; i < mid.size(); ++i) {

cnt[mid[i]]++;

mp[mid[i]].push\_back(i);

}

int i = 0;

unordered\_map<int, int> c;

vector<int> index(pre.size(), 0);

for (int k = 0; k < pre.size(); ++k) {

c[pre[k]]++;

index[k] = c[pre[k]] - 1;

}

function<Node\*(int, int)> build = [&](int lc, int rc) -> Node\* {

if (i >= pre.size()) return nullptr;

int val = pre[i];

int idx = mp[val][index[i]];

Node\* now = new Node(val);

if (lc <= idx - 1) {

++i;

now->left = build(lc, idx - 1);

}

if (rc >= idx + 1) {

++i;

now->right = build(idx + 1, rc);

}

return now;

};

return build(0, mid.size() - 1);

}

// 深度优先遍历，为每个节点计算 last 值

void dfs(Node\* now) {

if (now->left != nullptr) {

dfs(now->left);

now->last += now->left->last + now->left->val;

}

if (now->right != nullptr) {

dfs(now->right);

now->last += now->right->last + now->right->val;

}

}

// 中序遍历，获取每个节点的 last 值

void zhong(Node\* now, vector<int>& ans) {

if (now == nullptr) return;

zhong(now->left, ans);

ans.push\_back(now->last);

zhong(now->right, ans);

}

vector<int> split(string& s){

int now = 0;

int f=1;

vector<int> res;

for(auto&i:s){

if(i==' '){

res.push\_back(now\*f);

now=0;

f=1;

}else if(i=='-'){

f=-1;

}else{

now=now\*10+i-'0';

}

}

res.push\_back(now\*f);

return res;

}

int main() {

string a,b;

getline(cin,a);

getline(cin,b);

vector<int> mid=split(a), pre=split(b);

Node\* root = buildTree(pre, mid);

dfs(root);

vector<int> ans;

zhong(root, ans);

for (int i = 0; i < ans.size(); ++i) {

if (i > 0) cout << " ";

cout << ans[i];

}

cout << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 节点结构定义

typedef struct Node {

int val;

struct Node\* left;

struct Node\* right;

int last;

} Node;

// 创建新节点

Node\* newNode(int val) {

Node\* node = (Node\*)malloc(sizeof(Node));

node->val = val;

node->left = NULL;

node->right = NULL;

node->last = 0;

return node;

}

// 构建树的递归函数

Node\* buildTree(int\* pre, int\* mid, int preLen, int midLen) {

int\* mp[1000] = {0};

int cnt[1000] = {0};

int idx[1000] = {0};

int i;

for (i = 0; i < midLen; ++i) {

cnt[mid[i]]++;

}

for (i = 0; i < preLen; ++i) {

idx[i] = --cnt[pre[i]];

}

i = 0;

int c[1000] = {0};

Node\* build(int lc, int rc) {

if (i >= preLen) return NULL;

int val = pre[i];

int k;

for (k = lc; k <= rc; ++k) {

if (mid[k] == val) break;

}

Node\* now = newNode(val);

if (lc <= k - 1) {

i++;

now->left = build(lc, k - 1);

}

if (rc >= k + 1) {

i++;

now->right = build(k + 1, rc);

}

return now;

}

return build(0, midLen - 1);

}

// 深度优先遍历，为每个节点计算 last 值

void dfs(Node\* now) {

if (now->left != NULL) {

dfs(now->left);

now->last += now->left->last + now->left->val;

}

if (now->right != NULL) {

dfs(now->right);

now->last += now->right->last + now->right->val;

}

}

// 中序遍历，获取每个节点的 last 值

void zhong(Node\* now, int\* ans, int\* idx) {

if (now == NULL) return;

zhong(now->left, ans, idx);

ans[(\*idx)++] = now->last;

zhong(now->right, ans, idx);

}

int main() {

int n;

scanf("%d", &n);

int mid[n], pre[n];

for (int i = 0; i < n; ++i) scanf("%d", &mid[i]);

for (int i = 0; i < n; ++i) scanf("%d", &pre[i]);

Node\* root = buildTree(pre, mid, n, n);

dfs(root);

int ans[n];

memset(ans, 0, sizeof(ans));

int idx = 0;

zhong(root, ans, &idx);

for (int i = 0; i < n; ++i) {

if (i > 0) printf(" ");

printf("%d", ans[i]);

}

printf("\n");

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

class Node {

constructor(val = 0) {

this.val = val;

this.left = null;

this.right = null;

this.last = 0;

}

}

// 构建树的递归函数

function buildTree(pre, mid) {

const mp = new Map();

const cnt = new Map();

mid.forEach((v, k) => {

if (!cnt.has(v)) cnt.set(v, 0);

cnt.set(v, cnt.get(v) + 1);

if (!mp.has(v)) mp.set(v, []);

mp.get(v).push(k);

});

let i = 0;

const c = new Map();

const index = new Array(pre.length).fill(0);

pre.forEach((v, k) => {

if (!c.has(v)) c.set(v, 0);

c.set(v, c.get(v) + 1);

index[k] = c.get(v) - 1;

});

function build(lc, rc) {

if (i >= pre.length) return null;

const val = pre[i];

const idx = mp.get(val)[index[i]];

const now = new Node(val);

if (lc <= idx - 1) {

i++;

now.left = build(lc, idx - 1);

}

if (rc >= idx + 1) {

i++;

now.right = build(idx + 1, rc);

}

return now;

}

return build(0, mid.length - 1);

}

// 深度优先遍历，为每个节点计算 last 值

function dfs(now) {

if (now.left !== null) {

dfs(now.left);

now.last += now.left.last + now.left.val;

}

if (now.right !== null) {

dfs(now.right);

now.last += now.right.last + now.right.val;

}

}

// 中序遍历，获取每个节点的 last 值

function zhong(now, ans) {

if (now === null) return;

zhong(now.left, ans);

ans.push(now.last);

zhong(now.right, ans);

}

rl.question("", (midStr) => {

const mid = midStr.split(" ").map(Number);

rl.question("", (preStr) => {

const pre = preStr.split(" ").map(Number);

const root = buildTree(pre, mid);

dfs(root);

const ans = [];

zhong(root, ans);

console.log(ans.join(" "));

rl.close();

});

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// 节点结构定义

type Node struct {

val int

left \*Node

right \*Node

last int

}

// 构建新节点

func newNode(val int) \*Node {

return &Node{val: val}

}

// 构建树的递归函数

func buildTree(pre, mid []int) \*Node {

mp := map[int][]int{}

cnt := map[int]int{}

index := make([]int, len(pre))

for k, v := range mid {

cnt[v]++

mp[v] = append(mp[v], k)

}

c := map[int]int{}

for k, v := range pre {

c[v]++

index[k] = c[v] - 1

}

var i int

var build func(lc, rc int) \*Node

build = func(lc, rc int) \*Node {

if i >= len(pre) {

return nil

}

val := pre[i]

idx := mp[val][index[i]]

now := newNode(val)

if lc <= idx-1 {

i++

now.left = build(lc, idx-1)

}

if rc >= idx+1 {

i++

now.right = build(idx+1, rc)

}

return now

}

return build(0, len(mid)-1)

}

// 深度优先遍历，为每个节点计算 last 值

func dfs(now \*Node) {

if now.left != nil {

dfs(now.left)

now.last += now.left.last + now.left.val

}

if now.right != nil {

dfs(now.right)

now.last += now.right.last + now.right.val

}

}

// 中序遍历，获取每个节点的 last 值

func zhong(now \*Node, ans \*[]int) {

if now == nil {

return

}

zhong(now.left, ans)

\*ans = append(\*ans, now.last)

zhong(now.right, ans)

}

func main() {

reader := bufio.NewReader(os.Stdin)

// 读取中序遍历数组

midStr, \_ := reader.ReadString('\n')

midStr = strings.TrimSpace(midStr)

midArr := strings.Split(midStr, " ")

mid := make([]int, len(midArr))

for i, v := range midArr {

mid[i], \_ = strconv.Atoi(v)

}

// 读取前序遍历数组

preStr, \_ := reader.ReadString('\n')

preStr = strings.TrimSpace(preStr)

preArr := strings.Split(preStr, " ")

pre := make([]int, len(preArr))

for i, v := range preArr {

pre[i], \_ = strconv.Atoi(v)

}

root := buildTree(pre, mid)

dfs(root)

var ans []int

zhong(root, &ans)

ansStr := make([]string, len(ans))

for i, v := range ans {

ansStr[i] = strconv.Itoa(v)

}

fmt.Println(strings.Join(ansStr, " "))

}