Leetcode(c++) 深度优先搜索 简单题

690员工的重要性

题目：

给定一个保存员工信息的数据结构，它包含了员工唯一的id，重要度 和 直系下属的id。

比如，员工1是员工2的领导，员工2是员工3的领导。他们相应的重要度为15, 10, 5。那么员工1的数据结构是[1, 15, [2]]，员工2的数据结构是[2, 10, [3]]，员工3的数据结构是[3, 5, []]。注意虽然员工3也是员工1的一个下属，但是由于并不是直系下属，因此没有体现在员工1的数据结构中。

现在输入一个公司的所有员工信息，以及单个员工id，返回这个员工和他所有下属的重要度之和。

示例 1:

输入: [[1, 5, [2, 3]], [2, 3, []], [3, 3, []]], 1

输出: 11

解释:

员工1自身的重要度是5，他有两个直系下属2和3，而且2和3的重要度均为3。因此员工1的总重要度是 5 + 3 + 3 = 11。

注意:

一个员工最多有一个直系领导，但是可以有多个直系下属

员工数量不超过2000。

思路：

代码：

1. /\*
2. // Employee info
3. class Employee {
4. public:
5. // It's the unique ID of each node.
6. // unique id of this employee
7. int id;
8. // the importance value of this employee
9. int importance;
10. // the id of direct subordinates
11. vector<int> subordinates;
12. };
13. \*/
14. **class** Solution {
15. **public**:
16. **int** getImportance(vector<Employee\*> employees, **int** id) {
17. **int** res=0;
18. Employee\* cur=**new** Employee();
19. **if**(employees.size()<=0){
20. **return** 0;
21. }**else**{
22. **for**(auto i:employees){
23. **if**(i->id==id){
24. cur=i;
25. **break**;
26. }
27. }
28. }
29. res+=cur->importance;
30. **for**(auto a:cur->subordinates){
31. res+=getImportance(employees,a);
32. }
33. **return** res;
34. }
35. };

733图像渲染

题目：

有一幅以二维整数数组表示的图画，每一个整数表示该图画的像素值大小，数值在 0 到 65535 之间。

给你一个坐标 (sr, sc) 表示图像渲染开始的像素值（行 ，列）和一个新的颜色值 newColor，让你重新上色这幅图像。

为了完成上色工作，从初始坐标开始，记录初始坐标的上下左右四个方向上像素值与初始坐标相同的相连像素点，接着再记录这四个方向上符合条件的像素点与他们对应四个方向上像素值与初始坐标相同的相连像素点，……，重复该过程。将所有有记录的像素点的颜色值改为新的颜色值。

最后返回经过上色渲染后的图像。

示例 1:

输入:

image = [[1,1,1],[1,1,0],[1,0,1]]

sr = 1, sc = 1, newColor = 2

输出: [[2,2,2],[2,2,0],[2,0,1]]

解析:

在图像的正中间，(坐标(sr,sc)=(1,1)),

在路径上所有符合条件的像素点的颜色都被更改成2。

注意，右下角的像素没有更改为2，

因为它不是在上下左右四个方向上与初始点相连的像素点。

注意:

image 和 image[0] 的长度在范围 [1, 50] 内。

给出的初始点将满足 0 <= sr < image.length 和 0 <= sc < image[0].length。

image[i][j] 和 newColor 表示的颜色值在范围 [0, 65535]内。

思路：

代码：

1. **class** Solution {
2. **public**:
3. vector<vector<**int**>> floodFill(vector<vector<**int**>>& image, **int** sr, **int** sc, **int** newColor) {
4. **int** oldColor=image[sr][sc];
5. **if**(oldColor!=newColor)
6. dfs(image,sr,sc,oldColor,newColor);
7. **return** image;
8. }
9. **void** dfs(vector<vector<**int**>>& image, **int** i, **int** j, **int** oldColor,**int** newColor){
10. **int** m=image.size();
11. **int** n=image[0].size();
12. **if**(i<0||i>=m||j<0||i>=n||image[i][j]!=oldColor) **return** ;
13. image[i][j]=newColor;
14. dfs(image,i-1,j,oldColor,newColor);
15. dfs(image,i,j-1,oldColor,newColor);
16. dfs(image,i+1,j,oldColor,newColor);
17. dfs(image,i,j+1,oldColor,newColor);
18. }
19. };

Leetcode(c++) 深度优先搜索 中等题

109有序链表转换二叉搜索树

题目：

给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。

示例:

给定的有序链表： [-10, -3, 0, 5, 9],

一个可能的答案是：[0, -3, 9, -10, null, 5], 它可以表示下面这个高度平衡二叉搜索树：

0

/ \

-3 9

/ /

-10 5

思路：

首先看到题目就能想到递归。

递归的子问题： 找到一个数组的中间的位置，其中间往左的数组为其左边的节点，中间往右的数组为其右边的节点。

再从中间往左的数组中找出找出中间位置，这个中间位置的左边为左边的节点，右边为右边的节点，以此类推。

代码：

1. /\*\*
2. \* Definition for singly-linked list.
3. \* struct ListNode {
4. \*     int val;
5. \*     ListNode \*next;
6. \*     ListNode(int x) : val(x), next(NULL) {}
7. \* };
8. \*/
9. /\*\*
10. \* Definition for a binary tree node.
11. \* struct TreeNode {
12. \*     int val;
13. \*     TreeNode \*left;
14. \*     TreeNode \*right;
15. \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
16. \* };
17. \*/
18. **class** Solution {
19. **public**:
20. TreeNode\* sortedListToBST(ListNode\* head) {
21. TreeNode \*root=NULL;
22. vector<**int**>res;
23. **while**(head){
24. res.push\_back(head->val);
25. head=head->next;
26. }
27. **if**(res.size()==0)**return** root;
28. **return** sort(root,res,0,res.size()-1);
29. }
30. TreeNode\*sort(TreeNode\*root,vector<**int**>res,**int** le,**int** ri){//le,ri分别是数组的起始位置和结束位置
31. **if**(le<=ri){
32. vector<**int**>right\_res,left\_res;
33. **int** mid=(ri+le)/2;
34. root=**new** TreeNode(res[mid]);
35. root->left=sort(root->left,res,le,mid-1);
36. root->right=sort(root->right,res,mid+1,ri);
37. }
38. **return** root;
39. }
40. };

199二叉树的右视图

题目：

给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例:

输入: [1,2,3,null,5,null,4]

输出: [1, 3, 4]

解释:

1 <---

/ \

2 3 <---

\ \

5 4 <---

思路：

代码：

1. /\*\*
2. \* Definition for a binary tree node.
3. \* struct TreeNode {
4. \*     int val;
5. \*     TreeNode \*left;
6. \*     TreeNode \*right;
7. \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8. \* };
9. \*/
10. **class** Solution {
11. **public**:
12. vector<**int**> rightSideView(TreeNode\* root) {
13. vector<**int**> res;
14. **if**(!root) {
15. **return** res;
16. }
17. queue<TreeNode\*> que;
18. que.push(root);
19. **while**(!que.empty()) {
20. **int** n = que.size();
21. **while**(n--) {
22. TreeNode\* cur = que.front();
23. que.pop();
24. **if**(n == 0) {
25. res.push\_back(cur->val);
26. }
27. **if**(cur->left) {
28. que.push(cur->left);
29. }
30. **if**(cur->right) {
31. que.push(cur->right);
32. }
33. }
34. }
35. **return** res;
37. }
38. };

473火柴拼正方形

题目：

还记得童话《卖火柴的小女孩》吗？现在，你知道小女孩有多少根火柴，请找出一种能使用所有火柴拼成一个正方形的方法。不能折断火柴，可以把火柴连接起来，并且每根火柴都要用到。

输入为小女孩拥有火柴的数目，每根火柴用其长度表示。输出即为是否能用所有的火柴拼成正方形。

示例 1:

输入: [1,1,2,2,2]

输出: true

解释: 能拼成一个边长为2的正方形，每边两根火柴。

示例 2:

输入: [3,3,3,3,4]

输出: false

解释: 不能用所有火柴拼成一个正方形。

注意:

给定的火柴长度和在 0 到 10^9之间。

火柴数组的长度不超过15。

思路：

题目可以理解为：能不能把一组数字分成4组，每组的和是相同的。

我们注意到题目给出的数组长度最多只有15个，基本上可以使用O(N!)的时间复杂度去解决。所以我们可以直接使用回溯法。回溯的思路是先设置好4条边，然后把每一个火柴看看能不能放到4条边中的一个去，如果可以的话就继续向后扫描，直到所有的火柴全部用上为止。

在使用记录四组的和的方式，进行遍历的时候保存各个组的和，如果不能满足就把这个数字再加上，相当于跳过这个数字的方式。最后结束的条件就是所有的数字全部都用完了，因为如果用完了，说明我们把所有的火柴都放到了4条边中的一个，所以得到每组都满足我们条件的结论。

代码：

形式：int sum = accumulate(vec.begin() , vec.end() , 42);

accumulate带有三个形参：头两个形参指定要累加的元素范围，第三个形参则是累加的初值。

1. **class** Solution {
2. **public**:
3. **bool** makesquare(vector<**int**>& nums) {
4. **if**(nums.size()<4){
5. **return** **false**;
6. }
7. **int** sum=accumulate(nums.begin(),nums.end(),0);
8. **if**(sum%4!=0)
9. **return** **false**;
10. **int** edge=sum/4;
11. vector<**int**> target(4,edge);
12. **return** helper(nums,0,target);
14. }
15. **bool** helper(vector<**int**>& nums,**int** index,vector<**int**>& target){
16. **if**(index=nums.size()){
17. **return** **true**;
18. }
19. **int** num=nums[index];
20. **for**(**int** i=0;i<4;i++){
21. **if**(target[i]>=num){
22. target -=num;
23. **if**(helper(nums,index+1,target)){
24. **return** **true**;
25. }
26. target +=num;
27. }
28. }
29. **return** **false**;
30. }
31. };

491递增子序列

题目：

给定一个整型数组, 你的任务是找到所有该数组的递增子序列，递增子序列的长度至少是2。

示例:

输入: [4, 6, 7, 7]

输出: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]]

说明:

给定数组的长度不会超过15。

数组中的整数范围是 [-100,100]。

给定数组中可能包含重复数字，相等的数字应该被视为递增的一种情况。

思路：

代码：

1. **class** Solution {
2. public:
3. vector<vector<int>> findSubsequences(vector<int>& nums) {
4. int n=nums.size();
5. set<vector<int>> res;
6. vector<int> temp;
7. dfs(nums,res,temp,0);
8. **return** vector<vector<int>>(res.begin(),res.end());
9. }
10. void dfs(vector<int>& nums,set<vector<int>>& res,vector<int>& temp,int index){
11. **if**(temp.size()>=2){
12. res.insert(temp);
13. }
14. **for**(int i=index;i<nums.size();i++){
15. **if**(temp.empty()||nums[i]>=temp.back()){
16. temp.push\_back(nums[i]);
17. dfs(nums,res,temp,i+1);
18. temp.pop\_back();
20. }
21. }
22. }
23. };