【第三十周】字典树（Trie）与双数组字典树（Double-Array-Trie）

```cpp
/*************************************************************************
    > File Name: 1.trie.cpp
    > Author: huguang
mZ> Mail: hug@haizeix.com
    > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

#define BASE 26
class node {
public :
    node() {
        flag = false;
        for (int i = 0; i < BASE; i++) next[i] = nullptr;
    }
    ~node() {}
    bool flag;
    node *next[BASE];
};

class Trie {
public :
    Trie() {
        root = new node();
    }
    bool insert(string word) {
        node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            if (p->next[ind] == nullptr) p->next[ind] = new node();
            p = p->next[ind];
        }
        if (p->flag) return false;
        p->flag = true;
        return true;
```

```
    }
    bool search(string word) {
        node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            p = p->next[ind];
            if (p == nullptr) return false;
        }
        return p->flag;
    }
    static void clearTrie(node *root) {
        if (root == nullptr) return ;
        for (int i = 0; i < BASE; i++) clearTrie(root->next[i]);
        delete root;
        return ;
    }
    ~Trie() {
        clearTrie(root);
    }
private:
    node *root;
};

int main() {
    Trie t;
    int op;
    string s;
    while (cin >> op >> s) {
        switch (op) {
            case 1: t.insert(s); break;
            case 2: cout << "search word = " << s << ", result : "  << t.search(s)
<< endl;
        }
    }
    return 0;
}
```

```
/*************************************************************************
    > File Name: 1.trie.cpp
    > Author: huguang
mZ> Mail: hug@haizeix.com
    > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
```

```cpp
#include <set>
#include <vector>
using namespace std;

#define BASE 26
class node {
public :
    node() {
        flag = false;
        for (int i = 0; i < BASE; i++) next[i] = nullptr;
    }
    ~node() {}
    bool flag;
    node *next[BASE];
};

class Trie {
public :
    Trie() {
        root = new node();
    }
    bool insert(string word) {
        node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            if (p->next[ind] == nullptr) p->next[ind] = new node();
            p = p->next[ind];
        }
        if (p->flag) return false;
        p->flag = true;
        return true;
    }
    bool search(string word) {
        node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            p = p->next[ind];
            if (p == nullptr) return false;
        }
        return p->flag;
    }
    void output() {
        __output(root, "");
        return ;
    }
    static void __output(node *root, string s) {
        if (root == nullptr) return ;
        if (root->flag) cout << "find : " << s << endl;
        for (int i = 0; i < BASE; i++) {
            __output(root->next[i], s + char(i + 'a'));
        }
        return ;
```

```
    }
    static void clearTrie(node *root) {
        if (root == nullptr) return ;
        for (int i = 0; i < BASE; i++) clearTrie(root->next[i]);
        delete root;
        return ;
    }
    ~Trie() {
        clearTrie(root);
    }
private:
    node *root;
};

int main() {
    int n;
    cin >> n;
    string s;
    Trie t;
    for (int i = 0; i < n; i++) {
        cin >> s;
        t.insert(s);
    }
    t.output();
    return 0;
}
```

```
/*************************************************************************
    > File Name: 1.trie.cpp
    > Author: huguang
mZ> Mail: hug@haizeix.com
    > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

#define BASE 26
class node {
public :
    int flag;
    int next[BASE];
    void clear() {
```

```
        flag = 0;
        for (int i = 0; i < BASE; i++) {
            next[i] = 0;
        }
    }
} trie[10000];
int cnt, root;
void clearTrie() {
    cnt = 2, root = 1;
    trie[root].clear();
    return ;
}

int getNewNode() {
    trie[cnt].clear();
    return cnt++;
}

void insert(string s) {
    int p = root;
    for (auto x : s) {
        int ind = x - 'a';
        if (trie[p].next[ind] == 0) trie[p].next[ind] = getNewNode();
        p = trie[p].next[ind];
    }
    trie[p].flag = 1;
    return ;
}

bool search(string s) {
    int p = root;
    for (auto x : s) {
        int ind = x - 'a';
        p = trie[p].next[ind];
        if (p == 0) return false;
    }
    return trie[p].flag;
}

int main() {
    cout << "trie version 3 : " << endl;
    clearTrie();
    int op;
    string s;
    while (cin >> op >> s) {
        switch (op) {
            case 1: insert(s); break;
            case 2: cout << "search word = " << s << ", result : "  << search(s) <<
endl;
        }
    }
    return 0;
```

```
}
```

```
/*************************************************************************
    > File Name: 1.trie.cpp
    > Author: huguang
mZ> Mail: hug@haizeix.com
    > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

#define BASE 26
#define MAX_CNT 10000
class node {
public :
    int flag;
    int next[BASE];
    void clear() {
        flag = 0;
        for (int i = 0; i < BASE; i++) {
            next[i] = 0;
        }
    }
} trie[MAX_CNT];
int cnt, root;
void clearTrie() {
    cnt = 2, root = 1;
    trie[root].clear();
    return ;
}

int getNewNode() {
    trie[cnt].clear();
    return cnt++;
}

void insert(string s) {
    int p = root;
    for (auto x : s) {
        int ind = x - 'a';
        if (trie[p].next[ind] == 0) trie[p].next[ind] = getNewNode();
        p = trie[p].next[ind];
```

```
    }
    trie[p].flag = 1;
    return ;
}

bool search(string s) {
    int p = root;
    for (auto x : s) {
        int ind = x - 'a';
        p = trie[p].next[ind];
        if (p == 0) return false;
    }
    return trie[p].flag;
}

int *base, *check, da_root = 1;

int getBaseValue(int root, int *base, int *check) {
    int b = 1, flag = 0;
    while (flag == 0) {
        flag = 1;
        b += 1;
        for (int i = 0; i < BASE; i++) {
            if (trie[root].next[i] == 0) continue;
            if (check[b + i] == 0) continue;
            flag = 0;
            break;
        }
    }
    return b;
}

int ConvertToDoubleArrayTrie(int root, int da_root, int *base, int *check) {
    if (root == 0) return 0;
    int max_ind = da_root;
    base[da_root] = getBaseValue(root, base, check);
    for (int i = 0; i < BASE; i++) {
        if (trie[root].next[i] == 0) continue;
        check[base[da_root] + i] = da_root;
        if (trie[trie[root].next[i]].flag) {
            check[base[da_root] + i] = -da_root;
        }
    }
    for (int i = 0; i < BASE; i++) {
        if (trie[root].next[i] == 0) continue;
        max_ind = max(
            max_ind,
            ConvertToDoubleArrayTrie(
                trie[root].next[i],
                base[da_root] + i,
                base, check
            )
```

```
        );
    }
    return max_ind;
}

bool searchDATrie(string s) {
    int p = da_root;
    for (auto x : s) {
        int ind = x - 'a';
        if (abs(check[base[p] + ind]) != p) return false;
        p = base[p] + ind;
    }
    return check[p] < 0;
}

int main() {
    cout << "trie version 4 : " << endl;
    clearTrie();
    int n;
    cin >> n;
    string s;
    for (int i = 0; i < n; i++) {
        cin >> s;
        insert(s);
    }
    base  = new int[MAX_CNT];
    check = new int[MAX_CNT];
    memset(base, 0, sizeof(int) * MAX_CNT);
    memset(check, 0, sizeof(int) * MAX_CNT);
    int max_ind = ConvertToDoubleArrayTrie(root, da_root, base, check);
    printf("trie usage : %lu byte\n", cnt * sizeof(node));
    printf("double array trie usage : %lu byte\n", (max_ind + 1) * sizeof(int) * 2);
    while (cin >> s) {
        cout << "find " << s << ", from trie : " << search(s) << endl;
        cout << "find " << s << ", from da trie : " << searchDATrie(s) << endl;
    }
    return 0;
}
```

208. 实现 Trie (前缀树)

Trie（发音类似 "try"）或者说 前缀树 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

- `Trie()` 初始化前缀树对象。
- `void insert(String word)` 向前缀树中插入字符串 `word` 。

- `boolean search(String word)` 如果字符串 `word` 在前缀树中，返回 `true`（即，在检索之前已经插入）；否则，返回 `false` 。
- `boolean startsWith(String prefix)` 如果之前已经插入的字符串 `word` 的前缀之一为 `prefix` ，返回 `true` ；否则，返回 `false` 。

示例：

```
输入
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出
[null, null, true, false, true, null, true]

解释
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True
```

```
class Node {
public :
    Node() {
        flag = 0;
        for (int i = 0; i < 26; i++) {
            next[i] = nullptr;
        }
    }
    int flag;
    Node *next[26];
};
class Trie {
public:
    Node *root;
    Trie() {
        root = new Node();
    }

    void insert(string word) {
        Node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            if (p->next[ind] == nullptr) p->next[ind] = new Node();
            p = p->next[ind];
        }
        p->flag = true;
        return ;
    }

```

```
    bool search(string word) {
        Node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            p = p->next[ind];
            if (p == nullptr) return false;
        }
        return p->flag;
    }

    bool startsWith(string prefix) {
        Node *p = root;
        for (auto x : prefix) {
            int ind = x - 'a';
            p = p->next[ind];
            if (p == nullptr) return false;
        }
        return true;
    }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

1268. 搜索推荐系统

给你一个产品数组 `products` 和一个字符串 `searchWord` ， `products` 数组中每个产品都是一个字符串。

请你设计一个推荐系统，在依次输入单词 `searchWord` 的每一个字母后，推荐 `products` 数组中前缀与 `searchWord` 相同的最多三个产品。如果前缀相同的可推荐产品超过三个，请按字典序返回最小的三个。

请你以二维列表的形式，返回在输入 `searchWord` 每个字母后相应的推荐产品的列表。

示例 **1**：

```
输入：products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord = "mouse"
输出：[
["mobile","moneypot","monitor"],
["mobile","moneypot","monitor"],
["mouse","mousepad"],
["mouse","mousepad"],
["mouse","mousepad"]
]
解释：按字典序排序后的产品列表是 ["mobile","moneypot","monitor","mouse","mousepad"]
输入 m 和 mo，由于所有产品的前缀都相同，所以系统返回字典序最小的三个产品 ["mobile","moneypot","monitor"]
输入 mou， mous 和 mouse 后系统都返回 ["mouse","mousepad"]
```

```cpp
#define BASE 26
class node {
public :
    node() {
        flag = false;
        for (int i = 0; i < BASE; i++) next[i] = nullptr;
    }
    ~node() {}
    set<string> s;
    bool flag;
    node *next[BASE];
};

class Trie {
public :
    Trie() {
        root = new node();
    }
    bool insert(string word) {
        node *p = root;
        for (auto x : word) {
            int ind = x - 'a';
            if (p->next[ind] == nullptr) p->next[ind] = new node();
            p = p->next[ind];
            p->s.insert(word);
            if (p->s.size() > 3) {
                auto iter = p->s.end();
                iter--;
                p->s.erase(iter);
            }
        }
        if (p->flag) return false;
        p->flag = true;
        return true;
    }
    vector<vector<string>> search(string word) {
        node *p = root;
```

```
        vector<vector<string>> ret;
        for (auto x : word) {
            if (p == nullptr) {
                ret.push_back(vector<string>());
                continue;
            }
            int ind = x - 'a';
            p = p->next[ind];
            vector<string> temp;
            if (p != nullptr) {
                for (auto s : p->s) temp.push_back(s);
            }
            ret.push_back(temp);
        }
        return ret;
    }
    static void clearTrie(node *root) {
        if (root == nullptr) return ;
        for (int i = 0; i < BASE; i++) clearTrie(root->next[i]);
        delete root;
        return ;
    }
    ~Trie() {
        clearTrie(root);
    }
private:
    node *root;
};

class Solution {
public:
    vector<vector<string>> suggestedProducts(vector<string>& products, string
searchWord) {
        Trie tree;
        for (auto s : products) tree.insert(s);
        return tree.search(searchWord);
    }
};
```

241. 为运算表达式设计优先级

给定一个含有数字和运算符的字符串，为表达式添加括号，改变其运算优先级以求出不同的结果。你需要给出所有可能的组合的结果。有效的运算符号包含 `+`，`-` 以及 `*` 。

示例 **1:**

```
输入:"2-1-1"输出:[0, 2]解释:
((2-1)-1) = 0
(2-(1-1)) = 2
```

```cpp
class Solution {
public:
    vector<int> diffWaysToCompute(string expression) {
        vector<int> ret;
        for (int i = 0; expression[i]; i++) {
            char op = expression[i];
            if (op != '+' && op != '-' && op != '*') continue;
            string a_str = expression.substr(0, i);
            string b_str = expression.substr(i + 1, expression.size());
            vector<int> a = diffWaysToCompute(a_str);
            vector<int> b = diffWaysToCompute(b_str);
            for (auto x : a) {
                for (auto y : b) {
                    switch (op) {
                        case '+': ret.push_back(x + y); break;
                        case '-': ret.push_back(x - y); break;
                        case '*': ret.push_back(x * y); break;
                    }
                }
            }
        }
        if (ret.size() == 0) {
            int num = 0;
            for (auto x : expression) {
                num = num * 10 + (x - '0');
            }
            ret.push_back(num);
        }
        return ret;
    }
};
```

剑指 Offer II 067. 最大的异或

给定一个整数数组 `nums` ，返回 ** `nums[i] XOR nums[j]` 的最大运算结果，其中 `0 ≤ i ≤ j < n` 。

示例 **1**：

```
输入：nums = [3,10,5,25,2,8]
输出：28
解释：最大运算结果是 5 XOR 25 = 28.
```

```cpp
class Node {
public :
    Node() {
        for (int i = 0; i < 2; i++) {
            next[i] = nullptr;
        }
    }
```

```
    Node *next[2];
};
class Trie {
public:
    Node *root;
    Trie() {
        root = new Node();
    }

    void insert(int x) {
        Node *p = root;
        for (int i = 30; i >= 0; --i) {
            int ind = !!(x & (1 << i));
            if (p->next[ind] == nullptr) p->next[ind] = new Node();
            p = p->next[ind];
        }
        return ;
    }

    int search(int x) {
        Node *p = root;
        int ans = 0;
        for (int i = 30; i >= 0; --i) {
            int ind = !!(x & (1 << i));
            if (p->next[!ind]) {
                ans |= (1 << i);
                p = p->next[!ind];
            } else {
                p = p->next[ind];
            }
        }
        return ans;
    }
};
class Solution {
public:
    int findMaximumXOR(vector<int>& nums) {
        Trie tree;
        for (auto x : nums) tree.insert(x);
        int ans = 0;
        for (auto x : nums) ans = max(ans, tree.search(x));
        return ans;
    }
};
```

133. 克隆图

给你无向 连通 图中一个节点的引用，请你返回该图的 深拷贝（克隆）。

图中的每个节点都包含它的值 `val`（`int`）和其邻居的列表（`list[Node]`）。

```java
class Node {
    public int val;
    public List<Node> neighbors;
}
```

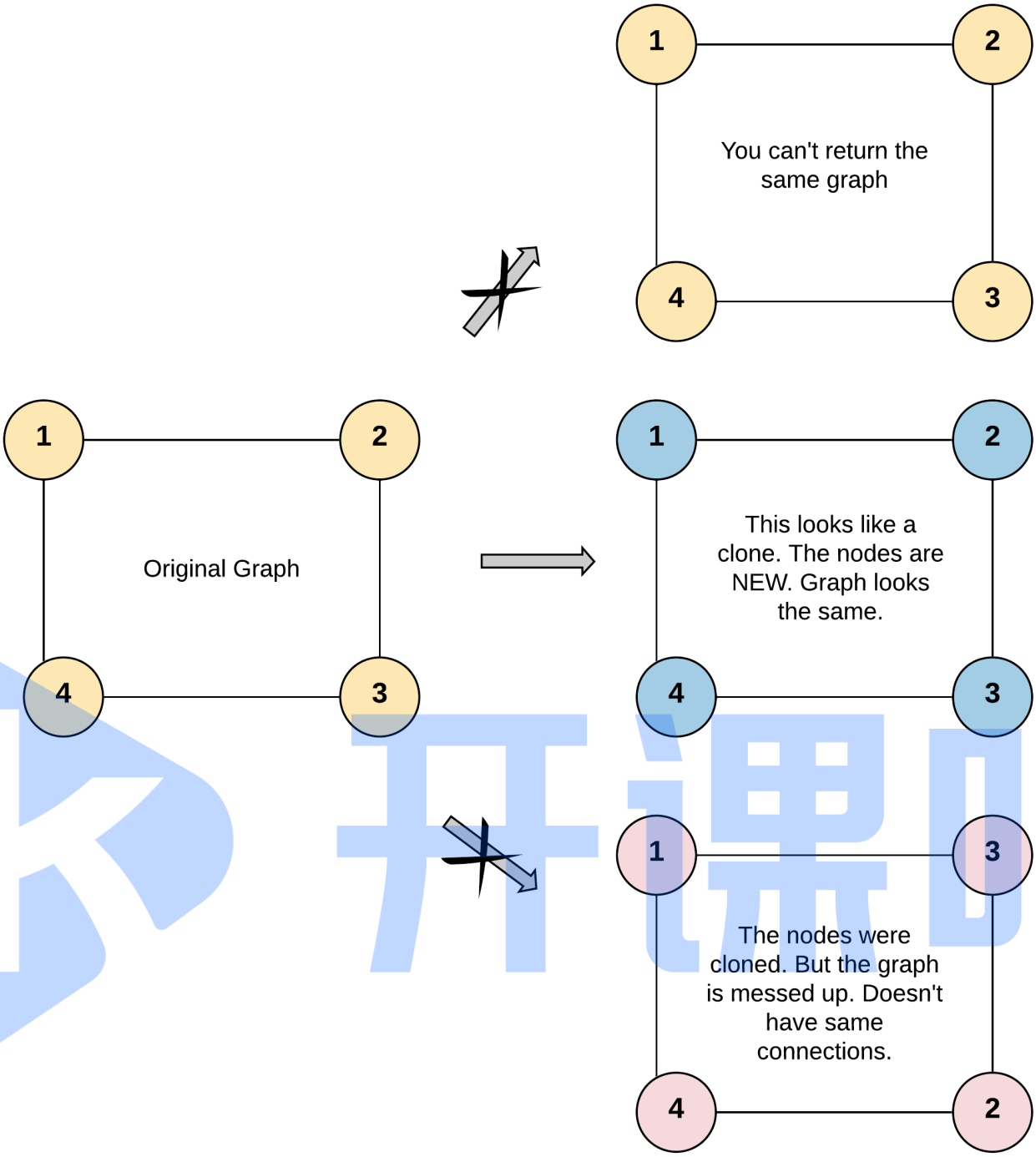
测试用例格式：

简单起见，每个节点的值都和它的索引相同。例如，第一个节点值为 1（`val = 1`），第二个节点值为 2（`val = 2`），以此类推。该图在测试用例中使用邻接列表表示。

邻接列表 是用于表示有限图的无序列表的集合。每个列表都描述了图中节点的邻居集。

给定节点将始终是图中的第一个节点（值为 1）。你必须将 给定节点的拷贝 作为对克隆图的引用返回。

示例 **1**：




```
输入：adjList = [[2,4],[1,3],[2,4],[1,3]]
输出：[[2,4],[1,3],[2,4],[1,3]]
解释：
图中有 4 个节点。
节点 1 的值是 1，它有两个邻居：节点 2 和 4 。
节点 2 的值是 2，它有两个邻居：节点 1 和 3 。
节点 3 的值是 3，它有两个邻居：节点 2 和 4 。
节点 4 的值是 4，它有两个邻居：节点 1 和 3 。
```

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> neighbors;
    Node() {
```

```
        val = 0;
        neighbors = vector<Node*>();
    }
    Node(int _val) {
        val = _val;
        neighbors = vector<Node*>();
    }
    Node(int _val, vector<Node*> _neighbors) {
        val = _val;
        neighbors = _neighbors;
    }
};
*/

class Solution {
public:
    unordered_map<Node *, Node *> h;
    Node *getResult(Node *node) {
        if (node == nullptr) return nullptr;
        if (h[node]) return h[node];
        h[node] = new Node(node->val);
        for (auto x : node->neighbors) {
            getResult(x);
            h[node]->neighbors.push_back(h[x]);
        }
        return h[node];
    }
    Node* cloneGraph(Node* node) {
        h.clear();
        return getResult(node);
    }
};
```

987. 二叉树的垂序遍历

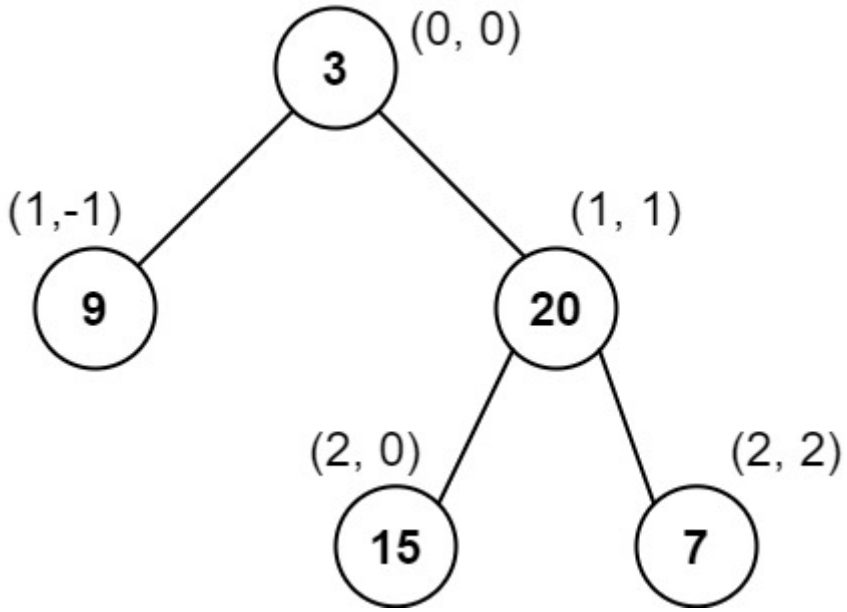
给你二叉树的根结点 `root` ，请你设计算法计算二叉树的 垂序遍历** 序列。

对位于 `(row, col)` 的每个结点而言，其左右子结点分别位于 `(row + 1, col - 1)` 和 `(row + 1, col + 1)` 。树的根结点位于 `(0, 0)` 。

二叉树的 垂序遍历 从最左边的列开始直到最右边的列结束，按列索引每一列上的所有结点，形成一个按出现位置从上到下排序的有序列表。如果同行同列上有多个结点，则按结点的值从小到大进行排序。

返回二叉树的 垂序遍历 序列。

示例 **1：**




```
输入：root = [3,9,20,null,null,15,7]
输出：[[9],[3,15],[20],[7]]
解释：
列 -1 ：只有结点 9 在此列中。
列  0 ：只有结点 3 和 15 在此列中，按从上到下顺序。
列  1 ：只有结点 20 在此列中。
列  2 ：只有结点 7 在此列中。
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    typedef pair<int, int> PII;
    map<int, vector<PII>> h;
    void getResult(TreeNode *root, int i, int j) {
        if (root == nullptr) return ;
        h[j].push_back(PII(i, root->val));
        getResult(root->left,  i + 1, j - 1);
        getResult(root->right, i + 1, j + 1);
        return ;
    }
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        h.clear();
        getResult(root, 0, 0);
        vector<vector<int>> ret;
```

```
        for (auto item : h) {
            vector<PII> &arr = item.second;
            sort(arr.begin(), arr.end());
            vector<int> temp;
            for (auto x : arr) temp.push_back(x.second);
            ret.push_back(temp);
        }
        return ret;
    }
};
```