

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

```
1  /*****
2      > File Name: 1.trie.cpp
3      > Author: huguang
4  mZ> Mail: hug@haizeix.com
5      > Created Time:
6      *****/
7
8  #include <iostream>
9  #include <cstdio>
10 #include <cstdlib>
11 #include <queue>
12 #include <stack>
13 #include <algorithm>
14 #include <string>
15 #include <map>
16 #include <set>
17 #include <vector>
18 using namespace std;
19
20 #define BASE 26
21 class node {
22 public:
23     node() {
24         flag = false;
25         for (int i = 0; i < BASE; i++) next[i] = nullptr;
26     }
27     ~node() {}
28     bool flag;
29     node *next[BASE];
30 };
31
32 class Trie {
33 public:
34     Trie() {
35         root = new node();
36     }
37     bool insert(string word) {
38         node *p = root;
39         for (auto x : word) {
40             int ind = x - 'a';
41             if (p->next[ind] == nullptr) p->next[ind] = new node();
42             p = p->next[ind];
43         }
44         if (p->flag) return false;
45         p->flag = true;
46         return true;
```

```

47     }
48     bool search(string word) {
49         node *p = root;
50         for (auto x : word) {
51             int ind = x - 'a';
52             p = p->next[ind];
53             if (p == nullptr) return false;
54         }
55         return p->flag;
56     }
57     static void clearTrie(node *root) {
58         if (root == nullptr) return ;
59         for (int i = 0; i < BASE; i++) clearTrie(root->next[i]);
60         delete root;
61         return ;
62     }
63     ~Trie() {
64         clearTrie(root);
65     }
66 private:
67     node *root;
68 };
69
70 int main() {
71     Trie t;
72     int op;
73     string s;
74     while (cin >> op >> s) {
75         switch (op) {
76             case 1: t.insert(s); break;
77             case 2: cout << "search word = " << s << ", result : " << t.search(s)
<< endl;
78         }
79     }
80     return 0;
81 }

```

```

1  /*****
2      > File Name: 1.trie.cpp
3      > Author: huguang
4  mZ> Mail: hug@haizeix.com
5      > Created Time:
6      *****/
7
8  #include <iostream>
9  #include <cstdio>
10 #include <cstdlib>
11 #include <queue>
12 #include <stack>
13 #include <algorithm>
14 #include <string>
15 #include <map>

```

```
16 #include <set>
17 #include <vector>
18 using namespace std;
19
20 #define BASE 26
21 class node {
22 public :
23     node() {
24         flag = false;
25         for (int i = 0; i < BASE; i++) next[i] = nullptr;
26     }
27     ~node() {}
28     bool flag;
29     node *next[BASE];
30 };
31
32 class Trie {
33 public :
34     Trie() {
35         root = new node();
36     }
37     bool insert(string word) {
38         node *p = root;
39         for (auto x : word) {
40             int ind = x - 'a';
41             if (p->next[ind] == nullptr) p->next[ind] = new node();
42             p = p->next[ind];
43         }
44         if (p->flag) return false;
45         p->flag = true;
46         return true;
47     }
48     bool search(string word) {
49         node *p = root;
50         for (auto x : word) {
51             int ind = x - 'a';
52             p = p->next[ind];
53             if (p == nullptr) return false;
54         }
55         return p->flag;
56     }
57     void output() {
58         __output(root, "");
59         return ;
60     }
61     static void __output(node *root, string s) {
62         if (root == nullptr) return ;
63         if (root->flag) cout << "find : " << s << endl;
64         for (int i = 0; i < BASE; i++) {
65             __output(root->next[i], s + char(i + 'a'));
66         }
67         return ;
68     }
69 }
```

```

68     }
69     static void clearTrie(node *root) {
70         if (root == nullptr) return ;
71         for (int i = 0; i < BASE; i++) clearTrie(root->next[i]);
72         delete root;
73         return ;
74     }
75     ~Trie() {
76         clearTrie(root);
77     }
78 private:
79     node *root;
80 };
81
82 int main() {
83     int n;
84     cin >> n;
85     string s;
86     Trie t;
87     for (int i = 0; i < n; i++) {
88         cin >> s;
89         t.insert(s);
90     }
91     t.output();
92     return 0;
93 }

```

```

1  /*****
2  > File Name: 1.trie.cpp
3  > Author: huguang
4  mZ> Mail: hug@haizeix.com
5  > Created Time:
6  *****/
7
8  #include <iostream>
9  #include <cstdio>
10 #include <cstdlib>
11 #include <queue>
12 #include <stack>
13 #include <algorithm>
14 #include <string>
15 #include <map>
16 #include <set>
17 #include <vector>
18 using namespace std;
19
20 #define BASE 26
21 class node {
22 public :
23     int flag;
24     int next[BASE];
25     void clear() {

```

```
26     flag = 0;
27     for (int i = 0; i < BASE; i++) {
28         next[i] = 0;
29     }
30 }
31 } trie[10000];
32 int cnt, root;
33 void clearTrie() {
34     cnt = 2, root = 1;
35     trie[root].clear();
36     return ;
37 }
38
39 int getNewNode() {
40     trie[cnt].clear();
41     return cnt++;
42 }
43
44 void insert(string s) {
45     int p = root;
46     for (auto x : s) {
47         int ind = x - 'a';
48         if (trie[p].next[ind] == 0) trie[p].next[ind] = getNewNode();
49         p = trie[p].next[ind];
50     }
51     trie[p].flag = 1;
52     return ;
53 }
54
55 bool search(string s) {
56     int p = root;
57     for (auto x : s) {
58         int ind = x - 'a';
59         p = trie[p].next[ind];
60         if (p == 0) return false;
61     }
62     return trie[p].flag;
63 }
64
65 int main() {
66     cout << "trie version 3 : " << endl;
67     clearTrie();
68     int op;
69     string s;
70     while (cin >> op >> s) {
71         switch (op) {
72             case 1: insert(s); break;
73             case 2: cout << "search word = " << s << ", result : " << search(s) <<
endl;
74         }
75     }
76     return 0;
```

```
1  /*****
2      > File Name: 1.trie.cpp
3      > Author: huguang
4  mZ> Mail: hug@haizeix.com
5      > Created Time:
6      *****/
7
8  #include <iostream>
9  #include <cstdio>
10 #include <cstdlib>
11 #include <queue>
12 #include <stack>
13 #include <algorithm>
14 #include <string>
15 #include <map>
16 #include <set>
17 #include <vector>
18 using namespace std;
19
20 #define BASE 26
21 #define MAX_CNT 10000
22 class node {
23 public :
24     int flag;
25     int next[BASE];
26     void clear() {
27         flag = 0;
28         for (int i = 0; i < BASE; i++) {
29             next[i] = 0;
30         }
31     }
32 } trie[MAX_CNT];
33 int cnt, root;
34 void clearTrie() {
35     cnt = 2, root = 1;
36     trie[root].clear();
37     return ;
38 }
39
40 int getNewNode() {
41     trie[cnt].clear();
42     return cnt++;
43 }
44
45 void insert(string s) {
46     int p = root;
47     for (auto x : s) {
48         int ind = x - 'a';
49         if (trie[p].next[ind] == 0) trie[p].next[ind] = getNewNode();
50         p = trie[p].next[ind];
```

```

51     }
52     trie[p].flag = 1;
53     return ;
54 }
55
56 bool search(string s) {
57     int p = root;
58     for (auto x : s) {
59         int ind = x - 'a';
60         p = trie[p].next[ind];
61         if (p == 0) return false;
62     }
63     return trie[p].flag;
64 }
65
66 int *base, *check, da_root = 1;
67
68 int getBaseValue(int root, int *base, int *check) {
69     int b = 1, flag = 0;
70     while (flag == 0) {
71         flag = 1;
72         b += 1;
73         for (int i = 0; i < BASE; i++) {
74             if (trie[root].next[i] == 0) continue;
75             if (check[b + i] == 0) continue;
76             flag = 0;
77             break;
78         }
79     }
80     return b;
81 }
82
83 int ConvertToDoubleArrayTrie(int root, int da_root, int *base, int *check) {
84     if (root == 0) return 0;
85     int max_ind = da_root;
86     base[da_root] = getBaseValue(root, base, check);
87     for (int i = 0; i < BASE; i++) {
88         if (trie[root].next[i] == 0) continue;
89         check[base[da_root] + i] = da_root;
90         if (trie[trie[root].next[i]].flag) {
91             check[base[da_root] + i] = -da_root;
92         }
93     }
94     for (int i = 0; i < BASE; i++) {
95         if (trie[root].next[i] == 0) continue;
96         max_ind = max(
97             max_ind,
98             ConvertToDoubleArrayTrie(
99                 trie[root].next[i],
100                 base[da_root] + i,
101                 base, check
102             )

```

```

103     );
104 }
105 return max_ind;
106 }
107
108 bool searchDATrie(string s) {
109     int p = da_root;
110     for (auto x : s) {
111         int ind = x - 'a';
112         if (abs(check[base[p] + ind]) != p) return false;
113         p = base[p] + ind;
114     }
115     return check[p] < 0;
116 }
117
118 int main() {
119     cout << "trie version 4 : " << endl;
120     clearTrie();
121     int n;
122     cin >> n;
123     string s;
124     for (int i = 0; i < n; i++) {
125         cin >> s;
126         insert(s);
127     }
128     base = new int[MAX_CNT];
129     check = new int[MAX_CNT];
130     memset(base, 0, sizeof(int) * MAX_CNT);
131     memset(check, 0, sizeof(int) * MAX_CNT);
132     int max_ind = ConvertToDoubleArrayTrie(root, da_root, base, check);
133     printf("trie usage : %lu byte\n", cnt * sizeof(node));
134     printf("double array trie usage : %lu byte\n", (max_ind + 1) * sizeof(int) * 2);
135     while (cin >> s) {
136         cout << "find " << s << ", from trie : " << search(s) << endl;
137         cout << "find " << s << ", from da trie : " << searchDATrie(s) << endl;
138     }
139     return 0;
140 }

```

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：

```
1  输入
2  ["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
3  [[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
4  输出
5  [null, null, true, false, true, null, true]
6
7  解释
8  Trie trie = new Trie();
9  trie.insert("apple");
10 trie.search("apple"); // 返回 True
11 trie.search("app"); // 返回 False
12 trie.startsWith("app"); // 返回 True
13 trie.insert("app");
14 trie.search("app"); // 返回 True
```

```
1  class Node {
2  public :
3      Node() {
4          flag = 0;
5          for (int i = 0; i < 26; i++) {
6              next[i] = nullptr;
7          }
8      }
9      int flag;
10     Node *next[26];
11 };
12 class Trie {
13 public:
14     Node *root;
15     Trie() {
16         root = new Node();
17     }
18
19     void insert(string word) {
20         Node *p = root;
21         for (auto x : word) {
22             int ind = x - 'a';
23             if (p->next[ind] == nullptr) p->next[ind] = new Node();
24             p = p->next[ind];
25         }
26         p->flag = true;
27         return ;
28     }
29 }
```

```

30     bool search(string word) {
31         Node *p = root;
32         for (auto x : word) {
33             int ind = x - 'a';
34             p = p->next[ind];
35             if (p == nullptr) return false;
36         }
37         return p->flag;
38     }
39
40     bool startsWith(string prefix) {
41         Node *p = root;
42         for (auto x : prefix) {
43             int ind = x - 'a';
44             p = p->next[ind];
45             if (p == nullptr) return false;
46         }
47         return true;
48     }
49 };
50
51 /**
52  * Your Trie object will be instantiated and called as such:
53  * Trie* obj = new Trie();
54  * obj->insert(word);
55  * bool param_2 = obj->search(word);
56  * bool param_3 = obj->startsWith(prefix);
57  */

```

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:

```
1  输入: products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord =  
    "mouse"  
2  输出: [  
3      ["mobile","moneypot","monitor"],  
4      ["mobile","moneypot","monitor"],  
5      ["mouse","mousepad"],  
6      ["mouse","mousepad"],  
7      ["mouse","mousepad"]  
8  ]  
9  解释: 按字典序排序后的产品列表是 ["mobile","moneypot","monitor","mouse","mousepad"]  
10 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"]  
11 输入 mou,  mou 和 mouse 后系统都返回 ["mouse","mousepad"]
```

```
1  #define BASE 26  
2  class node {  
3  public :  
4      node() {  
5          flag = false;  
6          for (int i = 0; i < BASE; i++) next[i] = nullptr;  
7      }  
8      ~node() {}  
9      set<string> s;  
10     bool flag;  
11     node *next[BASE];  
12 };  
13  
14 class Trie {  
15 public :  
16     Trie() {  
17         root = new node();  
18     }  
19     bool insert(string word) {  
20         node *p = root;  
21         for (auto x : word) {  
22             int ind = x - 'a';  
23             if (p->next[ind] == nullptr) p->next[ind] = new node();  
24             p = p->next[ind];  
25             p->s.insert(word);  
26             if (p->s.size() > 3) {  
27                 auto iter = p->s.end();  
28                 iter--;  
29                 p->s.erase(iter);  
30             }  
31         }  
32         if (p->flag) return false;  
33         p->flag = true;  
34         return true;  
35     }  
36     vector<vector<string>> search(string word) {  
37         node *p = root;
```

```

38     vector<vector<string>>> ret;
39     for (auto x : word) {
40         if (p == nullptr) {
41             ret.push_back(vector<string>());
42             continue;
43         }
44         int ind = x - 'a';
45         p = p->next[ind];
46         vector<string> temp;
47         if (p != nullptr) {
48             for (auto s : p->s) temp.push_back(s);
49         }
50         ret.push_back(temp);
51     }
52     return ret;
53 }
54 static void clearTrie(node *root) {
55     if (root == nullptr) return ;
56     for (int i = 0; i < BASE; i++) clearTrie(root->next[i]);
57     delete root;
58     return ;
59 }
60 ~Trie() {
61     clearTrie(root);
62 }
63 private:
64     node *root;
65 };
66
67 class Solution {
68 public:
69     vector<vector<string>>> suggestedProducts(vector<string>& products, string
searchWord) {
70         Trie tree;
71         for (auto s : products) tree.insert(s);
72         return tree.search(searchWord);
73     }
74 };

```

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。有效的运算符包含 `+`，`-` 以及 `*`。

示例 1:

```

1 | 输入:"2-1-1"输出:[0, 2]解释:
2 | ((2-1)-1) = 0
3 | (2-(1-1)) = 2

```

```

1  class Solution {
2  public:
3      vector<int> diffWaysToCompute(string expression) {
4          vector<int> ret;
5          for (int i = 0; expression[i]; i++) {
6              char op = expression[i];
7              if (op != '+' && op != '-' && op != '*') continue;
8              string a_str = expression.substr(0, i);
9              string b_str = expression.substr(i + 1, expression.size());
10             vector<int> a = diffWaysToCompute(a_str);
11             vector<int> b = diffWaysToCompute(b_str);
12             for (auto x : a) {
13                 for (auto y : b) {
14                     switch (op) {
15                         case '+': ret.push_back(x + y); break;
16                         case '-': ret.push_back(x - y); break;
17                         case '*': ret.push_back(x * y); break;
18                     }
19                 }
20             }
21         }
22         if (ret.size() == 0) {
23             int num = 0;
24             for (auto x : expression) {
25                 num = num * 10 + (x - '0');
26             }
27             ret.push_back(num);
28         }
29         return ret;
30     }
31 };

```

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:

```

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

```

```

1  class Node {
2  public :
3      Node() {
4          for (int i = 0; i < 2; i++) {
5              next[i] = nullptr;
6          }
7      }

```

```

8     Node *next[2];
9 };
10 class Trie {
11 public:
12     Node *root;
13     Trie() {
14         root = new Node();
15     }
16
17     void insert(int x) {
18         Node *p = root;
19         for (int i = 30; i >= 0; --i) {
20             int ind = !(x & (1 << i));
21             if (p->next[ind] == nullptr) p->next[ind] = new Node();
22             p = p->next[ind];
23         }
24         return ;
25     }
26
27     int search(int x) {
28         Node *p = root;
29         int ans = 0;
30         for (int i = 30; i >= 0; --i) {
31             int ind = !(x & (1 << i));
32             if (p->next[ind]) {
33                 ans |= (1 << i);
34                 p = p->next[ind];
35             } else {
36                 p = p->next[ind];
37             }
38         }
39         return ans;
40     }
41 };
42 class Solution {
43 public:
44     int findMaximumXOR(vector<int>& nums) {
45         Trie tree;
46         for (auto x : nums) tree.insert(x);
47         int ans = 0;
48         for (auto x : nums) ans = max(ans, tree.search(x));
49         return ans;
50     }
51 };

```

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]`）。

```
1 class Node {  
2     public int val;  
3     public List<Node> neighbors;  
4 }
```

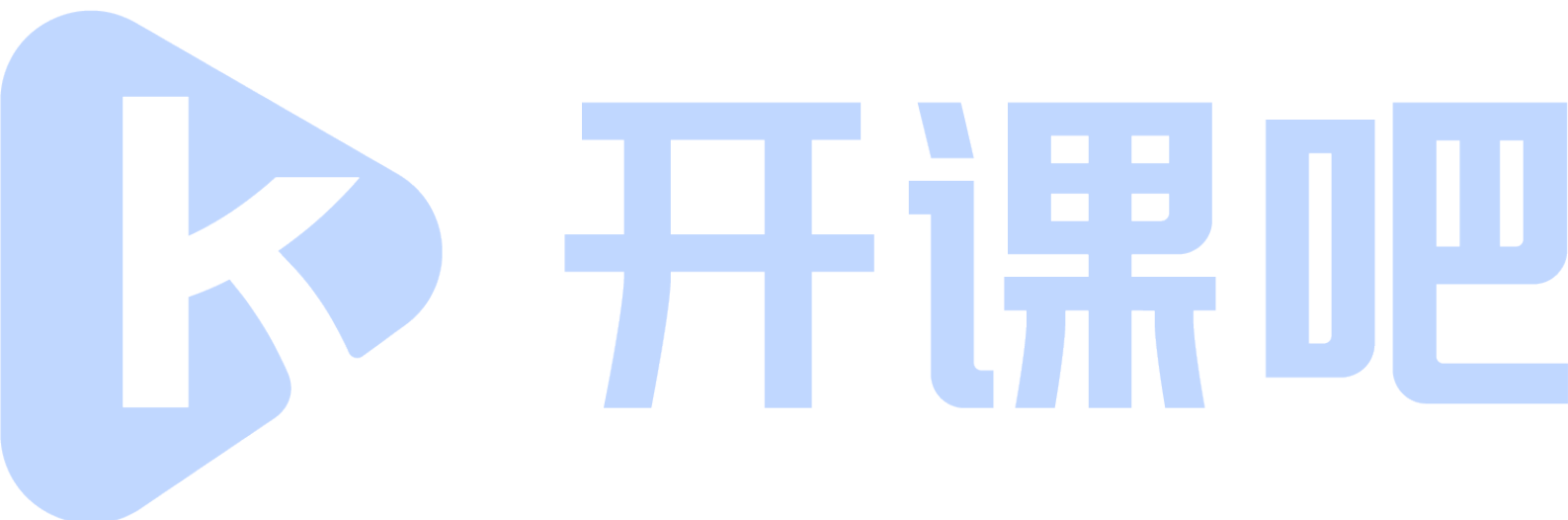
测试用例格式：

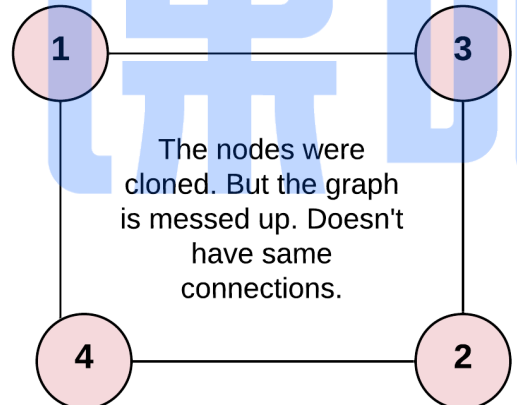
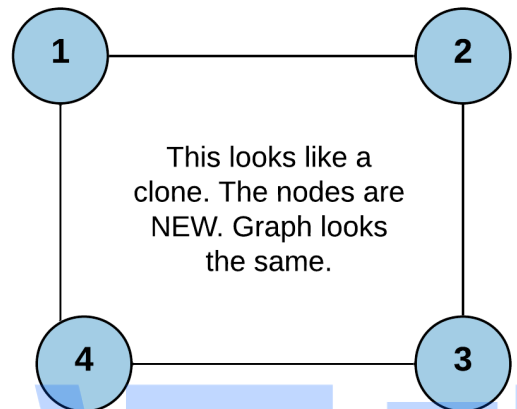
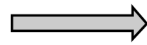
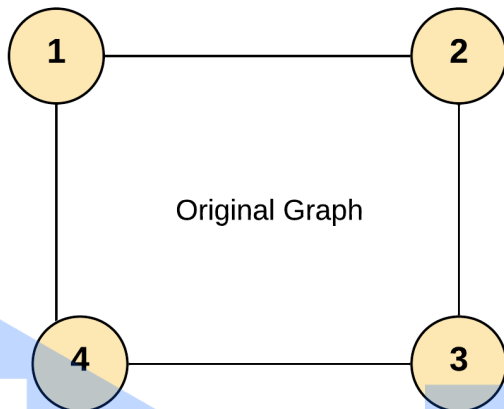
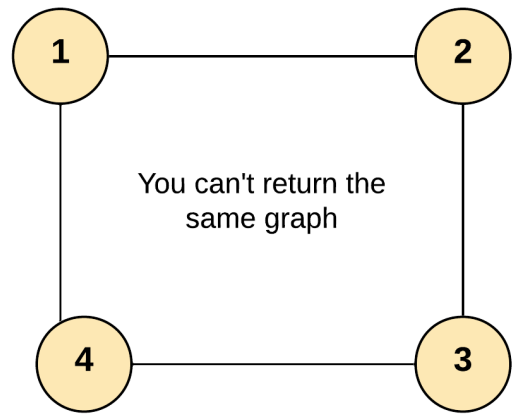
简单起见，每个节点的值都和它的索引相同。例如，第一个节点值为 1（`val = 1`），第二个节点值为 2（`val = 2`），以此类推。该图在测试用例中使用邻接列表表示。

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:





```

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

```

```

1  /*
2  // Definition for a Node.
3  class Node {
4  public:
5      int val;
6      vector<Node*> neighbors;
7      Node() {

```



```

8         val = 0;
9         neighbors = vector<Node*>();
10    }
11    Node(int _val) {
12        val = _val;
13        neighbors = vector<Node*>();
14    }
15    Node(int _val, vector<Node*> _neighbors) {
16        val = _val;
17        neighbors = _neighbors;
18    }
19 };
20 */
21
22 class Solution {
23 public:
24     unordered_map<Node *, Node *> h;
25     Node *getResult(Node *node) {
26         if (node == nullptr) return nullptr;
27         if (h[node]) return h[node];
28         h[node] = new Node(node->val);
29         for (auto x : node->neighbors) {
30             getResult(x);
31             h[node]->neighbors.push_back(h[x]);
32         }
33         return h[node];
34     }
35     Node* cloneGraph(Node* node) {
36         h.clear();
37         return getResult(node);
38     }
39 };

```

987. 二叉树的垂序遍历

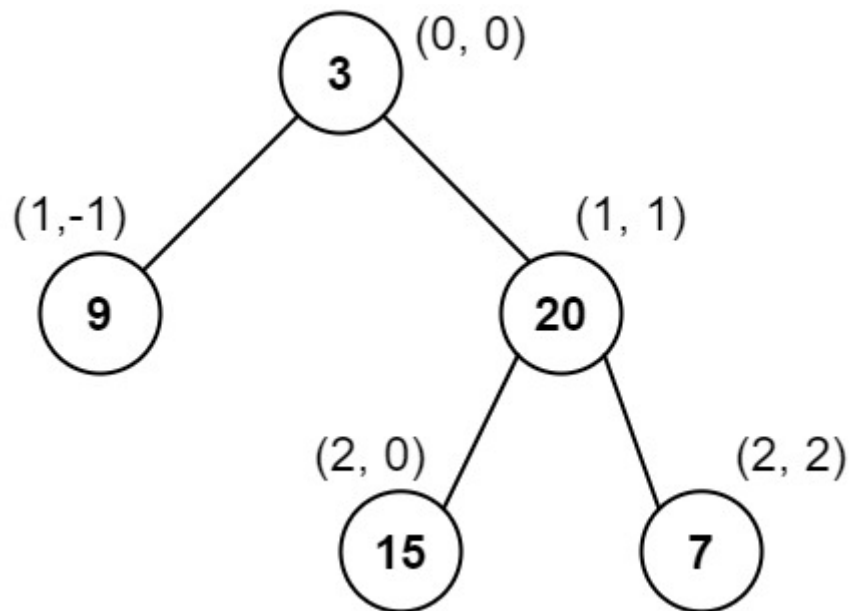
给你二叉树的根结点 `root`，请你设计算法计算二叉树的 垂序遍历** 序列。

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:



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

```
1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8   *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9   *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
10  * };
11  */
12  class Solution {
13  public:
14      typedef pair<int, int> PII;
15      map<int, vector<PII>> h;
16      void getResult(TreeNode *root, int i, int j) {
17          if (root == nullptr) return ;
18          h[j].push_back(PII(i, root->val));
19          getResult(root->left, i + 1, j - 1);
20          getResult(root->right, i + 1, j + 1);
21          return ;
22      }
23      vector<vector<int>> verticalTraversal(TreeNode* root) {
24          h.clear();
25          getResult(root, 0, 0);
26          vector<vector<int>> ret;
```

```
27     for (auto item : h) {
28         vector<PII> &arr = item.second;
29         sort(arr.begin(), arr.end());
30         vector<int> temp;
31         for (auto x : arr) temp.push_back(x.second);
32         ret.push_back(temp);
33     }
34     return ret;
35 }
36 };
```

