# LeetCode 1365. How Many Numbers Are Smaller Than the Current Number

1365. 有多少小于当前数字的数字

难度 简单 6

给你一个数组 `nums`，对于其中每个元素 `nums[i]`，请你统计数组中比它小的所有数字的数目。

换而言之，对于每个 `nums[i]` 你必须计算出有效的 `j` 的数量，其中 `j` 满足 `j != i` 且 `nums[j] < nums[i]` 。

以数组形式返回答案。

**示例 1：**

```
输入：nums = [8,1,2,2,3]
输出：[4,0,1,1,3]
解释：
对于 nums[0]=8 存在四个比它小的数字：（1，2，2 和 3）。
对于 nums[1]=1 不存在比它小的数字。
对于 nums[2]=2 存在一个比它小的数字：（1）。
对于 nums[3]=2 存在一个比它小的数字：（1）。
对于 nums[4]=3 存在三个比它小的数字：（1，2 和 2）。
```

**示例 2：**

```
输入：nums = [6,5,4,8]
输出：[2,1,0,3]
```

**示例 3：**

```
输入：nums = [7,7,7,7]
输出：[0,0,0,0]
```

**提示：**

- `2 <= nums.length <= 500`
- `0 <= nums[i] <= 100`

```
class Solution {
public:
    vector<int> smallerNumbersThanCurrent(vector<int>& nums) {
        vector<int> res(nums.size());
        for(int i = 0;i < nums.size();i ++)
        {
            for(int x : nums)
```

```
            {
                if(nums[i] > x)
                    res[i] ++;
            }
        }
        return res;
    }

};
```

# LeetCode 1366. Rank Teams by Votes !!!

1366. 通过投票对团队排名

难度 中等 11

现在有一个特殊的排名系统，依据参赛团队在投票人心中的次序进行排名，每个投票者都需要按从高到低的顺序对参与排名的所有团队进行排位。

排名规则如下：

- 参赛团队的排名次序依照其所获「排位第一」的票的多少决定。如果存在多个团队并列的情况，将继续考虑其「排位第二」的票的数量。以此类推，直到不再存在并列的情况。
- 如果在考虑完所有投票情况后仍然出现并列现象，则根据团队字母的字母顺序进行排名。

给你一个字符串数组 `votes` 代表全体投票者给出的排位情况，请你根据上述排名规则对所有参赛团队进行排名。

请你返回能表示按排名系统 **排序后** 的所有团队排名的字符串。

**示例 1：**

**输入：** votes = ["ABC","ACB","ABC","ACB","ACB"]
**输出：** "ACB"
**解释：** A 队获得五票「排位第一」，没有其他队获得「排位第一」，所以 A 队排名第一。
B 队获得两票「排位第二」，三票「排位第三」。
C 队获得三票「排位第二」，两票「排位第三」。
由于 C 队「排位第二」的票数较多，所以 C 队排第二，B 队排第三。

**示例 2：**

**输入：** votes = ["WXYZ","XYZW"]
**输出：** "XWYZ"
**解释：** X 队在并列僵局打破后成为排名第一的团队。X 队和 W 队的「排位第一」票数一样，但是 X 队有一票「排位第二」，而 W 没有获得「排位第二」。

```
class Solution {
public:
```

```
    string rankTeams(vector<string>& votes) {
        int n = votes[0].size();
        vector<vector<int>> ranks(26, vector<int>(n + 1));//二维数组，每个同学在各种排名下的次数。ranks[i][n]   ranks[选手][排名]

        for (int i = 0; i < 26; i ++ ) {
            ranks[i][n] = -i;
        }

        for (auto &vote : votes)
            for (int i = 0; i < n; i ++ )
                ranks[vote[i] - 'A'][i] ++ ;

        sort(ranks.begin(), ranks.end(), greater<vector<int>>());//从小到大排序。

        string res;
        for (int i = 0; i < n; i ++ ) res += -ranks[i][n] + 'A';
        return res;
    }
};
```

## LeetCode 1367. Linked List in Binary Tree

1367. 二叉树中的列表

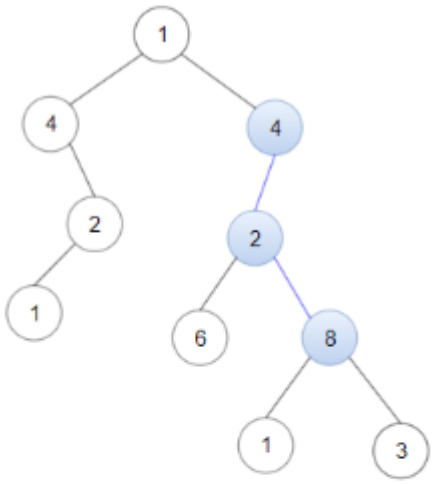
难度 中等 9

给你一棵以 `root` 为根的二叉树和一个 `head` 为第一个节点的链表。

如果在二叉树中，存在一条一直向下的路径，且每个点的数值恰好一一对应以 `head` 为首的链表中每个节点的值，那么请你返回 `True` ，否则返回 `False` 。

一直向下的路径的意思是：从树中某个节点开始，一直连续向下的路径。

**示例 1：**




**输入：** head = [4,2,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,3]
**输出：** true
**解释：** 树中蓝色的节点构成了与链表对应的子路径。

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
//枚举起点，然后往下继续找。
class Solution {
public:
```

```
    bool isSubPath(ListNode* head, TreeNode* root) {
        if (dfs(head, root)) return true;

        if (!root) return false;
        return isSubPath(head, root->left) || isSubPath(head, root->right);  //
枚举起点
    }

    bool dfs(ListNode* cur, TreeNode* root) {  // 起点已经固定，匹配的过程，从当前点匹
配能不能成功
        if (!cur) return true;//链表已经匹配完。
        if (!root) return false;//匹配不到。

        if (cur->val != root->val) return false;

        return dfs(cur->next, root->left) || dfs(cur->next, root->right);
    }
};
```