总结

1. 整体还行，最后困难题没第一时间想到动规，浪费了点时间写bfs

1455. 检查单词是否为句中其他单词的前缀（简单）

---

1. 题目描述

   给你一个字符串 `sentence` 作为句子并指定检索词为 `searchWord` ，其中句子由若干用 **单个空格** 分隔的单词组成。

   请你检查检索词 `searchWord` 是否为句子 `sentence` 中任意单词的前缀。

   - 如果 `searchWord` 是某一个单词的前缀，则返回句子 `sentence` 中该单词所对应的下标（**下标从 1 开始**）。
   - 如果 `searchWord` 是多个单词的前缀，则返回匹配的第一个单词的下标（**最小下标**）。
   - 如果 `searchWord` 不是任何单词的前缀，则返回 **-1** 。

   字符串 `S` 的「前缀」是 `S` 的任何前导连续子字符串。

   示例 1：

   ```
   输入：sentence = "i love eating burger", searchWord = "burg"
   输出：4
   解释："burg" 是 "burger" 的前缀，而 "burger" 是句子中第 4 个单词。
   ```

   示例 2：

   ```
   输入：sentence = "this problem is an easy problem", searchWord = "pro"
   输出：2
   解释："pro" 是 "problem" 的前缀，而 "problem" 是句子中第 2 个也是第 6 个单词，但是应该返回最小下标 2 。
   ```

   示例 3：

   ```
   输入：sentence = "i am tired", searchWord = "you"
   输出：-1
   解释："you" 不是句子中任何单词的前缀。
   ```

   示例 4：

   ```
   输入：sentence = "i use triple pillow", searchWord = "pill"
   输出：4
   ```

   示例 5：

```
输入：sentence = "hello from the other side", searchWord = "they"
输出：-1
```

提示：

- `1 <= sentence.length <= 100`
- `1 <= searchWord.length <= 10`
- `sentence` 由小写英文字母和空格组成。
- `searchWord` 由小写英文字母组成。
- 前缀就是紧密附着于词根的语素，中间不能插入其它成分，并且它的位置是固定的——-位于词根之前。（引用自 前缀_百度百科 ）

2. 比赛实现

一次遍历匹配

```
class Solution {
public:
    int isPrefixOfWord(string sentence, string searchWord) {
        int len1 = sentence.size();
        int len2 = searchWord.size
        int idx = 0;//遍历sentence的下标
        int ans = 1;//遍历到sentence的第几个单词
        while(idx < len1){
            if(sentence[idx] == searchWord[0]){
                int cur = 0;//匹配searchWord
                while(idx < len1 && cur < len2 &&
sentence[idx]==searchWord[cur]){
                    idx++;
                    cur++;
                }
                if(cur >= len2)//匹配上前缀
                    return ans;
            }
            while(idx < len1 && sentence[idx] != ' ')//找到下一个单词
                idx++;
            idx++;
            ans++;
        }
        return -1;
    }
};
```

1456. 定长子串中元音的最大数目（中等）

1. 题目描述

   给你字符串 `s` 和整数 `k` 。

   请返回字符串 `s` 中长度为 `k` 的单个子字符串中可能包含的最大元音字母数。

   英文中的 **元音字母** 为（`a`，`e`，`i`，`o`，`u`）。

   示例 1：

   ```
   输入：s = "abciiidef", k = 3
   输出：3
   解释：子字符串 "iii" 包含 3 个元音字母。
   ```

   示例 2：

   ```
   输入：s = "aeiou", k = 2
   输出：2
   解释：任意长度为 2 的子字符串都包含 2 个元音字母。
   ```

   示例 3：

   ```
   输入：s = "leetcode", k = 3
   输出：2
   解释："lee"、"eet" 和 "ode" 都包含 2 个元音字母。
   ```

   示例 4：

   ```
   输入：s = "rhythms", k = 4
   输出：0
   解释：字符串 s 中不含任何元音字母。
   ```

   示例 5：

   ```
   输入：s = "tryhard", k = 4
   输出：1
   ```

   提示：

   - `1 <= s.length <= 10^5`
   - `s` 由小写英文字母组成
   - `1 <= k <= s.length`

2. 比赛实现——滑动窗口

   ```cpp
   class Solution {
   public:
       int maxVowels(string s, int k) {
   ```

```
            int len = s.size();
            unordered_set<char> dic = {'a', 'e', 'i', 'o', 'u'};
            int l = 0, r = 0;
            int ans = 0;
            while(r < len && r < k){
                if(dic.find(s[r]) != dic.end())
                    ans++;
                r++;
            }
            if(ans == k)
                    return ans;
            int cnt = ans;
            while(r < len){
                if(dic.find(s[r]) != dic.end())
                    cnt++;
                if(dic.find(s[l]) != dic.end())
                    cnt--;
                ans = max(ans, cnt);
                l++;
                r++;
                if(ans == k)
                    return ans;
            }
            return ans;
        }
    };
```

1457. 二叉树中的伪回文路径（中等）

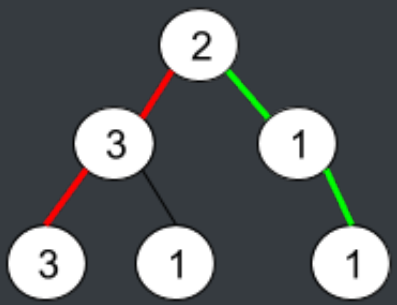
---

1. 题目描述

   给你一棵二叉树，每个节点的值为 1 到 9 。我们称二叉树中的一条路径是 「**伪回文**」的，当它满足：路径经过的所有节点值的排列中，存在一个回文序列。

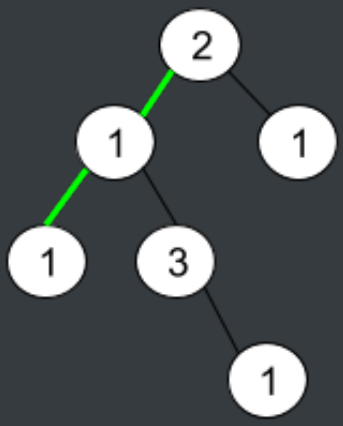
   请你返回从根到叶子节点的所有路径中 **伪回文** 路径的数目。

   示例 1：



输入：root = [2,3,1,3,1,null,1]
输出：2
解释：上图为给定的二叉树。总共有 3 条从根到叶子的路径：红色路径 [2,3,3] ，绿色路径 [2,1,1] 和路径 [2,3,1] 。
在这些路径中，只有红色和绿色的路径是伪回文路径，因为红色路径 [2,3,3] 存在回文排列 [3,2,3] ，绿色路径 [2,1,1] 存在回文排列 [1,2,1] 。

示例 2：



输入：root = [2,1,1,1,3,null,null,null,null,null,1]
输出：1
解释：上图为给定二叉树。总共有 3 条从根到叶子的路径：绿色路径 [2,1,1] ，路径 [2,1,3,1] 和路径 [2,1] 。
这些路径中只有绿色路径是伪回文路径，因为 [2,1,1] 存在回文排列 [1,2,1] 。

示例 3：

输入：root = [9]
输出：1

提示：

- 给定二叉树的节点数目在 `1` 到 `10^5` 之间。
- 节点值在 `1` 到 `9` 之间。

2. 比赛实现——DFS

判读伪回文路径的方法：至多有一个数出现奇数次

```
class Solution {
public:
    int ans = 0;
    bool judge(vector<int>& cnt){//判断是否为伪回文路径
        bool f = false;
        for(int i = 1; i <= 9; i++){
            if(cnt[i] % 2 == 1){
```

```
                    if(f) return false;
                    f = true;
                }
            }
            return true;
        }
        void dfs(TreeNode* root, vector<int>& cnt){
            if(!root->left && !root->right){
                if(judge(cnt))
                    ans++;
                return;
            }
            if(root->left){
                cnt[root->left->val]++;
                dfs(root->left, cnt);
                cnt[root->left->val]--;
            }
            if(root->right){
                cnt[root->right->val]++;
                dfs(root->right, cnt);
                cnt[root->right->val]--;
            }
        }
        int pseudoPalindromicPaths (TreeNode* root) {
            vector<int> cnt(10, 0);
            cnt[root->val]++;
            dfs(root, cnt);
            return ans;
        }
    };
```

1458. 两个子序列的最大点积（困难）

---

1. 题目描述

   给你两个数组 `nums1` 和 `nums2` 。

   请你返回 `nums1` 和 `nums2` 中两个长度相同的 **非空** 子序列的最大点积。

   数组的非空子序列是通过删除原数组中某些元素（可能一个也不删除）后剩余数字组成的序列，但不能改变数字间相对顺序。比方说， `[2,3,5]` 是 `[1,2,3,4,5]` 的一个子序列而 `[1,5,3]` 不是。

   示例 1：

```
输入：nums1 = [2,1,-2,5], nums2 = [3,0,-6]
输出：18
解释：从 nums1 中得到子序列 [2,-2] ，从 nums2 中得到子序列 [3,-6] 。
它们的点积为 (2*3 + (-2)*(-6)) = 18 。
```

示例 2：

```
输入：nums1 = [3,-2], nums2 = [2,-6,7]
输出：21
解释：从 nums1 中得到子序列 [3] ，从 nums2 中得到子序列 [7] 。
它们的点积为 (3*7) = 21 。
```

示例 3：

```
输入：nums1 = [-1,-1], nums2 = [1,1]
输出：-1
解释：从 nums1 中得到子序列 [-1] ，从 nums2 中得到子序列 [1] 。
它们的点积为 -1 。
```

提示：

- `1 <= nums1.length, nums2.length <= 500`
- `-1000 <= nums1[i], nums2[i] <= 100`

2. 比赛实现

动态规划，`dp[i][j]` 表示 `nums1[0...i]` 与 `nums2[0...j]` 的非空子序列最大点积，则状态转移为：

`dp[i][j] = max(dp[i][j-1], dp[i-1][j], dp[i-1][j-1] + nums1[i]*nums2[j], nums1[i]*nums2[j])`，max函数里依次表示

1. 只可能选取nums1[i]
2. 只可能选取nums2[j]
3. 两个数都选
4. 选且只选这两个数

要注意的是，情况1和2已经包含了两个数都不选的情况，之所以要单独列出情况4，是为了防止之前累加了负数，而现在出现了正数，例如 `nums1=[-3,-8,3], nums2=[9,2,3]` 在遍历到 `nums1[2]` 之前的情况

```cpp
class Solution {
public:
    int maxDotProduct(vector<int>& nums1, vector<int>& nums2) {
        int l1 = nums1.size();
        int l2 = nums2.size();
        vector<vector<int>> dp(l1, vector<int>(l2));
        //初始化选择单个元素的情况
```

```
        dp[0][0] = nums1[0] * nums2[0];
        for(int i = 1; i < l2; i++)
            dp[0][i] = max(dp[0][i-1], nums1[0] * nums2[i]);
        for(int i = 1; i < l1; i++)
            dp[i][0] = max(dp[i-1][0], nums1[i] * nums2[0]);
        //dp
        for(int i = 1; i < l1; i++)
            for(int j = 1; j < l2; j++){
                dp[i][j] = max(max(dp[i-1][j-1], dp[i-1][j]), dp[i][j-
1]);
                dp[i][j] = max(dp[i][j], dp[i-1][j-
1]+nums1[i]*nums2[j]);
                dp[i][j] = max(dp[i][j], nums1[i]*nums2[j]);
            }
        return dp[l1-1][l2-1];
    }
};
```