总结

1. 本此周赛比较简单，五十分钟完成，前三道题都是暴力模拟就能过，第四道题就正好开窍的想到动态规划，还总结出了规律

1408. 数组中的字符串匹配（简单）

1. 题目描述

   给你一个字符串数组 `words` ，数组中的每个字符串都可以看作是一个单词。请你按 **任意** 顺序返回 `words` 中是其他单词的子字符串的所有单词。

   如果你可以删除 `words[j]` 最左侧和/或最右侧的若干字符得到 `word[i]` ，那么字符串 `words[i]` 就是 `words[j]` 的一个子字符串。

   示例 1：

   ```
   输入：words = ["mass","as","hero","superhero"]
   输出：["as","hero"]
   解释："as" 是 "mass" 的子字符串，"hero" 是 "superhero" 的子字符串。
   ["hero","as"] 也是有效的答案。
   ```

   示例 2：

   ```
   输入：words = ["leetcode","et","code"]
   输出：["et","code"]
   解释："et" 和 "code" 都是 "leetcode" 的子字符串。
   ```

   示例 3：

   ```
   输入：words = ["blue","green","bu"]
   输出：[]
   ```

   提示：

   - `1 <= words.length <= 100`
   - `1 <= words[i].length <= 30`
   - `words[i]` 仅包含小写英文字母。
   - 题目数据 **保证** 每个 `words[i]` 都是独一无二的。

2. 比赛实现

   看数据规模后直接暴力循环O(n^2)

   ```cpp
   class Solution {
   public:
       vector<string> stringMatching(vector<string>& words) {
           vector<string> ans;
   ```

```
        unordered_set<string> s;//set暂存答案去重
        for(int i = 0; i < words.size(); i++){
            for(int j = 0; j < words.size(); j++){
                if(i == j) continue;
                if(words[i].find(words[j]) != words[i].npos)//j是i的子串
                    s.insert(words[j]);//可能出现重复，所以先用set存
            }
        }
        for(auto it = s.begin(); it != s.end(); it++)
            ans.push_back(*it);
        return ans;
    }
};
```

3. 更优解法——两次遍历

第一次遍历数组拼接成一个长字符串S 第二次遍历数组查找字符串在S中出现的位置，如果是别人的子串，那么在S中的出现次数一定 >= 2，那么起始跟结束的位置索引一定是不一样的，如果一样说明不是子串。 注意：为了避免前一字符串的尾部与后一字符串的头部构成混淆项，各字符串用,隔开拼接。

```
class Solution {
    public List<String> stringMatching(String[] words) {
        List<String> list = new ArrayList<>();
        if (words.length == 0) return list;
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < words.length; i++){
            String str = words[i];
            builder.append(str + ",");
        }

        for (int i = 0; i < words.length; i++){
            String str = words[i];
            if (builder.toString().indexOf(str) !=
builder.toString().lastIndexOf(str)) list.add(str);
        }
        return list;
    }
}
```

1409. 查询带键的排列（中等）

1. 题目描述

给你一个待查数组 queries ， 数组中的元素为 1 到 m 之间的正整数。 请你根据以下规则处理所有待查项 queries[i]（从 i=0 到 i=queries.length-1）：

- 一开始，排列 P=[1,2,3,...,m]。
- 对于当前的 i ，请你找出待查项 queries[i] 在排列 P 中的位置（下标从 0 开始），然后将其从原位置移动到排列 P 的起始位置（即下标为 0 处）。注意， queries[i] 在 P 中的位置就是 queries[i] 的查询结果。

请你以数组形式返回待查数组 queries 的查询结果。

```
示例 1:
输入: queries = [3,1,2,1], m = 5
输出: [2,1,2,1]
解释: 待查数组 queries 处理如下:
对于 i=0: queries[i]=3, P=[1,2,3,4,5], 3 在 P 中的位置是 2, 接着我们把 3 移动到 P 的起始位
置, 得到 P=[3,1,2,4,5] 。
对于 i=1: queries[i]=1, P=[3,1,2,4,5], 1 在 P 中的位置是 1, 接着我们把 1 移动到 P 的起始位
置, 得到 P=[1,3,2,4,5] 。
对于 i=2: queries[i]=2, P=[1,3,2,4,5], 2 在 P 中的位置是 2, 接着我们把 2 移动到 P 的起始位
置, 得到 P=[2,1,3,4,5] 。
对于 i=3: queries[i]=1, P=[2,1,3,4,5], 1 在 P 中的位置是 1, 接着我们把 1 移动到 P 的起始位
置, 得到 P=[1,2,3,4,5] 。
因此, 返回的结果数组为 [2,1,2,1] 。

示例 2:
输入: queries = [4,1,2,2], m = 4
输出: [3,1,2,0]

示例 3:
输入: queries = [7,5,5,8,3], m = 8
输出: [6,5,0,7,5]
```

提示:

- 1 <= m <= 10^3
- 1 <= queries.length <= m
- 1 <= queries[i] <= m

2. 比赛实现

暴力模拟O(m^2)，利用Map记录各个数的索引可以稍微降低复杂度，但是量级不会变，所以先不写了

```cpp
class Solution {
public:
    vector<int> processQueries(vector<int>& queries, int m) {
        vector<int> p(m);
        for(int i = 0; i < m; i++)
            p[i] = i+1;
        vector<int> ans(queries.size());
        for(int i = 0; i < queries.size(); i++){
            for(int j = 0; j < m; j++){//查找queries[i]
                if(p[j] == queries[i]){//找到
                    ans[i] = j;
                    p.erase(p.begin() + j);//去掉
                    p.insert(p.begin(), queries[i]);//插在第一个
                    break;
                }
            }
        }
        return ans;
    }
};
```

1410. HTML实体解析器（中等）

1. 题目描述

   「HTML 实体解析器」 是一种特殊的解析器，它将 HTML 代码作为输入，并用字符本身替换掉所有这些特殊的字符实体。

   HTML 里这些特殊字符和它们对应的字符实体包括：

   - **双引号**：字符实体为 `&quot;` ，对应的字符是 `"` 。
   - **单引号**：字符实体为 `&apos;` ，对应的字符是 `'` 。
   - **与符号**：字符实体为 `&amp;` ，对应对的字符是 `&` 。
   - **大于号**：字符实体为 `&gt;` ，对应的字符是 `>` 。
   - **小于号**：字符实体为 `&lt;` ，对应的字符是 `<` 。
   - **斜线号**：字符实体为 `&frasl;` ，对应的字符是 `/` 。

   给你输入字符串 `text` ，请你实现一个 HTML 实体解析器，返回解析器解析后的结果。

   示例 1：

   ```
   输入：text = "&amp;amp; is an HTML entity but &amp;ambassador; is not."
   输出："& is an HTML entity but &amp;ambassador; is not."
   解释：解析器把字符实体 &amp;amp; 用 & 替换
   ```

   示例 2：

   ```
   输入：text = "and I quote: &amp;quot;...&amp;quot;"
   输出："and I quote: \"...\""
   ```

   示例 3：

   ```
   输入：text = "Stay home! Practice on Leetcode :)"
   输出："Stay home! Practice on Leetcode :)"
   ```

   示例 4：

   ```
   输入：text = "x &amp;gt; y &amp;amp;&amp;amp; x &amp;lt; y is always false"
   输出："x > y && x < y is always false"
   ```

   示例 5：

   ```
   输入：text = "leetcode.com&amp;frasl;problemset&amp;frasl;all"
   输出："leetcode.com/problemset/all"
   ```

   提示：

   - `1 <= text.length <= 10^5`
   - 字符串可能包含 256 个ASCII 字符中的任意字符。

2. 比赛实现

   按照题目规则遍历替换即可

```
class Solution {
public:
    string entityParser(string text) {
        unordered_map<string, char> dic;//替换列表，不含&
        dic["quot;"] = '\"';
        dic["apos;"] = '\'';
        dic["amp;"] = '&';
        dic["gt;"] = '>';
        dic["lt;"] = '<';
        dic["frasl;"] = '/';
        string ans = "";
        int len = text.size();
        for(int i = 0; i < len; i++){
            if(text[i] == '&'){//可能发生替换
                bool changed = false;//标记是否替换
                for(auto it = dic.begin(); it != dic.end(); it++){//遍历dic
                    if(i + it->first.size() < len
                       && text.substr(i+1, it->first.size()) == it->first){//需要替
换
                        ans += it->second;//替换
                        i += it->first.size();//i前移
                        changed = true;
                        break;
                    }
                }
                if(!changed)//未替换
                    ans += text[i];
            }
            else
                ans += text[i];
        }
        return ans;
    }
};
```

其实当时没发现每个要替换的后面有分号，不然就直接找分号就能确定替换的结尾了

3. 怪咖trick

可惜C++没有这么好的api

```
class Solution {
    public String entityParser(String text) {
        return text.replace("&quot;", "\"").replace("&apos;", "'").replace("&amp;",
"&").replace("&gt;", ">").replace("&lt;", "<").replace("&frasl;", "/");
    }
}
```

1411. 给N*3网格图涂色的方案数（困难）

1. 题目描述

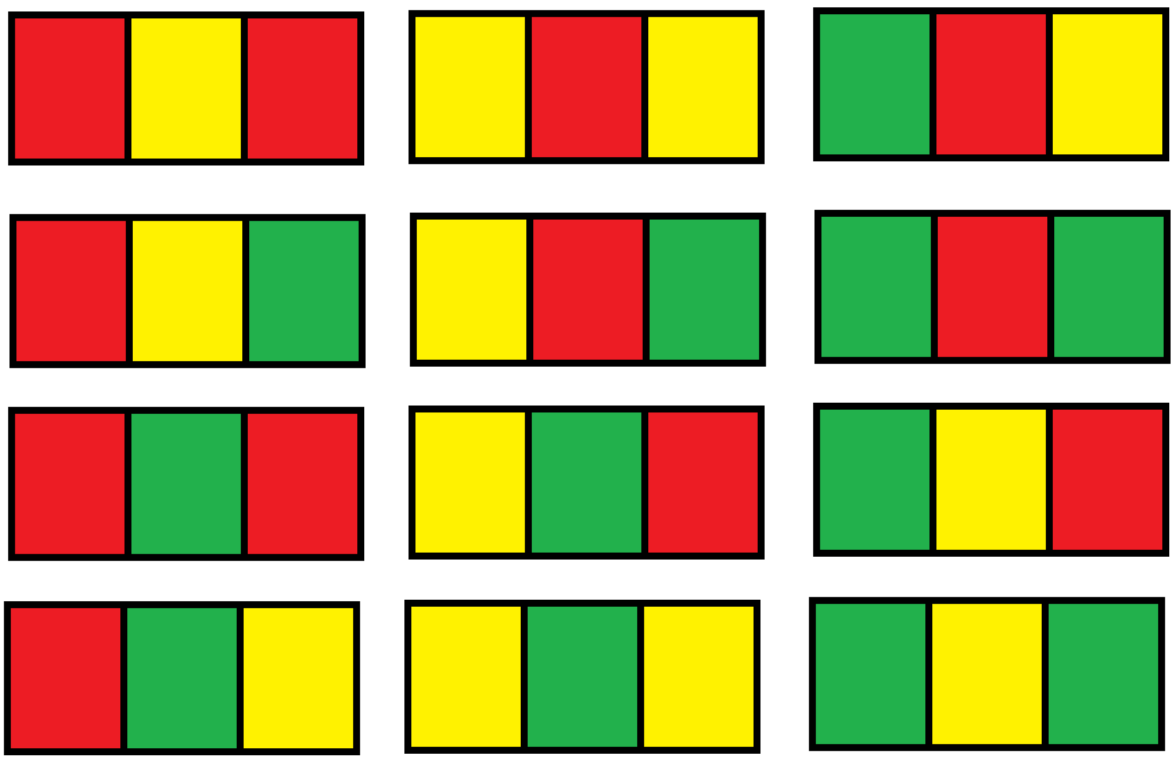
你有一个 `n x 3` 的网格图 `grid` ，你需要用 **红，黄，绿** 三种颜色之一给每一个格子上色，且确保相邻格子颜色不同（也就是有相同水平边或者垂直边的格子颜色不同）。

给你网格图的行数 `n` 。

请你返回给 `grid` 涂色的方案数。由于答案可能会非常大，请你返回答案对 `10^9 + 7` 取余的结果。

示例 1：

```
输入：n = 1
输出：12
解释：总共有 12 种可行的方法
```



示例 2：**

```
输入：n = 2
输出：54
```

示例 3：

```
输入：n = 3
输出：246
```

示例 4：

```
输入：n = 7
输出：106494
```

示例 5：

```
输入：n = 5000
输出：30228214
```

提示：

- `n == grid.length`
- `grid[i].length == 3`
- `1 <= n <= 5000`

2. 比赛实现
   - 本题一看答案很大要取余，再加上发现第i行的涂色方案实际上只受第i-1行的限制，因此想到可以用dp
   - 初始的想法是，列出第i-1行每一个可能的涂色情况（共12种情况，示例一已经给出）下，对应的第i行可以有哪些涂色方案，这样就可以推导出第`dp[i][case1..12]`与`dp[i-1][case1...12]`关系了
   - 于是开始老老实实地列举这12种情况下的下一行可能的涂色方案数，幸运地发现了新的规律：
     - 这12种涂色方案可以分为两大类：对称的（例如红/黄/红）和不对称的（例如红/黄/绿）
     - 对于对称的情况而言，它们的下一行可以有5种涂色方案，其中有三种对称的和两种不对称的，例如，红/黄/红的下一行可以是 黄/红/黄、绿/红/绿、黄/绿/黄、黄/红/绿、绿/红/黄
     - 对于不对称的情况而言，它们的下一行可以有4种涂色方案，其中有两种对称的和两种不对称的，例如，红/黄/绿的下一行可以是 黄/红/黄、黄/绿/黄、黄/绿/红、绿/红/黄
   - 因此我们只要知道i-1行网格的所有涂色方案中，最后一行为对称的方案的数量a，以及最后一行为不对称的方案的数量b，则i行网格的所有涂色方案数为`5*a+4*b`，而每次迭代a和b的变换公式为`a=3a+2b, b=2a+2b`

```cpp
class Solution {
public:
    int MOD = 1e9 + 7;
    int numOfWays(int n) {
        long a = 6, b = 6;
        long ans = 12;
        while(--n){
            ans = (5*a + 4*b) % MOD;
            int tmp_a = (3*a + 2*b) % MOD;
            b = (2*a +2*b) % MOD;
            a = tmp_a;
        }
        return ans;
    }
};
```