1175. 质数排列（简单）

1. 题目描述

   请你帮忙给从 `1` 到 `n` 的数设计排列方案，使得所有的「质数」都应该被放在「质数索引」（索引从 1 开始）上；你需要返回可能的方案总数。

   让我们一起来回顾一下「质数」：质数一定是大于 1 的，并且不能用两个小于它的正整数的乘积来表示。

   由于答案可能会很大，所以请你返回答案 **模 mod 10^9 + 7** 之后的结果即可。

   示例 1：

   ```
   输入：n = 5
   输出：12
   解释：举个例子，[1,2,5,4,3] 是一个有效的排列，但 [5,2,3,4,1] 不是，因为在第二种情况里质数 5 被错误地放在索引为 1 的位置上。
   ```

   示例 2：

   ```
   输入：n = 100
   输出：682289015
   ```

   提示：

   - `1 <= n <= 100`

2. 简单实现

   结果就是 质数个数的阶乘 * 非质数个数的阶乘

   ```
   class Solution {
   public:
       int MOD = 1e9 + 7;
       bool isPrime(int n){//判断质数
           for(int i = 2; i <= sqrt(n); i++)
               if(n%i == 0)
                   return false;
           return true;
       }
       int numPrimeArrangements(int n) {
           int cnt = 0;
           for(int i = 2; i <= n; i++)
               if(isPrime(i))
                   cnt++;
           cnt = max(cnt, n-cnt);
           long ans = 1;
           for(int i = 2; i <= cnt; i++){
               ans *= i;
               if(i <= n-cnt)
                   ans *= i;
               ans %= MOD;
   ```

```
        }
        return ans;
    }
};
```

1177. 构建回文串检测（中等）

1. 题目描述

   给你一个字符串 `s`，请你对 `s` 的子串进行检测。

   每次检测，待检子串都可以表示为 `queries[i] = [left, right, k]`。我们可以 **重新排列** 子串 `s[left], ..., s[right]`，并从中选择 **最多** `k` 项替换成任何小写英文字母。

   如果在上述检测过程中，子串可以变成回文形式的字符串，那么检测结果为 `true`，否则结果为 `false`。

   返回答案数组 `answer[]`，其中 `answer[i]` 是第 `i` 个待检子串 `queries[i]` 的检测结果。

   注意：在替换时，子串中的每个字母都必须作为 **独立的** 项进行计数，也就是说，如果 `s[left..right] = "aaa"` 且 `k = 2`，我们只能替换其中的两个字母。（另外，任何检测都不会修改原始字符串 `s`，可以认为每次检测都是独立的）

   示例：

```
输入：s = "abcda", queries = [[3,3,0],[1,2,0],[0,3,1],[0,3,2],[0,4,1]]
输出：[true,false,false,true,true]
解释：
queries[0] : 子串 = "d"，回文。
queries[1] : 子串 = "bc"，不是回文。
queries[2] : 子串 = "abcd"，只替换 1 个字符是变不成回文串的。
queries[3] : 子串 = "abcd"，可以变成回文的 "abba"。 也可以变成 "baab"，先重新排序变成 "bacd"，然后把 "cd" 替换为 "ab"。
queries[4] : 子串 = "abcda"，可以变成回文的 "abcba"。
```

   提示：

   - `1 <= s.length, queries.length <= 10^5`
   - `0 <= queries[i][0] <= queries[i][1] < s.length`
   - `0 <= queries[i][2] <= s.length`
   - `s` 中只有小写英文字母

2. 简单实现

   对于某个子字符串的判断，既然可以任意重排，因此字母出现的顺序就不重要，只需要统计每个字母出现的个数，每两个相同的字母就叫可以凑一对，最后单出来的每两个需要消耗一次替换

   为了防止统计s种字母出现重复，达到时间限制，需要开辟数组`cnt[s.size()+1][26]`进行预统计，其中`cnt[i][j]`表示s的前i个字符中字母'a'+j的个数

```
class Solution {
public:
    vector<bool> canMakePaliQueries(string s, vector<vector<int>>& queries) {
        vector<bool> ans(queries.size(), true);
        vector<vector<int>> cnt;//最终大小为[s.size()+1, 26]
        vector<int> cur(26, 0);//累加统计器
```

```
        cnt.push_back(cur);
        for(int i = 1; i <= s.size(); i++){
            cur[s[i-1]-'a']++;
            cnt.push_back(cur);
        }
        for(int i = 0; i < queries.size(); i++){
            int l = queries[i][0];
            int r = queries[i][1];
            int k = queries[i][2];
            int cur_cnt = 0;
            for(int j = 0; j < 26; j++){//依次查看子字符串中各个字母的情况
                if((cnt[r+1][j]-cnt[l][j]) % 2 == 1)//落单
                    cur_cnt++;
            }
            ans[i] = cur_cnt/2 <= k;
        }
        return ans;
    }
};
```

1178. 猜字谜（困难）

1. 题目描述

外国友人仿照中国字谜设计了一个英文版猜字谜小游戏，请你来猜猜看吧。

字谜的谜面 `puzzle` 按字符串形式给出，如果一个单词 `word` 符合下面两个条件，那么它就可以算作谜底：

- 单词 `word` 中包含谜面 `puzzle` 的第一个字母。
- 单词 `word` 中的每一个字母都可以在谜面 `puzzle` 中找到。 例如，如果字谜的谜面是 "abcdefg"，那么可以作为谜底的单词有 "faced", "cabbage", 和 "baggage"；而 "beefed"（不含字母 "a"）以及 "based"（其中的 "s" 没有出现在谜面中）则不能作为谜底。

返回一个答案数组 `answer`，数组中的每个元素 `answer[i]` 是在给出的单词列表 `words` 中可以作为字谜迷面 `puzzles[i]` 所对应的谜底的单词数目。

示例：

```
输入：
words = ["aaaa","asas","able","ability","actt","actor","access"],
puzzles = ["aboveyz","abrodyz","abslute","absoryz","actresz","gaswxyz"]
输出：[1,1,3,2,4,0]
解释：
1 个单词可以作为 "aboveyz" 的谜底 : "aaaa"
1 个单词可以作为 "abrodyz" 的谜底 : "aaaa"
3 个单词可以作为 "abslute" 的谜底 : "aaaa", "asas", "able"
2 个单词可以作为 "absoryz" 的谜底 : "aaaa", "asas"
4 个单词可以作为 "actresz" 的谜底 : "aaaa", "asas", "actt", "access"
没有单词可以作为 "gaswxyz" 的谜底，因为列表中的单词都不含字母 'g'。
```

提示：

- `1 <= words.length <= 10^5`
- `4 <= words[i].length <= 50`
- `1 <= puzzles.length <= 10^4`
- `puzzles[i].length == 7`
- `words[i][j]`, `puzzles[i][j]` 都是小写英文字母。
- 每个 `puzzles[i]` 所包含的字符都不重复。

2. 正确解法

- 仅考虑包含关系，因此字符串内字母顺序不重要，且只关心每个字母是否出现而不关心出现次数，因此，可以用二进制表示某个字符串的模式数，例如，aaaa表示为1，ac表示为101
- 集合的运算：`A&B=A` 表示A是B的子集
- 子集的遍历：**用 `for(int i=A; i; i=(i-1)&A)` 可以快速遍历A的子集，此处非常巧妙，值得思考记住**

```
class Solution {
public:
    vector<int> findNumOfValidWords(vector<string> &words, vector<string> &puzzles)
{
        int n = puzzles.size();
        vector<int>ret;
        int wn = words.size();
        unordered_map<int, int>count;//words中所有word产生的模式数的数量
        for (auto word : words) {//统计words
            int num = 0;//模式数
            for (auto c : word)
                num |= 1 << c - 'a';
            count[num]++;
        }
        for (auto p : puzzles) {//遍历puzzles统计符合条件的word数量
            int num = 0;//模式数
            for (auto c : p)
                num |= 1 << c - 'a';
            auto first_num = 1 << p[0] - 'a';
            int cnt = 0;
            for (int j = num; j; j = (j - 1)&num)//遍历num的子集
                if ((j&first_num) && count.count(j))//满足题目要求的两个条件
                    cnt += count[j];
            ret.push_back(cnt);
        }
        return ret;
    }
};
```