2022.11.8

每日一题1684 统计一致的字符串的数目

简单题，构造哈希表储存后直接比较遍历即可，没啥好说的。

```
class Solution {
private:
    bool hash[28];
public:
    int countConsistentStrings(string allowed, vector<string>& words) {
        for(auto ch :allowed)
            hash[ch-'a'] = true;
        int ans = 0;
        for(auto str : words) {
            bool isright = true;
                for(auto ch :str)
                    if(!hash[ch-'a']) {isright = false; break;}
            if(isright) ans += 1;
        }
        return ans;
    }
};
```

就着哈希我们可以聊点别的。

- 首先就是因为哈希表只储存26个英文小写字母，因此可以直接用字符串哈希的思想，将每一个字母对应到每一个二进制位，然后再去判断。这也是状压DP的基本思想。

  ```
  class Solution {
  public:
      int countConsistentStrings(string allowed, vector<string>& words) {
          int mask = 0;
          for (auto c : allowed)
              mask |= 1 << (c - 'a');
          int res = 0;
          for (auto &&word : words) {
              int mask1 = 0;
              for (auto c : word)
                  mask1 |= 1 << (c - 'a');
              if ((mask1 | mask) == mask) res++;
          }
          return res;
      }
  };
  ```

- 能不能再短点？来点高科技——

```
class Solution {
public:
    int countConsistentStrings(string_view a, vector<string>& words) {
        return count_if(begin(words), end(words), [&](auto&& s){return
s.find_first_not_of(a) == -1;});
    }
};
```

1. 首先对 $count\_if$ 给出作用：返回即找出容器内满足条件的元素个数。

   ```
   int std::count_if(a.begin(), a.end(), [&](int n) {return n > 3; });
   ```

   即找出容器 a 内满足大于3的元素。

2. 然后对 $find\_first\_not\_of(const\ char\ \&str)$ 给出作用：

   返回在字符串中首次出现的不匹配 str 中的任何一个字符的首字符索引，， 如果全部匹配则返回$string :: npos$。

- 也可以小学一下 $python$ 的写法：

```
class Solution:
    def countConsistentStrings(self, allowed: str, words: List[str]) -> int:
        mask = 0
        for c in allowed:
            mask |= 1 << (ord(c) - ord('a'))
        res = 0
        for word in words:
            mask1 = 0
            for c in word:
                mask1 |= 1 << (ord(c) - ord('a'))
            res += (mask1 | mask) == mask
        return res
```

方法 $ord(char)$ 为返回对应字符的ASCII码，对应方法 $chr(int)$ 为返回对应ASCII码下的字符。

650 只有两个键的键盘

方法一

对其状态转移进行考虑，发现影响转移的一是现在复制的内容，二是当前的长度。因此可以设计：

$DP[i][j]$ 表示达到当前字符长度为 i 且已经复制的字符串长度为 j 的状态时所用的最小步数。由于题目中只给了两种操作，因此转移方程也只有两个：

$$Paste: \quad dp[i+j][j] = min\{dp[i+j][j], dp[i][j]+1\}$$
$$CopyAll: \quad dp[i][i] = min\{dp[i][i], dp[i][j]+1\}$$

遍历后找出数组 $dp[n]$ 的最小值即可。时间复杂度 $O(n^2)$。

```
class Solution {
public:
    int minSteps(int n) {
        vector<vector<int>> dp(n + 1,vector<int>(n + 1,n + 1));
        dp[1][0] = 0;    dp[1][1] = 1;
```

```
        for(int i = 1; i < n; i++)
            for(int j = 1; j <= i; j++) {
                if(i + j <= n)  dp[i+j][j] = min(dp[i+j][j],dp[i][j] + 1);
                dp[i][i] = min(dp[i][i],dp[i][j] + 1);
            }
        int ans = n + 1;
        for(auto p : dp.back()) ans = min(ans,p);
        return ans;
    }
};
```

方法二

仔细想想不难发现，上面的DP遍历时有不少状态是空的不存在的，因此可以进一步简化算法。

如果我们将 **一次 $CopyAll + x$ 次的 $Paste$** 看做是"一个"动作的话。

那么一次动作带来的效果就是**把字符串长度变为原来的(x+1)倍。**最后的最小操作数方案可以等效为如下流程：

- 起始对长度为 1 的记事本字符进行 1次 $CopyAll$ 和 $k_1 - 1$次 $Paste$ 操作（消耗次数为 k_1，得到长度为 k_1 的字符串）
- 再对对长度为 k_1 的记事本字符进行 1次 $CopyAll$ 和 $k_2 - 1$次 $Paste$ 操作（消耗次数为 k_2，得到长度为 k_1k_2 的字符串）
- ...

最终经过 x 次动作后应该有:

$$n = k_1 * k_2 * k_3 * \cdots * k_x$$

问题转化为：如何拆分 n 使得 $k_1 + k_2 + k_3 + \cdots + k_x$ 最小。

而对于任意合数 $k = a * b$ 而言，由于 $a * b \geq a + b$ 可知拆分一定不会使结果变差。

那就对 n 使用试除法进行分解质因数操作，累加所有操作次数即为答案。时间复杂度$O(\sqrt{n})$。

```
class Solution {
public:
    int minSteps(int n) {
        int ans = 0;
        for(int i = 2; i * i <= n; i++) {
            while(n % i == 0) {
                ans += i;
                n /= i;
            }
        }
        return ans + (n != 1 ? n : 0);
    }
};
```