总结

1. 前三题难度不大，拼手速的题，发挥可以说是正常吧，因为没想到会这么简单，所以也没快做
2. 第四题后来要被自己蠢死了，没有及时想到用get()判断是否属于同一集合，各种瞎试，相当于浪费了十分钟的罚时+十分钟的思考时间，唉，不过好歹最后做出来了，也就还凑合吧

替换所有的问号（简单）

---

1. 题目描述

   给你一个仅包含小写英文字母和 `'?'` 字符的字符串 `s` ，请你将所有的 `'?'` 转换为若干小写字母，使最终的字符串不包含任何 **连续重复** 的字符。

   注意：你 **不能** 修改非 `'?'` 字符。

   题目测试用例保证 **除** `'?'` 字符 **之外**，不存在连续重复的字符。

   在完成所有转换（可能无需转换）后返回最终的字符串。如果有多个解决方案，请返回其中任何一个。可以证明，在给定的约束条件下，答案总是存在的。

   示例 1：

   ```
   输入：s = "?zs"
   输出："azs"
   解释：该示例共有 25 种解决方案，从 "azs" 到 "yzs" 都是符合题目要求的。只有 "z" 是无效的修改，因为字符串 "zzs" 中有连续重复的两个 'z' 。
   ```

   示例 2：

   ```
   输入：s = "ubv?w"
   输出："ubvaw"
   解释：该示例共有 24 种解决方案，只有替换成 "v" 和 "w" 不符合题目要求。因为 "ubvvw" 和 "ubvww" 都包含连续重复的字符。
   ```

   示例 3：

   ```
   输入：s = "j?qg??b"
   输出："jaqgacb"
   ```

   示例 4：

   ```
   输入：s = "??yw?ipkj?"
   输出："acywaipkja"
   ```

   提示：

   - `1 <= s.length <= 100`
   - `s` 仅包含小写英文字母和 `'?'` 字符

2. 比赛实现

`s[i]`只受`s[i-1]`和`s[i+1]`的影响，因此即使出现多个相邻的?，也依旧可以有很多可以解的答案，直接贪心即可

```
class Solution {
public:
    string modifyString(string s) {
        string ans = s;
        int n = ans.size();
        for(int i = 0; i < n; i++){
            if(ans[i] == '?'){
                if(i == 0){
                    for(int j = 0; j < 26; j++){
                        if('a'+j != ans[i+1]){
                            ans[i] = 'a'+j;
                            break;
                        }
                    }
                }
                else if(i == n-1){
                    for(int j = 0; j < 26; j++){
                        if('a'+j != ans[i-1]){
                            ans[i] = 'a'+j;
                            break;
                        }
                    }
                }
                else{
                    for(int j = 0; j < 26; j++){
                        if('a'+j != ans[i+1] && 'a'+j != ans[i-1]){
                            ans[i] = 'a'+j;
                            break;
                        }
                    }
                }
            }
        }
        return ans;
    }
};
```

数的平方等于两数乘积的方法数（中等（

1. 题目描述

给你两个整数数组`nums1`和`nums2`，请你返回根据以下规则形成的三元组的数目（类型 1 和类型 2 ）：

- 类型 1：三元组`(i, j, k)`，如果`nums1[i]2 == nums2[j] * nums2[k]`其中`0 <= i < nums1.length`且`0 <= j < k < nums2.length`
- 类型 2：三元组`(i, j, k)`，如果`nums2[i]2 == nums1[j] * nums1[k]`其中`0 <= i < nums2.length`且`0 <= j < k < nums1.length`

示例 1：

```
输入：nums1 = [7,4], nums2 = [5,2,8,9]
输出：1
解释：类型 1：(1,1,2), nums1[1]^2 = nums2[1] * nums2[2] (4^2 = 2 * 8)
```

示例 2：

```
输入：nums1 = [1,1], nums2 = [1,1,1]
输出：9
解释：所有三元组都符合题目要求，因为 1^2 = 1 * 1
类型 1：(0,0,1), (0,0,2), (0,1,2), (1,0,1), (1,0,2), (1,1,2), nums1[i]^2 = nums2[j]
* nums2[k]
类型 2：(0,0,1), (1,0,1), (2,0,1), nums2[i]^2 = nums1[j] * nums1[k]
```

示例 3：

```
输入：nums1 = [7,7,8,3], nums2 = [1,2,9,7]
输出：2
解释：有两个符合题目要求的三元组
类型 1：(3,0,2), nums1[3]^2 = nums2[0] * nums2[2]
类型 2：(3,0,1), nums2[3]^2 = nums1[0] * nums1[1]
```

示例 4：

```
输入：nums1 = [4,7,9,11,23], nums2 = [3,5,1024,12,18]
输出：0
解释：不存在符合题目要求的三元组
```

提示：

- `1 <= nums1.length, nums2.length <= 1000`
- `1 <= nums1[i], nums2[i] <= 10^5`

2. 比赛实现

根据数据量判断需要O(n^2)的复杂度，因此不能暴力遍历，可以采用hashmap对两个数组的所有`j<k`对的乘积结果进行计数，再遍历求结果即可，具体见代码，思路很清晰

```cpp
class Solution {
public:
    int numTriplets(vector<int>& nums1, vector<int>& nums2) {
        unordered_map<long, int> cnt1;
        for(int i = 0; i < nums1.size(); i++){
            for(int j = i+1; j < nums1.size(); j++){
                cnt1[long(nums1[i])*nums1[j]]++;
            }
        }
        unordered_map<long, int> cnt2;
        for(int i = 0; i < nums2.size(); i++){
            for(int j = i+1; j < nums2.size(); j++){
                cnt2[long(nums2[i])*nums2[j]]++;
            }
        }
```

```
        int ans = 0;
        for(int i = 0; i < nums1.size(); i++)
            ans += cnt2[long(nums1[i]) * nums1[i]];
        for(int i = 0; i < nums2.size(); i++)
            ans += cnt1[long(nums2[i]) * nums2[i]];
        return ans;
    }
};
```

避免重复字母的最小删除成本（中等）

1. 题目描述

   给你一个字符串 `s` 和一个整数数组 `cost` ，其中 `cost[i]` 是从 `s` 中删除字符 `i` 的代价。

   返回使字符串任意相邻两个字母不相同的最小删除成本。

   请注意，删除一个字符后，删除其他字符的成本不会改变。

   示例 1：

   ```
   输入：s = "abaac", cost = [1,2,3,4,5]
   输出：3
   解释：删除字母 "a" 的成本为 3，然后得到 "abac"（字符串中相邻两个字母不相同）。
   ```

   示例 2：

   ```
   输入：s = "abc", cost = [1,2,3]
   输出：0
   解释：无需删除任何字母，因为字符串中不存在相邻两个字母相同的情况。
   ```

   示例 3：

   ```
   输入：s = "aabaa", cost = [1,2,3,4,1]
   输出：2
   解释：删除第一个和最后一个字母，得到字符串 ("aba") 。
   ```

   提示：

   - `s.length == cost.length`
   - `1 <= s.length, cost.length <= 10^5`
   - `1 <= cost[i] <= 10^4`
   - `s` 中只含有小写英文字母

2. 比赛实现

   对于连续相同的`s[l...r]`，它们中只能留下一个，剩下的都要删掉，因此留下cost最大的那个才能达到最小删除成本

   ```
   class Solution {
   public:
       int minCost(string s, vector<int>& cost) {
   ```

```
        int ans = 0;
        int n = s.size();
        if(n == 1) return 0;
        int l = 0, r = 0;
        while(r < n){
            while(r < n && s[r] == s[l]) r++;
            if(r-l > 1){//超过1个连续相同的字符
                int cur_max = -1;
                int cur_sum = 0;
                while(l < r){
                    cur_max = max(cur_max, cost[l]);
                    cur_sum += cost[l];
                    l++;
                }
                ans += cur_sum - cur_max;
            }
            l = r;
        }
        return ans;
    }
};
```

保证图可完全遍历（困难）

1. 题目描述

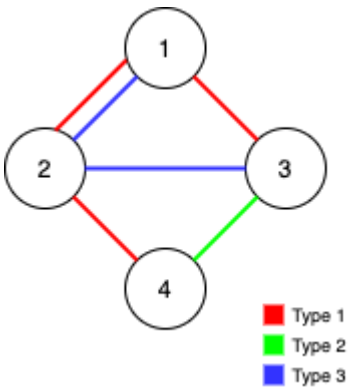
   Alice 和 Bob 共有一个无向图，其中包含 n 个节点和 3 种类型的边：

   - 类型 1：只能由 Alice 遍历。
   - 类型 2：只能由 Bob 遍历。
   - 类型 3：Alice 和 Bob 都可以遍历。

   给你一个数组 `edges` ，其中 `edges[i] = [typei, ui, vi]` 表示节点 `ui` 和 `vi` 之间存在类型为 `typei` 的双向边。请你在保证图仍能够被 Alice和 Bob 完全遍历的前提下，找出可以删除的最大边数。如果从任何节点开始，Alice 和 Bob 都可以到达所有其他节点，则认为图是可以完全遍历的。

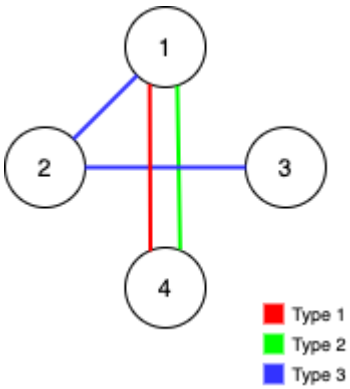
   返回可以删除的最大边数，如果 Alice 和 Bob 无法完全遍历图，则返回 -1 。

   示例 1：

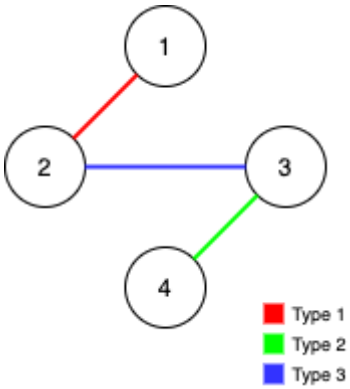

```
输入：n = 4, edges = [[3,1,2],[3,2,3],[1,1,3],[1,2,4],[1,1,2],[2,3,4]]
输出：2
解释：如果删除 [1,1,2] 和 [1,1,3] 这两条边，Alice 和 Bob 仍然可以完全遍历这个图。再删除任何其他的边都无法保证图可以完全遍历。所以可以删除的最大边数是 2 。
```

示例 2：




```
输入：n = 4, edges = [[3,1,2],[3,2,3],[1,1,4],[2,1,4]]
输出：0
解释：注意，删除任何一条边都会使 Alice 和 Bob 无法完全遍历这个图。
```

示例 3：




```
输入：n = 4, edges = [[3,2,3],[1,1,2],[2,3,4]]
输出：-1
解释：在当前图中，Alice 无法从其他节点到达节点 4 。类似地，Bob 也不能达到节点 1 。因此，图无法完全遍历。
```

提示：

- `1 <= n <= 10^5`
- `1 <= edges.length <= min(10^5, 3 * n * (n-1) / 2)`
- `edges[i].length == 3`
- `1 <= edges[i][0] <= 3`
- `1 <= edges[i][1] < edges[i][2] <= n`
- 所有元组 `(typei, ui, vi)` 互不相同

2. 比赛实现

误打误撞实现的方法，看了题解发现，这其实是在找最小生成树：

- 首先，类型3的线要尽可能保留，因为它同时服务于两个人，其次对类型1/2的分别做取舍
- 取舍的过程可以通过找两个人的最小生成树实现，并查集的方法对应的是Prime算法，依次判断各个边是否能加进去

本题最大的收获应该就是意识到并查集在最小生成树上的应用，如果之前有这个知识储备，这个题就不难了

```
class UnionFind{//并查集
private:
    vector<int> father;
    int size;
public:
    UnionFind(int n){
        father = vector<int>(n);
        size = n;
        for(int i = 0; i < n; i++)
            father[i] = i;
    }
    int get(int x){
        if(father[x] == x)
            return x;
        return father[x] = get(father[x]);//路径压缩
    }
    void merge(int x, int y){
        x = get(x);
        y = get(y);
        if(x != y){
            father[y] = x;
            size--;
        }
    }
    int getSize() {return size;}
};


class Solution {
public:
    static bool cmp(const vector<int>& v1, const vector<int>& v2) {
        return v1[0] < v2[0]; //从小到大排序
    }
    int maxNumEdgesToRemove(int n, vector<vector<int>>& edges) {
        int size = edges.size();
        sort(edges.begin(), edges.end(), cmp);//按类型排序
        UnionFind u1(n);//Alice
        UnionFind u2(n);//Bob
        int cnt = 0;//加入u1或u2的边数
        for(int i = size-1; i >= 0; i--){//逆序，保证优先处理类型3
            if(u1.getSize() == 1 && u2.getSize() == 1) break;//两者都找完了
            if(edges[i][0] == 3){//只要能服务于Alice或Bob中的任意一个人，就可以保留
                if(u1.get(edges[i][1]-1) != u1.get(edges[i][2]-1) ||
u2.get(edges[i][1]-1) != u2.get(edges[i][2]-1)){
                    u1.merge(edges[i][1]-1, edges[i][2]-1);
                    u2.merge(edges[i][1]-1, edges[i][2]-1);
                    cnt++;
```

```
                }
            }
            if(edges[i][0] == 2 && u2.getSize() > 1){//Bob
                if(u2.get(edges[i][1]-1) != u2.get(edges[i][2]-1)){
                    u2.merge(edges[i][1]-1, edges[i][2]-1);
                    cnt++;
                }
            }
            if(edges[i][0] == 1 && u1.getSize() > 1){//Alice
                if(u1.get(edges[i][1]-1) != u1.get(edges[i][2]-1)){
                    u1.merge(edges[i][1]-1, edges[i][2]-1);
                    cnt++;
                }
            }
        }
        if(u1.getSize() > 1 || u2.getSize() > 1)
            return -1;
        return size - cnt;
    }
};
```