1247. 交换字符使得字符串相同（中等）

1. 题目描述

   有两个长度相同的字符串 `s1` 和 `s2`，且它们其中 **只含有** 字符 `"x"` 和 `"y"`，你需要通过「交换字符」的方式使这两个字符串相同。

   每次「交换字符」的时候，你都可以在两个字符串中各选一个字符进行交换。

   交换只能发生在两个不同的字符串之间，绝对不能发生在同一个字符串内部。也就是说，我们可以交换 `s1[i]` 和 `s2[j]`，但不能交换 `s1[i]` 和 `s1[j]`。

   最后，请你返回使 `s1` 和 `s2` 相同的最小交换次数，如果没有方法能够使得这两个字符串相同，则返回 `-1` 。

   示例 1:

   ```
   输入: s1 = "xx", s2 = "yy"
   输出: 1
   解释:
   交换 s1[0] 和 s2[1]，得到 s1 = "yx", s2 = "yx"。
   ```

   示例 2:

   ```
   输入: s1 = "xy", s2 = "yx"
   输出: 2
   解释:
   交换 s1[0] 和 s2[0]，得到 s1 = "yy", s2 = "xx" 。
   交换 s1[0] 和 s2[1]，得到 s1 = "xy", s2 = "xy" 。
   注意，你不能交换 s1[0] 和 s1[1] 使得 s1 变成 "yx"，因为我们只能交换属于两个不同字符串的字符。
   ```

   示例 3:

   ```
   输入: s1 = "xx", s2 = "xy"
   输出: -1
   ```

   示例 4:

   ```
   输入: s1 = "xxyyxyxyxx", s2 = "xyyxyxxxyx"
   输出: 4
   ```

   提示:

   - `1 <= s1.length, s2.length <= 1000`
   - `s1, s2` 只包含 `'x'` 或 `'y'`。

2. 题目解答

   嘤嘤嘤想多了，还想用BFS，我就是个憨憨

- 第一步：普遍规律
  由于要求**交换次数尽量少**，故：
  - 本来相同位置就有相同的字符，不需要交换。
  - 本来相同位置字符不同，需要交换。交换为**两组字符交换**，本质上只有两种情形：

    (a) 2组相同(2组 `xy` 或2组 `yx` ，等价于示例1)：
    此时， `s1[0]` 与 `s2[1]` 交换即可，需要进行1次交换
    (b) 2组不同(1组 `xy` ，1组 `yx` ，等价于示例2)：
    此时，将 `s1[0]` 与 `s2[0]` 交换后与a)相同，需要进行2次交换

    综上所述，我们可以得出如下结论：
    - `xy` 与 `yx` 的组数之和必须为偶数，否则返回 `-1` (**两两交换**)
    - 优先进行(a)类交换，剩余的进行(b)类交换(**贪心算法**)

- 第二步：得出结论
  由于匹配是两两进行，因此，在确定有M对 `xy` ，N对 `yx` ，且$M+N$为偶数后：
  - 若M为偶数，则N也为偶数，则**全部为(a)类交换**。总匹配数为：
    $$\frac{M+N}{2}$$
  - 若M为奇数，则N也为奇数，则**各拿一组进行(b)类交换，其余(a)类交换**。总匹配数为：
    $$\frac{M-1}{2}+\frac{N-1}{2}+2=\frac{M+N}{2}+1$$
  - 两者均可写作：
    $$\frac{M+1}{2}+\frac{N+1}{2}$$

```
class Solution {
public:
    int minimumSwap(string s1, string s2) {
        int ans = 0, n = s1.size();
        int cnt1 = 0,cnt2 = 0;//统计有多少对x-y和y-x
        for(int i = 0; i < n; i++){
            if(s1[i]=='x'&&s2[i]=='y') cnt1++;
            else if(s1[i]=='y'&&s2[i]=='x') cnt2++;
        }
        //对于每一对 x-y 和 x-y 以及 y-x 和 y-x 都只需要一次操作即可完成匹配
        ans += cnt1/2+cnt2/2;//所需要的操作数
        cnt1%=2;//剩余未匹配的对数
        cnt2%=2;
        if(cnt1+cnt2==1) return -1;//只剩一个时无法匹配
        else if(cnt1+cnt2==2) ans+=2;//只剩了 x-y和y-x 需要两次匹配
        return ans;
    }
};
```

1248. 统计「优美子数组」（中等）

1. 题目描述

   给你一个整数数组 `nums` 和一个整数 `k`。

   如果某个 **连续** 子数组中恰好有 `k` 个奇数数字，我们就认为这个子数组是「**优美子数组**」。

   请返回这个数组中「优美子数组」的数目。

   示例 1：

   ```
   输入：nums = [1,1,2,1,1], k = 3
   输出：2
   解释：包含 3 个奇数的子数组是 [1,1,2,1] 和 [1,2,1,1] 。
   ```

   示例 2：

   ```
   输入：nums = [2,4,6], k = 1
   输出：0
   解释：数列中不包含任何奇数，所以不存在优美子数组。
   ```

   示例 3：

   ```
   输入：nums = [2,2,2,1,2,2,1,2,2,2], k = 2
   输出：16
   ```

   提示：

   - `1 <= nums.length <= 50000`
   - `1 <= nums[i] <= 10^5`
   - `1 <= k <= nums.length`

2. 简单实现

   可以找到以下规律：

   - 假如找到一段连续子数组恰好有k个数字，且其两端为奇数，则这个子数组是一个优美子数组，且任意删除两端的任何一个元素后将无法构成优美子数组，我将其称为**元优美子数组**，例如 `nums = [1,2,1,4,4,1,6,6,6,1], k= 2` 中，有三个元优美子数组，即[1,2,1], [1,4,4,1]和[1,6,6,6,1]
   - 将元优美子数组向两端**连续扩展**，以构成新的优美子数组
     - 向左：可以选择向左**连续**扩展0个偶数(即不向左扩展)，向左**连续**扩展1个偶数，....，直至遇见奇数为止，如上例中，元优美子数组[1,4,4,1]最多可向左连续扩展1个偶数，构成2种情况
     - 向右同理，如上例中，元优美子数组[1,4,4,1]最多可向右连续扩展3个偶数，构成4种情况
     - 因此，由元优美子数组[1,4,4,1]可以扩展出 `2*4 = 8` 个优美子数组（包含其本身）
   - 以此类推，[1,2,1]可扩展出 `1*3 = 3` 个，[1,6,6,6,1]可扩展出 `3*1 = 3` 个，因此nums共有 `3+8+3=14` 个优美子数组

   ```cpp
   class Solution {
   public:
       int numberOfSubarrays(vector<int>& nums, int k) {
           vector<int> idxs;//记录nums中从左至右各个奇数对应的索引
   ```

```
        int len = nums.size();
        for(int i = 0; i < len; i++){
            if(nums[i] % 2 == 1)
                idxs.push_back(i);
        }
        int num = idxs.size();//nums中奇数个数
        if(num < k) return 0;//奇数的个数不足以构成优美子数组
        int ans = 0;
        int l = 0;
        int r = k -1;//nums[idxs[l]...idxs[r]]构成一个元优美子数组
        while(r < num){
            int l_num, r_num;
            if(l == 0)
                l_num = idxs[l] + 1;
            else
                l_num = idxs[l] - idxs[l-1];
            if(r == num - 1)
                r_num = len - idxs[r];
            else
                r_num = idxs[r+1] - idxs[r];
            ans += l_num * r_num;
            l++;
            r++;
        }
        return ans;
    }
};
```

1249. 移除无效的括号（中等）

1. 题目描述

给你一个由 `'('`、`')'` 和小写字母组成的字符串 `s`。

你需要从字符串中删除最少数目的 `'('` 或者 `')'` （可以删除任意位置的括号），使得剩下的「括号字符串」有效。

请返回任意一个合法字符串。

有效「括号字符串」应当符合以下 **任意一条** 要求：

- 空字符串或只包含小写字母的字符串
- 可以被写作 `AB`（`A` 连接 `B`）的字符串，其中 `A` 和 `B` 都是有效「括号字符串」
- 可以被写作 `(A)` 的字符串，其中 `A` 是一个有效的「括号字符串」

示例 1：

```
输入：s = "lee(t(c)o)de)"
输出："lee(t(c)o)de"
解释："lee(t(co)de)" , "lee(t(c)ode)" 也是一个可行答案。
```

示例 2：

```
输入：s = "a)b(c)d"
输出："ab(c)d"
```

示例 3：

```
输入：s = "))(("
输出：""
解释：空字符串也是有效的
```

示例 4：

```
输入：s = "(a(b(c)d)"
输出："a(b(c)d)"
```

提示：

- `1 <= s.length <= 10^5`
- `s[i]` 可能是 `'('`、`')'` 或英文小写字母

2. 简单实现

通过，但速度慢

```
class Solution {
public:
    string minRemoveToMakeValid(string s) {
        stack<string> cache;
        int idx = 0;
        while(idx < s.size()){
            if(s[idx] == '('){//直接入栈
                string temp = "(";
                cache.push("(");
                idx++;
            }
            else if(s[idx] == ')'){//在栈里一直找到第一个'('，构成一个有效「括号字符串」
                string temp = "";
                string cur = "";
                while(!cache.empty()){
                    cur = cache.top();
                    cache.pop();
                    temp = cur + temp;
                    if(cur == "("){//找到了"("
                        temp = temp + ")";
                        break;
                    }
                }//若没找到"("，则忽视该位置的")"，代码逻辑与找到"("相同
                cache.push(temp);
                idx++;
            }
            else{//字母，连起来存入栈即可
                string temp = "";
                while(idx < s.size() && s[idx] != '(' && s[idx] != ')'){
                    temp += s[idx];
```

```
                    idx++;
                }
                cache.push(temp);
            }
        }
        string ans = "";
        while(!cache.empty()){
            string cur = cache.top();
            cache.pop();
            if(cur != "("){
                ans = cur + ans;
            }
        }
        return ans;
    }
};
```

3. 自我改进

   实际上只关心 (与) 的配对情况

```
class Solution {
public:
    string minRemoveToMakeValid(string s) {
        stack<int> cache;//记录'('的索引
        for(int i = 0; i < s.size(); i++){
            if(s[i] == '(')
                cache.push(i);
            else if(s[i] == ')'){
                if(!cache.empty())//配对成功
                    cache.pop();
                else{//无法配对，删除该')'
                    s.erase(i, 1);
                    i--;
                }
            }
        }
        while(!cache.empty()){//存在未配对的'('
            s.erase(cache.top(), 1);
            cache.pop();
        }
        return s;
    }
};
//PS：由于从左到右遍历，不用担心删除字符导致索引值对不上
```

4. 其他解法

   从左到右遍历记录左括号-右括号数量，可以删除多余的右括号

   从右到左遍历记录右括号-左括号数量，可以删除多余的左括号

1250. 检查「好数组」（困难）

1. 题目描述

给你一个正整数数组 `nums`，你需要从中任选一些子集，然后将子集中每一个数乘以一个 **任意整数**，并求出他们的和。

假如该和结果为 `1`，那么原数组就是一个「**好数组**」，则返回 `True`；否则请返回 `False`。

示例 1:

```
输入: nums = [12,5,7,23]
输出: true
解释: 挑选数字 5 和 7。
5*3 + 7*(-2) = 1
```

示例 2:

```
输入: nums = [29,6,10]
输出: true
解释: 挑选数字 29, 6 和 10。
29*1 + 6*(-3) + 10*(-1) = 1
```

示例 3:

```
输入: nums = [3,6]
输出: false
```

提示:

- `1 <= nums.length <= 10^5`
- `1 <= nums[i] <= 10^9`

2. 题目解答

考验数学功底。。。。

斐蜀定理:
n个整数时
设$a_1, a_2, a_3, \cdots, a_n$为n个整数，d是它们的最大公约数，那么存在整数$x_1, \cdots, x_n$使得$x_1 * a_1 + x_2 * a_2 + ...x_n * a_n = d$
特别来说，$a_1, a_2, a_3, \cdots, a_n$互质（不是两两互质）等价于存在整数$x_1, \cdots, x_n$使得$x_1 * a_1 + x_2 * a_2 + ...x_n * a_n = 1$

由于数越多，互质的可能越高，因此，直接遍历整个数组直到最大公因数为1即可

```
class Solution {
public:
    int gcd(int a,int b){
        if(b > a) return gcd(b,a);
        if(b) return gcd(b,a%b);
        else return a;
    }

    bool isGoodArray(vector<int>& nums) {
        int res = nums[0];
        if(nums.size() == 1)
```

```
            return res == 1;

        for(int i = 1; i < nums.size(); ++i){
            res = gcd(res,nums[i]);
            if(res == 1)
                return true;
        }
        return false;
    }
};
```