基础算法

双指针及其使用条件

双指针使用的条件为两个指针都必须具有单调性，所谓单调性，就是当右指针向右走的时候，左指针**不允许**出现向左走的情况，具体需要依据题目来确定，但必须要满足这一点才能够用双指针

不能用双指针

原题连接：LeetCode 1590. 使数组和能被 P 整除

求区间和，首先用前缀和处理，将区间和转化为两个数的差，有：

```
int n = nums.size();
vector<int>prefix(n + 1, 0);
for(int i = 1; i <= n; i ++)
    prefix[i] = (prefix[i - 1] + nums[i - 1]) % p;
```

设需要删除的区间为 $[l,r]$ ，此时区间和为 $prefix[r]-prefix[l-1]$ ，若满足

$$prefix[n]-(prefix[r]-prefix[l-1])\equiv 0(\mod p)$$

那么该区间便是局部解

这里不能用双指针的点在于，$prefix[i]\mod p$ 并不是单调的，也就是说当右指针递增时，我们**无法保证左指针一定不会出现递减的情况**

关于「单调性」的理解：

在这里我将其理解为「二分性」

回顾二分查找算法：在一个单调递增的数组中，找到某个数 x 的最小下标 对于一个单调递增的区间而言，它是具有「二分性」的，整个数组可以划分为两个集合——「满足性质 p」以及「不满足性质 p」，在这个案例中，「性质 p」为是否「大于等于 x」

这两个集合没有交集，而二分找的就是两个集合的两个分界点

回到这里，当右指针递增的时候，如果此时左右指针构成的区间能够保证二分性，那么便说明可以使用双指针

对于区间 $[l,r]$ ，我们无法找到某个点 idx 将区间划分为两个**不相交**的集合使得左集合内所有元素均满足 $prefix[n]-(prefix[r]-prefix[l])\equiv 0(\mod p)$

这道题正确的做法是用哈希表

由于每次枚举的是右端点，因此为确定值，将上式移项，有：

$$ prefix[l-1]\equiv prefix[r]-prefixn $$

因此，只需要用哈希表记录**最近一次**$prefix[r]-prefixn$出现的下标，然后取最小值即可

这道题有两个细节：

- 哈希表需要预先将 0 存入，因为前缀和是从 0 开始的
- 求前缀和时，需要对 p 取模，因为会爆 `int`

完整代码如下：

```
class Solution {
public:
    int minSubarray(vector<int>& nums, int p)
    {
        int n = nums.size();
        vector<int>prefix(n + 1, 0);
        for(int i = 1; i <= n; i ++)
            prefix[i] = (prefix[i - 1] + nums[i - 1]) % p;
        int cnt = 0x3f3f3f3f;
        unordered_map<int, int>Hash;
        for(int i = 0; i <= n; i ++)//需要将0提前放入哈希表中
        {
            Hash[prefix[i]] = i;//用于记录prefix[i]在哈希表中最后一次出现得到下标
            int left = (prefix[i] - prefix[n] + p) % p;//保证不出现负数
            if(Hash.find(left) != Hash.end())
                cnt = min(cnt, i - Hash[left]);
        }
        return cnt == 0x3f3f3f3f || cnt == n ? -1 : cnt;
    }
};
```

如果用 `long long`，不对前缀和取模，需要在哈希表插入时取模，因为 $prefix[r]-prefixn$的值不会超过p$

```
class Solution {
public:
    int minSubarray(vector<int>& nums, int p)
    {
        int n = nums.size();
        vector<long long>prefix(n + 1, 0);
        for(int i = 1; i <= n; i ++)
            prefix[i] = (prefix[i - 1] + nums[i - 1]);
        int cnt = 0x3f3f3f3f;
        unordered_map<int, int>Hash;
        for(int i = 0; i <= n; i ++)//需要将0提前放入哈希表中
        {
            Hash[prefix[i] % p] = i;//用于记录prefix[i]在哈希表中最后一次出现得到下标
            int left = ((prefix[i] - prefix[n]) % p + p) % p;//前面一定要先取一次
模，因为前面一定会超出p
```

```
            if(Hash.find(left) != Hash.end())
                cnt = min(cnt, i - Hash[left]);
        }
        return cnt == 0x3f3f3f3f || cnt == n ? -1 : cnt;
    }
};
```

原题链接：LeetCode 面试题 17.05. 字母与数字

是字母还是数字并不重要，我们将字母设成 -1 ，数字设成 1 此时原数组便可以转化成一个**只含** $0,1$ **的整数数组**

我们需要求最长的子数组保证里面的数字与字母相同，等价于，在整数数组中求一段子数组使得 0 与 1 的个数相同，即这段区间和为 0

考虑用前缀和优化，这时问题转变成求两个数 $prefix[r],prefinx[l-1]$ 使得 $prefix[r]=prefix[l-1]$ 并且期望区间最大

子区间并不具有「二分性」，因此不能用双指针，考虑用哈希表

对于当前枚举到的数 $prefix[i]$ 而言

- 如果不在哈希表中，那么将其加入哈希表内
- 如果在哈希表中，则计算一遍区间大小，取最大的区间赋值给 l,r

完整代码：

```
class Solution {
public:
    vector<string> findLongestSubarray(vector<string>& array)
    {
        int n = array.size();
        vector<int>num(n + 1, 0), prefix(n + 1, 0);
        for(int i = 0; i < n; i ++)
        {
            if(isalpha(array[i][0])) num[i] = -1;
            else num[i] = 1;
            prefix[i + 1] = prefix[i] + num[i];
        }
        unordered_map<int, int>Hash;
        int r = 0, l = 0;
        for(int i = 0; i <= n; i++)
        {
            auto it = Hash.find(prefix[i]);
            if(it == Hash.end())//第一次遇到，则对哈希表赋值
                Hash[prefix[i]] = i;
            else if(i - it->second > r - l)//第二次遇到，对区间赋值
                r = i, l = it->second;
        }
        return {array.begin() + l, array.begin() + r};
    }
};
```