

基础算法

- 基础算法
 - 双指针及其使用条件
 - 不能用双指针

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

$$prefix[n] - (prefix[r] - prefix[l - 1]) \equiv 0 \pmod{p}$$

那么该区间便是局部解

这里不能用双指针的点在于， $prefix[i] \bmod p$ 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**

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：

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

因此，只需要用哈希表记录**最近一次** $prefix[r] - prefix[n]$ 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 $\text{prefix}[r] - \text{prefix}[l]$ 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;
    }
};

```