# 题目

给定一个二进制数组 nums , 找到含有相同数量的 0 和 1 的最长连续子数组，并返回该子数组的长度。

示例 1：

输入: nums = [0,1]

输出: 2

说明: [0, 1] 是具有相同数量 0 和 1 的最长连续子数组。

示例 2：

输入: nums = [0,1,0]

输出: 2

说明: [0, 1] (或 [1, 0]) 是具有相同数量 0 和 1 的最长连续子数组。

提示：

1 <= nums.length <= 105

nums[i] 不是 0 就是 1

注意：本题与主站 525 题相同：

<https://leetcode-cn.com/problems/contiguous-array/>

# 分析

## 方法一：前缀和+哈希表

思路：

由于「0和1的数量相同」等价于「1的数量减去0的数量等于0」，我们可以将数组中的0视作−1，则原问题转换成「求最长的连续子数组，其元素和为0」。

设数组nums的长度为n，将数组nums进行转换得到长度相等的新数组newNums：对于0≤i<n，当nums[i]=1时newNums[i]=1，当nums[i]=0时newNums[i]=−1。

为了快速计算newNums的子数组的元素和，需要首先计算newNums的前缀和。用prefixSums[i]表示newNums从下标0到下标i的前缀和，则newNums从下标j+1到下标k（其中j<k）的子数组的元素和为prefixSums[k]−prefixSums[j]，该子数组的长度为k−j。

当 prefixSums[k]−prefixSums[j]=0时，即得到newNums的一个长度为k−j的子数组元素和为0，对应nums的一个长度为k−j的子数组中有相同数量的0和1。

实现方面，不需要创建数组newNums和prefixSums，只需要维护一个变量counter存储newNums的前缀和即可。具体做法是，遍历数组nums，当遇到元素1时将counter的值加1，当遇到元素0时将counter的值减1，遍历过程中使用哈希表存储每个前缀和第一次出现的下标。

规定空的前缀的结束下标为−1，由于空的前缀的元素和为0，因此在遍历之前，首先在哈希表中存入键值对(0,−1)。遍历过程中，对于每个下标i，进行如下操作：

**如果counter的值在哈希表中已经存在，则取出counter在哈希表中对应的下标prevIndex，nums从下标prevIndex+1到下标i的子数组中有相同数量的0和1**，该子数组的长度为i−prevIndex，使用该子数组的长度更新最长连续子数组的长度；

如果counter的值在哈希表中不存在，则将当前余数和当前下标i的键值对存入哈希表中。

由于哈希表存储的是counter的每个取值第一次出现的下标，因此当遇到重复的前缀和时，根据当前下标和哈希表中存储的下标计算得到的子数组长度是以当前下标结尾的子数组中满足有相同数量的0和1的最长子数组的长度。遍历结束时，即可得到nums中的有相同数量的0和1的最长子数组的长度。

代码：

class Solution {

public:

int findMaxLength(vector<int>& nums) {

int maxLength = 0;

unordered\_map<int, int> mp;

int counter = 0;

mp[counter] = -1;

int n = nums.size();

for (int i = 0; i < n; i++) {

int num = nums[i];

if (num == 1) {

counter++;

} else {

counter--;

}

if (mp.count(counter)) { // 存在这个counter前缀和

int prevIndex = mp[counter];

maxLength = max(maxLength, i - prevIndex);

} else {

mp[counter] = i;

}

}

return maxLength;

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是数组nums的长度。需要遍历数组一次。

空间复杂度：O(n)，其中n是数组nums的长度。空间复杂度主要取决于哈希表，哈希表中存储的不同的counter的值不超过n个。