# 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

# 分析

这里是含有相同数量的最长连续数组，因此符合前缀和的算法。

## 方法一：前缀和+哈希+动态规划

思路：

代码：

int findMaxLength(vector<int>& nums) {

size\_t n = nums.size();

vector<int> preSum(n + 1);

unordered\_map<int, int> mp;

mp[0] = -1;

size\_t maxLen = 0;

for (size\_t i = 0; i < n; ++i) {

if (nums[i] == 0) nums[i] = -1;

preSum[i + 1] = preSum[i] + nums[i];

// 首次出现保存到unordered\_map中

if (mp.find(preSum[i + 1]) == mp.end())

mp[preSum[i + 1]] = i;

// 在mp的就是preSum值之前首次出现的下标，更新最大值

else maxLen = max(maxLen, i - mp[preSum[i + 1]]);

}

return maxLen;

}

另一种写法：

1、关于两种东西的个数相同====个数的差值diff=0

类似的问题：

平衡二叉树的平衡因子 = 左子树depth - 右子树depth

等价于：左子的depth每个深度单位（1）贡献为+1，右子的depth每个深度单位（1）贡献为-1

数值 ==== 平衡因子 ==== cnt1 - cnt2

2、本题---------> 区间和==0 的最大区间长度

3、区间和---->前缀和

4、加速------>哈希（万金油）

5、动态规划dp---->往前探---->往左找匹配

class Solution

{

public:

int findMaxLength(vector<int>& nums)

{

int n = nums.size();

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

if (nums[i] == 0)

nums[i] = -1;

vector<int> presum(n + 1, 0);

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

presum[i + 1] = presum[i] + nums[i];

unordered\_map<int, int> presum\_first\_idx;

for (int i = 0; i < n + 1; i ++)

{

if (presum\_first\_idx.find(presum[i]) == presum\_first\_idx.end())

presum\_first\_idx[presum[i]] = i;

}

int res = 0;

for (int i = 1; i < n + 1; i ++)

res = max(res, i - presum\_first\_idx[presum[i]]);

return res;

}

};

## 方法二：前缀和+哈希表/路径压缩

思路：

路径压缩（针对前缀和数组，但hash还是要空间）。

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（路径压缩）：

class Solution {

public:

int findMaxLength(vector<int>& nums) {

unordered\_map<int, int> mp;

int preSum = 0, maxLen = 0;

mp[0] = -1;

for (int i = 0; i < nums.size(); ++i){

preSum += nums[i] ? 1 : -1;

if(mp.find(preSum) != mp.end()) maxLen = max(maxLen, i - mp[preSum]);

else mp[preSum] = i;

}

return maxLen;

}

};

**或（官方答案，推荐）：**

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。

## 方法三：前缀和

**思路：**

算法步骤（这个方法感觉理解不是很好）：

1、创建一个哈希表，用key来储存cur值，value来储存当前index。

2、假设我们碰到0就将cur decrement(减一), 碰到1则increment(加一)。

3、如果我们能在哈希表中找到当前的cur值, 则取出对应的pos, 在看当前的index - pos是否比ans大, 取其中的最优解。

核心：由于以上碰1加一，碰0减一的操作，当0与1数量一致时(连续数组), 其连续数组的和为零。因此我们知道数组前面的cur值是什么，在到达该连续数组尾部时就不会变。因此我们只需要检查哈希表中是否存在其相同的cur值即可！

**代码：**

class Solution {

public:

int findMaxLength(vector<int>& nums) {

unordered\_map<int, int> m = {{0,-1}};

int cur = 0, ans = 0;

for(int i = 0; i < nums.size(); ++i)

{

nums[i] == 0? --cur : ++cur;

if(m.count(cur))

ans = max(ans, i - m[cur]);

else

m[cur] = i;

}

return ans;

}

};

复杂度：

时间复杂度：O(n)，需要遍历整个数组  
 空间复杂度：O(n)，空间复杂度取决与哈希表中键值的数量

或：

class Solution {

public:

int findMaxLength(vector<int>& nums) {

unordered\_map<int,int>maps;

maps[0] = -1;

int sum = 0,ret = 0;

for(int i = 0;i<nums.size();i++){

sum += nums[i] ? 1: -1;

auto tmp = maps.find(sum);

if(tmp != maps.end()){

ret = max(ret,i - tmp->second);

}else{

maps[sum] = i;

}

}

return ret;

}

};