# 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

## 方法一：前缀和+哈希+动态规划

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+路径压缩

路径压缩(针对前缀和数组，但hash还是要空间)

代码：

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;

}

};

复杂度：

空间复杂度O(n)

时间复杂度O(n)

## 方法三：前缀和

**思路：**

算法步骤：

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;

}

};