# 题目

给定一个二进制数组，你可以最多将 1 个 0 翻转为 1，找出其中最大连续 1 的个数。

示例 1：

输入：[1,0,1,1,0]

输出：4

解释：翻转第一个 0 可以得到最长的连续 1。

  当翻转以后，最大连续 1 的个数为 4。

注：

输入数组只包含 0 和 1.

输入数组的长度为正整数，且不超过 10,000

**进阶：**

如果输入的数字是作为 无限流 逐个输入如何处理？换句话说，内存不能存储下所有从流中输入的数字。您可以有效地解决吗？

# 分析

## 方法一：预处理 + 枚举

直观的想法就是枚举每个0的位置，把这个位置变成1，统计它能把前后连成的最多的1的个数。考虑位置i(0<i<n−1) ，从这个位置切开可以将整个数组（数组下标[0,n−1]）分成三个部分：

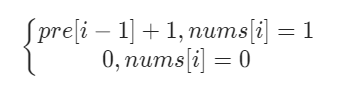
1.[0,i−1]

2.[i,i]

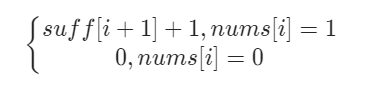
3.[i+1,n−1]

则区间包含i的最大连续1的个数就是由第一部分的数组末尾往前延伸最大的连续1的个数加上第三部分的数组开头往后延伸最大的连续1的个数再加上把第i个位置变成1的总和。第一部分和第二部分我们都可以通过预处理数组来实现O(1)查询。

预处理pre[i]数组表示以i结尾往前延伸最大连续1的个数，则可以列出递推式：



预处理suff[i]数组表示以i开头往后延伸最大连续1的个数，则可以列出递推式：



则位置i的答案就是pre[i−1]+1+suff[i+1]

对于这一类最多改变一个位置的题都可以这么考虑，当然我们肯定还要统计原数组的最大连续1的个数，这个直接在我们预处理两个数组的时候就可以统计出来了，即maxin=0pre[i] 。

class Solution {

public:

int findMaxConsecutiveOnes(vector<int>& nums) {

int n=(int)nums.size(),ans=0;

vector<int>pre(n,0),suff(n,0);

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

if (i) pre[i]=pre[i-1];

if (nums[i]) pre[i]++;

else pre[i]=0;

ans=max(ans,pre[i]);

}

for (int i=n-1;i>=0;--i){

if (i<n-1) suff[i]=suff[i+1];

if (nums[i]) suff[i]++;

else suff[i]=0;

}

for (int i=0;i<n;++i)if(!nums[i]){

int res=0;

if (i>0) res+=pre[i-1];

if (i<n-1) res+=suff[i+1];

ans=max(ans,res+1);

}

return ans;

}

};

复杂度分析

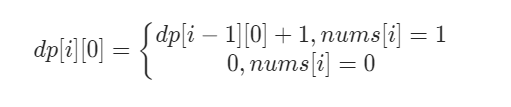
时间复杂度：O(n) ，预处理两个数组O(n)， 最后统计答案也是O(n) 。

空间复杂度：O(n)，需要额外两个辅助数组。

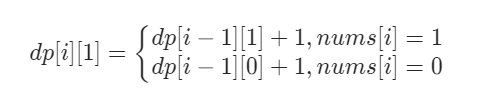
## 方法二：动态规划

方法一其实没有办法解决进阶问题：如果输入的数字是作为无限流逐个输入如何处理？换句话说，内存不能存储下所有从流中输入的数字。您可以有效地解决吗？ 因为它需要预先知道所有的数，而我们如果用动态规划则可以有效解决进阶问题。

定义dp[i][0]为考虑到以i为结尾未使用操作将[0,i]某个0变成1的最大的连续1的个数，dp[i][1]为考虑到以i为结尾使用操作将[0,i]某个0变成1的最大的连续1的个数。则我们可以列出转移式：



和



解释一下，针对dp[i][0] ，如果当前位置是0，由于未使用操作，所以肯定是0，如果是1，则从前一个位置未使用操作的状态转移过来即可。针对 dp[i][1] ，如果当前位置是0 ，则我们操作肯定是要用在这个位置，把它变成1，所以只能从前一个未使用过操作的状态转移过来，如果是1，则从前一个已经使用过操作的状态转移过来。

最后答案就是



到这里其实还并不能解决进阶问题，因为开dp数组仍然需要提前知道数组的大小，但是我们注意到每次转移只与前一个位置有关，所以我们并不需要开数组，只需要额外两个变量记录一下前一个位置的两个状态即可，这样我们就可以有效解决进阶的问题。

class Solution {

public:

int findMaxConsecutiveOnes(vector<int>& nums) {

int n=(int)nums.size(),ans=0,dp0=0,dp1=0;

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

if (nums[i]){

dp1++;

dp0++;

}

else{

dp1=dp0+1;

dp0=0;

}

ans=max(ans,max(dp0,dp1));

}

return ans;

}

};

复杂度分析

时间复杂度：由上面的转移式我们知道转移是O(1)的，一共有2n个状态，所以时间复杂度为O(n)。

空间复杂度：O(1)。

## 方法三：双指针+一次遍历

class Solution {

public:

int findMaxConsecutiveOnes(vector<int>& nums) {

int zeroCount = 0;

int ans = 0;

int left = 0;

int right = 0;

int size = nums.size();

int preZero = -1;

if(size == 1) return 1;

while(right < size)

{

if(nums[right] == 0)

zeroCount++;

if(nums[right] == 0 && preZero == -1)

preZero = right;

if(zeroCount > 1)

{

zeroCount--;

left = preZero+1;

preZero = right;

}

ans = max(ans, right - left + 1);

right++;

}

return ans;

}

};

## 方法四：滑动窗口

解题思路

用滑动窗口，l为窗口左边界，r为窗口右边界，在窗口中一直保证一直最多一个0存在，然后r-l就是包括这个零的最大长度，最后别忘了对尾巴处理

代码

class Solution {

public:

//用滑动窗口去做

int findMaxConsecutiveOnes(vector<int>& nums) {

int l=0,r=0;

int count\_0=0;//统计0的个数

int max=0;

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

if(nums[r]==0){

if(r-l>max) max=r-l;

if(count\_0>=1){//已经有1个0了

while(nums[l]==1) l++;

l++;

}

else{

count\_0++;

}

}

}

if(r-l>max) max=r-l;

return max;

}

};