# #978. 最长湍流子数组【动态规划】

## 疯狂超时---滑动窗口（繁琐了？）--最后一个用例没法通过

class Solution {  
public:  
 int maxTurbulenceSize(vector<int>& arr) {  
 int length=arr.size();//获取数组长度  
//设置一个dp数组  
vector<int>dp(length,1);//所有数组初始化为1  
  
//逐一遍历  
for(int i=0;i<length;i++)  
{  
 if(i!=0&&dp[i]!=dp[i-1])  
 dp[i]=2;  
 if(i==1&&(arr[i]>arr[i-1]||arr[i]<arr[i-1]))  
 dp[i]=max(dp[i],dp[i-1]+1);  
 else if(i>1&&((arr[i]>arr[i-1]&&arr[i-1]<arr[i-2])||(arr[i]<arr[i-1]&&arr[i-1]>arr[i-2])))  
 dp[i]=max(dp[i],dp[i-1]+1);  
   
}  
//找最大值  
int Max=dp[0];  
for(int i=1;i<length;i++)  
{  
 if(Max<dp[i])Max=dp[i];  
}  
return Max;  
 }  
};

## 疯狂超时-动态规划繁琐了？

class Solution {  
public:  
 int maxTurbulenceSize(vector<int>& arr) {  
//动态规划  
int n=arr.size();  
vector<vector<int>>dp(n,vector<int>(n,0));  
//遍历  
int res=0;  
for(int length=0;length<arr.size();length++){  
 for(int i=0;i<n-length;i++){  
 int j=i+length;  
 if(length==0){  
 //初始化--长度为1  
 dp[i][j]=1;  
 }  
 else if(length==1){  
 //初始化-长度为2  
 if(arr[i]!=arr[j]){  
 dp[i][j]=1;  
 }  
 }  
 else {  
 //int left=0,right=0;  
 if(dp[i+1][j]){  
  
 //左边  
 if(arr[i+1]>arr[i+2]&&arr[i]<arr[i+1]){  
 dp[i][j]=1;  
 // left=1;  
 }  
 else if(arr[i+1]<arr[i+2]&&arr[i]>arr[i+1])  
 {  
 dp[i][j]=1;//left=1;  
 }  
 }  
 else if(dp[i][j-1]){  
 //右边  
 if(arr[j-1]>arr[j-2]&&arr[j-1]>arr[j]){  
 dp[i][j]=1;  
 //right=1;  
 }  
 else if(arr[j-1]<arr[j-2]&&arr[j-1]<arr[j]){  
 dp[i][j]=1;  
 //right=1;   
 }  
 }  
   
   
 }  
  
 if(dp[i][j]){  
 res=max(res,length+1);  
 }  
  
 }  
}  
  
return res;  
 }  
};

## 动态规划也会繁琐？---如何评价和寻找最有动态规划

第一个超时应该是由于大小比较的次数有点多，尤其是在用vector模板的情况下；

第二个虽然说是动态规划，但是这个动态规划的效能很低啊！不仅开销的空间大，而且比较的次数也多；

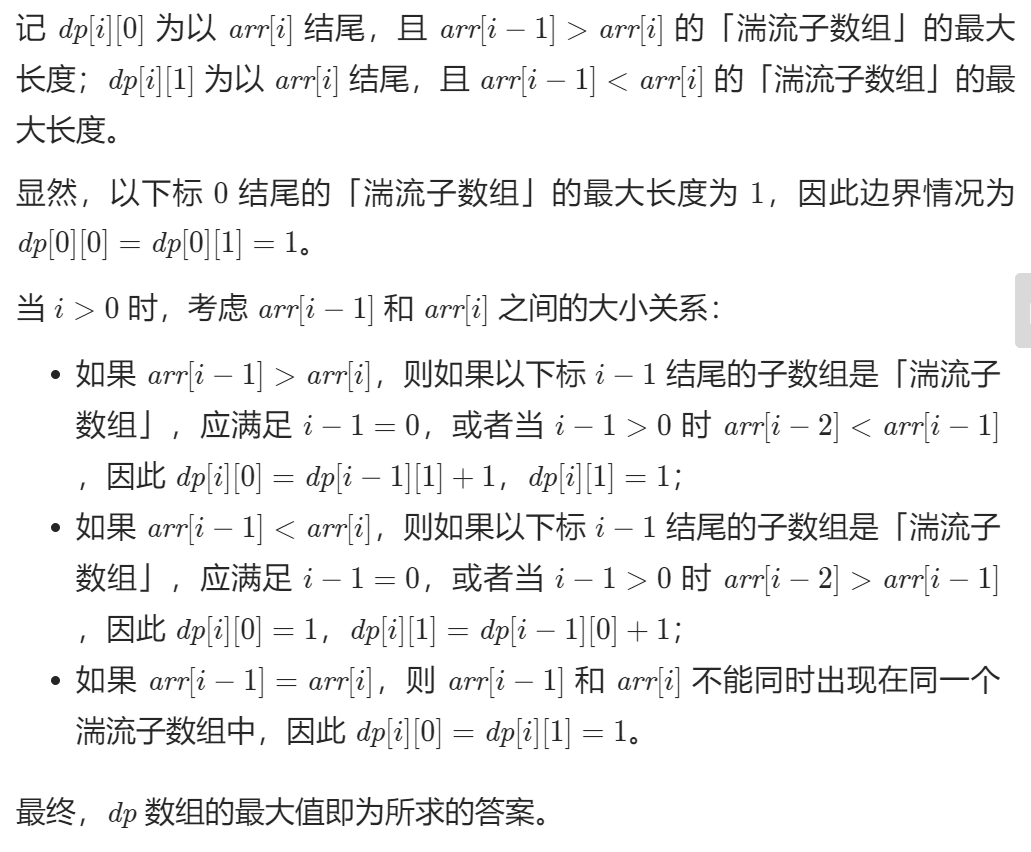
**所以说，动态规划也是有优劣之分的，在进行动态规划设计的时候还是要注意找到一个有时间优势或空间优势的方案**

## 滑动窗口优化

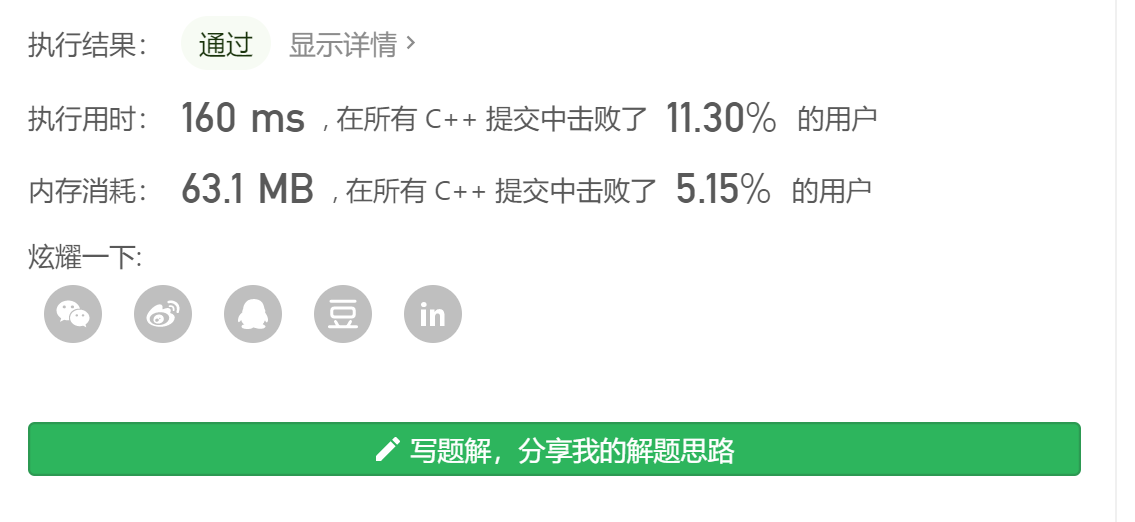
class Solution {  
public:  
 int maxTurbulenceSize(vector<int>& arr) {  
 int n = arr.size();  
 int ret = 1;  
 int left = 0, right = 0;//只设置两个指针，而不开销空间，空间消耗很小  
  
 while (right < n - 1) {  
 if (left == right) {  
 if (arr[left] == arr[left + 1]) {  
 left++;  
 }  
 right++;  
 } else {  
 if (arr[right - 1] < arr[right] && arr[right] > arr[right + 1]) {  
 right++;  
 } else if (arr[right - 1] > arr[right] && arr[right] < arr[right + 1]) {  
 right++;  
 } else {  
 left = right;  
 }  
 }  
 ret = max(ret, right - left + 1);  
 }  
 return ret;  
 }  
};

## 动态规划优化

1. 首先是要减少空间的花销；从左往右看，线性思维；但是花销其实也很大；（这道题用动态规划并不是最优解）
2. 思路分析

* 
* class Solution {  
  public:  
   int maxTurbulenceSize(vector<int>& arr) {  
   int n = arr.size();  
   vector<vector<int>> dp(n, vector<int>(2, 1));  
   dp[0][0] = dp[0][1] = 1;  
   for (int i = 1; i < n; i++) {  
   if (arr[i - 1] > arr[i]) {  
   dp[i][0] = dp[i - 1][1] + 1;  
   } else if (arr[i - 1] < arr[i]) {  
   dp[i][1] = dp[i - 1][0] + 1;  
   }  
   }  
    
   int ret = 1;  
   for (int i = 0; i < n; i++) {  
   ret = max(ret, dp[i][0]);  
   ret = max(ret, dp[i][1]);  
   }  
   return ret;  
   }  
  };

1. 不是很优



## 反思总结

1：动态规划也有优劣之分，在数据较大的时候速度会有欠缺

2：vector模板的使用会在一定程度上限制速度

3：动态规划也不是所有题目的最优解，还是要随机应变