## 300.最长递增子序列

题目链接：https://leetcode-cn.com/problems/longest-increasing-subsequence/

给你一个整数数组 nums ，找到其中最长严格递增子序列的长度。

子序列是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，[3,6,2,7] 是数组 [0,3,1,6,2,2,7] 的子序列。

示例 1：
输入：nums = [10,9,2,5,3,7,101,18]
输出：4
解释：最长递增子序列是 [2,3,7,101]，因此长度为 4 。

> dp数组及其下标：

**dp[i]表示下标0....i-1且以i结尾的最长递增子序列长度**

> 递推公式：
> if(nums[j]<nums[i]) dp[i]=max(dp[i],dp[j]+1);
> 初始化:**由于以下标为i结尾的递增子序列长度至少为1即本身，所以初始化为1**
> 遍历顺序一定是**从0开始到nums.size()**

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
 if(nums.size()==0) return 0;
 int result=1;
 vector<int> dp(nums.size(),1);//初始化为1
 for(int i=1;i<nums.size();i++){
 for(int j=0;j<i;j++){
 if(nums[j]<nums[i]) dp[i]=max(dp[i],dp[j]+1);
  }
  result=max(dp[i],result);
  }
  return result;
     }
};
```

## 674. 最长连续递增序列

题目链接：https://leetcode-cn.com/problems/longest-continuous-increasing-subsequence/

给定一个未经排序的整数数组，找到最长且 连续递增的子序列，并返回该序列的长度。

连续递增的子序列 可以由两个下标 l 和 r（l < r）确定，如果对于每个 l <= i < r，都有 nums[i] < nums[i + 1] ，那么子序列 [nums[l], nums[l + 1], ..., nums[r - 1], nums[r]] 就是连续递增子序列。

> 加了一个限制条件就是**连续**
> **dp数组定义及其下标含义：**
> **dp[i]表示以i结尾的最长连续递增子序列长度**
> **递推公式**
> **if(nums[j]<nums[i])dp[i]=max(dp[i],dp[j]+1);**
> **初始化:初始为1**

```
class Solution {
public:
    int findLengthOfLCIS(vector<int>& nums) {
 if(nums.size()==0) return 0;
 vector<int> dp(nums.size(),1);
 int result=1;
 for(int i=1;i<nums.size();i++){
 if(nums[i-1]<nums[i]) dp[i]=dp[i-1]+1;
 result=max(dp[i],result);
  }
  return result;
     }
};

或者用贪心算法求
if(nums.size()==0) return 0;
int result=1;
int count=1;
for(int i=1;i<nums.size();i++){
  if(nums[i-1]<nums[i]) count++;
  else count=1;
  result=max(count,result);
}
return result;
```

**科普:不同编程语言数组元素在内存中的分配是不同的，C++的数组元素在内存中是连续分配的**

1、数组是存放**连续存储空间**中**相同数据类型**的**集合**

**2、数组元素是不能删除是覆盖**

## 718. 最长重复子数组

题目链接：https://leetcode-cn.com/problems/maximum-length-of-repeated-subarray/

给两个整数数组 A 和 B ， 返回两个数组中公共的、长度最长的子数组的长度。

示例：

输入：
A: [1,2,3,2,1]
B: [3,2,1,4,7]
输出：3
解释：
长度最长的公共子数组是 [3, 2, 1] 。
提示：

- 1 <= len(A), len(B) <= 1000
- 0 <= A[i], B[i] < 100

# 如果子数组不是连续则用下面方法

> dp数组及其下标:
> dp[i][j]表示数组A前i个元素和数组B前j个元素公共的，长度最长的子数组长度
> 递推公式
> if(ArrayA[i]==ArrayB[j])dp[i][j]=dp[i-1][j-1]+1;
> if(ArrayA[i]!=ArrayB[j])dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
> 初始化
> dp[0][0]=0,dp[0][1]=0,dp[1][0]=0;
> 确定遍历顺序
> 外层for循环遍历A,内层for循环遍历B
> 最终结果:dp[ArrayA.size()][ArrayB.size()];

# 如果子数组是连续的则用下面方法

> 1、dp数组及其下标:
> dp[i][j]表示以数组A下标i-1结尾和数组B下标j-1结尾，最长重复子数组长度为dp[i][j]
> 2、递推公式:
> if(numsA[i-1]==numsB[j-1])//注意i不是指数组中的下标而是指数组的第几个元素
> dp[i][j]=dp[i-1][j-1]+1;
> 3、初始化
> dp[0][0]=0;

```
1 class Solution {
2 public:
```

```
    int findLength(vector<int>& nums1, vector<int>& nums2) {
            vector<vector<int>> dp(nums1.size()+1,vector<int>(nums2.size()+1,0));
            int result=0;
            if(nums1.size()==0||nums2.size()==0) return 0;
            for(int i=1;i<=nums1.size();i++){
                for(int j=1;j<=nums2.size();j++){
                    //注意i不是指数组下标,是指数组中的第几个元素
                     if(nums1[i-1]==nums2[j-1]) dp[i][j]=dp[i-1][j-1]+1;
                     result=max(result,dp[i][j]);
                 }
             }
             return result;
    }
};
```

## 1143.最长公共子序列

给定两个字符串 text1 和 text2，返回这两个字符串的最长公共子序列的长度。

一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。

例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。两个字符串的「公共子序列」是这两个字符串所共同拥有的子序列。

若这两个字符串没有公共子序列，则返回 0。

> 由于最长公共子序列可以不是连续的，那么就可以dp数组的定义就如下：
> 1、dp数组的定义及其下标含义
> dp[i][j]表示数组A下标为0....i-1和数组B0....j-1最长公共子序列长度
> 2、递推公式
> if(nums[i-1]==nums[j-1])dp[i][j]=dp[i-1][j-1]+1;
> else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
> 3、初始化
> dp[0][0]=dp[0][1]=dp[1][0]=0;
> 4、遍历顺序先遍历数组A后遍历数组B

```cpp
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
 if(text1.size()==0||text2.size()==0) return 0;
 vector<vector<int>> dp(text1.size()+1,vector<int>(text2.size()+1,0));
 for(int i=1;i<=text1.size();i++){
 for(int j=1;j<=text2.size();j++){
 if(text1[i-1]==text2[j-1]) dp[i][j]=dp[i-1][j-1]+1;
 else dp[i][j]=max(dp[i][j-1],dp[i-1][j]);
  }
  }
  return dp[text1.size()][text2.size()];
    }
};
```

# 总结：

**如果求得是连续子序列的最长公共长度则dp[i][j]定义为以下标i-1数组A结尾且以下标j-1数组B结尾的最长公共子序列长度**

**如果求的是不连续子序列的最长公共长度则dp[i][j]定义为以下标0....i-1数组A且以下标0...j-1数组B的最长公共子序列长度**

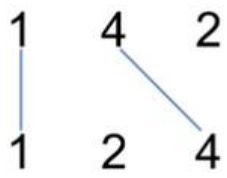
## 1035.不相交的线

我们在两条独立的水平线上按给定的顺序写下 A 和 B 中的整数。

现在，我们可以绘制一些连接两个数字 A[i] 和 B[j] 的直线，只要 A[i] == B[j]，且我们绘制的直线不与任何其他连线（非水平线）相交。

以这种方法绘制线条，并返回我们可以绘制的最大连线数。

**示例 1：**




**输入：** A = [1,4,2], B = [1,2,4]
**输出：** 2
**解释：**
我们可以画出两条不交叉的线，如上图所示。
我们无法画出第三条不相交的直线，因为从 A[1]=4 到 B[2]=4 的直线将与从 A[2]=2 到 B[1]=2 的直线相交。

其实就是不连续的最长公共子序列长度
因为：**直线不能相交,这就是说明在字符串A中找到一个与字符串B相同的子序列**，且子序列的相对顺序不改变，**链接相同数字的直线就不会相交**
1、dp[i][j]表示下标0..
.i-1数组A和下标0....j-1数组B，我们可以绘制的最大连线数
2、递推公式 if(A[i-1]==B[j-1]) dp[i][j]=dp[i-1][j-1]+1;else dp[i][j]= max(dp[i-1][j],dp[i][j-1]);
3、初始化,所以初始化为0;

```
class Solution {
public:
    int maxUncrossedLines(vector<int>& A, vector<int>& B) {
        //其实就是不连续的最长公共子序列
            vector<vector<int>> dp(A.size()+1,vector<int>(B.size()+1,0));
            for(int i=1;i<=A.size();i++){
                for(int j=1;j<=B.size();j++){
                    if(A[i-1]==B[j-1]) dp[i][j]=dp[i-1][j-1]+1;
```

```
                    else dp[i][j]=max(dp[i][j-1],dp[i-1][j]);
                }
            }
            return dp[A.size()][B.size()];
    }
};
```

## 53. 最大子序和

题目地址：https://leetcode-cn.com/problems/maximum-subarray/

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例:
输入: [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

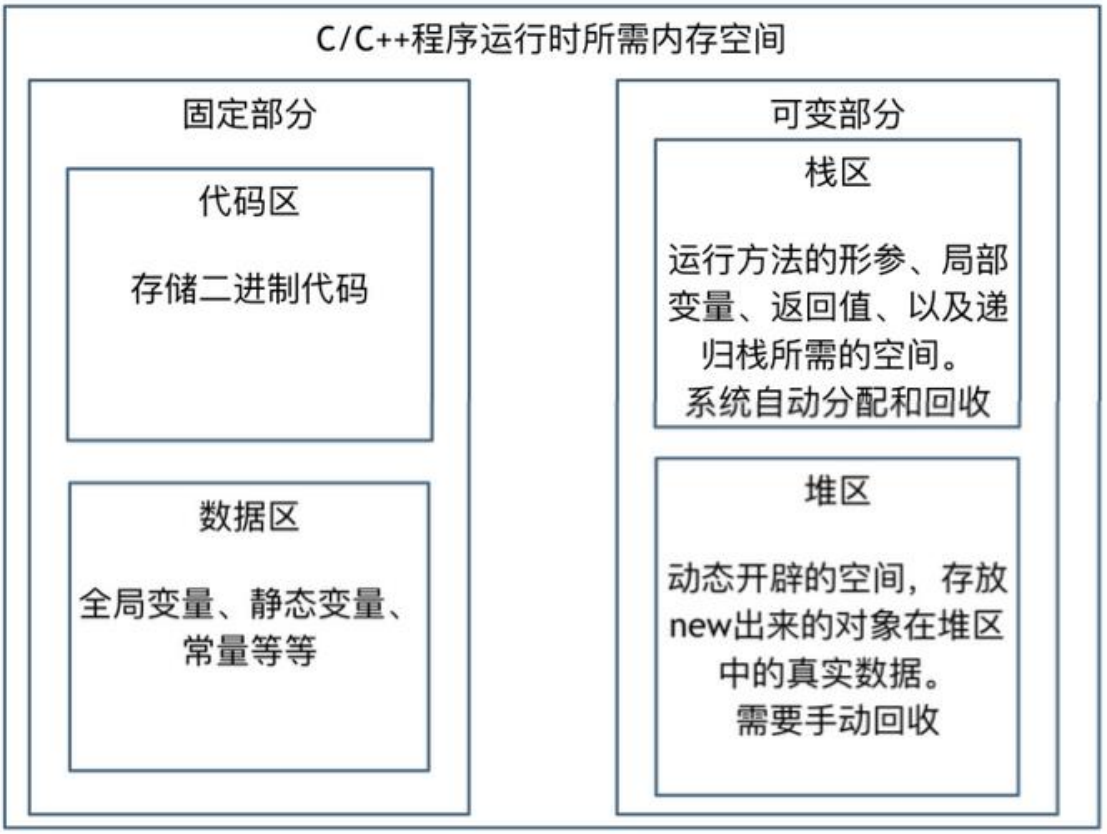
## 思路

```
dp数组定义及其含义:
 dp[i]表示以下标为i-1结尾的整数数组nums，最大和的连续子数组

 class Solution {
public:
    int maxSubArray(vector<int>& nums) {
 vector<int> dp(nums.size()+1,0);
 int result=INT_MIN;
 for(int i=1;i<=nums.size();i++){
  dp[i]=max(dp[i-1]+nums[i-1],nums[i-1]);
  result=max(dp[i],result);
 }
 return result;
    }
};
```

> 刷了这么多题，你了解自己代码的内存消耗么?
> C++的程序就要知道堆栈的概念，程序运行时所需的内存空间可分为固定部分和可变部分

如果我们写C++的程序，就要知道栈和堆的概念，程序运行时所需的内存空间分为 固定部分，和可变部分，如下：




固定区表示**固定部分的内存消耗**是不会随着代码产生变化的，**可变部分**则是会变化的。

> 在可变部分中，栈区间的数据在代码块执行结束之后，系统会自动回收，而堆区间数据是需要程序员回收的，所以也就是内存泄露的发源地。

```
1、栈区(Stack):由编译器自动分配释放，存放函数的参数值，局部变量的值等，其操作方式类似于数据结果中的栈
2、堆区(Heap):一般由程序员分配释放，若程序员不释放，程序结束时可能由OS收回
3 代码区和数据区占用内存空间都是固定，而且占用的空间非常小，那么看运行时消耗的内存主要看可变部分
```

# 内存对齐:

> 不要以为只有C/C++才会有内存对齐，只要可以**跨平台的编程语言**都需要做**内存对齐**，Java、Python都是一样的。

**为什么会有内存对齐?**

1、平台原因：**不是所有的硬件平台**都能访问**任意内存地址上**的**任意数据**，某些硬件平台只能在某些地址处取某些特定类型的数据，否则**抛出硬件异常**。为了同一个程序可以在多平台运行，需要内存对齐。

2、硬件原因:经过**内存对齐后**，**CPU访问内存的速度**大大提升。