4.1-5

题目：设计一个非递归、线性时间的算法，寻找一个数组中的最大子数组。

伪代码如下:

//arr为入参的数组

linearTimeFind(arr)

1. maxStart =maxEnd=0 //目标返回值
2. curStart =curEnd =0 //当前最大子数组索引
3. curSum =maxSum =sum = 0; //当前最大子数组和，最大子数组和，连续串和
4. for i=0 to arr.length
5. sum+=arr[i]
6. //sum入不敷出，前方所有位置看作是一个负数，被舍弃
7. if(sum<=0)
8. if(maxSum<curSum)
9. (maxSum,maxStart,maxEnd) = (curSum,curStart,curEnd)
10. //重新记录一个可能的最大子数组
11. sum=curSum=0;
12. if(i+1<arr.length)
13. curStart = curEnd = i+1
14. //如果当前位置导致和增加（要保证连续串大于当前和），那么重新记录当前尾指针
15. if(curSum+arr[i]>cursum && sum>cursum)
16. curEnd = i; cursum = sum
17. Return (maxStart,maxEnd)

算法复杂度分析:

由于for循环中的每一个操作都是θ(1)的，而for循环总共执行n次，可得，复杂度为θ(n)，也即线性时间。

实例：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 2 | 3 | 7 | -38 | 11 | -10 | 7 | 44 | 3 | 4 | -5 |

我们用(cs,ce)指代(curStart,curEnd),用(ms,me)指代(maxStart,maxEnd)

由算法：(下方图解位置对应上图，例如第二行的ce，意思是指向第四个位置的元素7)

(ms,me,cs,ce)

(ms,me,cs) ce

ce的下一个位置是 -38 这会导致sum<0,因此

ms me (cs,ce)

cs ce

ms me

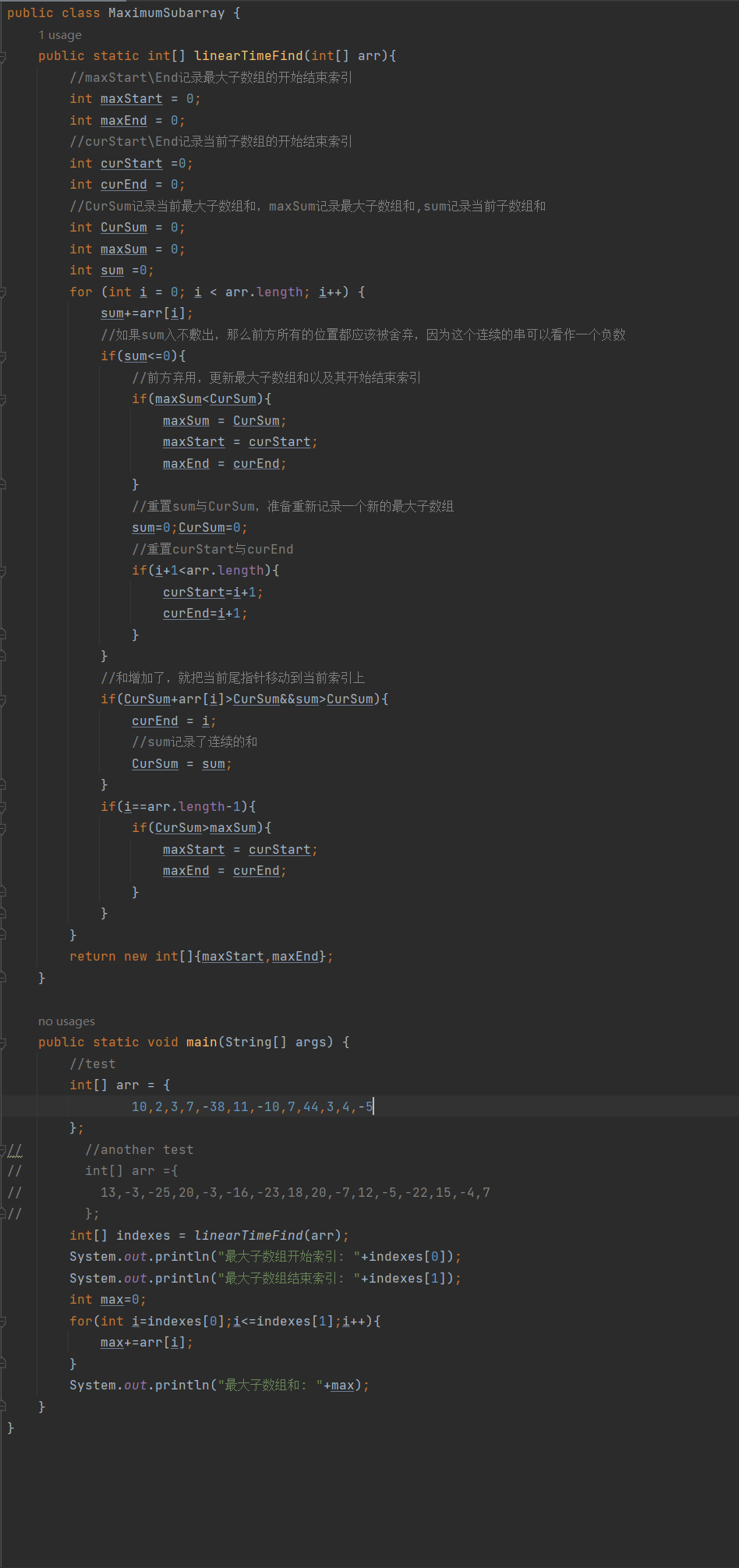
这样我们就找出最大子数组是元素11一直到元素4了。

代码具体实现与测试:

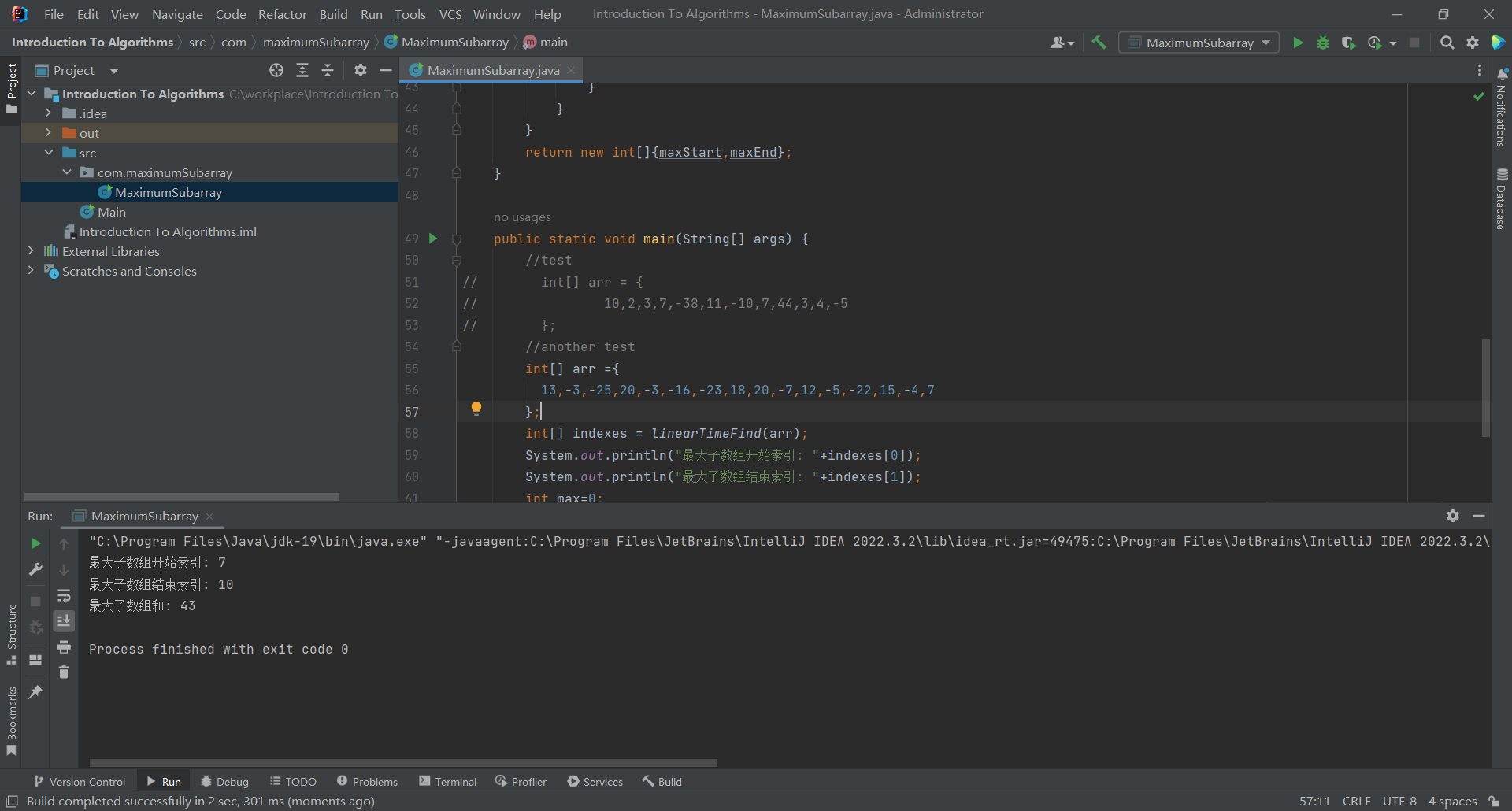
用的是java编写

package com.maximumSubarray;  
  
public class MaximumSubarray {  
 public static int[] linearTimeFind(int[] arr){  
 //maxStart\End记录最大子数组的开始结束索引  
 int maxStart = 0;  
 int maxEnd = 0;  
 //curStart\End记录当前子数组的开始结束索引  
 int curStart =0;  
 int curEnd = 0;  
 //CurSum记录当前最大子数组和，maxSum记录最大子数组和,sum记录当前子数组和  
 int CurSum = 0;  
 int maxSum = 0;  
 int sum =0;  
 for (int i = 0; i < arr.length; i++) {  
 sum+=arr[i];  
 //如果sum入不敷出，那么前方所有的位置都应该被舍弃，因为这个连续的串可以看作一个负数  
 if(sum<=0){  
 //前方弃用，更新最大子数组和以及其开始结束索引  
 if(maxSum<CurSum){  
 maxSum = CurSum;  
 maxStart = curStart;  
 maxEnd = curEnd;  
 }  
 //重置sum与CurSum，准备重新记录一个新的最大子数组  
 sum=0;CurSum=0;  
 //重置curStart与curEnd  
 if(i+1<arr.length){  
 curStart=i+1;  
 curEnd=i+1;  
 }  
 }  
 //和增加了，就把当前尾指针移动到当前索引上  
 if(CurSum+arr[i]>CurSum&&sum>CurSum){  
 curEnd = i;  
 //sum记录了连续的和  
 CurSum = sum;  
 }  
 if(i==arr.length-1){  
 if(CurSum>maxSum){  
 maxStart = curStart;  
 maxEnd = curEnd;  
 }  
 }  
 }  
 return new int[]{maxStart,maxEnd};  
 }  
  
 public static void main(String[] args) {  
 //test  
// int[] arr = {  
// 10,2,3,7,-38,11,-10,7,44,3,4,-5  
// };  
 //another test  
 int[] arr ={  
 13,-3,-25,20,-3,-16,-23,18,20,-7,12,-5,-22,15,-4,7  
 };  
 int[] indexes = *linearTimeFind*(arr);  
 System.*out*.println("最大子数组开始索引: "+indexes[0]);  
 System.*out*.println("最大子数组结束索引: "+indexes[1]);  
 int max=0;  
 for(int i=indexes[0];i<=indexes[1];i++){  
 max+=arr[i];  
 }  
 System.*out*.println("最大子数组和: "+max);  
 }  
}

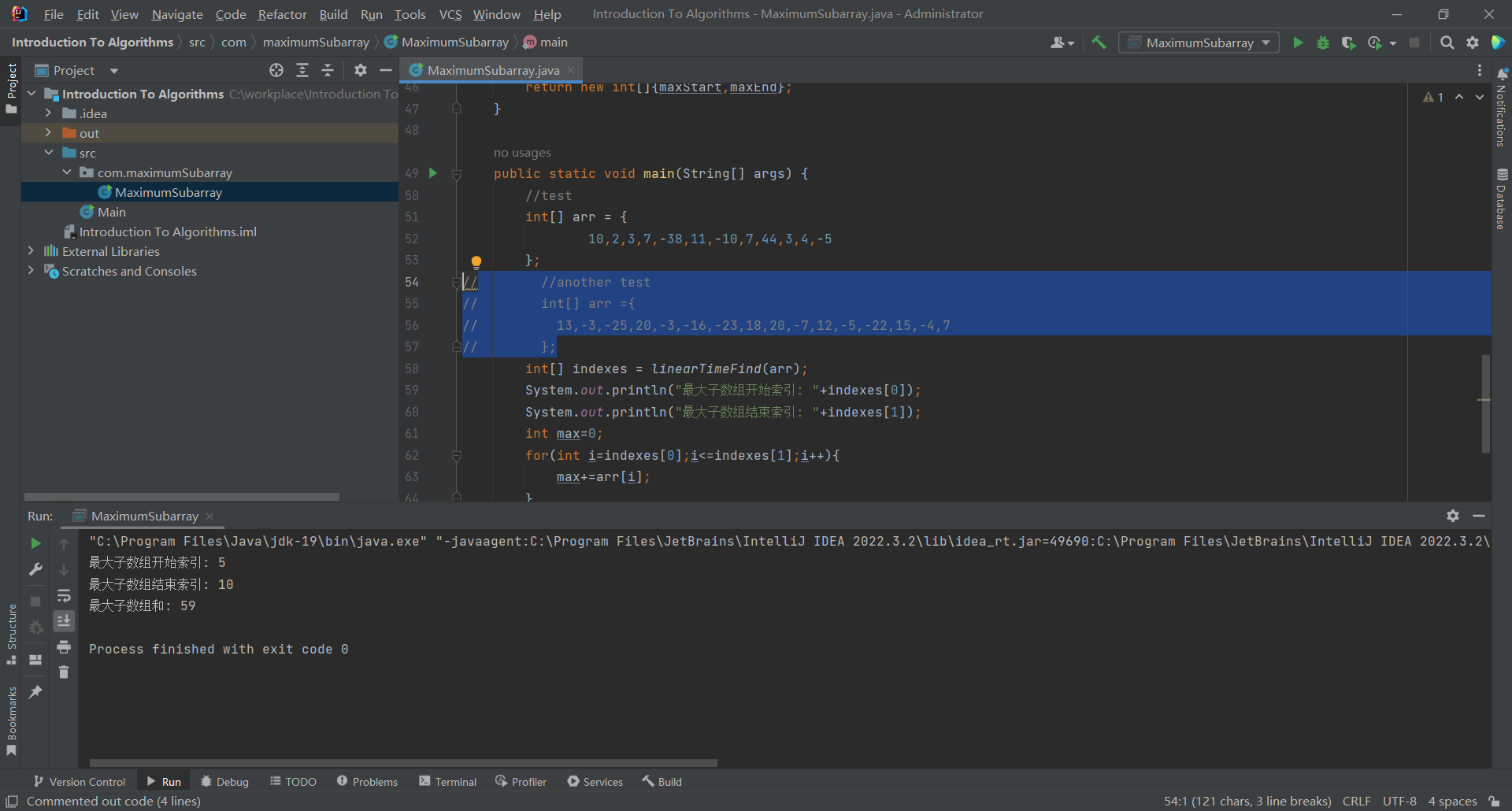
代码截图：



测试：



这是课本上的例子。答案正确。



这是我自己构造的例子，可以验证，答案正确。

算法思想（摘自题目）：

从数组左边界开始，由左至右，记录目前已处理过的最大子数组。若已知A[1,…,j]的最大子数组，可以这样扩展得到A[1,…,j+1]的最大子数组：

1. 也为A[1,…,j]
2. 为A[i,…,j+1] 其中(1<=i<=j+1)

我认为第二种情况的直观感受是，第j+1个元素起到了“扭转乾坤”的作用，它非常大，以至于之前的最大子数组需要放弃其“最大”的地位，把这个位置让渡给以该元素为代表的子数组。例如在实例中的44元素，就是这么一个“扭转乾坤”的元素。