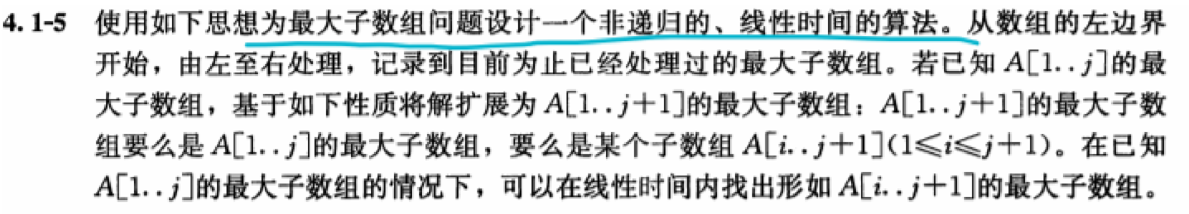
作业4.1-5的解释



4.1-5 使用如下思想为最大子数组问题设计一个非递归的、线性时间的算法。从数组的左边界开始，由左至右处理，记录到目前为止已经处理过的最大子数组。若已知 $A[1..j]$的最大子数组，基于如下性质将解扩展为 $A[1..j+1]$的最大子数组：$A[1..j+1]$的最大子数组要么是 $A[1..j]$的最大子数组，要么是某个子数组 $A[i..j+1]$($1\leqslant i\leqslant j+1$)。在已知 $A[1..j]$的最大子数组的情况下，可以在线性时间内找出形如 $A[i..j+1]$的最大子数组。

> Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far. Knowing a maximum subarray $A[1..j]$, extend the answer to find a maximum subarray ending at index $j+1$ by using the following observation: a maximum subarray $A[i..j+1]$, for some $1 \le i \le j+1$. Determine a maximum subarray of the form $A[i..j+1]$ in constant time based on knowing a maximum subarray ending at index j.

> 线性时间：算法总运行时间为O(n)

中英文意思相同，翻译没问题。这题的思想就是课上讲的O(n)算法，我把代码post出来，你们尝试解释并给出伪码：

```
import math

# ========== 分治法 ==============


def FindMaxCrossingSubArray(A, low, mid, high):
    # 寻找max sum[i,mid]
    lowPartMaxSum = A[mid]
    tmpSum = A[mid]
    i = mid
    if low < mid:
        for k in range(mid-1, low-1, -1):
            tmpSum += A[k]
            if tmpSum > lowPartMaxSum:
                lowPartMaxSum = tmpSum
                i = k
    # 寻找下半部分的max sum[mid+1,j]
    tmpSum = A[mid+1]
    highPartMaxSum = A[mid+1]
    j = mid+1
    if mid+1 < high:
        for k in range(mid+2, high+1):
            tmpSum += A[k]
            if tmpSum > highPartMaxSum:
                highPartMaxSum = tmpSum
                j = k
    return i, j, lowPartMaxSum+highPartMaxSum


def FindMaxSubArrayByDivision(A, low, high):
    if low == high:
```

```python
        return low, high, A[low]
    mid = math.floor((low+high)/2)
    lowPartMaxI, lowPartMaxJ, lowPartMaxSum = FindMaxSubArrayByDivision(
        A, low, mid)
    highPartMaxI, highPartMaxJ, highPartMaxSum = FindMaxSubArrayByDivision(
        A, mid+1, high)
    crossingMaxI, crossingMaxJ, crossingMaxSum = FindMaxCrossingSubArray(
        A, low, mid, high)
    # 比较三部分的max，确定全局max
    if lowPartMaxSum > highPartMaxSum:
        if lowPartMaxSum > crossingMaxSum:
            return lowPartMaxI, lowPartMaxJ, lowPartMaxSum
        else:
            return crossingMaxI, crossingMaxJ, crossingMaxSum
    else:
        if highPartMaxSum > crossingMaxSum:
            return highPartMaxI, highPartMaxJ, highPartMaxSum
        else:
            return crossingMaxI, crossingMaxJ, crossingMaxSum


# ========== O(n)方法的两个版本 ============


def FindMaxSubArray(A, n):
    # 完全按照课上思路走的O(n)方法

    # max变量记录全局最大sum和下标
    # maxSum初始为A[0]而不是0，应对全为负数的数组
    maxSum = A[0]
    maxI, maxJ = 0, 0
    # current变量记录当前最大sum和下标
    curSum = 0
    curI, curJ = 0, 0
    for k in range(n):
        curSum += A[k]
        curJ = k
        if curSum > maxSum:
            maxSum = curSum
            maxI, maxJ = curI, curJ
        # curSum如果≤0，对后面的贡献非正，舍弃，从k+1重新开始
        if curSum <= 0:
            curI = curJ = k+1
            # 赋值为0，相当于舍弃之前的和
            curSum = 0
    print(f"max = {maxSum} in [{maxI},{maxJ}]")


def FindMaxSubArray2(A, n):
    # O(n)方法的另一个版本，与上个版本思想是一致的
    # 这个版本更直接反映题意

    # max变量记录全局最大sum和下标
    # # maxSum初始为A[0]而不是0，应对全为负数的数组
    maxSum = A[0]
    maxI, maxJ = 0, 0
    # maxSumEndAtRightBound变量记录迭代过程中，以右边界结尾的最大子数组和
    maxSumEndAtRightBound = 0
    mseI, mseJ = 0, 0
    for k in range(n):
        # 由上次迭代的maxSumEndAtRightBound，确定这次迭代的maxSumEndAtRightBound
        if maxSumEndAtRightBound + A[k] > A[k]:
            maxSumEndAtRightBound += A[k]
            mseJ = k
        else:
            maxSumEndAtRightBound = A[k]
            mseI = mseJ = k
        # 由上次迭代的maxSum，即A[..k-1]的最大子数组和
        # 以及这次迭代的maxSumEndAtRightBound
        # 确定这次迭代的maxSum
        if maxSumEndAtRightBound > maxSum:
            maxSum = maxSumEndAtRightBound
            maxI, maxJ = mseI, mseJ
    print(f"max = {maxSum} in [{maxI},{maxJ}]")


# ================ 测试 ===============
testCaseNo = 0
for A in [
    [1, 2, 3, 4, -5, 10, -1, -1],
    [1, -1, 1, -1],
    [-3, -2, -1],
    [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7],
    [-99, 100]
]:
    testCaseNo += 1
    print(f"==========Test Case {testCaseNo} =========")
    n = len(A)

    print("分治法O(nlogn)：")
    i, j, maxSum = FindMaxSubArrayByDivision(A, 0, n-1)
    print(f"max = {maxSum} in [{i},{j}]")

    print("O(n)方法的两个版本：")
    FindMaxSubArray(A, n)
    FindMaxSubArray2(A, n)

    print()
```