程序设计艺术与方法

第六讲　专题解析

——代码优化

合肥工业大学计算机与信息学院　2013-12

The Problem and a Simple Program

- **The problem arose in one-dimensional pattern recognition**
- 输入是**n**维实数数组***x***; 输出连续子数组的和为最大的（最大字段和）.
- 例如，输入数组如下：

31	−41	59	26	−53	58	97	−93	−23	84
		↑ 3				↑ 7			

- 则返回子数组 **X[3 .. 7]**, 和为 **187.**

简单求解程序

- **程序显然简单**: 对于每一对整数 L 和 U (1 < L < U < N)
  - 计算出 *X*[*L*.. U] 的和
  - 检查该和是否大于到目前为止的最大和
- **代码示意如下： 时间复杂度 O(n^3)**

```
MaxSoFar = 0
for i = [0, n)
  for j = [i, n)
      sum = 0
      for k = [i, j]
          sum += x[k]
          /* Sum now contains the sum of X[i..j] */
          MaxSoFar := max(MaxSoFar, Sum)
```

平方级算法（一）

- 注意到X[L .. U] 之间的内在联系
- **X[L .. U]的和等于X[L .. U-1]的和 + X[U]**
- 时间复杂度可以降为平方级

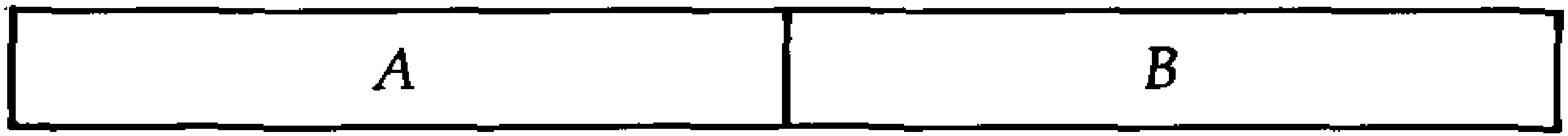
```
MaxSoFar = 0
for i = [0, n)
  sum = 0
  for j = [i, n)
      sum += x[j]
      MaxSoFar := max(MaxSoFar, Sum)
```

平方级算法（二）

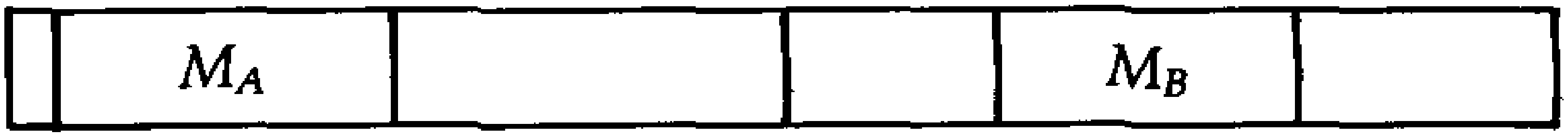
- 另一平方计算法
- 建立辅助数组**cumarr**， cumarr[i] = S **X[1..*i*]**
- S ***X***[i..j] = cumarr[***j***] - cumarr[***j* - 1]**

```
cumarr[-1] = 0
for i = [0, n)
    cumarr[i] = cumarr[i-1] + x[i]
MaxSoFar = 0
for i = [0, n)
    for j = [i, n)
      sum = cumarr[ j ] – cumarr[i-1]
      /* sum is sum of x[i..j] */
      MaxSoFar := max(MaxSoFar, Sum)
```

- It is based on the following divide-and-conquer schema:
  - *To solve a problem of size N, recursively solve two subproblems of size approximately N/2, and combine their solutions to yield a solution to the complete problem.*
- The original problem deals with a vector of size N, the natural way to divide it into subproblems is to create two subvectors of approximately equal size, called A and B:



- We then recursively find the maximum subvectors in A and B, which we'll call M_A and M_B:



- It is tempting to think that we have solved the problem because the maximum sum subvector of the entire vector must be either M_A or M_B, or it crosses the border between A and B (which called M_C for the maximum *crossing* the border):

分治算法

- **Thus our divide-and-conquer algorithm will compute M_A and M_B recursively, compute M_C by some other means, and then return the maximum of the three**

```
int maxsum(l, u)
    if (l > u) /* zero elements */
        return 0
    if (l == u) /* 1 elements */
        return max(0, x[l])
    m = (l + u) / 2
    /* find max crossing to left */
    lmax = sum = 0
    for (i=m; i>0; --i)
        sum += x[i]
        lmax = max(lmax, sum)
    /* find max crossing to right */
    rmax = sum = 0
    for (i=m+1; i<=u; ++i)
        sum += x[i]
        rmax = max(rmax, sum)
    return max( lmax+rmax, maxsum(l, m), maxsum(m+1, u) )
```

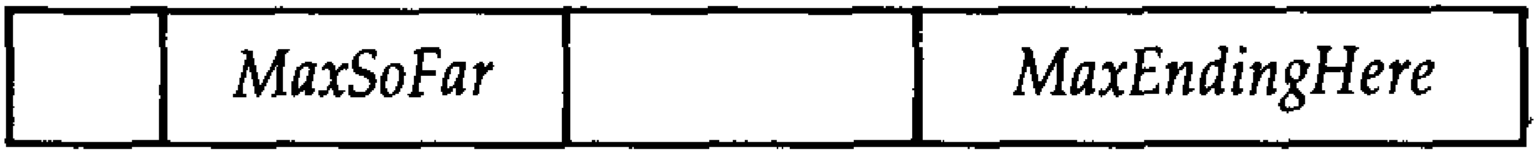
分治算法

- answer = maxsum(0, n-1)

- T(1) = O(1) and
  T(N) = 2T(N/2) + *O(N)*
  so T(N) = *O(N* log N)

扫描算法

- The maximum is initially zero
- Suppose we've solved the problem for X[1 .. *i* - 1]
- how can we extend that to a solution for the first *i* elements?
- the maximum sum in the first *i* elements is either the maximum sum in the first *i* - 1 elements (call *MaxSoFar*)



- … on *i* (call *MaxEndingHere*):

扫描算法

```
maxsofar = 0
maxendinghere = 0
for i = [0, n)
    // invariant: maxendinghere and maxsofar
    //are accurate for x[0..i-1]
    maxendinghere = max(maxendinghere+x[i], 0)
    maxsofar = max(maxsofar, maxendinghere)
```

最大子矩阵和问题

- 最大子段和推广到二维
- 描述：

  给定一个**2**维的整数矩阵，找出其中具有最大和的子矩阵。一个矩阵的和就是矩阵中所有元素的和。本题中，具有最大和的子矩阵称为最大子矩阵。子矩阵是指位于整个矩阵中任何一个**1**×**1**或更大的连续的子矩阵。

最大子矩阵和示例

- 例如，在矩阵

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

中，最大子矩阵在其左下角

9	2
-4	1
-1	8

最大子矩阵和解析

- **最大子段和问题的二维推广.即给定一个m行n列的矩阵,求其一个子矩阵,行数从r_1~r_2,列数从c_1~c_2,使之全部元素之和为最大**
- **可以将最大子段和的动态规划解法推广到上述二维情况.其基本思路为:**
  - 若始行i_1与末行i_2已给定,则求以i_1起始以i_2结束的最大子矩阵之和,即等于一个一维的最大子段和问题,
  - 数组x中元素x[j]是第j列里从第i_1行加到第i_2行的所有元素之和. 令$t[i_1,i_2]$表示这个行从i_1到i_2的最大子矩阵和,则求全矩阵的最大子矩阵之和的问题就等于在$1<=i_1<=i_2<=m$的范围中使$t[i_1,i_2]$最大化
  - 上述算法的时间复杂度为O(m^2*n)
  - 整个问题的解决本质上一维最大子段和的问题, 不过在另一个维度——行上面,则还是枚举所有的$1<=i_1<=i_2<=m$用打擂的方法比较出最大者.
  - 此方法仍然只是在列这个维度上用到了动态规划.