4.1 最大子数组问题

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
1  left-sum = −∞
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > left-sum
6          left-sum = sum
7          max-left = i
8  right-sum = −∞
9  sum = 0
10 for j = mid + 1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
1  if high == low
2      return (low, high, A[low])
3  else mid = ⌊(low + high)/2⌋
4      (left-low, left-high, left-sum) = FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      (right-low, right-high, right-sum) = FIND-MAXUMUM-SUBARRAY(A, mid + 1, high)
6      (cross-low, cross-high, cross-sum) = FIND-MAXIMUM-SUBARRAY(A, low, mid, high)
7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
8          return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10         return (right-low, right-high, right-sum)
11     else return (corss-low, cross-high, cross-sum)
```

分治算法的分析

首先，第 1 行花费常量时间，对于 $n = 1$ 的基本情况，也很简单：第 2 行花费常量时间，因此，

$$T(1) = \Theta(1)$$

当 $n > 1$ 时为递归情况。第 1 行和第 3 行花费常量时间，第 4 行和第 5 行求解的子问题均为 $n/2$ 个元素的子数组 (假定原问题规模为 2 的幂，保证了 $n/2$ 为整数)，因此每个子问题的求解时间为 $T(n/2)$。因为需要求解两个子问题，因此第 4 行和第 5 行给总运行时间增加了 $2T(n/2)$。又因为第 6 行花费

$\Theta(n)$ 时间，第 7-11 行仅花费 $\Theta(1)$ 时间。所以对于递归情况，我们有：

$$T(n) = \begin{cases} \Theta(1) & \text{若 } n = 1 \\ 2T(n/2) + \Theta(n) & \text{若 } n > 1 \end{cases}$$

上面递归式的解为 $T(n) = \Theta(nlgn)$。

练习

4.1-1 当 A 的所有元素均为负数时，FIND-MAXIMUM-SUBARRAY 返回什么？
返回其中最大的那个负数及其在数组中的位置。

4.1-2 对最大子数组问题，编写暴力求解方法的伪代码，其运行时间应该为 $\Theta(n^2)$

```
FIND-MAXIMUM-SUBARRAY-VIOLENCE(A)
max-sum = A[0]
left = 0
right = 0
temp-sum = 0
for i = 1 to A.length - 1
    temp-sum = A[i]
    for j = i + 1 to A.length
        temp-sum = temp-sum + A[j]
        if temp-sum > max-sum
            max-sum = temp-sum
            left = i
            right = j
return (left, right, max-sum)
```

4.1-4 假定修改最大子数组问题的定义，允许结果为空子数组，其和为 0。你应该如何修改现有算法，使它们能允许空子数组为最终结果？
只需将 FIND-MAX-CROSSING-SUBARRAY 修改如下 (注意，这里假定数组下标从 1 开始)：

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

```
left-sum = 0
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = 0
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

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 $A[1\ldots j]$ 的最大子数组，基于如下性质讲解扩展为 $A[1\ldots j+1]$ 的最大子数组:$A[1\ldots j+1]$ 的最大子数组要么是 $A[1\ldots j]$ 的最大子数组，要么是某个子数组 $A[i\ldots j+1](1\leq i\leq j+1)$。在已知 $A[1\ldots j]$ 的最大子数组的情况下，可以在线性时间内找出形如 $A[i\ldots j+1]$ 的最大子数组。

FIND-MAXIMUM-SUBARRAY-LINEAR(A)

```
left = 0
right = 0
temp-sum = A[0]
max-sum = temp-sum
for j = 1 to A.length
    temp-sum = temp-sum + a[j]
    if max-sum < temp-sum
        right = j
        max-sum = temp-sum
    else
        if A[j] > temp-sum
            left = j
            right = j
            temp-sum = A[j]
            max-sum = temp-sum
return (left, right, max-sum)
```

4.2 矩阵乘法的 Strassen 算法

SQUARE-MATRIX-MULTIPLY(A, B)

1 $n = A.rows$
2 let C be a new $n \times n$ matrix
3 **for** $i = 1$ **to** n
4   **for** $j = 1$ **to** n
5         $c_{ij} = 0$
6   **for** $k = 1$ **to** n
7             $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
8 **return** C

上面过程中，由于三重 for 循环的每一重都恰好执行 n 步，而第 7 行每次执行都花费常量时间，因此花费 $\Theta(n^3)$ 时间。

矩阵乘法的一个简单的分治算法

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

1 $n = A.rows$
2 Let C be a new $n \times n$ matrix
3 **if** $n == 1$
4     $c_{11} = a_{11} \cdot b_{11}$
5 **else** *patition* A, B, *and* C
6     $C_{11} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{11})
        +SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{21})
7     $C_{12} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{12})
        +SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{22})
8     $C_{21} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{11})
        +SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{21})
9     $C_{22} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{12})
        +SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{22})
10 **return** C

注意，在具体的 C 实现中，不会真的把 A, B, C 矩阵分为 12 个新的子矩阵，我们可以计算下标来限定子矩阵的范围，例如，一对坐标表示子矩阵的起始位置，再用一个整数表示该子矩阵的行或者列数，就可以知道子矩阵的范围了，这就是上述伪代码隐藏的实现细节。另外矩阵 C 在程序开始时初始化为具有 A × B 大小的矩阵，每一次递归到基本情况时，就在 C 上进行计算，待 8 次递归调用完成后，矩阵 C 就是 A × B 的结果了。

现在来分析一下算法的运行时间。由于使用下标计算对 A, B, C 进行子矩阵分解，所以第 5 行只需 $\Theta(1)$ 的时间，注意，使用下标计算而非通过复制元素 (如果这样做则分解矩阵则需要 $\Theta(n^2)$ 的时间)

来分解矩阵对总渐进运行时间并无影响。令 $T(n)$ 表示此过程的运行时间。对 $n = 1$ 的基本情况，只需要进行一次标量乘法 (第 4 行)，因此

$$T(n) = \Theta(1)$$

当 $n > 1$ 时是递归情况，第 5 行花费 $\Theta(1)$ 的时间，第 6-9 行共 8 次递归调用，由于每次递归调用完成两个 $n/2 \times n/2$ 矩阵乘法，因此花费时间为 $T(n/2)$，8 次递归调用总时间为 $8T(n/2)$。还需要计算 6-9 行的 4 次矩阵加法。每个矩阵包含 $n^2/4$ 个元素，因此每次矩阵加法花费 $\Theta(n^2)$ 时间。由于矩阵加法的的次数是常数，第 6-9 行进行矩阵假发的总时间为 $\Theta(n^2)$(这里仍然使用下标计算的方法讲矩阵加法的结果放置于 C 的正确位置，由此带来的额外开销为每个元素 $\Theta(1)$)。因此，递归情况的总时间为分解时间、递归调用时间以及矩阵加法时间之和：

$$T(n) = \Theta(1) + 8\Theta(n/2) + \Theta(n^2) = 8T(n/2) + \Theta(n^2)$$

注意，如果通过复制元素来实现矩阵的分解，额外开销为 $\Theta(n^2)$，递归式不会发生改变，只是总运行时间将会提高常数倍。综上，

$$T(n) = \begin{cases} \Theta(1) & \text{若 n=1} \\ 8\Theta(n/2) + \Theta(n^2) & \text{若 n>1} \end{cases}$$

利用后面学习的主定理，得到的解为 $T(n) = \Theta(n^3)$，因此简单的分治算法并不优于暴力算法。注意，因子 8 决定了递归树中每个节点有几个子节点，进而决定了树的每一层为总和贡献了多少项。如果省略因子 8，递归树就变为线性结构，而不是“茂盛的”了，树的每一层只为总和贡献了一项。

Strassen 方法

Strassen 方法的核心思想是用常数次的矩阵加法的代价减少一次矩阵乘法。其递归式如下所示:

$$T(n) = \begin{cases} \Theta(1) & \text{若 n=1} \\ 7\Theta(n/2) + \Theta(n^2) & \text{若 n>1} \end{cases}$$

再次利用后面的主定理，$T(n) = \Theta(n^{lg7})$。

练习

4.2-1 使用 Strassen 算法计算如下矩阵乘法:

$$\begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \quad \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$$

给出计算过程。

解: 我们从步骤 2 开始计算 $S_1, S_2, \cdots, S_{10}$:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} = \begin{bmatrix} 8 \end{bmatrix} - \begin{bmatrix} 2 \end{bmatrix} = \begin{bmatrix} 6 \end{bmatrix} \\
S_2 &= A_{11} + A_{12} = \begin{bmatrix} 1 \end{bmatrix} + \begin{bmatrix} 3 \end{bmatrix} = \begin{bmatrix} 4 \end{bmatrix} \\
S_3 &= A_{21} + A_{22} = \begin{bmatrix} 7 \end{bmatrix} + \begin{bmatrix} 5 \end{bmatrix} = \begin{bmatrix} 12 \end{bmatrix} \\
S_4 &= B_{21} + B_{11} = \begin{bmatrix} 4 \end{bmatrix} - \begin{bmatrix} 6 \end{bmatrix} = \begin{bmatrix} -2 \end{bmatrix} \\
S_5 &= A_{11} + A_{22} = \begin{bmatrix} 1 \end{bmatrix} + \begin{bmatrix} 5 \end{bmatrix} = \begin{bmatrix} 6 \end{bmatrix} \\
S_6 &= B_{11} + B_{22} = \begin{bmatrix} 6 \end{bmatrix} + \begin{bmatrix} 2 \end{bmatrix} = \begin{bmatrix} 8 \end{bmatrix} \\
S_7 &= A_{12} - A_{22} = \begin{bmatrix} 3 \end{bmatrix} - \begin{bmatrix} 5 \end{bmatrix} = \begin{bmatrix} -2 \end{bmatrix} \\
S_8 &= B_{21} + B_{22} = \begin{bmatrix} 4 \end{bmatrix} + \begin{bmatrix} 2 \end{bmatrix} = \begin{bmatrix} 6 \end{bmatrix} \\
S_9 &= A_{11} - A_{21} = \begin{bmatrix} 1 \end{bmatrix} - \begin{bmatrix} 7 \end{bmatrix} = \begin{bmatrix} -6 \end{bmatrix} \\
S_{10} &= B_{11} + B_{12} = \begin{bmatrix} 6 \end{bmatrix} + \begin{bmatrix} 8 \end{bmatrix} = \begin{bmatrix} 14 \end{bmatrix}
\end{aligned}
$$

然后进行步骤 3 的计算: