目录

湖南工商大学 算法导论 课程教案

授课题目（教学章、节或主题） 课时安排: 2学时

第四讲：分治策略 授课时间 :第四周周一第1、2节

教学内容（包括基本内容、重点、难点）：

基本内容：（1）基本概念：分治策略；

（2）最大子数组问题；

（3）用Strassen算法求解矩阵乘法问题

（4）求解递归式方法：代入式，递归树法，主方法

教学重点、难点：重点为分治策略原理、递归式求解方法

教学媒体的选择：本章使用大数据分析软件Jupyter教学，Jupyter集课件、Python程序运行、HTML网页制作、Pdf文档生成、Latex文档编译于一身，是算法导论课程教学的最佳选择。

板书设计：黑板分为上下两块，第一块基本定义，推导证明以及例子放在第二块。第一块 整个课堂不擦洗，以便学生随时看到算法流程图以及基本算法理论等内容。

课程过程设计： （1）讲解基本算法理论；（2）举例说明；（3）程序设计与编译；（4）对本课堂进行总 结、讨论；（5）布置作业与实验报告

第四讲：分治策略

一. 分治策略简介

分治法三个步骤：

1. 分解：将问题分解为若干子问题
2. 解决：递归地求解子问题
3. 合并：将子问题的解组合成原问题的解
   若 n 足够小，则运行时间 $T(n)=\Theta(1)$；若分解的问题个数 a，每个子问题的规模为 n/b，分解时间为 $D(n)$，组合时间为 $C(n)$，则

$$T(n)=aT(n/b)+D(n)+C(n)$$

二. 最大子数组问题

例 1 下图给出某支股票17天价格。第0天价格是每股100美元，你可以在此后任何时间买进股票。你当然希望“低价买进，高价卖出”，即在最低价价格时买进股票，之后在最高价格时卖出，这样可以最大化收益。但遗憾的是，在一段给定时期内，可能无法做到在最低价格时买进股票，然后在最高价格时卖出。例如，下图中，最低价格发生在第7天，而最高价格发生在第1天。最高价格在前，最低价在后。

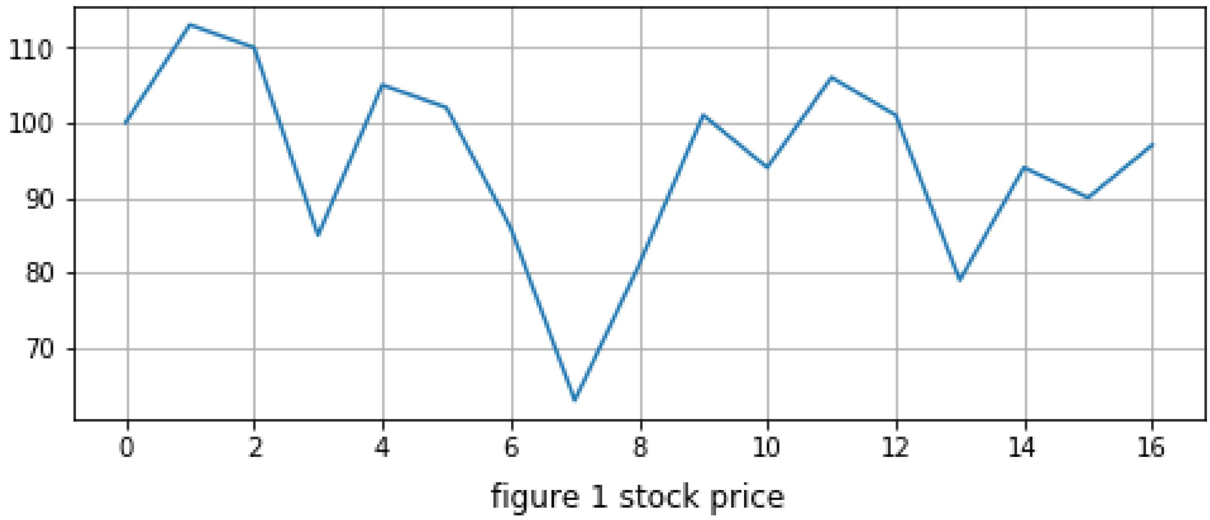
天	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
价格	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
变化		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

In [14]:

```python
# 股票价格图像
import matplotlib.pyplot as plt
import numpy as np
x=range(17)
y=[100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90,97]
plt.figure(figsize=(8,3))
plt.grid()
plt.plot(x,y)
plt.title("figure 1 stock price",y=-0.25)
```

Out[14]:

```
Text(0.5, -0.25, 'figure 1 stock price')
```



通过股票的购买与出售，经过问题转换，将前一天的当天的股票差价重新表示出来，即转为了一个最大子数组的问题，即对如下日收益数组

$$A = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]$$

找到这连续的16个数里面的连续和最大的子数组。

暴力求解方法：

如果我们用暴力求解，则简单地尝试每对可能的买进和卖出日期组合，只要卖出日期在买入日期之后即可。n 天中共有 $\binom{n}{2}$ 种日期组合。因为 $\binom{n}{2} = \Theta(n^2)$，而处理每对日期所花费的时间至少也是常量，因此，这种暴力求解方法的运行时间为 $\Omega(n^2)$。有更好的方法吗？

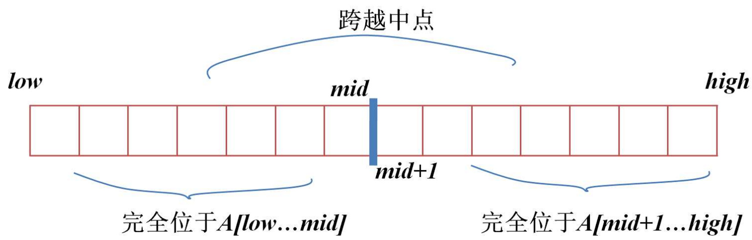
1. 最大子数组问题描述

问题： 求解一个数组中最大之和的连续子数组

思路： 分治策略求解算法，算法复杂度 $O(nlgn)$

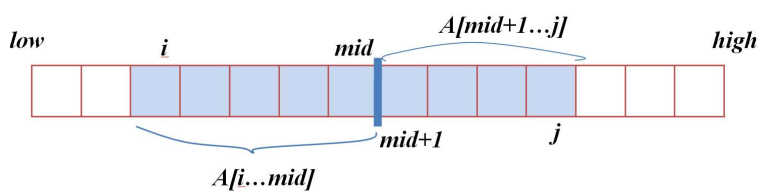
分析： 数组 $A[low, high]$（假设mid是数组 A 的中间位置）的任何连续子数组 $A[i..j]$ 所处的位置必然是以下三种情况之一：

- 完全位于子数组 $A[low..mid]$中，因此 $low \leq i \leq j \leq mid$
- 完全位于子数组 $A[mid+1..high]$中，因此 $mid \leq i \leq j \leq high$ 中，因此 $mid \leq i \leq j \leq high$
- 跨越了中点，因此 $low \leq i \leq mid < j \leq high$



我们可以递归的求解 $A[low..mid]$ 和 $A[mid+1..high]$ 的最大子数组问题，因为这两个子问题仍然是最大子数组问题，剩下的全部工作是先需要寻找跨越中点的最大子数组问题，然后在三种情况中选取和最大者。

任何跨越中点的子数组都由两个子数组 $A[i..mid]$ 和 $A[mid+1..j]$ 组成，其中 $low \leq i \leq mid$ 且 $mid < j \leq high$，因此我们只需要找出 $A[i..mid]$ 和 $A[mid+1..j]$ 的最大子数组，然后将其合并即可。



2. 最大子数组问题代码实现

分治策略伪代码：求解最大数组问题

```
findMaxCrossingSubarray(A,low,mid,high)
    sum=0
    leftSum=-9999//假设是无穷小
    for i=mid downto low
        sum=sum+A[i]
        if sum>leftSum
            leftSum=sum
            maxLeft=i
    sum=0
    rightSum=-9999
    for j=mid+1 to high
        sum=sum+A[j]
        if sum>rightSum
            rightSum=sum
            maxRight=j
    return [maxLeft,maxRight,leftSum+rightSum]

findMaximumSubArray(A,low,high)
    if low==high
        return [low,high,A[low]]
    else
        mid=floor((low+high)/2)
        [leftLow,leftHigh,leftSum]= findMaximumSubArray(A,low,mid)
        [rightLow,rightHigh,rightSum]= findMaximumSubArray(A,mid+1,high)
        [midLow,midHigh,midSum]= findMaxCrossingSubarray(A,low,mid,high)
        if(leftSum>=midSum and leftSum>=rightSum)
            return [leftLow,leftHigh,leftSum]
        else if(rightSum>=leftSum and rightSum>=midSum)
            return [rightLow,rightHigh,rightSum]
        else
            return [midLow,midHigh,midSum]
```

分治策略Python代码：求解最大数组问题

In [22]:

```
import sys
#存放初始化中的最小值
mins = -sys.maxsize

#跨越中点的子数组求解
def find_max_cross_subarray(A, low, mid, high):
    left_sum = mins
    sums = 0
    # [low, mid]区间，low-1才可以取到low值
    for i in range(mid, low-1, -1):
        sums+=A[i]
        if sums > left_sum:
            left_sum = sums
            max_left = i
    right_sum = mins
    sums = 0
    #[mid+1, high]区间，high+1才可以取到high值
    for i in range(mid+1, high+1):
        sums+=A[i]
        if sums > right_sum:
            right_sum = sums
            max_right = i
    return (max_left, max_right, left_sum + right_sum)

def find_max_subarray(A, low, high):
    if low == high:
        return (low, high, A[low])
    else:
        mid = (low+high) / 2
        mid = int(mid)
        (left_low, left_high, left_sum) = find_max_subarray(A, low, mid)
        (right_low, right_high, right_sum) = find_max_subarray(A, mid+1, high)
        (cross_low, cross_high, cross_sum) = find_max_cross_subarray(A, low,mi
        #在三种情况中选取和最大者，返回最大子数组索引值以及子数组和
        if left_sum >= right_sum and left_sum >= cross_sum:
            return (left_low, left_high, left_sum)
        elif right_sum >= left_sum and right_sum >= cross_sum:
            return (right_low, right_high, right_sum)
        else:
            return (cross_low, cross_high, cross_sum)


if __name__ == "__main__":
    A = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
    low, high, sums = find_max_subarray(A,0,len(A)-1)
    print("日收益数组为 A = ", A)
```

```
    print("最大子数组为：", A[low:high+1])
    print("买入日期、卖出日期和最大总收益分别为：",str(low)+',  '+str(high)+',
```

```
日收益数组为 A =  [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -
22, 15, -4, 7]
最大子数组为： [18, 20, -7, 12]
买入日期、卖出日期和最大总收益分别为： 7,  10,  43
```

3. 分治策略算法分析

设 $T(n)$ 表示 FIND-MAXIMUM-SUBARRAY 求解 n 个元素的最大子数组的运行时间：

- 第1、2行花费常量时间,且 $T(1) = \Theta(1)$;
- 第4、5行求解答每个子问题的求解时间为 $T(n/2)$。因为我们需要求解左右两个子问题--左子数组和右子数组，因此总运行时间为 $2T(n/2)$;
- 调用 FIND-MAX-CROSSING-SUBARRAY 花费 $\Theta(n)$;
- 其他花费 $\Theta(1)$.

因此，对于递归情况，我们有

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(n/2) + \Theta(n) & n > 1 \end{cases}$$

利用递归树或主方法，可以推得其解为 $T(n) = \Theta(n \lg n)$。

三. 矩阵乘法的Strassen算法

1. 矩阵乘法问题

若矩阵 $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$ 均为 $n \times n$ 方阵，$i, j = 1, 2, \cdots, n$。定义矩阵乘积 $C = A \cdot B$ 为:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

我们需要计算 n^2 个矩阵元素， 每个元素又是 n 个值的和。**以下为矩阵乘法伪代码：**

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

```
    for j=1 to n
        c_ij = 0
        for k=1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

因为有三重for 循环，因此SQUARE-MATRIX-MULTIPLY花费时间 $\Theta(n^3)$时间。是否有更节省时间的计算方法呢?

接下来我们将会看到Strass的著名 $n \times n$ 矩阵相乘的递归算法的运行时间为 $\Theta(n^{\lg 7})$。由于 $\lg 7$ 介于 2.80 和 2.81 之间，因此，Strass算法的运行时间为 $O(n^{2.81})$, 渐近复杂性优于简单的SQUARE-MATRIX-MULTIPLY过程。

2. 一个简单的分治算法

为简单起见，我们假设 n 为 2 的幂。将每个矩阵都分块成4个大小相等的子矩阵，每个子矩阵都是$n/2 \times n/2$ 的方阵，

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

将矩阵乘积公式改写为:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

分治降阶，子矩阵阶数大于2时，可将其分块，直到子矩阵的阶为2。计算2个 n 阶方阵的乘积转化为计算8个 $n/2$ 阶方阵的乘积和4个 $n/2$ 阶方阵的加法。时间复杂度

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

从而由主方法可推得，$T(n) = \Theta(n^3)$, 简单递归式不比原始定义方法直接计算更有效，只是总运行时间会提高常数倍。**以下为简单递归分治算法的伪代码:**

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A,B)
n=A.rows
let C be a new n × n matrix
```

if n==1

$c_{11} = a_{11} b_{11}$

else partition A,B and C as in equations (4.9)

C_{11} = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{11})
+SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{21})

C_{12} = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{12})
+SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{22})

C_{21} = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{11})
+SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{21})

C_{22} = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{12})
+SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{22})

return C

3. Strassen 算法

- 我们假设 n 为 2 的幂。也将每个矩阵都分块成4个大小相等的子矩阵，每个子矩阵都是 $n/2 \times n/2$ 的方阵，

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- 然后创建如下10个矩阵：

$$\begin{aligned} S_1 &= B_{12} - B_{22}, & S_2 &= A_{11} + A_{12} \\ S_3 &= A_{21} + A_{22}, & S_4 &= B_{21} - B_{11} \\ S_5 &= A_{11} + A_{22}, & S_6 &= B_{11} + B_{22} \\ S_7 &= A_{12} - A_{22}, & S_8 &= B_{21} + B_{22} \\ S_9 &= A_{11} - A_{21}, & S_{10} &= B_{11} + B_{12} \end{aligned}$$

由于必须进行 10 次 $n/2 \times n/2$ 矩阵的加减法，因此，该步骤花费 $\Theta(n^2)$ 时间。

- 接下来递归地计算 7 次 $n/2 \times n/2$ 矩阵的乘法：

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\ P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \\ P_7 &= S_9 \cdot S_10 = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \end{aligned}$$

该步骤花费 $7T(n/2)$ 时间。上述公式中，只有中间一列的乘法时需要计算的，右边这一列只是用于说明这些乘积与原始子矩阵的关系。

- 接下来对创建的 P_i 矩阵进行加减法运算：

$$\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}$$

由于进行8次 $n/2 \times n/2$ 矩阵的加减法，因此，该步骤花费 $\Theta(n^2)$ 时间。

综合上述，我们看到由4个步骤构成的 Strassen 算法，时间复杂度为：

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

从而由主方法可推得，$T(n) = \Theta(n^{\lg 7})$，Strassen算法的渐近复杂性低于简单的SQUARE-MATRIX-MULTIPLY过程。

Strassen算法Python代码1：实现矩阵乘法

In [7]:

```
import numpy as np
import math

def strassen_algorithm(A, B, L1, L2):
    n = int(L1[1]) - int(L1[0]) + 1
    d = math.floor(n / 2 - 1)  # the half length of matrix width/height minus
    # if the matrix's length is 1, then set whose value
    C = np.zeros((n, n))
    if n <= 0:
        return
    if n == 1:
        C[n - 1, n - 1] = A[L1[3], L1[0]] * B[L2[3], L2[0]]
    else:
        a11 = [L1[0], L1[0] + d, L1[2], L1[2] + d]
        a12 = [int((L1[0] + L1[1] + 1) / 2), L1[1], L1[2], L1[2] + d]
        a21 = [L1[0], L1[0] + d, int((L1[2] + L1[3] + 1) / 2), L1[3]]
        a22 = [int((L1[0] + L1[1] + 1) / 2), L1[1], int((L1[3] + L1[2] + 1) /
        b11 = [L2[0], L2[0] + d, L2[2], L2[2] + d]
        b21 = [L2[0], L2[0] + d, int((L2[2] + L2[3] + 1) / 2), L2[3]]
        b12 = [int((L2[0] + L2[1] + 1) / 2), L2[1], L2[2], L2[2] + d]
        b22 = [int((L2[0] + L2[1] + 1) / 2), L2[1], int((L2[2] + L2[3] + 1) /

        P1 = strassen_algorithm(A, B, a11, b12) - strassen_algorithm(A, B, a11
        P2 = strassen_algorithm(A, B, a11, b22) + strassen_algorithm(A, B, a12
        P3 = strassen_algorithm(A, B, a21, b11) + strassen_algorithm(A, B, a22
        P4 = strassen_algorithm(A, B, a22, b21) - strassen_algorithm(A, B, a22
        P5 = strassen_algorithm(A, B, a11, b11) + strassen_algorithm(A, B, a11
             strassen_algorithm(A, B, a22, b11) + strassen_algorithm(A, B, a22
        P6 = strassen_algorithm(A, B, a12, b21) + strassen_algorithm(A, B, a12
             strassen_algorithm(A, B, a22, b21) - strassen_algorithm(A, B, a22
        P7 = strassen_algorithm(A, B, a11, b11) + strassen_algorithm(A, B, a11
             strassen_algorithm(A, B, a21, b11) - strassen_algorithm(A, B, a21

        C[0:d + 1, 0:d + 1] = P5 + P4 - P2 + P6
        C[0:d + 1, int(n / 2):int(n / 2 + d + 1)] = P1 + P2
        C[int(n / 2):int(n / 2 + d + 1), 0:d + 1] = P3 + P4
        C[int(n / 2):int(n / 2 + d + 1), int(n / 2):int(n / 2 + d + 1)] = P5
    return C

n = 8
A = np.random.randint(0, 100, size=[n, n])
B = np.random.randint(0, 100, size=[n, n])

L1 = [0, n - 1, 0, n - 1]
L2 = [0, n - 1, 0, n - 1]
ret = strassen_algorithm(A, B, L1, L2)
```

```
# ret2 = np.dot(A, B)
# print(ret2)

print(ret)
```

```
[[16022. 13965. 12515. 16861. 20177. 13328.  8132. 13178.]
 [18666. 21852. 14121. 18533. 23265. 19257. 10862. 12676.]
 [20290. 19674. 15689. 14594. 19916. 16158. 15040. 14293.]
 [17845. 12900. 12275. 16874. 13079. 11870.  7226.  9665.]
 [27941. 24177. 22977. 26368. 28560. 20476. 16144. 20524.]
 [26049. 24209. 22954. 23286. 27722. 21450. 17543. 18415.]
 [22792. 22226. 20373. 15806. 18506. 17114. 17157. 13008.]
 [22781. 24848. 17724. 19887. 25144. 17809. 17135. 18096.]]
```

Strassen算法Python代码2：实现矩阵乘法

In [6]:

```
import numpy as np
def divide11(a,n):
    k=int(n/2)
    a11=[ [ [0] for i in range(0,k)]  for j in range(0,k)]#初始化矩阵
    for i in range(0,k):
        for j in range(0,k):
            a11[i][j]=a[i][j]
    return a11

def divide12(a,n):
    k=int(n/2)
    a12=[ [ [0] for i in range(0,k)]  for j in range(0,k)]
    for i in range(0,k):
        for j in range(0,k):
            a12[i][j]=a[i][j+k]
    return a12

def divide21(a,n):
    k=int(n/2)
    a21=[ [ [0] for i in range(0,k)]  for j in range(0,k)]
    for i in range(0,k):
        for j in range(0,k):
            a21[i][j]=a[i+k][j]
    return a21

def divide22(a,n):
    k=int(n/2)
    a22=[ [ [0] for i in range(0,k)]  for j in range(0,k)]
    for i in range(0,k):
        for j in range(0,k):
            a22[i][j]=a[i+k][j+k]
    return a22

def Merge(a11,a12,a21,a22,n):
    k=int(2*n)
    a = [[[0] for i in range(0, k)] for j in range(0, k)]
    for i in range(0,n):
        for j in range(0,n):
            a[i][j]=a11[i][j]
            a[i][j+n]=a12[i][j]
            a[i+n][j]=a21[i][j]
            a[i+n][j+n]=a22[i][j]
    return a

def Plus(a,b,n):
    c=[[[0] for i in range(0, n)] for j in range(0, n)]
```

```
    for i in range(0,n):
        for j in range(0,n):
            c[i][j]=a[i][j]+b[i][j]
    return  c

def Minus(a,b,n):
    c=[[[0] for i in range(0, n)] for j in range(0, n)]
    for i in range(0,n):
        for j in range(0,n):
            c[i][j]=a[i][j]-b[i][j]
    return  c

def Strassen(a, b, n):
    k = n
    if k == 2:
        d = [[[0] for i in range(2)] for i in range(2)]
        d[0][0] = a[0][0] * b[0][0] + a[0][1] * b[1][0]
        d[0][1] = a[0][0] * b[0][1] + a[0][1] * b[1][1]
        d[1][0] = a[1][0] * b[0][0] + a[1][1] * b[1][0]
        d[1][1] = a[1][0] * b[0][1] + a[1][1] * b[1][1]
        return d
    else:
        a11 = divide11(a, n)
        a12 = divide12(a, n)
        a21 = divide21(a, n)
        a22 = divide22(a, n)
        b11 = divide11(b, n)
        b12 = divide12(b, n)
        b21 = divide21(b, n)
        b22 = divide22(b, n)
        k = int(n / 2)
        m1 = Strassen(a11, Minus(b12, b22, k), k)
        m2 = Strassen(Plus(a11, a12, k), b22, k)
        m3 = Strassen(Plus(a21, a22, k), b11, k)
        m4 = Strassen(a22, Minus(b21, b11, k), k)
        m5 = Strassen(Plus(a11, a22, k), Plus(b11, b22, k), k)
        m6 = Strassen(Minus(a12, a22, k), Plus(b21, b22, k), k)
        m7 = Strassen(Minus(a11, a21, k), Plus(b11, b12, k), k)

        c11 = Plus(Minus(Plus(m5, m4, k), m2, k), m6, k)
        c12 = Plus(m1, m2, k)
        c21 = Plus(m3, m4, k)
        c22 = Minus(Minus(Plus(m5, m1, k), m3, k), m7, k)
        c = Merge(c11, c12, c21, c22, k)
        return c

a=np.array([[1,2,3,4],[5,6,7,8],[4,3,2,1],[8,7,6,5]],dtype=int)
b=np.array([[1,2,3,4],[5,6,7,8],[4,3,2,1],[8,7,6,5]],dtype=int)
```

```
print(Strassen(a, b, 4))
print(np.dot(a,b))
```

```
[[55, 51, 47, 43], [127, 123, 119, 115], [35, 39, 43, 47], [107, 111, 1
15, 119]]
[[ 55  51  47  43]
 [127 123 119 115]
 [ 35  39  43  47]
 [107 111 115 119]]
```

四. 求解递归式

1 用代入法求递归式

(1) 猜测解：确定常数c
(2) 将猜测的解代入递归关系中
例：求 递归式$T_n = 2T(\lfloor n/2 \rfloor) + n$的上界
解：猜测$T_n = O(nlgn)$,利用数学归纳法
　　假设$T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor lg(\lfloor n/2 \rfloor)$成立
　　$T(n) \le 2c\lfloor n/2 \rfloor lg(\lfloor n/2 \rfloor) + n \le cnlg(n/2) + n$
　　$= cnlgn - cnlg2 + n \le cnlgn$　(当c>1)
　　边界条件：$T(1) = 1$，$T(2) = 2$，$T(3) = 5$
　　取$c \ge 2$,$T(2) \le c2lg2$且$T(3) \le c3lg3$

做出好的猜测：

1. 与见过解类似
   例如:$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$
   　　$T(n) = O(nlgn)$
2. 先证比较宽松的上下界，再减小猜测范围
   例如：$T(n) = 2T(n/2) + n$
   　下界$T(n) = \Omega(n)$
   　上界$T(n) = O(n^2)$
   　猜测$T(n) = \Theta(nlgn)$

微妙的细节：

猜测的解中减去一个低次项
例：$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
分析：猜测解为$T(n) = O(n)$

假设$T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor$，带入迭代式

$$T(n) \le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$
$$= cn + 1$$

(此时归纳法错误)

改进：猜测解为$T(n) \le cn - b$，用数学归纳法

$$T(n) \le (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$$
$$= cn - 2b + 1 \le cn - b, \quad (\text{取} b \ge 1)$$

避免陷阱:

不能直接使用渐近记号

例4：$T(n) = 2T(\lfloor n/2 \rfloor) + n$

分析：猜测解为$T(n) = O(n)$，即证明$T(n) \le cn$

$$T(n) \le c\lfloor n/2 \rfloor + n$$
$$\le cn + n$$
$$\le O(n)$$

（错误！） 结论：要证明$T(n) = O(n)$，需严格证明$T(n) \le cn$

变量替换:

例5：$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + lgn$

解：作变量替换$m = lgn$,可得

$$T(2^m) = 2T(2^{m/2}) + m$$

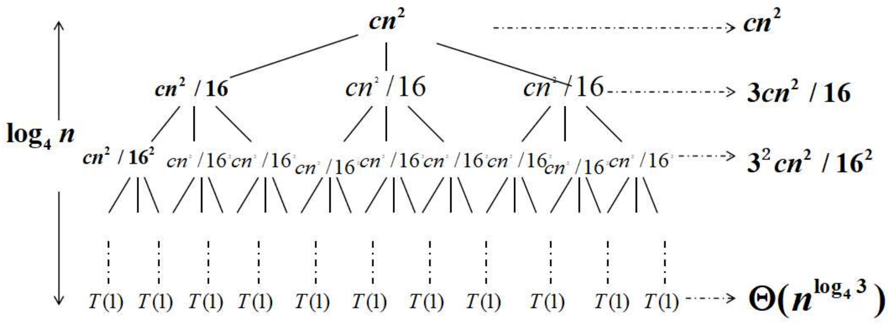
令$S(m) = T(2^m)$，上式变为$S(m) = 2S(m/2) + m$

从而可得解为$S(m) = O(mlgm)$

c$T(n) = T(2^m) = S(m) = O(mlgm) = O(lgnlglgn)$

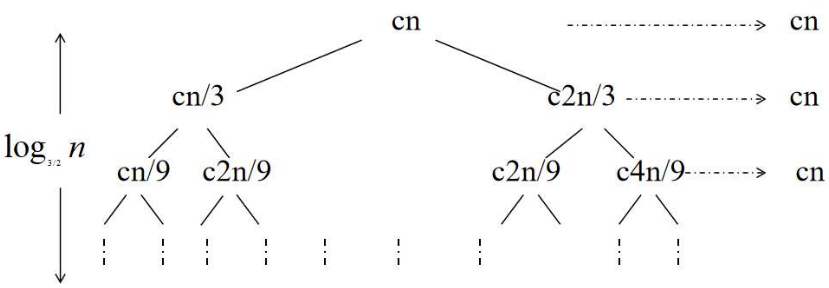
2. 用递归树法求递归式

例 6：求解递归式 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$



$$T(n) = cn^2 \sum_{k=0}^{log_4 n-1} (3/16)^k + \Theta(n^{log_4 3}) = O(n^2)$$

例 7：求解递归式 $T(n) = T(n/3) + T(2n/3) + cn^2$



$$T(n) = O(cnlog_{3/2}{}^2 n) = O(nlgn)$$

3. 用主方法求递归式

定理4.1（主定理） 令$a \geq 1$，$b > 1$是常数，$f(n)$是函数，
$T(n)$是定义在非负整数上的递归方程：
$T(n) = aT(n/b) + f(n)$
这里n/b解释为$\lfloor n \rfloor$或$\lceil n \rceil$，则$T(n)$的渐进界为：

1. 若对某常数$\varepsilon > 0$，$f(n) = O(n^{log_b a-\epsilon})$，则$T(n) = \Theta(n^{log_b a})$
2. 若$f(n) = O(n^{log_b a})$，则$T(n) = \Theta(n^{log_b a} \cdot lgn)$
3. 若对某常数$\varepsilon > 0$，$f(n) = O(n^{log_b a+\epsilon})$，且对某常数$c < 1$
   有$af(n/b) \leq cf(n)$，则$T(n) = \Theta(f(n))$

举例说明：

例1. $T(n) = 9T(n/3) + n$
　解：$a = 9$，$b = 3$，$n^{log_b a} = n^2$
　　$f(n) = n$，从而$T(n) = \Theta(n^2)$

例2. $T(n) = T(2n/3) + 1$
　解：$a = 1$，$b = 3/2$，$n^{log_b a} = 1$
　　$f(n) = \Theta(1)$，$T(n) = \Theta(lgn)$

例3. $T(n) = 3T(n/4) + nlgn$
　解：$a = 3$，$b = 4$，$n^{log_b a} = n^{log_4 3}$
　　$f(n) = nlgn$，从而$T(n) = \Theta(nlgn)$

例4. $T(n) = 2T(n/2) + nlgn$
　解：$a = 2$，$b = 2$，$n^{log_b a} = n$
　　$f(n) = nlgn$，不能使用主定理。

引用及参考：

[1]《Python数据结构与算法分析》布拉德利.米勒等著，人民邮电出版社，2019年9月.

[2] https://blog.csdn.net/zzz12122/article/details/53118822 (https://blog.csdn.net/zzz12122/article/details/53118822)

[3] https://www.52pojie.cn/thread-740397-1-1.html (https://www.52pojie.cn/thread-740397-1-1.html)

课后练习

1. 写出求解最大子数组的完整Python代码或 C语言代码。举例并代码实现

2. 写出Strassen算法求解矩阵乘法问题的完整Python代码或 C语言代码，举例并通过代码实现。

3. 利用递归树和主定理求解递归式。

讨论、思考题、作业：

参考资料（含参考书、文献等）：算法导论. Thomas H. Cormen等，机械工业出版社，2017.

授课类型（请打√）：理论课□ 讨论课□ 实验课□ 练习课□ 其他□

教学过程设计（请打√）：复习□ 授新课□ 安排讨论□ 布置作业□

教学方式（请打√）：讲授□ 讨论□ 示教□ 指导□ 其他□

教学资源（请打√）：多媒体□ 模型□ 实物□ 挂图□ 音像□ 其他□

填表说明：1、每项页面大小可自行添减；2、教学内容与讨论、思考题、作业部分可合二为一。