算法与复杂性　作业十三

516021910528 - SHEN Jiamin

2020 年 5 月 17 日

1　0511

1. 设计算法求出 n 个矩阵 $M_1, M_2, \ldots, M_n$ 相乘最多需要多少次乘法，请给出详细的算法描述和时间复杂性

给定 $n+1$ 个正整数 $c_0, c_1, \ldots, c_n$，其中 c_{i-1} 和 c_i 为矩阵 M_i 的行数和列数，$1 \le i \le n$。

记 M_{ij} 为 $M_i M_{i+1} \ldots M_j$ 的乘积，$Q(i,j)$ 为计算 M_{ij} 所需要的最多乘法数量，则

$$Q(i,j) = \begin{cases} \max_{i \le k < j} \{Q(i,k) + Q(k+1,j) + c_{i-1} c_k c_j\} & , i < j \\ 0 & , i = j \end{cases}$$

伪代码见算法1，算法的时间复杂度为 $O(n^3)$。

算法 1 矩阵最多乘法次数

输入: $n+1$ 个正整数 $c[0 \ldots n]$

输出: 最大乘法次数

```
Q[1...n][1...n] ← {0}
for j ← 2 to n do
    for i ← j − 1 down to 1 do
        precompute ← c[i − 1]c[j]
        for k ← i to j − 1 do
            t ← Q[i][k] + Q[k + 1][j] + c[k] · precompute
            Q[i][j] ← t if Q[i][j] < t
        end for
    end for
end for
return Q[1][n]
```

2. 设有算法 A 能够在 $O(i)$ 时间内计算一个 i 次多项式和一个 1 次多项式的乘积，算法 B 能够在 $O(i\log i)$ 时间内计算两个 i 次多项式的乘积。现给定 d 个整数 $n_1, n_2, \ldots, n_d$，设计算法求出满足 $P(n_1) = P(n_2) = \cdots = P(n_d) = 0$ 且最高次项系数为 1 的 d 次多项式 $P(x)$，并给出算法的时间复杂性。

满足 $P(n_1) = P(n_2) = \cdots = P(n_d) = 0$ 且最高次项系数为 1 的 d 次多项式 $P(x)$ 为

$$P(x) = (x - n_1)(x - n_2)\cdots(x - n_d)$$

记 $Q(i,j) = (x - n_i)\cdots(x - n_j)$，则 $P(x) = Q(1,d)$

$$Q(i,j) = \begin{cases} A(Q(i,j-1),(x-n_j)) & ,j-i+1 \equiv 1 \bmod 2 \\ B(Q(i,(i+j-1)/2),Q((i+j-1)/2+1,j)) & ,j-i+1 \equiv 0 \bmod 2 \end{cases}$$

伪代码见算法2

设该算法运行所需时间为 $T(d)$，则

$$T(d) = \begin{cases} 1 & ,d = 2 \\ T(d-1) + O(d-1) & ,d \equiv 1 \bmod 2 \\ 2T\left(\frac{d}{2}\right) + O\left(\frac{d}{2}\log\frac{d}{2}\right) & ,d \equiv 0 \bmod 2 \end{cases}$$

- 当 $d = 2^k$ 时，

$$\begin{aligned} T\left(2^k\right) =& 2T\left(2^{k-1}\right) + O\left((k-1)2^{k-1}\right) \\ =& 2^2T\left(2^{k-2}\right) + O\left(2(k-2)2^{k-2} + (k-1)2^{k-1}\right) \\ =& 2^2T\left(2^{k-2}\right) + O\left([(k-2)+(k-1)]\,2^{k-1}\right) \\ =& 2^{k-1}T(2) + O\left(\sum_{i=1}^{k-1}(k-i)2^{k-1}\right) \\ =& 2^{k-1} + O\left(k^2 2^{k-1}\right) = O\left(d\cdot\log^2 d\right) \end{aligned}$$

- 当 $d = 2^k - 1$ 时，记 $u_k = 2^k - 1$，且有 $u_k - 1 = 2u_{k-1}$

$$\begin{aligned} T\left(2^k-1\right) = T(u_k) =& T(u_k-1) + O(u_k-1) = T(2u_{k-1}) + O(2u_{k-1}) \\ =& 2T(u_{k-1}) + O(u_{k-1}\log(u_{k-1})) + O(2u_{k-1}) \\ =& 2^2T(u_{k-2}) + O(2u_{k-2}\log(u_{k-2}) + u_{k-1}\log(u_{k-1})) + O\left(2^2u_{k-2} + 2u_{k-1}\right) \\ =& 2^{k-2}T(u_2) + O\left(\sum_{i=2}^{k-1}2^{k-i-1}u_i\log(u_i)\right) + O\left(\sum_{i=2}^{k-1}2^{k-i}u_i\right) \\ =& O\left(3\cdot 2^{k-2}\right) + O\left(\sum_{i=2}^{k-1}2^{k-i-1}\left(2^i-1\right)i + \sum_{i=2}^{k-1}2^{k-i}\left(2^i-1\right)\right) \\ =& O\left(2^{k-2}k^2 + 3\times 2^{k-2}k + k - 3\times 2^k + 3\right) = O\left(2^k k^2\right) = O\left(d\log^2 d\right) \end{aligned}$$

算法 2 零点多项式

输入: d 个整数 $n[1 \dots d]$

输出: 多项式

```
1: procedure Q(n[1...d])
2:     if d = 1 then
3:         return (−n[d], 1)                                  ▷ x − n_d
4:     else if d ≡ 0 mod 2 then
5:         P_1 ← Q(n[1...d/2])
6:         P_2 ← Q(n[d/2+1...d])
7:         return GivenAlgoB(P_1, P_2)
8:     else
9:         P_1 ← Q(n[1...d−1])
10:        P_2 ← Q(n[d...d])
11:        return GivenAlgoA(P_1, P_2)
12:    end if
13: end procedure
14: return Q(n)
```

3. 将正整数 n 表示成一系列正整数之和: $n = n_1 + n_2 + \cdots + n_k$, 其中 $n_1 \geq n_2 \geq \cdots \geq n_k \geq 1$, $k \geq 1$。正整数 n 的这种表示称为正整数 n 的划分，例如正整数 6 有如下 11 种不同的划分:

$$
\begin{array}{ccccc}
6 & & & & \\
5+1 & & & & \\
4+2 & & 4+1+1 & & \\
3+3 & & 3+2+1 & & 3+1+1+1 \\
2+2+2 & & 2+2+1+1 & & 2+1+1+1+1 \\
1+1+1+1+1+1 & & & &
\end{array}
$$

设计算法求正整数 n 的不同划分个数并证明其时间复杂性为 $\Theta(n^2)$。

记 $Q(n, m)$ 为最大数为 m 的 n 的所有划分，即 $m = n_1 \geq n_2 \geq \cdots \geq n_k \geq 1$。当一个划分中最大的数为 k 时，划分中剩余的部分为 $Q(n-k, k)$，使用动态规划求解。

$$P(n) = \bigcup_{1 \leq m \leq n} Q(n, m)$$

其中

$$Q(n, m) = \begin{cases} \bigcup_{1 \leq k \leq m, k < n} \{k\} \times Q(n-k, k) & n > m \geq 1 \\ \{\{n\}\} & n = m \end{cases}$$

代码见算法3

求解 $P(n)$，需要求解 $\frac{n(n-1)}{2}$ 个子问题 Q，所以时间复杂度为 $\Theta(n^2)$

算法 3 正整数分划

```
Q_state = {}


def Q(n, m):
    if(n, m) in Q_state:        # 如果已经求解过该问题，不重复求解
        return Q_state[(n, m)]

    assert (n >= m)

    result = []
    if n == m:
        result = [[m]]
    else:
        r = n - m
        for k in range(1, min(r, m)+1):
            for q in Q(r, k):
                assert(max(q) <= m)
                result.append([m] + q)

    Q_state[(n, m)] = result
    return result


def solve(n):
    result = []
    for i in range(1, n+1):
        result += Q(n, i)

    return result
```

2 0514

4. 输入是由数轴上的区间所组成的集合，这些区间由它们的两个端点表示。设计 $O(n \log n)$ 算法识别所有包含在集合中其它某个区间的区间。这个问题与二维平面极大点问题有什么关系

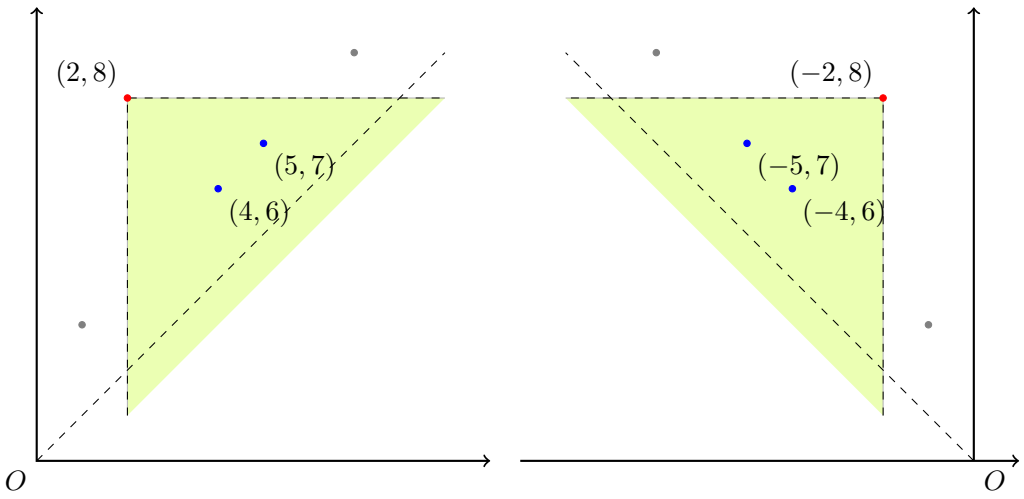
   例如输入：$(1,3),(2,8),(4,6),(5,7),(7,9)$，则输出为 $(4,6)$ 和 $(5,7)$

极大点的定义 设 $p_1 = (x_1, y_1)$ 和 $p_2 = (x_2, y_2)$ 是平面上的两个点，如果 $x_1 \leq x_2$ 并且 $y_1 \leq y_2$，则称 p_2 支配 p_1，记为 $p_1 \prec p_2$。点集 S 中的点 p 为极大点，意味着在 S 中找不到一个点 q，$q \neq p$ 并且 $p \prec q$，即 p 不被 S 中其它点支配。

区间 (x_1, y_1) 包含区间 (x_2, y_2) 当且仅当

$$x_1 \leq x_2 \wedge y_1 \geq y_2$$

将区间用平面上的点表示，所有的点都在直线 $y = x$ 上方。



(a) $(x, y) \longmapsto (x, y)$ (b) $(x, y) \longmapsto (-x, y)$

图 1: 将区间映射到二维平面上的点

为了与二维平面上极大点的定义相匹配，定义区间 (x, y) 映射到平面上的一点 $(-x, y)$。则上述区间包含的等价条件相应地变为

$$(x_2, y_2) \subseteq (x_1, y_1) \Longleftrightarrow -x_2 \leq -x_1 \wedge y_2 \leq y_1 \Longleftrightarrow (-x_2, y_2) \prec (-x_1, y_1)$$

所以该问题等价于在平面点集中，找出被其他任何一个点支配的所有点。即将区间集合映射到二维平面上的点集之后，找出二维平面中所有极大点的补集即可。

5. 证明 Graham 算法是求凸包问题的一种最优算法

在 n 个点中选取 k 个点按一定顺序构成凸包，可能有 $P(n,k)$ 种结果。由于 k 是未知的，所以共有

$$\sum_{k=3}^{n} P(n,k)$$

种可能性。任取其中一种，选中正确的凸包的概率为

$$p = \frac{1}{\sum_{k=3}^{n} P(n,k)}$$

其信息量为

$$I = -\log p = \log\left(\sum_{k=3}^{n} P(n,k)\right) = \log\left(\sum_{k=3}^{n} \frac{n!}{(n-k)!}\right) = \log\left(n!\sum_{k=3}^{n} \frac{1}{(n-k)!}\right)$$
$$= \log n! + \log\left(\sum_{k=3}^{n} \frac{1}{(n-k)!}\right) = O(n\log n)$$

一次比较操作能提供的信息量为 $\log 2$，所以凸包问题的信息论下界为 $O(n\log n)$。

Graham 算法的时间复杂度为 $O(n\log n)$，与信息论下界同阶，是最优算法。

6. 证明如果存在时间复杂度为 $O(T(n))$ 的两个 $n \times n$ 下三角矩阵的乘法，则存在时间复杂度为 $O(T(n)+n^2)$ 的任意两个 $n \times n$ 矩阵相乘的算法。

令 E 为 n 阶单位阵。设 A, B, C 均为 n 阶方阵，且 $C = AB$。则矩阵

$$Q = \begin{pmatrix} E & O & O & O \\ B & E & O & O \\ O & A & E & O \\ O & O & B & E \end{pmatrix}$$

是 $4n$ 阶的下三角方阵。则有

$$Q^2 = \begin{pmatrix} E & O & O & O \\ 2B & E & O & O \\ AB & 2A & E & O \\ O & BA & 2B & E \end{pmatrix}$$

构造矩阵 Q 需要时间 $(4n)^2$，作矩阵乘法需要时间 $T(4n)$，取出 AB 的值需要时间 n^2。

因此该算法的时间复杂度为 $O(T(4n)+17n^2)$。在假设 $T(cn) = T(n)$ 的条件下，该复杂度即为 $O(T(n)+n^2)$。

7. 如果在序列 $x_1, x_2, \ldots, x_n$ 中，存在某个 i 使 x_i 是序列中的最小者，且序列

$$x_i, x_{i+1}, \ldots, x_n, x_1, \ldots, x_{i-1}$$

是递增的，则称序列 $x_1, x_2, \ldots, x_n$ 是循环序列。设计算法找出循环序列中最小元素的位置。为简单起见，假设该位置是唯一的。证明你的算法是最优的。

遍历整个序列以寻找最小值的时间复杂度为 $O(n)$。

在 n 个数组中随机选取一个数，该数是序列中最小的值的概率为 $p = 1/n$，信息量为 $\log n$。下面给出一种时间复杂度为 $O(\log n)$ 的算法，该算法的时间复杂度达到了问题的信息论下界，是一个最优算法。

若序列 $\{x_i\}$ 是递增的，则 $\forall i, j : i < j \to x_i < x_j$。所以，若 $\exists i < j : x_i > x_j$，则该序列是非增的；又因为该序列是循环递增的，所以序列的极值点一定在 i, j 之间。

伪代码见算法4

该算法类似于二分查找，每次搜索范围减半直到找到最小值，其时间复杂度为 $O(\log n)$。

算法 4 循环递增序列的极值点

输入: 序列 $x[1 \ldots n]$

输出: $\text{argmin}_i x_i$

$left \leftarrow 1, right \leftarrow n$
if $x[left] < x[right]$ **then** ▷ 序列是有序的
  return $left$
else
  repeat
    $mid \leftarrow \frac{left+right}{2}$
    if $x[mid] < x[right]$ **then** ▷ $(mid, right)$ 是有序的
      $right \leftarrow mid$
    else ▷ $(left, mid)$ 是有序的
      $left \leftarrow mid$
    end if
  until $right - left \leq 1$
  return $right$
end if