傅申 PB20000051
2022 秋

算法基础 作业 8

4.2–3. 如何修改 Strassen 算法, 使之适应矩阵规模 n 不是 2 的幂的情况? 证明: 算法的运行时间为 $\Theta\left(n^{\lg 7}\right)$.

解: 只需要在矩阵最后追加行和列, 用零填充, 拓展到 2^k 的规模使用 Strassen 算法, 再将结果中多余的行和列去掉即可. 证明: 假设 $m = 2^k$ 是大于 n 的最小的 2 的幂, 则算法的运行时间为 $\Theta\left(m^{\lg 7}\right)$, 而显然有 $n \leqslant m < 2n$, 且

$$n^{\lg 7} \leqslant m^{\lg 7} < 7 \cdot n^{\lg 7} \tag{3.1}$$

所以算法的运行时间为 $\Theta\left(n^{\lg 7}\right)$.

30.1–1. 运用等式 30.1 和 30.2, 把下列两个多项式相乘: $A(x) = 7x^3 - x^2 + x - 10$ 和 $B(x) = 8x^3 - 6x + 3$.

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \tag{30.1}$$

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} \tag{30.2}$$

解: 如下:

$$\begin{aligned} C(x) = A(x)B(x) &= 56x^6 + (-8)x^5 + (8-42)x^4 + (-80+6+21)x^3 + (-6-3)x^2 + (60+3)x - 30 \\ &= 56x^6 - 8x^5 - 34x^4 - 53x^3 - 9x^2 + 63x - 30 \end{aligned} \tag{1.1}$$

30.1–2. 求一个次数界为 n 的多项式 $A(x)$ 在某给定点 x_0 的值存在另外一种方法: 把多项式 $A(x)$ 除以多项式 $(x - x_0)$, 得到一个次数界为 $n-1$ 的商多项式 $q(x)$ 和预想 r, 满足 $A(x) = q(x)(x - x_0) + r$. 很明显, $A(x_0) = r$. 请说明如何根据 x_0 和 A 的系数, 在 $\Theta(n)$ 的时间复杂度内计算出余项 r 以及 $q(x)$ 的系数.

解: 设 $A(x)$ 和 $q(x)$ 的系数分别为 $a_0, a_2, \cdots, a_n$ 和 $q_0, q_2, \cdots, q_{n-1}$. 可以从后向前计算, 依次有 $q_{n-1} = a_n$, $q_{n-i-1} = a_{n-i} + x_0 q_{n-i}$, $n \geqslant i \geqslant 1$, $r = a_0 + x_0 q_0$. 显然算法的时间复杂度为 $\Theta(n)$.

30.2–2. 计算向量 (0, 1, 2, 3) 的 DFT.

解: 写成多项式的形式: $A(x) = 3x^3 + 2x^2 + x$, 则

$$\begin{aligned} y_0 &= A(1) = 6 & y_1 &= A(\mathrm{i}) = -3i - 2 + i = -2 - 2\mathrm{i} \\ y_2 &= A(-1) = -3 + 2 - 1 = -2 & y_3 &= A(-\mathrm{i}) = 3\mathrm{i} - 2 - \mathrm{i} = 2\mathrm{i} - 2 \end{aligned} \tag{2.1}$$

则 (0, 1, 2, 3) 的 DFT 为 $(6, -2 - 2\mathrm{i}, -2, 2\mathrm{i} - 2)$.

30.2–5. 请把 FFT 推广到 n 是 3 的幂的情形, 写出运行时间的递归式并求解.

解: 首先有

$$\left(w_n^{k+n/3}\right)^3 = w_n^{3k+n} = w_n^{3k} = w_{n/3}^k \tag{5.1}$$

然后定义三个新的次数界为 $n/3$ 的多项式 $A^{[0]}(x)$, $A^{[1]}(x)$ 和 $A^{[2]}(x)$:

$$\begin{aligned} A^{[0]}(x) &= a_0 + a_3 x + a_6 x^2 + \cdots + a_{n-3} x^{n/3-1} \\ A^{[1]}(x) &= a_1 + a_4 x + a_7 x^2 + \cdots + a_{n-2} x^{n/3-1} \\ A^{[2]}(x) &= a_2 + a_5 x + a_8 x^2 + \cdots + a_{n-1} x^{n/3-1} \end{aligned} \tag{5.2}$$

则有

$$A(x) = A^{[0]}\left(x^3\right) + xA^{[1]}\left(x^3\right) + x^2 A^{[2]}\left(x^3\right) \tag{5.3}$$

问题就转换为了求次数界为 $n/3$ 的多项式 $A^{[0]}(x)$, $A^{[1]}(x)$ 和 $A^{[2]}(x)$ 在 $\left(w_n^0\right)^3, \left(w_n^1\right)^3, \cdots, \left(w_n^{n/3-1}\right)^3$ 上的取值. 推广后的 FFT 如下

Algorithm 1: 3-POWER-FFT(a)

```
n = a.length
if n = 1
    return a
w_n = e^{2πi/n}
w = 1
a^[0] = (a_0, a_3, ..., a_{n-3})
a^[1] = (a_1, a_4, ..., a_{n-2})
a^[2] = (a_2, a_5, ..., a_{n-1})
y^[0] = 3-POWER-FFT(a^[0])
y^[1] = 3-POWER-FFT(a^[1])
y^[2] = 3-POWER-FFT(a^[2])
for k = 0 to n/3 - 1
    y_k = y_k^[0] + w y_k^[1] + w^2 y_k^[2]
    y_{k+n/3} = y_k^[0] + w y_k^[1] e^{2iπ/3} + w^2 y_k^[2] e^{4iπ/3}
    y_{k+2n/3} = y_k^[0] + w y_k^[1] e^{4iπ/3} + w^2 y_k^[2] e^{2iπ/3}
    w = w w_n
return y
```

运行时间的递归式为

$$T(n) = 3T\left(\frac{n}{3}\right) + \Theta(n) = \Theta(n \lg n) \tag{5.4}$$

30.3–1. 请说明如何用 ITERATIVE-FFT 计算输入向量 $(0, 2, 3, -1, 4, 5, 7, 9)$ 的 DFT.

解: 首先 BIT-REVERSE-COPY 后得到 $(0, 4, 3, 7, 2, 5, -1, 9)$, 然后第一轮循环得到 $(4, -4, 10, -4, 7, -3, 8, -10)$, 第二轮循环得到 $(14, -4 - 4\mathrm{i}, -6, -4 + 4\mathrm{i}, 15, -3 - 10\mathrm{i}, -1, -3 + 10\mathrm{i})$, 最后一轮循环得到

$$y = \begin{pmatrix} 29 \\ \frac{7\sqrt{2}}{2} - 4 - \left(4 + \frac{13\sqrt{2}}{2}\right)\mathrm{i} \\ -6 - \mathrm{i} \\ -4 - \frac{7\sqrt{2}}{2} + \left(4 - \frac{13\sqrt{2}}{2}\right)\mathrm{i} \\ -1 \\ -4 - \frac{7\sqrt{2}}{2} + \left(\frac{13\sqrt{2}}{2} - 4\right)\mathrm{i} \\ -6 + \mathrm{i} \\ \frac{7\sqrt{2}}{2} - 4 + \left(4 + \frac{13\sqrt{2}}{2}\right)\mathrm{i} \end{pmatrix} \tag{1.1}$$

30.3–2. 请说明如何实现一个 FFT 算法, 注意把位逆序置换放在计算的最后而不是开始. (提示: 考虑逆 DFT)

解: 首先将 ITERATIVE-FFT 中的 w_n 修改为 w_n^{-1}, 并将计算结果的每个元素都除以 n, 就得到了求逆 DFT 的算法. 再将该算法逆序执行, 即先逆序执行外循环 ($s = \lg n \to 1$), 并将最内部的循环体作修改, 再将计算结果的每个元素都乘以 n, 最后进行位逆序置换, 就得到了 DFT. 算法如下:

Algorithm 2: ANOTHER-ITERATIVE-FFT(a)

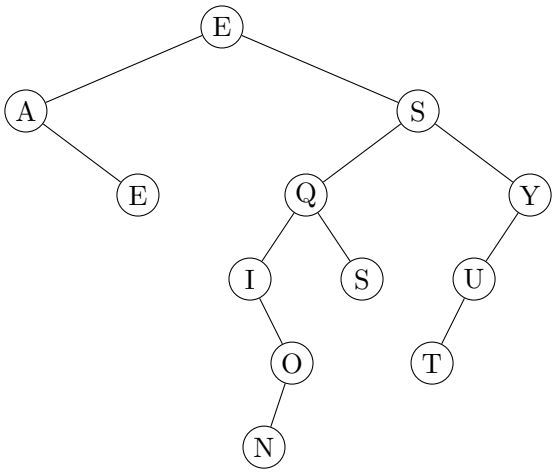
```
1  n = a.length
2  for s = lg n downto 1
3      m = 2^s
4      w_m^{-1} = e^{-2πi/m} for k = 0 to n − 1 by m
5          w = 1
6          for j = 0 to m/2 − 1
7              t = a[k + j] + a[k + j + m/2]
8              u = a[k + j] − a[k + j + m/2]
9              a[k + j] = t/2
10             a[k + j + m/2] = wu/2
11             w = w w_m^{-1}
12     for k = 0 to n − 1
13         a[k] = n a[k]
14     BIT-REVERSE-COPY(a, A)
15     return A
```

12.1–3. 因为在基于比较的排序模型中, 完成 n 个元素的排序, 其最坏情况下需要 $\Omega(n \lg n)$ 时间. 试证明: 任何基于比较的算法从 n 个元素的任意序列中构造一棵二叉搜索树, 其最坏情况下需要 $\Omega(n \lg n)$ 的时间.

解: 假设存在一个基于比较的算法, 在最坏情况下, 其能在 $O(n \lg n)$ 的时间内构造一棵二叉搜索树, 那么对于任何输入序列, 我们先调用该算法构造二叉搜索树, 再中序遍历该二叉搜索树, 并将遍历到的结点的值依次存储到数组中, 就完成了一次基于比较的排序. 其中算法的执行时间为 $O(n \lg n)$, 而中序遍历的时间为 $\Theta(n)$, 合起来是 $O(n \lg n)$, 与基于比较的排序算法最坏情况下的运行时间矛盾, 故假设不成立.

Extra.1–1. 将 "E A S Y Q U E S T I O N" 作为键按顺序插入到一棵空的二叉搜索树中, 画出生成的二叉搜索树. 构造这棵树需要多少次比较?

解: 如下



需要 $2 \times 1 + 3 \times 2 + 3 \times 3 + 2 \times 4 + 1 \times 5 = 30$ 次比较.