Homework06 2021.11.22

1

假定我们对一个数据结构执行一个由 n 个操作组成的操作序列，当 i 严格为 2 的幂时，第 i 个操作的代价为 i，否则代价为 1.

（1）使用聚合分析确定每个操作的摊还代价。

（2）使用核算法确定每个操作的摊还代价。

（3）使用势能法确定每个操作的摊还代价。

（1）由题意可知：$c_i = \begin{cases} i, i\text{为}2\text{的幂} \\ 1\text{，}otherwise \end{cases}$

$\sum_{i=1}^{n} c_i \leq \sum_{j=0}^{\lg n} 2^j + n = 2n - 1 + n < 3n$

故每个操作的摊还分析为$\frac{3n}{n} = O(1)$

（2）对于每一次操作，设置其摊还代价为3。当i不为2的幂时，用1支付操作的实际代价，剩余的2存入信用；当i为2的幂时，支付i作为操作的实际代价，不足的用信用来补全。

因此总摊还代价为$\sum_{i=1}^{n} c_i' = 3n$；由(1)可知实际摊还代价为$\sum_{i=1}^{n} c_i < 3n$，故信用始终为非负值。

所以每个操作的摊还代价为$O(1)$，n个操作的摊还代价为$O(n)$

（3）定义势函数为$\Phi(D_i) = \begin{cases} 0, i = 0 \\ 2i - 2^{1+\lfloor \lg i \rfloor}\text{，}i > 0 \end{cases}$

当$i = 1$时，$c_i' = c_i + \Phi(D_1) - \Phi(0) = 1 + 2i - 2^{1+\lfloor \lg i \rfloor} - 0 = 1$

当$i > 1$且i不是2的幂，
$c_i' = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2i - 2^{1+\lfloor \lg i \rfloor} - 2(i-1) - 2^{1+\lfloor \lg (i-1) \rfloor} = 3$

当$i > 1$且i是2的幂，$c_i' = c_i + \Phi(D_i) - \Phi(D_{i-1}) = i + 2i - 2^{1+j} - (2(i-1) - 2^{1+j-1}) = 2$

所以每个操作的摊还代价为$O(1)$，n个操作的摊还代价为$O(n)$

2

V.Pan 发现一种方法，可以用 132464 次乘法操作完成 68×68 的矩阵相乘，发现另一种方法，可以用 143664 次乘法操作完成 70×70 的矩阵相乘，还发现一种方法，可以用 155424 次乘法操作完成72×72 的矩阵乘法。当用于矩阵乘法的分治算法时，上述哪种方法会得到最佳的渐近运行时间？与Strassen 算法相比，性能如何？

由题意可知：

$log_{68} 132464 \approx 2.795128$

$log_{70} 143644 \approx 2.795162$

$log_{72} 155424 \approx 2.795147$

因此完成68×68矩阵乘法的算法有最佳渐近运行时间，且相比于Strassen算法，这种方法的性能更好一些。

3

我们可以将一维离散傅里叶变换 (DFT) 推广到 d维上。这时输入是一个d 维的数组$A = (a_{j_1,j_2,...,j_d})$，维数分别为 $n_1, n_2, ..., n_d$，其中 $n_1n_2...n_d = n$。定义d维离散傅里叶变换如下：

$y_{k_1,k_2,...,k_d} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} ... \sum_{j_d=0}^{n_d-1} a_{j_1,j_2,...,j_d} \omega_{n_1}^{j_1k_1} \omega_{n_2}^{j_2k_2} ... \omega_{n_d}^{j_dk_d}$

其中$0 \le k_1 < n_1, 0 \le k_2 < n_2, ..., 0 \le k_d < n_d$

a. 证明：我们可以依次在每个维度上计算一维的DFT 来计算一个 d 维的DFT。也就是说，首先沿着第 1维计算n/n_1 个独立的一维 DFT。然后，把沿着第 1 维的DFT 结果作为输入，我们计算沿着第 2维的 n/n_2个独立的一维 DFT。利用这个结果作为输入，我们计算沿着第三维的 n/n_3个独立的一维 DFT，如此下去，直到第 d 维。

b. 证明：维度的次序并无影响，于是可以通过在d个维度的任意顺序中计算一维 DFT来计算一个d维的 DFT。

c. 证明：如果采用计算快速傅里叶变换计算每个一维的DFT，那么计算一个d 维的 DFT 的总时间是 $O(nlgn)$，与 d 无关。

a.

$ \begin{aligned} y_{k_1,k_2,...,k_d} &=\sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}... \sum_{j_d=0}^{n_d-1}a_{j_1,j_2,...,j_d} \omega_{n_1}^{j_1k_1}\omega_{n_2}^{j_2k_2}...\omega_{n_d}^{j_dk_d} y_{k_1,k_2,...,k_d} \\&=\sum_{j_d=0}^{n_d-1}\sum_{j_{d-1}=0}^{n_{d-1}-1}... \sum_{j_1=0}^{n_1-1}a_{j_1,j_2,...,j_d} \omega_{n_1}^{j_1k_1}\omega_{n_2}^{j_2k_2}...\omega_{n_d}^{j_dk_d} \\ &=\sum_{j_d=0}^{n_d-1}\sum_{j_{d-1}=0}^{n_{d-1}-1}... (\sum_{j_1=0}^{n_1-1}a_{j_1,j_2,...,j_d} \omega_{n_1}^{j_1k_1})\omega_{n_2}^{j_2k_2}...\omega_{n_d}^{j_dk_d}\end{aligned}$

括号中计算的便是n/n_1 个独立的一维 DFT，问题的维度减一，然后将计算结果作为输入，计算沿着第 2维的 n/n_2个独立的一维 DFT，以此类推。

b.因为在$y_{k_1,k_2,...,k_d} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} ... \sum_{j_d=0}^{n_d-1} a_{j_1,j_2,...,j_d} \omega_{n_1}^{j_1k_1} \omega_{n_2}^{j_2k_2} ... \omega_{n_d}^{j_dk_d}$中求和的系数没有出现在不同求和和的边界中，所以求和顺序可以任意交换，故通过在d个维度的任意顺序中计算一维 DFT来计算一个d维的 DFT。

c.对于第k维的DFT，时间复杂度为$O(n_k \lg(n_k))$

因此，总的时间复杂度为

$T(n) = \sum_{i=1}^{d} O(n_i \lg(n_i)) = O(\sum_{i=1}^{d} n_i \lg(n_i)) \le O(n \prod_{i=1}^{d} \lg(n_i)) = O(n \lg n)$