

算法设计HomeWork_1

ZY2006109_姬轶

一、已知下列递推式：

$$C(n) = \begin{cases} 1 & \text{若 } n = 1 \\ 2C(n/2) + n - 1 & \text{若 } n \geq 2 \end{cases}$$

请由**定理 1**导出 $C(n)$ 的非递归表达式并指出其渐进复杂性。

定理 1：设 a, c 为非负整数， b, d, x 为非负常数，并对于某个非负整数 k ，令 $n = c^k$ ，则以下递推式

$$f(n) = \begin{cases} d & \text{若 } n = 1 \\ af(n/c) + bn^x & \text{若 } n \geq 2 \end{cases}$$

的解是

$$\begin{aligned} f(n) &= bn^x \log_c n + dn^x && \text{若 } a = c^x \\ f(n) &= \left(d + \frac{bc^x}{a - c^x}\right) n^{\log_c a} - \left(\frac{bc^x}{a - c^x}\right) n^x && \text{若 } a \neq c^x \end{aligned}$$

解：令 $F(n) = C(n) - 1$

即

$$F(n) = \begin{cases} 0 & \text{若 } n = 1 \\ 2(C(n/2) - 1) + n = 2F(n/2) + n & \text{若 } n \geq 2 \end{cases}$$

由**定理 1**可得： $a = 2, b = 1, c = 2, d = 0, x = 1$;

$$\therefore a = c^x = 2$$

$$\therefore F(n) = n \log_2 n + 2n$$

$$C(n) = F(n) + 1 = n \log_2 n + 2n + 1$$

$\therefore C(n)$ 的渐进复杂性为 $O(n \log_2 n)$ 。

二、由于 Prim 算法和 Kruskal 算法设计思路的不同，导致了其对不同问题实例的效率对比关系的不同。请简要论述：

- 1、如何将两种算法集成，以适应问题的不同实例输入；
- 2、你如何评价这一集成的意义？

解：1、Prim算法基于点找出有权重的图中的最小生成树，其时间复杂度为 $O(n^2)$ 。与图中边数无关，适合于稠密图。

Kruskal算法则是对图的边进行排序选择，找出有权重图中的最小生成树，其时间复杂度为 $O(e \log e)$ ，只和边有关系，适合稀疏图。

结合两种算法，共同维护一个点集与边集，我们将当前已生成的最小生成树看作点集中的一个点，将当前最小生成树已连接的边从边集中去除，根据点集与边集的情况，判断当前情况下属于稠密图还是稀疏图。分别使用Prim算法和Kruskal算法进行下一步操作。

2、综合了两个算法，使得可以动态的选择下一步操作的具体步骤，总体来说较使用单一算法更为智能。虽说继承两种算法后基本可以每种情况都能达到最优复杂度，但是在时间与空间开销上有所增加。每个算法都有自己适合的情况，所以在选择算法前需要对具体情况进行分析。

三、分析以下生成排列算法的正确性和时间效率：

```

HeapPermute(n)
//实现生成排列的 Heap 算法
//输入：一个正正整数 n 和一个全局数组 A[1..n]
//输出：A 中元素的全排列
if n = 1
    write A
else
for i ← 1 to n do
    HeapPermute(n - 1)
    if n is odd
        swap A[1] and A[n]
    else
        swap A[i] and A[n]

```

解：以下推导中 $[]$ 代表全排列，括号内内容为最后一次全排列结束后的元素排列。

$n = 1$ 时: $i = 1$: HeapPermute(1) 输出: a_1

$n = 2$ 时: $i = 1$: HeapPermute(1) 确定第二位, 输出 $[a_1]a_2$, 而后交换1和2的位置, 变为 a_2a_1 ;

$i = 2$: HeapPermute(1) 确定第二位, 输出 $[a_2]a_1$, 而后交换2和2的位置, 变为 a_2a_1 ;

$n = 3$ 时: $i = 1$: HeapPermute(2) 确定第三位, 输出 $[a_2a_1]a_3$, 而后交换1和3的位置, 变为 $a_3a_1a_2$;

$i = 2$: HeapPermute(2) 确定第三位, 输出 $[a_1a_3]a_2$, 而后交换1和3的位置, 变为 $a_2a_3a_1$;

$i = 3$: HeapPermute(2) 确定第三位, 输出 $[a_3a_2]a_1$, 而后交换1和3的位置, 变为 $a_1a_2a_3$;

$n = 4$ 时: $i = 1$: HeapPermute(3) 确定第四位, 输出 $[a_1a_2a_3]a_4$, 而后交换1和4的位置, 变为 $a_4a_2a_3a_1$;

$i = 2$: HeapPermute(3) 确定第四位, 输出 $[a_4a_2a_3]a_1$, 而后交换2和4的位置, 变为 $a_4a_1a_3a_2$;

$i = 3$: HeapPermute(3) 确定第四位, 输出 $[a_4a_1a_3]a_2$, 而后交换3和4的位置, 变为 $a_4a_1a_2a_3$;

$i = 4$: HeapPermute(3) 确定第四位, 输出 $[a_4 a_1 a_2] a_3$, 而后交换4和4的位置, 变为 $a_4 a_1 a_2 a_3$;

数学归纳法: 根据以上计算, 假设 n 为奇数时, 经过全排列输出后全局数组元素位置不变, n 为偶数时, 经过全排列输出后全局数组元素中第 n 个元素插到第一个位置。下面进行推导:

设当 n 为奇数时成立, 则 $n + 1$ 为偶数时, 要计算 HeapPermute($n+1$), 先确定 HeapPermute(n), 每次变换后仍为原序列, 此时依循环交换第 i 位和第 $n + 1$ 位, 可知, 全局数组每个元素均在 $A[n + 1]$ 位上过, 1至 n 位为剩余元素的全排列, 所以 n 为奇数时成立。

设当 n 为偶数时成立, 则 $n + 1$ 为奇数时, 要计算 HeapPermute($n+1$), 先确定 HeapPermute(n), 每次变换后第 n 位元素变为第1位, 此时依循环交换第1位和第 $n + 1$ 位, 可知, 全局数组每个元素均在 $A[n + 1]$ 位上过, 1至 n 位为剩余元素的全排列, 所以 n 为偶数时成立。

Q.E.D.

时间复杂度计算:

$$\begin{aligned} C(n) &= \begin{cases} 1 & \text{若 } n = 1 \\ n(C(n-1) + 1) & \text{若 } n \geq 2 \end{cases} \\ \therefore C(n) &= n! + (n + n(n-1) + n(n-1)(n-2) + \cdots + n!) \\ &= n! + (1^2 + 2^2 + \cdots + n^2) \\ &= n! + \frac{n(n+1)(2n+1)}{3} \\ \lim_{x \rightarrow \infty} C(n) &= n! \end{aligned}$$

所以该生成算法的渐近时间复杂度为 $O(n!)$ 。

四、对于求 n 个实数构成的数组中最小元素的位置问题, 写出你设计的具有减治思想算法的伪代码, 确定其时间效率, 并与该问题的蛮力算法相比较。

解: 先找出最小元素, 而后在原数组中进行位置匹配。

算法: 利用快排思想, 以第一个元素为基准进行快排的第一步操作, 若该元素下标为0, 则是最小元素, 若不是, 则对左半部分继续进行同种操作, 直至找到最小元素, 而后在元素组查找该元素的位置。

```

//寻找最小元素, A: 实数数组 high: 数组上界 pos: 基准数在数组中的位置
(elm, pos) = FindMinElement(A, 0, high)
//寻找最小元素位置
Find(elm)
//运用快排思想寻找最小元素
//high初始值为整个数组大小
if(pos == 0)
    Find(elm)
else
    (A[0], pos) = FindMinElement(A, 0, high)

```

算法复杂度计算（需再最后加上寻找最小元素的算法复杂度 $O(n) = \frac{n}{2}$ ）：

$$C(n) = \begin{cases} 1 & \text{若 } n = 1 \\ C(n/2) + n & \text{若 } n \geq 2 \end{cases}$$

$$\therefore C(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^k}$$

$$= 2n - \frac{n}{2^{n-1}}$$

\therefore 算法时间复杂度为 $O(\frac{5n}{2} - \frac{n}{2^{n-1}})$ ，比蛮力法的时间复杂度 $O(n)$ 要高，同时，也比蛮力法使用了更多的空间，主要原因在于减治思想增加了元素间的比较次数，同时也占用了额外的 $temp$ 存储基准数。

五、请给出约瑟夫斯问题的非递推公式 $J(n)$ ，并证明之。其中， n 为最初总人数， $J(n)$ 为最后幸存者的最初编号。

解：不妨设约瑟夫斯问题中 $m = 2$ ，对前16种情况进行简单分析可得：

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
J(n)	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1

当 $n = 1$ 时，此时有 $J(n) = 1$ ；

当 n 为偶数时，不妨设 $n = 2k$ ，此时有 $J(2k) = 2J(k) - 1$ ；

当 n 为奇数时，不妨设 $n = 2k + 1$ ，此时有 $J(2k + 1) = 2J(k) + 1$ ；

可以简化为以下形式：

$$J(n) = \begin{cases} 1 & \text{若 } n = 1 \\ 2J(\lfloor n/2 \rfloor) + n \% 2 * 2 - 1 & \text{若 } n \geq 2 \end{cases}$$

由上述情况寻找规律，不妨将 n 表示成 $n = 2^m + l$ 的形式，其中， m 为不超过 n 的最大幂次项，此时猜测有 $J(n) = 2l + 1$ ，其中 $l = n - 2^{\lfloor \log_2 n \rfloor}$ 。

数学归纳法： $n = 1$ 时： $J(n) = 1$ 成立。

$n = 2k$ 时：假设 $J(2k) = 2J(k) - 1$ 成立

$$\begin{aligned} J(2k) &= 2J(k) - 1 \\ &= 2(2(k - 2^{\lfloor \log_2 k \rfloor + 2}) + 1) - 1 \\ &= 2(2k - 2^{\lfloor \log_2 2k \rfloor + 1}) + 1 \\ &= 2l + 1 \end{aligned}$$

推论成立

$n = 2k + 1$ 时：假设 $J(2k + 1) = 2J(k) + 1$ 成立

$$\begin{aligned} J(2k + 1) &= 2J(k) + 1 \\ &= 2(2(k - 2^{\lfloor \log_2 k \rfloor + 2}) + 1) + 1 \\ &= 2((2k + 1) - 2^{\lfloor \log_2 2k + 1 \rfloor + 1}) + 1 \\ &= 2l + 1 \end{aligned}$$

推论成立

综上所述，约瑟夫问题的非递推公式 $J(n) = 2l + 1$ ，其中 $l = n - 2^{\lfloor \log_2 n \rfloor}$ 。

即 $J(n) = 2n - 2^{\lfloor \log_2 n \rfloor + 1} + 1$ 。