傅申 PB20000051
2022 秋

算法基础 作业 1

2.1–3. 考虑以下**查找问题**:
输入: n 个数的一个序列 $A = \langle a_1, a_2, \cdots, a_n \rangle$ 和一个值 v.
输出: 下标 i 使得 $v = A[i]$ 或者当 v 不在 A 中出现时, i 为特殊值 NIL.
写出**线性查找**的伪代码, 它扫描整个序列来查找 v. 使用一个循环不变式来证明你的算法是正确的. 确保你的循环不变式满足三条必要的性质.

解: 线性查找的伪代码如下

Algorithm 1: LINEAR-SEARCH(A, v)

```
1 i = NIL
2 for j = 1 to A.length
3     if A[j] = v
4         i = j
5         return i
6 return i
```

令循环不变式为 "在开始第 2 ~5 行 **for** 循环的每次迭代时, v 不在子数组 $A[1..j-1]$ 中出现且 $i = \text{NIL}$", 正确性证明如下:

初始化 在第一次循环迭代之前 (即 $j = 1$ 时), $i = \text{NIL}$, 子数组 $A[1..j-1]$ 中没有元素, 因此 v 不在子数组 $A[1..j-1]$ 中出现, 循环不变式成立.

保持 设第 j 次循环迭代之前循环不变式成立. 在第 j 次循环迭代中, 如果 $A[j] = v$, 则设置 $i = j$, 循环终止, 不再有下次循环迭代. 因此若存在下一次循环迭代, 有 $A[j] \neq j$, i 将不会被修改 (保持 NIL), 并且因为 v 也不在 $A[1..j-1]$ 中出现, v 不在子数组 $A[1..j]$ 中出现, 循环不变式成立.

终止 导致 **for** 循环终止的条件有两种: 一种情况是 $A[j] = v$. 这种情况下会在第 j 次循环迭代中进入第 3 行的 **if** 语句, 从而设置 $i = j$ 并返回, 因此得到的下标 i 满足 $v = A[i]$. 另一种情况是 $j > A.length$, 因为每次循环迭代 j 增加 1, 所以 $j = n + 1$. 在循环不变式的表述中将 j 用 $n + 1$ 替换, 得到 v 不在子数组 $A[1..n]$ 中出现且 $i = \text{NIL}$. 子数组 $A[1..n]$ 就是整个数组, 所以 v 不在整个数组中出现且 $i = \text{NIL}$. 在两种情况下都满足要求, 因此算法正确.

2.1–4. 考虑把两个 n 位二进制整数加起来的问题, 这两个整数分别存储在两个 n 元数组 A 和 B 中. 这两个整数的和应按二进制形式存储在一个 $(n + 1)$ 元数组 C 中. 请给出该问题的形式化描述, 并写出伪代码.

解: 形式化描述如下:

输入: 两个 n 元数组 $A = \langle a_1, a_2, \cdots, a_n \rangle, B = \langle b_1, b_2, \cdots, b_n \rangle$, 其中对于所有下标 i 都有 $a_i, b_i \in \{0, 1\}$.

输出: 一个 $(n+1)$ 元数组 $C = \langle c_1, c_2, \cdots, c_{n+1} \rangle$, 其中对于所有下标 i 都有 $c_i \in \{0, 1\}$, 且满足 $\sum_{i=1}^{n+1} c_i 2^{n+1-i} = \sum_{i=1}^{n} a_i 2^{n-i} + \sum_{i=1}^{n} b_i 2^{n-i}$.

算法伪代码如下:

Algorithm 2: BINARY-ADD (A, B)

```
carry = 0
for i = n downto 1
    C[i+1] = A[i] + B[i] + carry
    if C[i+1] ⩾ 2
        C[i+1] = C[i+1] - 2
        carry = 1
    else
        carry = 0
C[1] = carry
```

2.2–2. 考虑排序存储在数组 A 中的 n 个数: 首先找出 A 中最小元素并将其与 $A[1]$ 中的元素进行交换. 接着, 找出 A 中的次最小元素并将其与 $A[2]$ 中的元素进行交换. 对 A 中前 $n-1$ 个元素按该方式继续. 该算法称为**选择算法**, 写出其伪代码. 该算法维持的循环不变式是什么? 为什么它只需要对前 $n-1$, 而不是对所有 n 个元素进行? 用 Θ 记号给出选择排序的最好情况与最坏运行情况运行时间.

解: 选择算法的伪代码如下:

Algorithm 3: SELECTION-SORT (A)

```
for j = 1 to A.length - 1
    min = j
    for i = j + 1 to A.length
        if A[i] < A[min]
            min = i
    temp = A[j]
    A[j] = A[min]
    A[min] = temp
```

该算法维持的循环不变式为 “在第 1 ~8 行的 **for** 循环的每次迭代开始时, 子数组 $A[1..j-1]$ 由原数组 $A[1..n]$ 中前 $j-1$ 小元素组成, 且已按序排列, $A[j..n]$ 由原数组 $A[1..n]$ 中后 $n-j+1$ 小元素组成”.

该算法只需要对前 $n-1$ 个元素进行, 因为在第 $n-1$ 次迭代后, 子数组 $A[1..n-1]$ 由原数组 $A[1..n]$ 中前 $n-1$ 小元素组成, 且已按序排列, 而此时 $A[n]$ 中就是原数组中最大的元素, 整个数组已按序排列.

假设第 i 行的运行时间为 c_i. 当输入数组按升序排列时, 出现最佳情况, 运行时间为

$$\begin{aligned} T_{\min}(n) &= \sum_{j=1}^{n-1}\left(c_1+c_2+\sum_{i=j+1}^{n}(c_3+c_4)+c_6+c_7+c_8\right) \\ &= \frac{c_3+c_4}{2}\,n^2+\left(c_1+c_2+c_6+c_7+c_8-\frac{c_3+c_4}{2}\right)n \\ &= \Theta(n^2) \end{aligned} \tag{2.1}$$

当输入数组按降序排列时, 出现最坏情况, 运行时间为

$$\begin{aligned} T_{\max}(n) &= \sum_{j=1}^{n-1}\left(c_1+c_2+\sum_{i=j+1}^{n}(c_3+c_4+c_5)+c_6+c_7+c_8\right) \\ &= \frac{c_3+c_4+c_5}{2}\,n^2+\left(c_1+c_2+c_6+c_7+c_8-\frac{c_3+c_4+c_5}{2}\right)n \\ &= \Theta(n^2) \end{aligned} \tag{2.2}$$

2.2–3. 再次考虑线性查找问题 (参见练习 2.1–3). 假定要查找的元素等可能的为数组中的任意元素, 平均需要检查输入序列的多少元素? 最坏情况又如何呢? 用 Θ 记号给出线性查找的平均情况和最坏情况运行时间. 证明你的答案.

解: 数组中每个元素被查找的概率均为 $\dfrac{1}{n}$, 所以平均需要检查输入序列的 $\displaystyle\sum_{j=1}^{n}\frac{j}{n}=\frac{n+1}{2}$ 个元素. 在平均情况下, 运行时间

$$\begin{aligned} T_{\text{avg}}(n) &= c_1+\frac{n+1}{2}(c_2+c_3)+c_4+c_5 \\ &= \frac{c_2+c_3}{2}n+c_1+c_4+c_5+\frac{c_2+c_3}{2} \\ &= \Theta(n) \end{aligned} \tag{3.1}$$

当数组中没有需要查找的元素或者需要查找的元素为数组中最后一个元素时, 出现最坏情况, 需要检查输入序列的所有元素, 也就是 n 个元素. 在最坏情况下 (假设没有找到需要的元素), 运行时间为

$$T_{\max}(n)=(c_2+c_3)\,n+c_1+c_6=\Theta(n) \tag{3.2}$$

2.3–1. 使用书上图 2–4 作为模型, 说明归并排序在数组 $A=\langle 3,41,52,26,38,57,9,49\rangle$ 上的操作.

解: 操作如下:

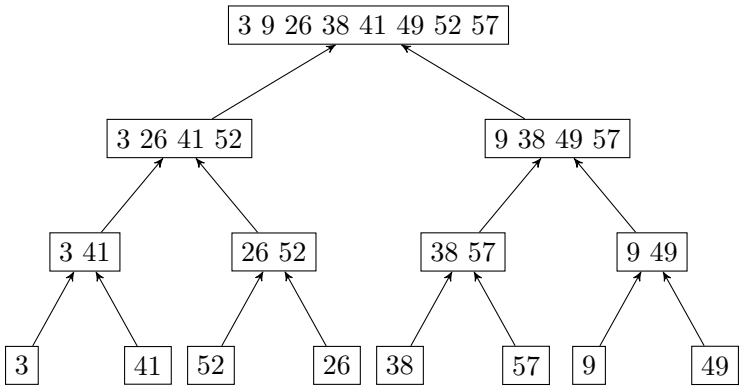

图 1.1: 归并排序的操作

2.3–5. 回顾查找问题 (参见练习 2.1–3). 注意到, 如果序列 A 已排好序, 就可以将该序列的中点与 v 进行比较. 根据比较的结果, 原序列有一半就可以不用再做进一步的考虑了. 二**分查找**算法重复这个过程, 每次都将序列剩余部分的规模减半. 为二分查找写出迭代或递归的伪代码, 证明: 二分查找的最坏情况运行时间为 $\Theta(\lg n)$.

解: 二分查找的伪代码如下, 其中初始调用为 BINARY-SEARCH(A, v, 1, n):

Algorithm 4: BINARY-SEARCH(A, v, p, r)

```
1 if p > r
2     return NIL
3 q = ⌊(p + r)/2⌋
4 if v = A[q]
5     return q
6 else if v > A[q]
7     return BINARY-SEARCH(A, v, q + 1, r)
8 else
9     return BINARY-SEARCH(A, v, p, q − 1)
```

分解过程只需要检查下标并计算中间下标, 需要常量时间, $D(n) = \Theta(1)$; 在最坏情况下, 解决过程每次都需要递归地求解一个规模为 $\frac{n}{2}$ 的子问题, 贡献 $T\left(\frac{n}{2}\right)$ 的运行时间, 直到子数组长度小于等于 1; 合并过程只需要返回下标, 需要常量时间, $C(n) = \Theta(1)$. 因此, 最坏情况运行时间的递推式为

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ T\left(\frac{n}{2}\right) + \Theta(1) & n > 1 \end{cases} \tag{5.1}$$

运用递归树, 每一层都只有一个结点, 代价均为 $\Theta(1)$, 高度至多为 $\lceil \lg n + 1 \rceil$, 因此 $T(n)$ 为 $\Theta(\lg n)$.

3.1–2. 证明: 对任意实常量 a 和 b, 其中 $b > 0$, 有

$$(n+a)^b = \Theta\left(n^b\right) \tag{2.1}$$

解: 取 $c_1 = \left(\frac{1}{2}\right)^b$, $c_2 = \left(\frac{3}{2}\right)^b$ 和 $n_0 = 2|a|$, 当 $n \geqslant n_0$ 时, 有

$$\frac{3}{2}n \geqslant n + |a| \geqslant n + a \geqslant n - |a| \geqslant \frac{1}{2}n \geqslant 0 \tag{2.2}$$

因此就有

$$c_2 n^b = \left(\frac{3}{2}n\right)^b \geqslant (n+a)^b \geqslant \left(\frac{1}{2}n\right)^b = c_1 n^b \geqslant 0 \tag{2.3}$$

所以 $(n+a)^b = \Theta\left(n^b\right)$

3.1–4. $2^{n+1} = O\left(2^n\right)$ 成立吗? $2^{2n} = O\left(2^n\right)$ 成立吗?

解: $2^{n+1} = O\left(2^n\right)$ 成立. 取 $c = 3$ 和 $n_0 = 1$, 当 $n \geqslant n_0$ 时, 有

$$0 \leqslant 2^{n+1} \leqslant 3 \times 2^n = c \times 2^n \tag{4.1}$$

$2^{2n} = O\left(2^n\right)$ 不成立. 对任意的实常量 c, 当 $n > \lg c$ 时, 有

$$2^{2n} = 2^n \cdot 2^n \geqslant c \times 2^n \tag{4.2}$$

3.1–6. 证明: 一个算法的运行时间为 $\Theta(g(n))$ 当且仅当其最坏情况运行时间为 $O(g(n))$ 且其最好情况运行时间为 $\Omega(g(n))$.

解: 记算法的运行时间, 最坏情况运行时间和最好情况运行时间分别为 $T(n)$, $T_{\max}(n)$ 和 $T_{\min}(n)$, 则有

$$T_{\min}(n) \leqslant T(n) \leqslant T_{\max}(n) \tag{6.1}$$

先证明 $T_{\max}(n) = O(g(n)), T_{\min}(n) = \Omega(g(n)) \Rightarrow T(n) = \Theta(g(n))$. 因为 $T_{\min}(n) = \Omega\left(g(n)\right)$, 所以存在正常量 c_1 和 n_1, 当 $n \geqslant n_1$ 时, 有

$$c_1 g(n) \leqslant T_{\min}(n) \tag{6.2}$$

同理, $T_{\max}(n) = O\left(g(n)\right)$, 所以存在正常量 c_2 和 n_2, 当 $n \geqslant n_2$ 时, 有

$$T_{\max}(n) \leqslant c_2 g(n) \tag{6.3}$$

因此, 取 $n_0 = \max\{n_1, n_2\}$, 当 $n \geqslant n_0$ 时, 有

$$c_1 g(n) \leqslant T_{\min}(n) \leqslant T(n) \leqslant T_{\max}(n) \leqslant c_2 g(n) \tag{6.4}$$

所以 $T(n)=\Theta\left(g(n)\right)$.

再证明 $T(n)=\Theta(g(n))\Rightarrow T_{\max}(n)=O(g(n)),T_{\min}(n)=\Omega(g(n))$. 因为 $T(n)=\Theta\left(g(n)\right)$, 所以存在正常量 c_1, c_2 和 n_0, 当 $n\geqslant n_0$ 时, 在所有情况下有

$$c_1g(n)\leqslant T_{\min}\leqslant T(n)\leqslant T_{\max}(n)\leqslant c_2g(n) \tag{6.5}$$

所以显然有 $T_{\max}(n)=O(g(n))$, $T_{\min}(n)=\Omega(g(n))$.

3.2–3. 证明下面的等式. 并证明 $n!=\omega\left(2^n\right)$ 且 $n!=o\left(n^n\right)$.

$$\lg\left(n!\right)=\Theta\left(n\lg n\right) \tag{3.1}$$

解: 首先证明 $\lg\left(n!\right)=\Theta\left(n\lg n\right)$. 取 $c_1=\dfrac{1}{2}$, $c_2=1$, 和 $n_0=\mathrm{e}^2$, 当 $n\geqslant n_0$ 时, 有

$$\lg\left(n!\right)=\lg\left(\sqrt{2\pi n}\Big(\frac{n}{\mathrm{e}}\Big)^n\mathrm{e}^{\alpha_n}\right)=\frac{1}{2}\lg\left(2\pi\right)+\frac{1}{2}\lg n+n\lg\Big(\frac{n}{\mathrm{e}}\Big)+\alpha_n\lg\mathrm{e}\geqslant n\lg\Big(\frac{n}{\mathrm{e}}\Big)\geqslant n\lg\left(\sqrt{n}\right)=\frac{1}{2}n\lg n \tag{3.2}$$

$$\lg\left(n!\right)=\sum_{i=1}^{n}\lg i\leqslant\sum_{i=1}^{n}\lg n=n\lg n=c_2n\lg n \tag{3.3}$$

然后证明 $n!=\omega\left(2^n\right)$. 对任意正常量 $c>0$, 取 $n_0=\max\{5,2c+1\}$, 当 $n\geqslant n_0$ 时, 有

$$n!=\begin{cases}\dfrac{n!}{5!}\times5!>2^{n-5}\times120>c\times2^n & 0<c\leqslant2\\ \dfrac{n!}{1!}\geqslant2^2\times\Big\lfloor\dfrac{n}{2}\Big\rfloor\times\Big\lfloor\dfrac{n-1}{2}\Big\rfloor\times\dfrac{(n-2)!}{1!}\geqslant2^2\times c\times2\times2^{n-3}=c\times2^n & c>2\end{cases} \tag{3.4}$$

最后证明 $n!=o\left(n^n\right)$. 对任意正常量 $c>0$, 取 $n_0=\max\left\{\dfrac{1}{c},1\right\}$, 当 $n\geqslant n_0$ 时, 有

$$n!\begin{cases}=\dfrac{n!}{1!}\leqslant n^{n-1}\leqslant n^{n-1}\times cn & 0<c\leqslant1\\ \leqslant n^n<cn^n & c>1\end{cases} \tag{3.5}$$

3.2–5. 如下两个函数中, 哪个渐进更大些: $\lg\left(\lg^*n\right)$ 还是 $\lg^*\left(\lg n\right)$?

解: 记 $n=2^m$, $t=\lg^*m$, 就有

$$\lg\left(\lg^*n\right)=\lg\left(1+\lg^*m\right)=\lg\left(1+t\right) \tag{5.1}$$

$$\lg^*\left(\lg n\right)=\lg^*m=t \tag{5.2}$$

因此, 原题转化为比较 $\lg\left(1+t\right)$ 和 t 的渐进大小, 有 $\lg\left(1+t\right)=o\left(t\right)$, 证明如下.

首先证明当 $x>1$ 时有 $\ln x<2\sqrt{x}-2$, 因为当 $x>1$ 时有 $\ln x<x-1$, 所以

$$\ln\sqrt{x}=\frac{1}{2}\ln x<\sqrt{x}-1\iff\ln x<2\sqrt{x}-2 \tag{5.3}$$

对任意正常量 $c > 0$, 记 $b = c \ln 2$, 取 $t_0 = \max\left\{\dfrac{4-4b}{b^2}, 1\right\}$, 当 $t \geqslant t_0$ 时, 有

$$\begin{aligned} \lg(1+t) \leqslant ct &\impliedby \ln(1+t) \leqslant ct \ln 2 = bt \\ &\impliedby 2\sqrt{1+t} - 2 \leqslant bt \\ &\impliedby b^2 t^2 + (4b-4)\, t \geqslant 0 \\ &\impliedby t \geqslant \frac{4-4b}{b^2} \text{ 且 } t \geqslant 1 \end{aligned} \tag{5.4}$$

综上所述, 有 $\lg(\lg^* n) = o(\lg^*(\lg n))$, 即 $\lg^*(\lg n)$ 的渐进更大.