

1. 线性时间排序算法

(a) 说明如何以 $O(n)$ 的最坏时间复杂度对 $[1, n^2]$ 范围内的 n 个整数排序。假设整数已经在你需要的基上, 也就是说, 你不需要对它们进行进制转换。

(b) 对于常数 $m \in \mathbb{N}^+$, 给定长度为 $n(n > m)$ 的数列 Z , 已知 Z 中每个元素都以这样的方式生成: $Z_i = X_i + Y_i$, 离散随机变量 X_i 在 $\{0, \frac{1}{m}, \frac{2}{m}, \dots, \frac{m-1}{m}\}$ 中随机均匀取值, 连续随机变量 Y_i 在 $(0, \frac{1}{n})$ 上随机均匀取值, 所有 X_i, Y_i 均独立。请设计一个桶排序算法, 以 $O(n)$ 的期望时间复杂度对 Z 进行排序。你只需给出算法简要流程, 不需要对这一时间复杂度进行证明。

提示: 第二问中, 为达到 $O(n)$ 的期望时间复杂度, 你需要设计 $\Theta(n)$ 个桶, 并尽可能均匀地将 Z 中的元素分配到桶中。

解:

(a) 以 n 为基底使用基数排序。由于数字以 n 为基底, 因此位数的范围为 1 到 n (准确地说是 0 到 $n-1$), 所以 $k=n$ 。由于 $n^2 = 100_n$, 所以 $d=3$ 。故基数排序的时间复杂度为 $\Theta(d \cdot n + d \cdot k) = \Theta(3n + 3n) = \Theta(n) \in O(n)$ 。

(b) 因为 $Y_i < \frac{1}{n} < \frac{1}{m}$, 易验证, 对任意 Z_i , $X_i = \lfloor mZ_i \rfloor$, 因此可以用这种方式将 Z_i 对应的 X_i, Y_i 分解出来。建立 m 个“大桶”, 每个“大桶”里有 n 个“小桶”。对于每个 Z_i , 根据 X_i 的值决定它进入哪个大桶, 再根据 Y_i 的值决定它进入哪个小桶。因为 X_i 和 Y_i 都是均匀的, 因此 Z_i 在期望意义上被均匀地分配到桶中, 从而具有线性时间复杂度。

具体而言, 对于桶 $B(x, y), x \in [0, m-1], y \in [0, n-1]$, 其内放置的元素范围应为 $(\frac{x}{m} + \frac{y}{n^2}, \frac{x}{m} + \frac{y+1}{n^2})$

2. 二叉搜索树一定程度上可以提高搜索效率，但在最坏情况下，仍需要 $O(n)$ 时间完成一次搜索，因此需要引入平衡二叉搜索树。AVL 树是一种高度平衡的二叉搜索树：对于每个结点，其左子树与右子树的高度差不超过 1。为实现一棵 AVL 树，需要在每个节点内维护一个额外的属性： $height[x]$ 为结点 x 的高度。此外，我们还定义平衡因子 (Balance Factor)： $BF[x]$ 为结点 x 左右子树的高度差，取值为 0、1、-1，分别对应着左右子树等高，左子树比右子树高，右子树比左子树高。

(a) 已知一颗 AVL 树上的结点 x' ， $BF[left[x']] = 0$ ， $BF[right[x']] = 1$ ，试问 $BF[x']$ 的值。

$BF[x']$ 的值可能是 0、1 或 -1。

(b) 证明：一棵有 n 个结点的 AVL 树高度为 $O(\lg n)$ 。

对于斐波那契数列有 $F(0) = 1$ ， $F(1) = 1$ ， $F(2) = 2$ ， $F(n) = F(n-1) + F(n-2)$ 。

设 $T(h)$ 为高度为 h 的 AVL 树的最少结点数，则有 $T(1) \geq F(1)$ ， $T(2) \geq F(2)$ 。

又有 $T(h) \geq T(h-1) + T(h-2) + 1 \geq F(h-1) + F(h-2) + 1 > F(h)$ 。

又有 $1.6^h \leq F(h) \leq 2^h$ ，故有 $T(h) \geq 1.6^h$ 。所以，一棵有 n 个结点的 AVL 树高度为 $O(\lg n)$ 。

(c) 在一棵 AVL 树中插入一个结点，首先以二叉搜索树的顺序把该结点放在适当的位置上。此时，可能某些结点的平衡因子可能为 2 或 -2。请给出算法 AVL-BALANCE(x) 的伪代码，输入一棵以 x 为根的子树，其左右子树都是高度平衡的，若 $BF[x]$ 为 2 或 -2，将其转变为高度平衡的，返回调整后的子树的根，否则直接返回 x 。

Algorithm 5 AVL-BALANCE(x)

```

1: if  $|BF[x]| \leq 1$  then
2:   Return  $x$ 
3: else if  $BF[x] = 2$  then
4:    $y = left[x]$ 
5:   if  $height[left[y]] < height[right[y]]$  then
6:     LEFT-ROTATE( $y$ )
7:   Return RIGHT-ROTATE( $x$ )
8: else
9:    $y = right[x]$ 
10:  if  $height[left[y]] > height[right[y]]$  then
11:    RIGHT-ROTATE( $y$ )
12:  Return LEFT-ROTATE( $x$ )

```

1. 快速排序

(a) 假设不是对整个数组排序，而只要确定一个给定的 n 个元素数列的第 m 个最小者。修改快速排序算法实现这一目的，以避免为进行一次完全排序所需要的额外操作。填写 QuickSelect 算法。

Algorithm 2 QUICKSORT(arr, low, high)

```
1: if low < high then
2:   pivotIndex = PARTITION(arr, low, high)
3:   QUICKSORT(arr, low, pivotIndex - 1)
4:   QUICKSORT(arr, pivotIndex + 1, high)
```

Algorithm 3 PARTITION(arr, low, high)

```
1: pivot = arr[high]
2: i = low - 1
3: for j = low to high - 1 do
4:   if arr[j] <= pivot then
5:     i = i + 1
6:     SWAP(arr[i], arr[j])
7: SWAP(arr[i + 1], arr[high])
8: Return i + 1
```

Algorithm 4 QuickSelect(arr, low, high, m)

```
1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
11:
12:
13:
```

(b) 给出 QuickSelect 的平均键码比较次数 C_{nm} 的递推形式表达式。
(c) 当 $m = 1$ 时，计算 $C_{n,1}$ 。

解:

(a)

Algorithm 5 QuickSelect(arr, low, high, m)

```

1: if low = high then
2:   Return arr[low]
3: pivotIndex = PARTITION(arr, low, high)
4: k = pivotIndex - low + 1
5: if m = k then
6:   Return arr[pivotIndex]
7: else if m < k then
8:   Return QuickSelect(arr, low, pivotIndex - 1, m)
9: else
10:  Return QuickSelect(arr, pivotIndex + 1, high, m - k)

```

(b) 递推式是 $C_{11} = 0$ 和当 $n > 1$ 时,

$$C_{nm} = n + 1 + \frac{1}{n} \left(\sum_{1 \leq s < m} C_{(n-s)(m-s)} + \sum_{m < s \leq n} C_{(s-1)m} \right)$$

(c) 首先, 根据递推公式, 当 $m = 1, n > 1$ 时, 我们有:

$$C_n = n + 1 + \frac{1}{n} \sum_{1 \leq s \leq n-1} C_s$$

可得:

$$\begin{aligned}
 nC_n &= n^2 + n + \sum_{1 \leq s \leq n-1} C_s \\
 (n-1)C_{n-1} &= n^2 - n + \sum_{1 \leq s \leq n-2} C_s
 \end{aligned}$$

两式相减:

$$nC_n - (n-1)C_{n-1} = 2n + C_{n-1}$$

得到:

$$C_n - C_{n-1} = 2$$

而 $C_1 = 0$ 故 $C_n = 2n - 2$

2. 二叉搜索树一定程度上可以提高搜索效率，但在最坏情况下，仍需要 $O(n)$ 时间完成一次搜索。为此，有人提出了 Treap 树结构，即二叉搜索树 (tree) 与最小堆 (heap) 的组合。具体而言，该树内的每个结点 x 都有一个关键字值 $x.key$ ，另外，还要为每个结点 x 指定一个优先级 $x.priority$ (**假设所有结点的优先级都是不同的，而且所有关键字也是不同的**)。Treap 树的结点被排列成让关键字遵循二叉搜索树的性质，且优先级遵循最小堆顺序性质：

- 如果 v 是 u 的左孩子，则 $v.key < u.key$ 。
- 如果 v 是 u 的右孩子，则 $v.key > u.key$ 。
- 如果 v 是 u 的孩子，则 $v.priority > u.priority$ 。

(a) 给定一个由 (关键字, 优先级) 对组成的结点集合 $\{(x_i.key, x_i.priority) | 1 \leq i \leq n\}$ ，存在多少颗不同的 Treap 树能表示这个集合。

(b) 给出 TREAP-INSERT(T, x) 的伪代码 (可以调用 LEFT-ROTATE 和 RIGHT-ROTATE)。

(c) 假设 Treap 树中各结点的优先级独立且均匀分布。求解 n 结点 Treap 树中的叶子节点的期望数量 $E(n)$ 。

解：

(a) 1

(b)

Algorithm 6 TREAP-INSERT(T, x)

```

1:  $key = x.key$ 
2:  $priority = x.priority$ 
3: if  $T$  is NULL then
4:   return NewNode( $key, priority$ )
5: if  $key < T.key$  then
6:    $T.left = \text{TREAP-INSERT}(T.left, x)$ 
7:   if  $T.left.priority < T.priority$  then
8:      $T = \text{RIGHT-ROTATE}(T)$ 
9: else
10:   $T.right = \text{TREAP-INSERT}(T.right, x)$ 
11:  if  $T.right.priority < T.priority$  then
12:     $T = \text{LEFT-ROTATE}(T)$ 
13: return  $T$ 

```

(c) $E(0) = 0, E(1) = E(2) = 1, E(3) = \frac{4}{3}$ ，又有

$$E(n) = \frac{1}{n} \sum_{k=0}^{n-1} (E(k) + E(n-1-k))$$

假设当 $1 < n \leq n_0$ 时， $E(n) = \frac{n+1}{3}$ ，则

$$\begin{aligned}
 E(n_0+1) &= \frac{1}{n_0+1} \sum_{k=0}^{n_0} (E(k) + E(n_0+1-1-k)) \\
 &= \frac{1}{n_0+1} \left(\sum_{k=2}^{n_0-2} \frac{k+1+n_0-k+1}{3} \right) + \frac{2}{n_0+1} \left(0 + \frac{n_0+1}{3} + 1 + \frac{n_0}{3} \right) = \frac{n_0+2}{3}
 \end{aligned}$$

故 $n = n_0 + 1$ 时， $E(n) = \frac{n+1}{3}$ 。递推得， $E(n) = \frac{n+1}{3}, (n > 1)$ 。综上，

$$E(n) = \begin{cases} \frac{n+1}{3}, & n > 1 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$$

1. 线性排序算法

(a) 基数排序 (Radix Sort) 的正确性依赖于一个稳定的子排序算法。假设我们使用计数排序 (Counting Sort) 作为 RADIX-SORT 所需的稳定子排序算法 STABLE-SORT, 请写出 COUNTING-SORT(A, d) 的伪代码, 该算法用于对一个包含 n 个元素的数组 A 进行排序, 其中每个元素都是不大于 m 的非负整数。

解:

Algorithm 2 COUNTING-SORT(A, m)

```

1: let  $B[0..n], C[0..m]$  be a new array
2: for  $i = 0$  to  $m$  do
3:    $C[i] = 0$ 
4: for  $j = 1$  to  $A.length$  do
5:    $C[A[j]] = C[A[j]] + 1$ 
6: for  $i = 1$  to  $m$  do
7:    $C[i] = C[i] + C[i - 1]$ 
8: for  $j = A.length$  to 1 do
9:    $B[C[A[j]]] = A[j]$ 
10:   $C[A[j]] = C[A[j]] - 1$ 
11: Return  $B$ 

```

(b) 已知待排序的 n 个元素中的每一个都有 d 位数字, 且每一位数字都可以取 0 到 $k-1$ 中的任意一个值 (即最多有 k 个可能的值), 请推导出 RADIX-SORT 的总时间复杂度。

解:

1. RADIX-SORT 的核心是一个从 1 到 d 的 for 循环。
2. 循环体内部调用一次 STABLE-SORT (即计数排序), 其时间复杂度为 $\Theta(n+k)$ 。由于循环执行 d 次, 总时间复杂度是 d 次 $\Theta(n+k)$ 的总和, 即 $\Theta(d(n+k))$ 。

(c) 在实际应用中, 我们经常处理 b 位的二进制数。一种高效的策略是不按 1 位排序, 而是将 b 位分成 d 组, 每组 r 位 (其中 $r \leq b$)。我们对这 $d = \lceil b/r \rceil$ 组分别使用基数排序 (同样使用计数排序作为子算法)。此时, RADIX-SORT 的总时间复杂度 $T(n, b, r)$ 是多少?

解:

- 每次计数排序的时间复杂度变为 $\Theta(n+k) = \Theta(n+2^r)$ 。
- 总共需要排序的“位数” $d = \lceil b/r \rceil$ 。
- 因此, 总时间复杂度 $T(n, b, r) = \Theta(d(n+2^r)) = \Theta(\lceil b/r \rceil (n+2^r))$ 。

2. 在标准二叉搜索树中, 我们可以高效支持 SEARCH、INSERT 和 DELETE 等操作, 但无法高效处理顺序统计问题, 如“第 i 小的关键字是什么?” 或者“关键字 x 的排名是多少?”

为解决此问题, 我们引入顺序统计树 (Order-Statistic Tree, OST): 它在每个结点 x 中额外维护一个属性 $x.size$, 表示以 x 为根的子树中包含的结点总数 (包括 x 自身)。空指针 (记为 NIL) 的 $size$ 定义为 0。利用 $size$ 信息可实现 OS-SELECT(x, i), 返回以 x 为根的子树中第 i 小的关键字对应的结点 ($1 \leq i \leq x.size$)。

(a) 在一棵 OST 中, 某结点 x 满足 $x.left.size = 3$, $x.right.size = 5$ 。

1. 计算 $x.size$ 。

2. 若调用 OS-SELECT($x, 4$), 将返回哪个结点? 简要说明理由。

(b) 用伪代码实现函数 OS-SELECT(x, i), 其中 x 是一棵 OST 的根结点, i 为正整数。若 $1 \leq i \leq x.size$, 返回对应结点; 否则返回 NIL。

(c) 假设 OST 由关键字 $1, 2, \dots, n$ 按均匀随机排列插入构建。令 D_i 表示第 i 小关键字对应结点的深度 (根深度为 0)。

1. 证明: 对任意 $1 \leq i \leq n$,

$$\mathbb{E}[D_i] = H_i + H_{n-i+1} - 2,$$

(提示: $\ln(k+1) \leq H_k \leq 1 + \ln k$ 对所有整数 $k \geq 1$ 成立, 其中 $H_k = \sum_{j=1}^k \frac{1}{j}$ 为第 k 个调和数。)

2. 利用 (1) 的结论, 推导 OS-SELECT($T.root, i$) 的时间复杂度。

解: (a)

1. 由 $size$ 定义:

$$x.size = x.left.size + x.right.size + 1 = 3 + 5 + 1 = \boxed{9}.$$

2. 设 $r = x.left.size + 1 = 3 + 1 = 4$ 。根据 OST 性质, x 是其子树中第 $r = 4$ 小的元素。因此, OS-SELECT($x, 4$) 返回结点 x 自身。

(b)

Algorithm 3 OS-SELECT(x, i)

Require: A node x and an integer i

```

1: if  $x = \text{NIL}$  or  $i < 1$  or  $i > x.size$  then
2:   return NIL
3: end if
4: if  $x.left = \text{NIL}$  then
5:    $r \leftarrow 1$ 
6: else
7:    $r \leftarrow x.left.size + 1$ 
8: end if
9: if  $i = r$  then
10:  return  $x$ 
11: else if  $i < r$  then
12:  return OS-SELECT( $x.left, i$ )
13: else
14:  return OS-SELECT( $x.right, i - r$ )
15: end if
```

(c)

1. 证明: 在随机 BST 中, 结点 $j \neq i$ 是结点 i 的祖先当且仅当在区间 $[\min(i, j), \max(i, j)]$ 中 j 最先插入, 故

$$\Pr[j \text{ 是 } i \text{ 的祖先}] = \frac{1}{|i - j| + 1}.$$

深度 D_i 等于祖先个数, 因此

$$\mathbb{E}[D_i] = \sum_{j=1}^{i-1} \frac{1}{i-j+1} + \sum_{j=i+1}^n \frac{1}{j-i+1} = \sum_{k=2}^i \frac{1}{k} + \sum_{k=2}^{n-i+1} \frac{1}{k} = (H_i - 1) + (H_{n-i+1} - 1) = H_i + H_{n-i+1} - 2.$$

2. OS-SELECT($T.\text{root}, i$) 访问 $D_i + 1$ 个结点 (深度为 D_i , 路径含 $D_i + 1$ 个结点), 故运行时间 $= D_i + 1$ 。取期望:

$$\mathbb{E}[\text{time}] = \mathbb{E}[D_i] + 1 = H_i + H_{n-i+1} + O(1).$$

由于 $H_k \leq \ln k + 1$, 有

$$\mathbb{E}[D_i] + 1 \leq 2(\ln i + \ln(n - i + 1)) + O(1) \leq 4\ln n + O(1) = O(\log n).$$

因此, 时间复杂度为 $O(\log n)$ 。