傅申 PB20000051
2022 秋

算法基础 作业 4

8.2–4. 设计一个算法, 它能够对于任何给定的介于 0 和 k 之间的 n 个整数先进行预处理, 然后在 $O(1)$ 时间内回答输入的 n 个整数中有多少个落在区间 $[a..b]$ 内. 你设计的算法的预处理时间应为 $\Theta(n+k)$.

解: 预处理算法如下算法 1, 回答算法如下算法 2.

Algorithm 1: PREPROCESS(A, k)

```
1 let C[0..k] be a new array
2 for i = 0 to k
3     C[i] = 0
4 for j = 1 to A.length
5     C[A[j]] = C[A[j]] + 1
6 for i = 1 to k
7     C[i] = C[i] + C[i − 1]
8 return C
```

Algorithm 2: ANSWER(A, k, a, b)

```
1 C = PREPROCESS(A, k)          // 预处理: Θ(n + k)
2 return C[b] − C[a − 1]        // 回答: O(1)
```

8.3–4. 说明如何在 $O(n)$ 时间内, 对 0 到 n^3-1 区间内的 n 个整数进行排序.

解: 首先将这 n 个整数转换为 n-进制数, 这需要 $O(n)$ 时间, 转换后每个整数至多有 3 位, 对它们基数排序, 使用稳定排序方法为计数排序, 每次都需要 $\Theta(n)$ 的时间, 共 3 次, 因此排序时间为 $O(n)$, 总共需要 $O(n)$ 的时间.

8.4–2. 解释为什么桶排序在最坏情况下运行时间是 $\Theta(n^2)$? 我们应该如何修改算法, 使其在保持平均情况为线性时间代价的同时, 最坏情况时间代价为 $O(n \lg n)$?

解: 在最坏情况下, 所有的 n 个元素都被放进同一个桶中, 再进行插入排序. 因为插入排序的时间复杂度为 $\Theta(n^2)$, 因此桶排序在最坏情况下运行时间为 $\Theta(n^2)$.

可以将插入排序换为同样对链表操作友好的归并排序, 这样最坏情况时间代价就减小到了 $O(n \lg n)$.

9.1–1. 证明: 最坏情况下, 找到 n 个元素中第二小的元素需要 $n + \lceil \lg n \rceil - 2$ 次比较.

解: 首先以这 n 个元素为叶结点自底向上构建一棵二叉树, 每个非叶结点为其两个子结点的较小值, 最后树的根结点就是整个数组的最小值. 这一部分需要的比较次数就是二叉树非叶结点数, 即 $n - 1$.

然后考虑与最小值比较过的数值, 它们在二叉树中表现为根结点到对应叶节点路径上面所有结点的兄弟结点, 共有 $h = \lceil \lg n \rceil$ 个, 找出它们中的最小值, 这就是 n 个元素中第二小的元素, 需要 $\lceil \lg n \rceil - 1$ 次比较, 因此总共需要 $n + \lceil \lg n \rceil - 2$ 次比较. 对应的算法如下, 使用堆中的 PARENT 操作:

Algorithm 3: SECOND-MIN(A)

```
1  n = A.length
2  let T[1..2n + 1] be a new array
3  let S[1..2n − 1] be a new array
4  T[2n] = T[2n + 1] = +∞
5  for i = 1 to n
6  |  T[n + i − 1] = A[i]
7  |  S[n + i − 1] = 2n                    // 所有叶节点的两个孩子都是无穷大
8  // 建树, 总共 n − 1 次比较
9  for i = 2n − 1 downto 3 by 2
10 |  if T[i] < T[i − 1]
11 |  |  T[PARENT(i)] = T[i]
12 |  |  S[PARENT(i)] = i
13 |  else
14 |  |  T[PARENT(i)] = T[i − 1]
15 |  |  S[PARENT(i)] = i − 1
16 // 查找次小值, 共 ⌈lg n⌉ − 1 次比较
17 idx = S[1]
18 sec-min = T[BROTHER(idx)]
19 for j = 1 to ⌈lg n⌉ − 1
20 |  idx = S[idx]
21 |  if T[BROTHER(idx)] < sec-min
22 |  |  sec-min = T[BROTHER(idx)]
23 return sec-min
```

其中 BROTHER(k) 返回 $4\lfloor k/2 \rfloor + 1 - k$.

9.2–3. 给出 RANDOMIZED-SELECT 的一个基于循环的版本.

解: 如下

Algorithm 4: LOOP-RANDOMIZED-SELECT(A, i)

```
1  p = 1
2  r = A.length
3  while p < r
4      q = RANDOMIZED-PARTITION(A, p, r)
5      k = q − p + 1
6      if i = k
7          return A[q]
8      if i < k
9          r = q − 1
10     else
11         p = q + 1
12         i = i − k
13 return A[p]
```

9.3–6. 对一个包含 n 个元素的集合来说, k **分位数**是指能把有序集合分为 k 个等大小集合的第 $k-1$ 个顺序统计量. 给出一个能找出某一集合的 k 分位数的 $O(n \lg k)$ 时间的算法.

解: 采用分治思想, 先寻找第 $k/2$ 个分位数, 并将数组分为两半再递归寻找其他分位数. 假设集合大小为 k 的整数倍减一, 算法如下:

Algorithm 5: K-QUANTITLES(A, k)

```
1  let Q[1..k − 1] be a new array
2  size = (A.length + 1) / k
3  K-QUANTITLES-REC(A, 1, A.length, 1, k − 1, size, Q)
4  return Q
```

Algorithm 6: K-QUANTITLES-REC$(A, p, r, qp, qr, size, Q)$

if $qp > qr$
  return
$qmid = \left\lfloor \frac{qp + qr}{2} \right\rfloor$
$rank = qmid \cdot size$
$Q[qmid] =$ SELECT$(A, p, r, rank)$
K-QUANTITLES-REC$(A, p, rank - 1, qp, qmid - 1, size, Q)$
K-QUANTITLES-REC$(A, rank + 1, r, qmid + 1, qr, size, Q)$
return

递归子算法的递归式为 $T(k) = T(\lfloor k/2 \rfloor) + T(\lfloor (k-1)/2 \rfloor) + O(k \cdot size)$, 可以解出 $T(k) = O(k \cdot size \lg k) = O(n \lg k)$, 而主算法的其他部分运行时间为 $O(1)$, 因此总的运行时间为 $O(n \lg k)$.