算法设计与分析 (4.6 作业)

智科三班　严中圣　222020335220177

2022 年 4 月 5 日

2.3　双 Hanoi 塔问题是 Hanoi 塔问题的一种推广，与 Hanoi 塔的不同点在于：$2n$ 个圆盘，分成大小不同的 n 对，每对圆盘完全相同. 初始，这些圆盘按照从大到小的次序从下到上放在 A 柱上，最终要把它们全部移到 C 柱，移动的规则与 Hanoi 塔相同.

(1) 设计一个移动的算法并给出伪码描述.

(2) 计算你的算法所需要的移动次数.

解.

(1) 运用分治算法，将全过程分解为以下步骤：

1. 将 A 柱上前 $2n-2$ 个圆盘移到 B 柱上
2. 将 A 柱剩余两个大圆盘移到 C 柱上
3. 将 B 柱上 $2n-2$ 个圆盘移到 C 柱上

伪码描述如下：

Algorithm 1 分治策略解决双 Hanoi 问题　Hanoi(A,C,n)

Require: n　//从 A 柱移动 $2n$ 个圆盘到 C 柱

```
if n==1 then
    Move(A,C) //从 A 到 C 移动 2 个圆盘
else
    Hanoi(A,B,2n − 2)
    Move(A,C)
    Hanoi(B,C,2n − 2)
end if
```

(2) 由题意可得递推方程为

$$\begin{cases} T(n) = 2T(n-1) + 2 \\ T(1) = 2 \end{cases} \tag{1}$$

对递推方程变形得：$T(n)+2 = 2(T(n-1)+2)$，由等比数列性质解之得：$T(n) = 2^{n+1} - 2$

2.4 给定含有 n 个不同的数的数组 $L=<x_1,x_2,\cdots,x_n>$. 如果 L 中存在 x_i 使得 $x_1<x_2<\cdots<x_{i-1}<x_i>x_{i+1}>\cdots>x_n$，则称 L 是单峰的，并称 x_i 是 L 的“峰顶”. 假设 L 是单峰的，设计一个算法找到 L 的峰顶.

解.

思路分析：由题意得 L 是单峰的，即认为 x_i 存在且仅存在一个，所以首先对数组进行遍历，若当前元素的下一个元素值比当前元素小则停止遍历；当前元素即为峰顶。伪代码如下：

Algorithm 2 单峰查找问题

Require: L 含有 n 个不同数的数组且存在单峰

Ensure: 峰顶元素

```
1: if L[1] > L[2] then
2:    return L[1]
3: else if L[n] > L[n − 1] then
4:    return L[n]
5: else
6:    for i ← 2 to n − 1 do
7:       if L[i + 1] < L[i] then
8:          return L[i]
9:       end if
10:   end for
11: end if
```

显然该方法算法时间复杂度时 $O(n)$ 的。

由于数组在峰顶两侧均是单调的，考虑利用**二分查找**的方法来优化算法：对该问题，首先对端点值进行判断防止峰顶出现的端点；若端点处不是峰顶，则对中间部分进行搜索，首先取 $k=\lfloor\frac{n}{2}\rfloor$，接着对其两侧的元素进行比较，若 $L[k]>L[k+1]$ 并且 $L[k]>L[k-1]$，则 $L[k]$ 为峰顶元素；若 $L[k-1]<L[k]<L[k+1]$，则下一步对 $L[k+1]\sim L[right]$ 进行搜索；若 $L[k-1]>L[k]>L[k+1]$，则下一步对 $L[left]\sim L[k-1]$ 进行搜索。如此递归至最后即可找出峰顶元素。

相应的伪代码见下页：

该方法时间复杂度上显然是 $O(logn)$ 的，较顺序查找方法效率上可以获得更优。

Algorithm 3 单峰查找问题（应用二分查找优化）

Require: L　含有 n 个不同数的数组且存在单峰

Ensure: 峰顶元素

if $L[1] > L[2]$ **then**
  return $L[1]$
else if $L[n] > L[n-1]$ **then**
  return $L[n]$
else
  $x \leftarrow Search(L, 1, n)$
  return x
end if

Algorithm 4 二分查找单峰　$Search(L, left, right)$

Require: $L, left, right$　L 为待搜索的数组，$left$ 和 $right$ 为左右边界

Ensure: 峰顶元素

$k \leftarrow \lfloor \frac{n}{2} \rfloor$
if $L[k] > L[k+1]$ *and* $L[k] > L[k-1]$ **then**
  return $L[k]$
else if $L[k-1] < L[k] < L[k+1]$ **then**
  $Search(L, k+1, right)$
else
  $Search(L, left, k-1)$
end if

2.5　设 A 是 n 个不同的数排好序的数组，给定数 L 和 U，$L<U$，设计一个算法找到 A 中满足 $L<x<U$ 的所有的数 x.

解.

直接利用二分查找找到 L 和 U 的位置，若未找到 L，则返回第一个比 L 大的数的位置；若未找到 U，则返回最后一个比 U 小的数的位置；将两位置之间的数输出即可。伪码如下所示：

Algorithm 5

Require: A, L, U　A 为待搜索的有序数组，L 和 U 为左右边界

Ensure: 满足条件的数组成的数组

$n \leftarrow |A|$ // 数组的长度
$i \leftarrow BinarySearch(A, n, L, 1)$
$j \leftarrow BinarySearch(A, n, U, 0)$
if $i == -1$ *or* $j == -1$ **then**
  return
else if $i == j$ **then**
  return $A[i]$

```
8: else
9:    return A[i, j]
10: end if
```

Algorithm 6 二分查找 $BinarySearch(A, n, x, flag)$

Require: $A, n, x, flag$　A 为待搜索的有序数组，n 为数组长度，x 为目标元素,$flag == 1$ 表示搜索大于等于 x 的第一个数，否则搜索小于等于 x 的最后一个数

Ensure: $index$　目标元素所处的位置

```
1: start ← 1, end ← n
2: if flag == 1 then
3:    while start ≤ end do
4:       mid ← ⌊(start+end)/2⌋
5:       if L[mid] ≤ x then
6:          start ← mid + 1
7:       else
8:          end ← mid − 1
9:       end if
10:   end while
11:   if start == 1 then
12:      return -1
13:   else
14:      return start − 1
15:   end if
16: else
17:   while start ≤ end do
18:      mid ← ⌊(start+end)/2⌋
19:      if L[mid] < x then
20:         start ← mid + 1
21:      else
22:         end ← mid − 1
23:      end if
24:   end while
25:   if end == n then
26:      return -1
27:   else
28:      return end + 1
29:   end if
30: end if
```

2.6 设 M 是一个 n 行 n 列的 0-1 矩阵，每行的 1 都排在 0 的前面.

(1) 设计一个最坏情况下 $O(n\log n)$ 时间的算法找到 M 中含有 1 最多的行，说明算法的设计思想，估计最坏情况下的时间复杂度.

(2) 对上述问题，能否找到一个最坏情况下 $O(n)$ 时间的算法？

解.

(2) 对于 $O(n)$ 的算法设计如下：从第一行开始查找至最后一个为 1 的位置，记录此时的列号，向下寻找，若寻找到某一行该位置也为 1，则在该行继续向后查找至最后一个为 1 的位置，重复该操作直至走到矩阵最下方。伪码如下：

Algorithm 7 2.6 $O(n)$ 算法

Require: M 题设矩阵，假设每一行 1 的个数都不相等

Ensure: $TargetRow$ M 种含有 1 最多的行

```
1: TargetRow ← 1, column ← 1, row ← 1
2: while row ≤ n do
3:   if M[row][column] == 0 then
4:     row ← row + 1
5:   else
6:     TargetRow ← row
7:     for i ← column to n do
8:       if M[row][i] == 1 then
9:         column ← i
10:      else
11:        break
12:      end if
13:    end for
14:    row ← row + 1
15:  end if
16: end while
17: return TargetRow
```

2.7 设 A 是含有 n 个元素的数组，如果元素 x 在 A 出现的次数大于 $n/2$，则称 x 是 A 的主元素.

(1) 如果 A 中元素是可以排序的，设计一个 $O(n\log n)$ 时间的算法，判断 A 中是否存在主元素.

(2) 对于(1)中可排序的数组，能否设计一个 $O(n)$ 时间的算法？

(3) 如果 A 中元素只能进行“是否相等”的测试，但是不能排序，设计一个算法判断 A 中是否存在主元素.

解.

(2) 利用 $O(n)$ 的排序算法，排序后再进行顺序查找，对相同元素计数，总时间复杂度为 $O(n)+O(n)=O(n)$，伪代码如下所示：

Algorithm 8 2.7　　$O(n)$ 算法

Require: A　题设数组

Ensure: $STATUS$　布尔量，若存在主元素则为 $True$

$STATUS \leftarrow True$
$A \leftarrow sort(A)$ //利用 $O(n)$
$count \leftarrow 0, number \leftarrow A[1], n \leftarrow |A|$
for $i \leftarrow 1$ **to** n **do**
  if $A[i] == number$ **then**
    $count \leftarrow count + 1$
    if $count > \frac{n}{2}$ **then**
      return $True$
    end if
  else
    $number \leftarrow A[i]$
    $count \leftarrow 1$
  end if
end for
return $False$

(3) 直接利用暴力算法即直接让每一个数和数组中每个元素进行相等测试，则可得到每个数出现的次数，再与 $n/2$ 比较即可得出结果，该算法时间复杂度显然是 $O(n^2)$ 的。伪码如下所示：

Algorithm 9 2.7(3)　　暴力算法

Require: A　题设数组

Ensure: $STATUS$　布尔量，若存在主元素则为 $True$

$STATUS \leftarrow True$
$n \leftarrow |A|$
for $i \leftarrow 1$ **to** $\frac{n+1}{2}$ **do**
  $number \leftarrow A[i], count \leftarrow 0$
  for $j \leftarrow 1$ **to** n **do**
    if $A[i] == number$ **then**
      $count \leftarrow count + 1$
    end if
    if $count > \frac{n}{2}$ **then**
      return $True$
    end if
  end for
end for
return $False$