计算机学院《算法设计与分析》
(2020 年秋季学期)

第一次作业参考答案

1 请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明，可以假定 n 是 2 的幂次。**(每小题 3 分，共 21 分)**

1.

$$T(1) = T(2) = 1$$

$$T(n) = T(n-2) + 1 \quad if \quad n > 2$$

2.

$$T(1) = 1$$

$$T(n) = T(n/2) + 1 \quad if \quad n > 1$$

3.

$$T(1) = 1$$

$$T(n) = T(n/2) + n \quad if \quad n > 1$$

4.

$$T(1) = 1$$

$$T(n) = 2T(n/2) + 1 \quad if \quad n > 1$$

5.

$$T(1) = 1$$

$$T(n) = 4T(n/2) + 1 \quad if \quad n > 1$$

6.

$$T(1) = 1$$

$$T(n) = T(n/2) + \log n \quad if \quad n > 1$$

7.

$$T(1) = 1$$

$$T(n) = 3T(n/2) + n^2 \quad if \quad n > 1$$

解：

1. $T(n) = O(n)$
2. $T(n) = O(\log n)$
3. $T(n) = O(n)$
4. $T(n) = O(n)$
5. $T(n) = O(n^2)$
6. $T(n) = O(\log^2 n)$

   原式展开为：

$$\log n + \log(n/2) + \log(n/4) + \cdots + \log 2 + 1 = 1 + (1 + 2 + \cdots + \log n) \leq \sum_{i=1}^{\log n} i = O(\log^2 n)$$

7. $T(n) = O(n^2)$

2 寻找中位数问题 (19 分)

一组数据 $x = (x_1, \cdots, x_n)$ 从小到大排序后的序列为 $x'_1, \cdots, x'_n$，则这组数据的**中位数** M 为

$$M = \begin{cases} x'_{\frac{n+1}{2}} & 若n为奇数； \\ \frac{1}{2}(x'_{\frac{n}{2}} + x'_{\frac{n}{2}+1}) & 若n为偶数。 \end{cases}$$

给定两个长度分别为 n、m 的有序数组 $A[1..n], B[1..m]$（A 与 B 均已按从小到大排序）。请设计尽可能高效的算法，求出这两个数组中所有数据的中位数，并分析其时间复杂度。

解：
根据中位数的定义，问题即为寻找两个有序数组中第 k 小的数，其中

$$k = \begin{cases} \frac{n+m+1}{2} & 若n+m为奇数； \\ \frac{n+m}{2}和\frac{n+m}{2}+1 & 若n+m为偶数。 \end{cases}$$

此时，尝试比较 $A[\lfloor \frac{k}{2} \rfloor]$ 和 $B[\lfloor \frac{k}{2} \rfloor]$。由于 $A[\lfloor \frac{k}{2} \rfloor]$ 和 $B[\lfloor \frac{k}{2} \rfloor]$ 的前面有 $A[1..\lfloor \frac{k}{2} \rfloor - 1]$ 和 $B[1..\lfloor \frac{k}{2} \rfloor - 1]$，故对于 $A[\lfloor \frac{k}{2} \rfloor]$ 和 $B[\lfloor \frac{k}{2} \rfloor]$ 中的较小值，最多只有 $\lfloor \frac{k}{2} \rfloor - 1 + \lfloor \frac{k}{2} \rfloor - 1 \leq k - 2$ 个元素，故其不可能是第 k 小的元素了。

根据如上观察，可以归纳出两种情况：

1 如果 $A[\lfloor \frac{k}{2} \rfloor] \leq B[\lfloor \frac{k}{2} \rfloor]$，则可以排除 $A[1..\lfloor \frac{k}{2} \rfloor]$。

2 如果 $A[\lfloor \frac{k}{2} \rfloor] > B[\lfloor \frac{k}{2} \rfloor]$，则可以排除 $B[1..\lfloor \frac{k}{2} \rfloor]$。

再排除之后，对于新的两个数组和新的 k，可以继续迭代去寻找第 k 小的数。该算法的伪代码如 **Algorithm** 2所示。

注意到，每轮循环后需要查找的 k 至少减小了一半，而初始时 $k = (m+n)/2$ 或 $k = (m+n)/2 + 1$。故时间复杂度为 $O(\log(m+n))$。

Algorithm 1 $getKthEle(A[1..n], B[1..m], k)$

Input:
两个数组 $A[1..n], B[1..m]$;
Output:
两个数组的第 k 小元素

```
i, j ← 1, 1
while True do
    if i = m then
        return B[i + k − 1]
    end if
    if j = n then
        return A[j + k − 1]
    end if
    if k = 1 then
        return min{A[i], B[j]}
    end if
    ni ← min{i + k/2 − 1, m}
    nj ← min{j + k/2 − 1, n}
    pi, pj ← A[ni], B[nj]
    if pi ≤ pj then
        k ← k − (ni − i + 1)
        i ← ni + 1
    else
        k ← k − (nj − j + 1)
        j ← nj + 1
    end if
end while
```

3 局部最小值问题 (20 分)

给定一个由 $n(n \geq 3)$ 个互不相同的整数组成的数组 $A[1..n]$，其满足 $A[1] > A[2]$ 并且 $A[n-1] < A[n]$。我们定义数组的**局部最小值**为比它的两个相邻元素 (如果存在) 都小的整数。换言之，$A[x]$ 是局部最小值当且仅当它满足 $A[x] < A[x-1]$ 并且 $A[x] < A[x+1](1 < x < n)$。例如，下图所示数组中包含两个局部最小值，分别为 3 和 1。

9	3	7	2	1	4	5

求局部最小值显然有一个 $O(n)$ 的做法，仅需要扫描一遍整个数组就可以找到所有的局部最小值。请你给出一个算法可以在 $O(\log n)$ 的时间复杂度内找出一个数组的局部最小值。如果局部最小值有多个，仅需要找出任意一个局部最小值即可。(提示：我们给出的限制条件保证数组至少有一个局部最小值。)

Algorithm 2 $FindMedian2(A[1..n], B[1..m])$

Input:
两个数组 $A[1..n], B[1..m]$;
Output:
两个数组的中位数

```
tot ← n + m
if tot is odd then
    return getKthEle(A, B, (tot + 1)/2)
else
    return (getKthEle(A,B,tot/2) + getKthEle(A,B,tot/2+1)) / 2
end if
```

解：

如果 n 等于 3，做法是显然的。考虑 n 大于 3 的情况，此时令 $m = \lfloor \frac{n}{2} \rfloor$，来检查 $A[m-1], A[m], A[m+1]$ 之间的大小关系：

1. 如果 $A[m-1] > A[m]$ 并且 $A[m] < A[m+1]$，那么 $A[m]$ 就是局部最小值，算法结束。
2. 如果 $A[m-1] < A[m] < A[m+1]$，那么 $A[1..m]$ 中一定存在局部最小值，我们递归处理数组 $A[1..m]$ 即可。
3. 如果 $A[m-1] > A[m] > A[m+1]$，那么 $A[m..n]$ 中一定存在局部最小值，我们递归处理数组 $A[m..n]$ 即可。
4. 如果 $A[m-1] > A[m]$ 并且 $A[m] < A[m+1]$，那么左右两个区间中都存在局部最小值，我们任选一个区间递归处理即可。

任何一种情况每次递归都缩减了一半的规模，因此可以得出递归式 $T(n) \leq T(n/2) + O(1)$，解出算法的时间复杂度为 $T(n) = O(\log n)$

4 字符串等价关系判定问题 (20 分)

给定两个长度为 n 的字符串 A 和 B，若称 A 与 B 是**等价**的，当且仅当它们满足如下关系之一：

1. A 和 B 完全相同;
2. 若将把 A 分成长度相等的两段 A_1 和 A_2，也将 B 分成长度相等的两段 B_1 和 B_2。且他们之间满足如下两种关系之一：
   a. A_1 和 B_1 等价且 A_2 和 B_2 等价;
   b. A_1 和 B_2 等价且 A_2 和 B_1 等价;

请你设计一个高效的算法来判断两个字符串是否等价并分析你的算法的时间复杂度。

解法 1：

通过观察可以发现，题目中所描述的**等价**事实上是一种等价关系。它满足自反性、对称性和传递性，这意味着我们可以把这些字符串划分为不同的等价类。如果对每个等价类找出它们中字典序最小的一个字符串作为其代表元，这样仅需判断它们的代表元是否相等就可以判断两个字符串是否等价了。

给出一个字符串，找和它属于同一等价类的字典序最小元的方法也很简单，就是递归处理字符串的左右两个子串，把字典序较小的子串放在前面。算法实现请参考 **Algorithm 3**。

Algorithm 3 $Smallest(S[1..n])$

Input:
一个长度为 n 的字符串, $S[1..n]$;
Output:
字符串 S 所在等价类的最小元;

```
if n%2 = 1 then
    return S;
end if
m ← n/2;
S1 ← Smallest(S[1..m]);
S2 ← Smallest(S[m+1..n]);
if S1 < S2 then
    return S1 + S2;
else
    return S2 + S1;
end if
```

注：伪代码中的 $<$ 运算为字典序的比较，$+$ 运算为两个字符串直接拼接。例如 $ab < ac, bd < cd, ab + cd = abcd, cd + ab = cdab$。

每次递归都将问题分解为两个规模缩减一半的问题，之后合并由于涉及到字符串的比较以及拼接，时间复杂度为 $O(n)$，可以写出递归式 $T(n) = 2T(n/2) + O(n)$，解出算法的时间复杂度为 $O(n \log n)$。

解法 2（答出本解法可得 16 分）：

直接从等价关系入手，根据定义直接进行四次递归判断，可以写出一个递归算法 **Algorithm** 4。

Algorithm 4 $Trivial\_equivalence(A, B)$

Input:

两个长度为 n 的字符串, $A[1..n], B[1..n]$;

Output:

两个字符串是否等价;

```
if A = B then
    return true;
end if
if n%2=1 then
    return false;
end if
m ← n/2;
A1 ← A[1..m];
A2 ← A[m + 1..n];
B1 ← B[1..m];
B2 ← B[m + 1..n];
return Trivial_equivalence(A1, B1) and Trivial_equivalence(A1, B1) or
Trivial_equivalence(A1, B2) and Trivial_equivalence(A2, B1);
```

上述算法在最坏的情况下需要进行四次递归计算，递归式为 $T(n) \leq 4T(n/2) + O(n)$，从而得到算法的时间复杂度为 $O(n^2)$。

然而，通过如下分析，可以发现该算法仍有可以改进的地方。

1. 如果 A1 和 B1 等价且 A2 和 B2 等价，那么可以直接返回 A 和 B 等价。
2. 在 A1 和 B1 等价且 A2 和 B2 不等价的时候，我们没有必要再判断 A2 和 B1 以及 A1 和 B2 是否等价了，因为如果 A1 和 B2 等价，又因为之前已经判断过了 A1 和 B1 是等价的，因此 B1 和 B2 一定是等价的，又因为之前已经知道 A2 和 B2 不等价，因此 A2 和 B1 一定是不等价的。
3. 在 A1 和 B1 不等价的时候，我们显然也不需要判断 A2 和 B2 是否等价了，仅需要判断 A2 和 B1 以及 A1 和 B2 是否等价。

该算法的伪代码如 **Algorithm** 5所示。

因此，该算法最多仅需进行 3 次递归计算。递归式为 $T(n) \leq 3T(n/2) + O(n)$，解出其时间复杂度为 $O(n^{\log_2 3})$。

5 数字消失问题 (20 分)

给定一长度为 n 的数组 $A[1..n]$，其包含 $[0, n]$ 闭区间内除某一特定数 (记做消失的数) 以外的所有数字（例如 $n = 3$ 时，$A = [1, 3, 0]$，则消失的数是 2）。这里假定 $n = 2^k - 1$。

1. 请设计一个尽可能高效的算法找到消失的数，并分析其时间复杂度。(8 分)
2. 若假定数组 A 用 k 位二进制方式存储（例如 $k = 2$，$A = [01, 11, 00]$ 则消失的数是 10），且不可以直接访存。目前**唯一**可以使用的操作是 `bit-lookup`(i, j)，其作用是用一个单位时间去查询 $A[i]$ 的第 j 个二进制位。请利用此操作设计一个尽可能高效的算法找到消失的数，并分析其时间复杂度。(12 分)

Algorithm 5 $Equivalence(A, B)$

Input:
两个长度为 n 的字符串, $A[1..n], B[1..n]$;
Output:
两个字符串是否等价;

if $A = B$ **then**
  return true;
end if
if n%2=1 **then**
  return false;
end if
$m \leftarrow \frac{n}{2}$;
$A1 \leftarrow A[1..m]$;
$A2 \leftarrow A[m+1..n]$;
$B1 \leftarrow B[1..m]$;
$B2 \leftarrow B[m+1..n]$;
if $Equivalence(A1, B1)$ **then**
  if $Equivalence(A2, B2)$ **then**
    return true;
  else
    return false;
  end if
else
  return $Equivalence(A1, B2)$ **and** $Equivalence(A2, B1)$;
end if

解:

1. 考虑目前数组对应的值域为 $[L, R]$，则可以利用中位数 $mid = \lfloor \frac{L+R}{2} \rfloor$ 将数组划分成两部分: $\leq mid$ 的部分和 $> mid$ 的部分。若 $\leq mid$ 的部分的元素个数少于 $mid - L + 1$，则说明消失的数在 $[L, mid]$ 之中，递归考虑，否则我们递归考虑 $> mid$ 的部分。算法伪代码请参考 **Algorithm** 6。每次仅有至多一半的元素需要进入下一层递归，故递归式可写为 $T(n) = T(\frac{n}{2}) + n$。由主定理可知，时间复杂度为 $T(n) = O(n)$。

2. 类似第一问的思路可以考虑将数组按照二进制位逐位的进行划分，然后迭代的考虑按位划分后每一位个数较少的一部分。算法伪代码请参考 **Algorithm** 7。注意到每次循环迭代 S 的过程时，都对 S 集合进行了一次折半操作。故 `bit-lookup` 的操作次数为 $T(n) = \sum_{i=0}^{k-1} \frac{n}{2^k}$。求解该式可得时间复杂度为 $O(n)$。

Algorithm 6 $MissingInteger(A, i, j)$

Input:
数组 A，待寻找的值域 $[i, j]$
Output:
消失的数

$mid \leftarrow \lfloor \frac{i+j}{2} \rfloor$
if mid 不在 A 数组中 **then**
  return mid
end if
把 A 数组划分成 $B(\leq x)$ 和 $C(> x)$ 两个部分
if $Size(B) < x - i + 1$ **then**
  return MissingInteger(B, i, x)
else
  return MissingInteger$(C, x + 1, j)$
end if

Algorithm 7 $MissingInteger2(A, k)$

Input:

数组 A 及其对应的值域 $[0, 2^k - 1]$

Output:

消失的数

$S \leftarrow 1, 2, \cdots, n$
$S_0 \leftarrow S_1 \leftarrow \varnothing$
$count0 \leftarrow count1 \leftarrow 0$
for $posn = k$ **downto** 1 **do**
  for $i \in S$ **do**
    $bit \leftarrow$ `bit-lookup`$(i, posn)$
    if $bit \leftarrow 0$ **then**
      $count0 \leftarrow count0 + 1$
      $S_0 \leftarrow S_0 \cup \{i\}$
    else
      $count1 \leftarrow count1 + 1$
      $S_1 \leftarrow S_1 \cup \{i\}$
    end if
  end for
  if $count0 > count1$ **then**
    $missing[posn] \leftarrow 1$
    $S \leftarrow S_1$
  else
    $missing[posn] \leftarrow 0$
    $S \leftarrow S_0$
  end if
  $S_0 \leftarrow S_1 \leftarrow \varnothing$
  $count0 \leftarrow count1 \leftarrow 0$
end for
return $missing$