算法与复杂性　作业七

516021910528 - SHEN Jiamin

2020 年 3 月 30 日

1　0330

1. 有 n 个数存放在两个有序数组中，如何找到这 n 个数的第 k 小的数

比较两个数组的头部，将最小的一个提取出来。重复该步骤 k 次，第 k 个数即为第 k 小的数。该算法的时间复杂度为 $O(k)$

算法 1 归并取第 k 小的数

输入: 数组 $A[1\dots p], B[1\dots q]$；目标 k　　▷ $p+q=n\geq k$

输出: A, B 中第 k 小的数 m

```
1: cnt ← 0, i ← 1, j ← 1
2: while cnt < k ∧ i ≤ p ∧ j ≤ q do
3:     if A[i] < B[j] then
4:         m ← A[i], i ← i + 1
5:     else
6:         m ← B[j], j ← j + 1
7:     end if
8:     cnt ← cnt + 1
9: end while
10: while cnt < k ∧ i < p do
11:     m ← A[i], i ← i + 1
12:     cnt ← cnt + 1
13: end while
14: while cnt < k ∧ j < q do
15:     m ← B[j], j ← j + 1
16:     cnt ← cnt + 1
17: end while
```

算法 2 计数 KMP

输入: $Text[1 \dots n]$（被查找字符串）, $Pattern[1 \dots m]$（查找目标）

输出: cnt（$Pattern$ 在 $Text$ 中出现的次数）

```
1: procedure StringMatch(Text, n, Pattern, m)
2:     next ← ComputeNext(Pattern, m)
3:     i ← 1, j ← 1
4:     cnt ← 0
5:     while i ≤ n do
6:         if Pattern[j] = Text[i] then
7:             j ← j + 1, i ← i + 1
8:             if j = m + 1 then
9:                 j ← next[j] + 1
10:                cnt ← cnt + 1
11:            end if
12:        else
13:            j ← next[j] + 1
14:            if j = 0 then
15:                j ← 1, i ← i + 1
16:            end if
17:        end if
18:    end while
19:    return cnt
20: end procedure

21: procedure ComputeNext(Pattern, m)
22:    next[1 ... m + 1]
23:    next[1] ← −1, next[2] ← 0
24:    for i ← 3 to m + 1 do
25:        j ← next[i − 1] + 1
26:        while j > 0 ∧ Pattern[i − 1] ≠ Pattern[j] do
27:            j ← next[j] + 1
28:        end while
29:        next[j] ← j
30:    end for
31:    return next
32: end procedure
```

2. 如何修改 KMP 算法，使之能够获得字符串 B 在字符串 A 中出现的次数

给定原有算法，已经能找到一个 s，使得 $\forall 1 \le i \le m, A[s+i-1] = B[i]$。

如果存在一个 $s' = s + \delta, 1 \le \delta < m$，使得 $\forall i \in [1, m], A[s'+i-1] = B[i]$ （即 A 中有两个重叠的 B，重叠长度为 $m - \delta$），那么必然有即 $\forall 1 \le i \le m - \delta, B[i] = B[i + \delta]$ （即 B 长度为 $m - \delta$ 的前后缀是相等的）。

伪代码见算法2

3. 设计高效算法求序列 T 和 P 的最长公共子序列和最短公共超序列的长度。

(a) 最长公共子序列（LCS）L 定义为 T 和 P 的共同子序列中最长的一个

记 $T[:i]$ 为序列 T 的前 i 个元素组成的序列。记 $l_{LCS}(T, P)$ 为 T 和 P 的最长公共子序列的长度。不妨设序列 T 和 P 的长度分别为 m, n。

则有归纳关系

$$l_{LCS}(T[:i], P[:j]) = \begin{cases} l_{LCS}(T[:i-1], P[:j-1]) + 1 & , T[i] = P[j] \\ \max\{l_{LCS}(T[:i], P[:j-1]), l_{LCS}(T[:i-1], P[:j])\} & , T[i] \neq P[j] \end{cases}$$

可用动态规划的方法解决此问题。该算法的时间复杂度为 $O(|T \times P|)$，空间复杂度为 $O(\min\{|T|, |P|\})$。

算法 3 最长公共子序列的长度

输入: $P[1 \dots m], T[1 \dots n]$（两个序列） ▷ 不妨设 $m \ge n$

输出: l（T, P 的最长公共子序列的长度）

```
dp[0...1][0...n] ← {0}
for i ← 1 to m do
    for j ← 1 to n do
        if T[i] = P[j] then
            dp[i mod 2][j] ← dp[(i − 1) mod 2][j − 1] + 1
        else if dp[i mod 2][j − 1] ≥ dp[(i − 1) mod 2][j] then
            dp[i mod 2][j] ← dp[i mod 2][j − 1]
        else
            dp[i mod 2][j] ← dp[(i − 1) mod 2][j]
        end if
    end for
end for
return dp[m mod 2][n]
```

(b) 最短公共超序列（SCS）S 定义为所有以 T 和 P 为子序列的序列中最短的一个

$T+P$ 一定是 T 和 P 的一个公共超序列，所以 $|S| \leq |T|+|P|$。

当 T, P 存在一个公共子序列 Q 时，只需从 P 中移除一个 Q，然后将 P 中其他元素按照与 T 中属于 Q 的元素相对顺序不变的顺序插入 T 中，使 P 成为该新序列的子序列。此时该新序列为 T 和 P 的公共超序列，其长度为 $|T|+|P|-|Q|$。

所以

$$|S| = \min(|T|+|P|-|Q|) = |T|+|P|-\max(|Q|) = |T|+|P|-|L|$$

求 $|L|$ 的算法见算法3。

4. 给定规模为 n 的整数数组，如何知道其中是否有**两个数之和**恰好等于 x。给出算法及相应的时间复杂性。

记该数组为 A

解 1 建立一个可以容纳 n 个整数的哈希表 H。扫描整个数组，到第 i 个数时，计算 $r := x - A[i]$。若 r 不在哈希表中，记 $H[r] = i$；否则有 $A[i] + A[H[r]] = x$。哈希表查找和插入的时间复杂度为 $O(1)$，所以遍历整个数组的时间复杂度为 $O(n)$。

算法 4 求补 1

输入: $A[1 \ldots n]$（被查找的数组）

输出: 是否存在两个数之和恰好等于 x

```
1:  H ← Hashmap(n)
2:  for i ← 1 to n do
3:      r ← x − A[i]
4:      if H.contains(r) then
5:          return true                          ▷ 此时 A[i] + A[H[r]] = x
6:      else
7:          H[r] ← i
8:      end if
9:  end for
10: return false
```

解 2 将数组中的所有数分成两个堆，超过 $x/2$ 的所有数插入最大化堆 S，不足 $x/2$ 的所有数插入最小化堆 I。去两个堆最顶端的数 s, i 的和并与 x 比较：

- 若 $s+i=x$，则恰好符合条件

- 若 $s+i>x$，则移除 S 最顶端的元素并重新调整堆。因为 I 中所有的元素都比 i 大，不可能有一个 i' 使得 $i'+s\leq i$
- 若 $s+i<x$，则移除 I 最顶端的元素并重新调整堆。因为 S 中所有的元素都比 s 大，不可能有一个 s' 使得 $i+s'\geq i$

建堆的时间复杂度为 $O(n\log n)$，出堆的时间复杂度最大为

$$a\log a+(n-a)\log(n-a)\leq a\log n+(n-a)\log n=n\log n$$

所以该算法的时间复杂度为 $O(n\log n)$。

算法 5 求补 2

输入: $A[1\ldots n]$（被查找的数组）

输出: 是否存在两个数之和恰好等于 x

```
1: S ← MaximizeHeap(), I ← MinimizeHeap(), halfcnt ← 0
2: for i ← 1 to n do                                          ▷ 建堆
3:     if 2A[i] > x then
4:         S.Push(A[i])
5:     else if 2A[i] < x then
6:         I.Push(A[i])
7:     else                                                   ▷ 2A[i] = x
8:         halfcnt ← halfcnt + 1
9:     end if
10: end for
11: if halfcnt ≥ 2 then                                       ▷ 数组中恰好存在 2 个或更多 x/2
12:     return true
13: end if
14: while ¬S.empty ∧ ¬I.empty do                              ▷ 出堆
15:     s ← S.top, i ← I.top                                  ▷ 取堆顶的元素
16:     if s + i > x then
17:         S.Pop()
18:     else if s + i < x then
19:         I.Pop()
20:     else                                                  ▷ s + i = x
21:         return true
22:     end if
23: end while
24: return false
```

5. 每个螺母需要一个螺栓配套使用，现有 n 个不同尺寸的螺母和相应的 n 个螺栓，如何快速地为为每一个螺母找到对应的螺栓？只能将一个螺母与一个螺栓进行匹配尝试，从而知道相互之间的大小关系，不能够比较两个螺母的大小，也不能比较两个螺栓的大小。给出算法及相应的时间复杂性。

（类似于快排）

分组方法：随机取一个螺栓（记其大小为 x_0），与每个螺母都进行一次比较。在找到配对的螺母的同时，把比这个螺栓小的螺母和比它大的螺母分成两部分 Y_0, Y_1。有

$$\forall y_i \in Y_0, y_j \in Y_1, y_i < x_0 < y_j$$

然后从 Y_0, Y_1 两堆螺母里各选一个螺母（y_0, y_1），和螺栓匹配，同时将螺栓按小、中、大分成三堆 X_0, X_1, X_2。且有

$$\forall x_i \in X_0, x_i < y_0 \qquad \forall x_j \in X_2, x_j > y_0$$

此时从 X_0 中任选一个螺栓，其配对的螺母必然在 Y_0 中，候选螺母的数量大约减半。

归纳假设：因为螺栓与螺母是一一配对的，即

$$\forall x_i \in X, \exists y_i \in Y : x_i = y_j$$

所以则若有 $\forall x_i \in \hat{X}, x_j \in \complement_X^{\hat{X}} : x_i > x_j$，$\forall y_i \in \hat{Y}, y_j \in \complement_Y^{\hat{Y}} : y_i > y_j$ 则必有

$$\left|\hat{X}\right| \leq \left|\hat{Y}\right| \Rightarrow \hat{X} \subseteq \hat{Y}$$

即与 $\hat{X}$ 中的螺栓匹配的螺母全部都在 $\hat{Y}$ 中。所以一定可以用 $\hat{X}$ 中的一个螺栓，将 $\hat{Y}$ 分为两组（除非选中的螺栓匹配的螺母是 $\hat{Y}$ 中最小或最大的）。

算法描述：见算法6

复杂性分析：若只考虑拆分一个集合，不考虑拆分另一个集合时从本集合中抽取的元素。

将一个规模为 n 的集合分为两部分需要进行 n 次比较，且每分一次元素的总个数少 1 个。若要将集合分为 2^k 个部分，需要比较的次数为 $\sum_{i=0}^{k-1}(n-i)$，元素总数减少了 $\sum_{i=1}^{k} i$。此时，剩余的元素数量有 $n - \frac{(1+k)k}{2}$。使拆分结束，有

$$n - \frac{(1+k)k}{2} = 2^k \Rightarrow 2n = 2^{k+1} + k(1+k) \geq 2^{k+1}$$

$$k \leq \log n$$

所以总的比较次数为

$$\begin{aligned}\sum_{i=0}^{k-1}(n-i) = \frac{k\left[n+n-(k-1)\right]}{2} &= -\frac{1}{2}k^2 + (n+\frac{1}{2})k = -\frac{1}{2}k\left[k-(2n+1)\right] \\ &\leq -\frac{1}{2}(\log n)^2 + (n+\frac{1}{2})(\log n) \\ &= n\log n - \frac{1}{2}(\log n)^2 + \frac{1}{2}(\log n) = O(n\log n)\end{aligned}$$

算法 6 螺栓螺母配对

输入: $X[1 \dots n]$（螺栓），$Y[1 \dots n]$（螺母）

输出: $P = \{(x_i, y_j)\}$（配对的螺栓螺母）

$\mathcal{X} \leftarrow \{X\}, \mathcal{Y} \leftarrow \{Y\}$ ▷ 初始时，螺栓、螺母都只有一组
repeat
  $\hat{X} \leftarrow \mathcal{X}.back, \hat{Y} \leftarrow \mathcal{Y}.back$ ▷ 取最后面的一组（尺寸最大的）
  if $\hat{X}.length \leq \hat{Y}.length$ **then**
    $x \leftarrow \textsc{Select}(\hat{X})$ ▷ 从 $\hat{X}$ 中随机选一个 x，并从 $\hat{X}$ 中删除 x
    $Y_1, Y_2, y \leftarrow \textsc{Split}(x, \hat{Y})$ ▷ Y_1 中的螺母更小一些，Y_2 中的螺母更大一些
    $\mathcal{Y}.PopBack()$
    $\mathcal{Y}.PushBack(Y_1)$ if $\neg Y_1.empty$ ▷ 确保 $\mathcal{Y}$ 中靠前的分组中任何一个螺母的尺寸
    $\mathcal{Y}.PushBack(Y_2)$ if $\neg Y_2.empty$ ▷ 严格小于靠后的分组中所有螺母的尺寸
    $P.PushBack((x, y))$ ▷ 配对的
  else
    $y \leftarrow \textsc{Select}(\hat{Y})$ ▷ 从 $\hat{Y}$ 中随机选一个 y，并从 $\hat{Y}$ 中删除 y
    $X_1, X_2, x \leftarrow \textsc{Split}(y, \hat{X})$
    $\mathcal{X}.PopBack()$
    $\mathcal{X}.PushBack(X_1)$ if $\neg X_1.empty$
    $\mathcal{X}.PushBack(X_2)$ if $\neg X_2.empty$
    $P.PushBack((x, y))$
  end if
until $\neg\mathcal{X}.empty \wedge \neg\mathcal{Y}.empty$ ▷ $\mathcal{X}, \mathcal{Y}$ 应当同时为空

procedure $\textsc{Split}(pivot, C)$
  $S \leftarrow \Phi, I \leftarrow \Phi, peer$
  while $\neg C.empty$ **do**
    $c \leftarrow C.PopBack()$
    if $c > pivot$ **then**
      $S.PushBack(c)$
    else if $c < pivot$ **then**
      $I.PushBack(c)$
    else
      $peer \leftarrow c$
    end if
  end while
  return $I, S, peer$
end procedure

2 0402

6. 用分治法找到数组中的最大数和最小数，若数组规模为 2 的幂，证明需要的比较次数为 $\frac{3}{2}n-2$

易判断初始条件 $T(1)=0, T(2)=1$，且有递推关系

$$T(n)=T(2^m)=2T(2^{m-1})+2 \Rightarrow T(2^m)+2=2T(2^{m-1})+4$$

即

$$\frac{T(2^m)+2}{T(2^{m-1})+2}=2$$

所以当 $m\geq 1$ 时，$T(2^m)+2=3\cdot 2^{m-1}$，可得 $T(2^m)=\begin{cases}3\cdot 2^{m-1}-2 & ,m\geq 1\\ 1 & ,m=0\end{cases}$

即

$$T(n)=\begin{cases}\frac{3}{2}n-2 & ,n\geq 2\\ 1 & ,n=1\end{cases}$$

7. 对于任意给定的 4 个 1-10 之间的整数（可以相同），判断是否可以通过整数四则运算得到 24

我们将运算中不具有交换律的情况算作两种运算，并规定 $x_1\leq x_2$，使两个数构成有序数对。则任一个有序数对的两个数之间可能有 6 种运算，即

$$x_1+x_2 \quad x_1\cdot x_2 \quad x_1-x_2 \quad x_2-x_1 \quad x_1/x_2 \quad x_2/x_1$$

记 $Q(a,b)$ 为 a 与 b 进行这 6 种计算得到的所有结果的集合，$|Q(a,b)|\leq 6$。

记 $J(A,B,O,n)$ 为判断是否存在 A 中的某个数能和 B 中的某个数通过 O 中的运算直接得到 n。对于每一种运算，使用类似于算法4中的算法，首先扫描一遍较小的数组并在哈希表中记录下互补的值，然后扫描另一个数组看是否存在于哈希表中。这一部分需要进行 $\min\{|A|,|B|\}$ 次计算和 $\max\{|A|,|B|\}$ 次哈希表查找。所以 $J(A,B)$ 的时间复杂度为 $|O|(|A|+|B|)$。

4 个数的结合次序有两种，可表示为后缀表达式

- $(x_1,x_2,op_1,x_3,x_4,op_2,op_3)$：4 个数两两一组分为 2 组共有 $\frac{\binom{4}{2}+\binom{2}{2}}{2}=3$ 种分组方式。所以讨论全部结果需要执行 3 次

  $$J\left(Q(x_1,x_2),Q(x_3,x_4),\left\{+,\times,-,\div,\hat{-},\hat{\div}\right\},24\right)$$

  总的运算次数为 $3\times 6\times 12=216$

- $(x_1, x_2, x_3, x_4, op_1, op_2, op_3)$：此时若 op_2 与 op_3 的运算顺序是可交换的，则与上一种情况相同。
  - op_2, op_3 同为加、减法（可通过增减括号后等价）
  - op_2, op_3 同为乘、除法

  因此，给定 op_2 后，op_3 只有 3 种选择排列方式共有 $\binom{4}{2}\binom{2}{1} = 12$ 种。

  $$J\left(Q(Q(x_3, x_4), x_2), \{x_1\}, \{+, -, \hat{-}\}, 24\right)$$

  $Q(x_3, x_4)$ 的运算结果可由上一种情况的运算结果查表得到。运算次数为 $12 \times 6^2 \times 3 = 1296$ 次

总运算次数为 1512。

8. 有 $n = 2^k$ 位选手参加一项单循环比赛，即每位选手都要与其他 $n-1$ 位选手比赛，且在 $n-1$ 天内每人每天进行一场比赛

   (a) 设计算法以得到赛程安排

赛程安排可表示为表格 M 如下（以 $k = 2$ 为例）：

	1	2	3	4
1	×	0	1	2
2	−	×	2	0
3	−	−	×	1
4	−	−	−	×

上表中，$m_{1,2} = 0$ 表示 1 号选手和 2 号选手在第 0 天进行一场比赛。由于自己不能和自己比赛，所以对角线上的值是无意义的。同时表格中的值应关于对角线对称，所以只需考虑对角线一侧的值（这里只考虑上方）。因此只要以某种算法，使用 $[0, n-1)$ 中的整数填写上表，使每行、每列都不存在重复的数即可。

填写表格方法为：

- 第一个数：

  $$m_{1,2} = 0$$

- 第一行其他的数：

  $$\forall j : 2 < j \leq n, m_{1,j} = [1 + m_{1,j-1} \bmod (n-1)]$$

- 表中其他的数：

  $$\forall i, j : i > 1, j > i, m_{i,j} = [1 + m_{i-1,j-1} \bmod (n-1)]$$

算法 7 比赛安排

输入: n（参加比赛的人数）

输出: $M[1\ldots n][1\ldots n]$（比赛的安排，i 号选手与 j 号选手在第 $M[i][j]$ 天比赛）

```
1: M[1][2] ← 0
2: for j ← 3 to n do
3:     M[1][j] ← M[1][j − 1] mod (n − 1)
4: end for
5: for i ← 2 to n do
6:     for j ← i + 1 to n do
7:         M[i][j] ← M[i − 1][j] mod (n − 1)
8:     end for
9: end for
```

由于第一行中只有 $n-1$ 个数，所以恰能使用 $[0, n-1)$ 中所有的数填满且不重复。由于每一列中最多只有 $n-1$ 个数，且有循环递增关系，所以一定能使用 $[0, n-1)$ 中的数填满而不产生重复。由于第一行从左到右有循环递增关系，第二行的值为第一行对应的值 $+1$，所以行中的递增关系保持，即每一行中也没有重复的数。因此由此方法填写的表格满足要求。

该表格中安排了 $\frac{n^2-n}{2} = 2^{k-1}(n-1)$，满足每天 k 场比赛，$n-1$ 天恰好比完。算法的时间复杂度为 $O(n^2)$

(b) 若比赛结果存放在一矩阵中，针对该矩阵设计 $O(n \log n)$ 的算法，为各个选手赋予次序 P_i，使得 P_1 打败 P_2，P_2 打败 P_3，依此类推。

对于任意两个选手 i, j，定义若 i 赢了 j 则 $i < j$。由于每两个人之间都有一场比赛，所以任意两个人之间都可以按上述定义比较大小，该大小关系可以通过查询比赛结果的数组得到。使用一种基于比较的排序算法（如快速排序），即可在 $O(n \log n)$ 的时间内排除顺序。

9. 数组 A 中包含 n 个互不相同的整数，用 $O(n)$ 的时间找出 A 中最大的 i 个数，并按从大到小的次序输出（$i \leq n^{1/2}$）

首先使用自底向上的方法建最大堆，然后出堆 i 次。

- 自底向上建堆过程中，每一步比较的次数最多是该节点高度的 2 倍。对于高度为 i 的节点，记其所有子节点的高度和为 $H(i)$，则有

$$H(i) = 2H(i-1) + 1 \xrightarrow{H(0)=0} H(i) = 2^{i+1} - i - 2$$

而 n 个节点的最大高度为 $\log n$，所以这一部分的时间复杂度为

$$T_1(n) = 4n - 2\lceil \log n \rceil - 4$$

- 出堆 i 次所需的时间 $T_2(i) \leq i \log n$。

所以整体的时间复杂度

$$\begin{aligned} T(n,i) = T_1(n) + T_2(i) &\leq (4n - 2\lceil \log n \rceil - 4) + i \log n \\ &\leq 4n - 2\lceil \log n \rceil - 4 + \sqrt{n} \log n \\ &\leq 4n - 2\lceil \log n \rceil - 4 + \sqrt{n} \cdot n^{1/2} \\ &= 5n - 2\lceil \log n \rceil - 4 = O(n) \end{aligned}$$