数据结构与算法第三次作业

麻超　201300066

南京大学人工智能学院

2021 年 9 月 23 日

1

1.1 a

$$
\begin{aligned}
&T(n) = 2T(n/2) + n \\
&= 2(2T(n/2*2) + n) + n \\
&= 2(2(2T(n/2*2*2) + n) + n) + n \\
&= 2^3 T(n/2^3) + 3n \\
&= ... \\
&= 2^k T(n/2^k) + kn \\
&= nT(1) + kn \\
&= n(lgn + T(1)) >= nlgn \\
&\therefore T(n) \in \Omega(nlgn)
\end{aligned}
$$

1.2 b

$T(n) = \sum_{i=0}^{n/2} 2^i \times (n-2i) = n\sum_{i=0}^{n/2} 2^i - 2\sum_{i=0}^{n/2} 2^i i = n2^{n/2+1} - n - n2^{n+1} + 2^{n/2+2} <= 2^{n/2+2} <= c2^n$, 故 T(n) 将近上界为 2^n.

使用代入法证明如下: 使用常数 d, 有 $T(n) <= 2T(n-2) + dn <= 2 \times c2^{n-2} + dn = c2^{n-1} + dn <= c2^n$, 故正确, 即 T(n)= $O(2^n)$.(这里本应有一张图, 但它不见了, 我还没有找到问题所在)

2

2.1 a

$T(n) = \Theta(n^2)$

2.2 b

$T(n) = \Theta(nlgn)$

3

由课上所讲 2 进制数的简便乘法即可得到. 首先将该二进制数 X 分为左右两部分 A 和 B, 每段长为 n/2, 则

$X = A \times 2^{n/2} + B$

由原乘法公式即可得 $X^2 = AA2^n + ((A-B)*(B-A) + AA + BB)2^{n/2} + BB$

Algorithm 1 Square of n-digit binary number X

Input: $Xwhichisn-bit$
  if $n == 1$ **then**
    $re = X * X$
  else
    $A = X_left[n/2], B = X_right[n/2]$
    $p1 = A*A, p2 = (A-B)*(A-B), p3 = B*B$
    $re = p1*2^n + (p1+p3-p2)2^{n/2} + p3$
  end if
Output: re

在计算过程中, 只有三次乘法,5 次加减法, 和 2 次移位操作, 由此可得:

$T(n) = 3T(n/2) + O(n), n > 1$, 易得 $T(n) = O(n^{lg3})$

4

首先需要确定 n 的值, 由于该数组是已经排好序的, 并且当 i>n 时, 数组的值都为 ∞. 由所需要的时间复杂度来看, 应使用二分查找法.

首先假定一个 i 的值, 当 A[i]!=∞ 时, 将 i 的值翻倍, 若 $A[i] == \infty$ 时, 将 i 的值减半, 则经过数次后可以确定 i 的唯一值. 当确定 i 的值后, 即可以通过二分查找法确定是否存在 x.

在第一步中, 确定 i 的值为 high(有穷数组的最高位), 确定 low=0, 从 high 和 low 通过二分法向中间逼近. 确定 $mid = (high + low)/2$. 若 A[mid]>X, 则令 high=mid, 若 A[mid]<X, 则令 low=mid, 再次递归查找. 当 A[mid]==X 时, 即可以返回 mid 的值, 当 low>=high 且没有查找到对应的 X 值时, 即可以返回没有该值的信息.

5

一共有 n 位代表，设一共有 m 个政党. 则选取其中一人, 将该人与其他人配对, 可以得到 n-1 个结果, 与该人意见相合的组成一部分, 意见不合的组成另一部分, 若相合的一部分人已经超过半数, 即可确定其对应的政党为多数党, 否则, 在另一部分中再次重复操作, 直至选出多数党或者 m>=n-1.

Algorithm 2 Select the majority party

Input: $A[n], m$
  function SEARCH($A[], k$)
    $i \leftarrow 1$
    $Agree[], Disagree[]$
    while $i < k$ **do**
      $CompareA[i], A[0]$ ▷ 判断是否为同一政党
      if $A[i] == A[0]$ **then** ▷ 为同一政党
        $Agree.append(A[i])$
      else
        $Disagree.append(A[i])$
      end if
    end while
    if $Agree.size() > Disagree.size()$ **then**
      return $Agree[], True$
    else
      return $Disagree[], False$
    end if
  end function
  $RE[], Judge \leftarrow Search(A[], n), num = 1$
  if $Judge == 1$ **then**
    return $RE[]$
  else
    while $num < m - 1 \&\& Judge = False$ **do**
      $Search(RE[], RE.size())$
      $num \leftarrow num + 1$
    end while
  end if
  return $RE[]$

在该算法中, 只有当迭代到了最后一步或者恰好能够判断多数党的时候才可以

退出.

当只有两个政党时, 可以直接决出多数党并返回, 算法是正确的. 当有三个政党时, 分为两种情况, 第一种情况是判断出的政党超过总人数一半, 即为多数党, 可以返回. 第二种情况是判断出的政党不足总人数一半, 此时需要对 Disagree 数组再次进行遍历即可得到多数党. 此时 num(遍历次数) 值为 1, 且 Judge(能否确定为多数党) 为 False, 所以再次遍历, 那么这之后 num 就等于 2, 便会退出循环, 得到人数占优的那一部分人员, 无论最后决出的 Judge 是 True 还是 False, 都得到了最后结果, 是正确的. 当政党数量更多时, 道理同 m=3 时, 可以得到其是正确的.

该算法最好的情况下时间复杂度为 O(n), 最差情况下为 $O(n^m)$.

6

如果要让时间复杂度减小到 O(n), 一个比较可行的办法是将横跨左右两个数组的左右数组分别分为在左数组的一部分和右数组的一部分, 即从 mid 这个位置开始, 向左右两边延申, 求出分别最大的数组, 再求和.

Algorithm 3 Modify the original algorithm

Require: $A[], low, high$
Ensure: $totalsum, max_left, max_right, crossingA$
  if $low == high$ **then**
    return $(A[l], A[l], A[l], A[l])$ ▷ 只有一个元素
  end if
  $mid \leftarrow (low + high)$
  $(lsum, l_lmax, l_rmax, leftmax) \leftarrow FMS(A, low, mid)$
  $(rsum, r_lmax, r_rmax, rightmax) \leftarrow FMS(A, mid+1, high)$
  $crossmax \leftarrow l_rmax + r_lmax$
  $lmax \leftarrow max(l - lmax, lsum + r - lmax)$
  $rmax \leftarrow max(r_rmax, rsum + l_rmax)$
  return $(lsum + rsum, lmax, rmax, max(leftmax, rightmax, crossmax))$

这个过程的输出是正确的, 因为 crossmax 的值等于 $max_{l<=u<=mid<v<=r} \sum_{i=u}^{v} a_i$, 因为 l_rmax 是到 mid 时的最大值, 同样地,r_rmax 也是从 mid+1 开始的最大值。其他的正确性类似.

时间复杂性: 在合并的过程中只用了 $\Theta(1)$time. 所以 $T(n) = 2T(n/2) + \Theta(1)$. 所以 $T(n) = \Theta(n)$.

7

7.1 a

由最大堆的性质 (每一个根节点对应的值都比叶节点对应的值更大), 可以确定第二大的值必然在根节点对应的两个叶节点里, 只需要比较这两个值的大小即可. 算法时间在常数时间内.

7.2 b

由最大堆的性质 (每一个根节点对应的值都比叶节点对应的值更大), 可以考虑从根节点开始遍历, 每次取出当前最大的一个值, 直到第 k 个最大的值.

方法如下: 首先该二叉树是标准的最大堆, 从根节点开始遍历取当前最大值, 第一个值肯定为根节点, 随后比较根节点的 left 和 right, 并将大的叶节点 (第 1 层) 定为新的根节点, 此时原叶节点对应的两个叶节点 (第 2 层) 通过同样的办法让小的叶变为大的叶的叶, 依此类推. 这样最后得到的二叉树还是一个最大堆, 再次通过同样的办法取出当前根节点, 依此类推, 直至找出第 k 个值. 可以利用交换根节点与对应叶节点的值来简便地完成构建新的最大堆.

Algorithm 4 Take the kth largest element in a max-heap

Require: $max-heap, k, n(size)$
Ensure: $A[k-1]$
  $A[], i = 1$
  while $i <= k$ **do**
    $n \leftarrow takemaxnum(root)$
    $A[i-1] \leftarrow n$
    $i \leftarrow i+1$
  end while
  function TAKEMAXNUM($root$)
    $m \leftarrow 0$
    while $root.left! = null \&\& root.right! = null$ **do**
      if then$root.left > root.right$
        $swap(root, root.left)$
      else$swap(root, root.right)$
      end if
    end while
    return $takemaxnum(root)$
  end function

在该算法中, 由于使用的是最大堆, 所以该树的深度最多为 $log_2 n+1$, 每次取出最大值需要经过这么多次交换, 一共需要执行 k 次, 所以时间复杂度 $T(n) = O(klgk)$