数据结构与算法第四次作业

麻超　201300066

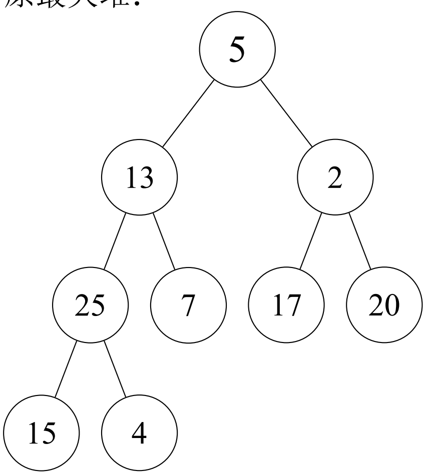
南京大学人工智能学院

2021 年 10 月 9 日

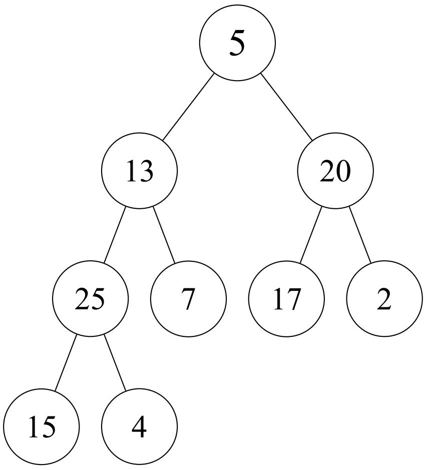
Problem 1

a

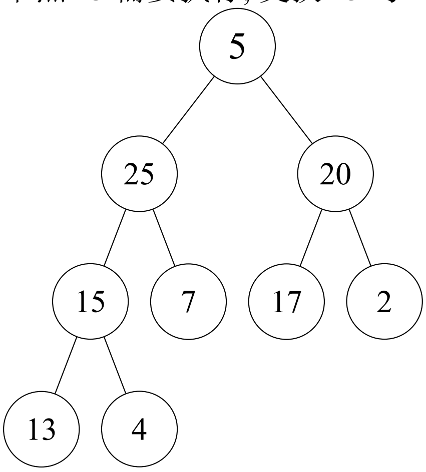
原最大堆：



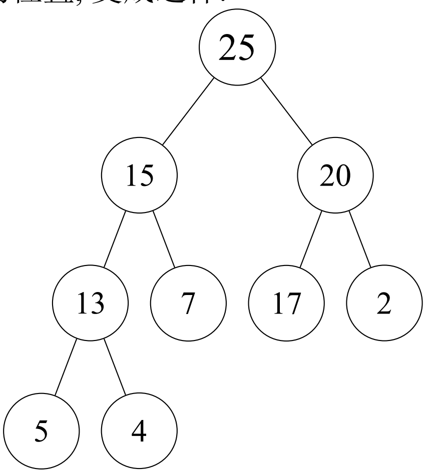
数组长度为 9，故从 4 开始向 1 循环，依次执行 MAX-HEAPIFY(A,i) 过程. 节点 25 不需要执行该过程, 节点 2 需要执行, 交换 2 与 20 的位置, 变成这样:



节点 13 需要执行, 交换 13 与 25 的位置, 再次交换 13 与 15 的位置, 变成这样:



节点 5 需要执行, 交换 5 与 25 的位置, 再次交换 5 与 15 的位置, 再次交换 5 与 13 的位置, 变成这样:



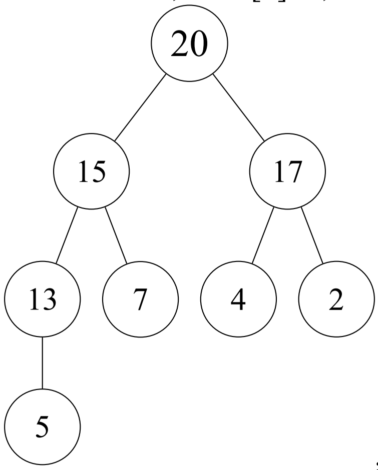
b

给定堆结构如 a) 中的结构.

从 A.length 开始循环到 2, 每次交换 A[i] 与 A[1] 的位置 (取出 A[1]), 再将其调

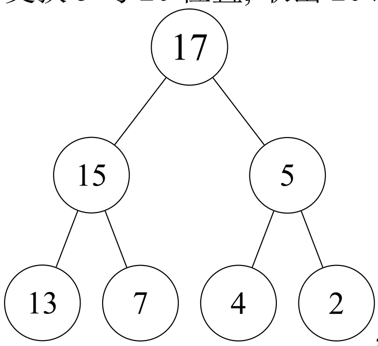
整为最大堆.

首先取出 25, 令 A[1]=4, 再交换 4 和 20 位置, 交换 4 和 17 位置. 如下:



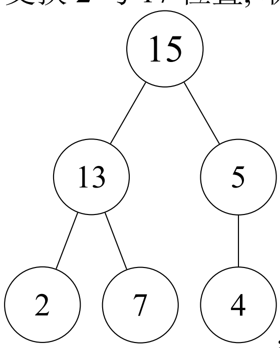
,A=[25]

交换 5 与 20 位置, 取出 20 之后, 交换 5 与 17 位置.



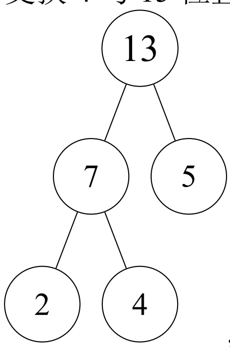
,A=[25,20]

交换 2 与 17 位置, 取出 17, 交换 2 与 15 位置, 交换 2 与 13 位置.



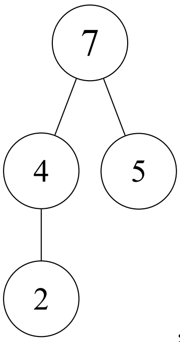
,A=[25,20,17]

交换 4 与 15 位置, 取出 15, 交换 4 与 13 位置, 交换 4 与 7 位置.



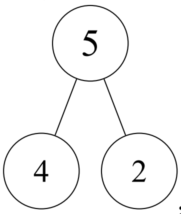
,A=[25,20,17,15]

交换 4 与 13 位置, 取出 13, 交换 4 与 7 位置.



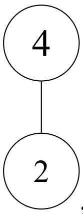
,A=[25,20,17,15,13]

交换 2 与 7 位置, 取出 7, 交换 2 与 5 位置.



,A=[25,20,17,15,13,7]

交换 2 与 5 位置, 取出 5, 交换 2 与 4 位置.



,A=[25,20,17,15,13,7,5]

交换 2 与 4 位置, 取出 4, 此时原堆中只有 2, 取出 2, 则 A=[25,20,17,15,13,7,5,4,2]

Problem 2

可以选择创建一个大小为 k 的数组，将 k 个排序链表中的第一个元素依次存放到数组中，然后将数组调整为最小堆，这样保证数组的第一个元素是最小的，也是这些链表中最小的元素，假设其为 min，将 min 从该最小堆取出并存放到空链表中，此时将 min 所在链表的下一个元素到插入的最小堆中，重复上面的操作，直到堆中没有元素为止。

假设初始链表为 [5,15,20][4,12,17][0,3,8,18], 则首先以每个链表的最小元素构建一个最小堆 [0,4,5], 从中取出 0, 将 0 对应链表中的第二个元素 3 加入最小堆中, 再次构建最小堆 [3,4,5], 取出 3, 将 8 加入最小堆, 再次构建为 [4,5,8], 取出 4..., 以此类推, 最后可以得到合并后的链表.

伪代码如下:

时间复杂度分析: 每次取出最小的元素只需要 O(1) 时间, 每次取出之后再次最小堆化需要 O(lg$k$) 时间, 一共有 n 个元素, 故最后的时间复杂度为 O(nlgk).

Algorithm 1 Merge k sorted lists

Require: $k-sorted-lists$
  $heaparr = [], rearr = []$
  for $i = 1 \rightarrow k$ **do**
    $heaparr[i] \leftarrow list[i][0]$
  end for
  $BUILD-MIN-HEAP(heaparr[])$
  while do$heaparr[].size! = 0$
    $MIN-HEAPIFY(heaparr[], 1)$
    $rearr[] \leftarrow heaparr[1]$
    if $heaparr[1].next! = NULL$ **then**
      $heaparr[1] \leftarrow heaparr[1].next$
    else
      $swap(heaparr[1], heaparr[k])$
      $k \leftarrow k-1$
    end if
  end while
Ensure: $rearr[]$

Problem 3

a

该算法利用了递归与分治思想, 将整个大的数组分为若干小的数组, 再次结合起来, 所以 Unusual() 函数的作用就是将每一部分排序为正确的顺序.

Cruel() 函数作用为分治, 其正确性显而易见.

以下证明 Unusual() 函数对 $n = 2^k$ 正确.

奠基: 当 n=2 时, 直接判断两个元素的大小并排序, 正确.

归纳假设: 设当 $n = 2^k$ 时该过程正确, 即可以正确排序, 则以下只需要证明 Unusual() 函数可以将已经排好序的左右两部分合并为正确的顺序.

由于左右两边已经排好序, 所以每一个部分内从左到右的数值都是依次增大的.

将原数组设为共有 4 个部分, 即左部分的左半边, 左部分的右半边, 右部分的左半边, 右部分的右半边.

1) 若左部分全小于右部分, 此时先将 2,3 部分对调, 两边分别排序后, 两部分依然符合顺序关系 (即可以跳过 7-8 行,) 再对 2,3 部分排序, 回复至原来的情况, 正确.

2) 若左部分全大于右部分, 则对调再分别排序后的顺序情况应为 3142(部分), 最后再给 1,4 部分排序 (即交换位置), 最后得到的结果为 3412(部分), 正确.

3) 介于 1 和 2 之间的情况, 该算法实质上会将左部分较小的半边和右部分较

小的半边进行比较得到正确结果, 将左部分较大的半边和右部分较大的半边进行比较, 最后再比较中间几个数字, 且由于归纳假设关系, 最左边 n/4 的数字一定都比后面的小, 最右边同理, 所以中间部分排序结束就可以决定整个数组的顺序, 故该算法正确, 得证.

b

如数组 [6,7,4,5], 若去掉 for 循环, 则最后结果为 [7,6,5,4]->[6,7,5,4]->[6,7,4,5]->[6,4,7,5]->[6,4,7,5].

c

如数组 [6,7,4,5], 若更换最后两行的位置, 则最后结果为 [6,7,4,5]->[6,4,7,5]->[4,6,7,5]->[4,6,7,5]->[4,6,5,7].

d

每次执行交换两数的操作需要 O(1) 时间.for 循环占用时间为 $\Theta(n)$. 则:

$T(n) = 3 \times T(n/2) + \Theta(n) = n + n \times \frac{3}{2} + n \times \frac{9}{4} + ... + n \times (\frac{3}{2})^{\lg n} = 2n^{\lg 3} = \Theta(n^{\lg 3})$

Cruel() 函数的时间复杂度为 $T(n) = 2T(n/2) + \Theta(n^{\lg 3})$, 递归树的层数为 lgn, 故其时间复杂度为 $T(n) = \Theta(n^{\lg 3} lgn)$

Problem 4

a

利用归纳法以证明结论:

奠基: 如果 A 只包含一个元素, 则 p=r. 算法立即终止, 已排序.

归纳假设: 设对于任意的 $1 \leq k \leq n-1$,TRQUICKSORT() 都能够正确地对有 k 个元素的数组 A 排序. 则设 A 的大小为 n, 设 q 为主元.

根据归纳假设, 左子数组的大小小于 n, 所以 TRQUICKSORT() 能对左子数组进行正确排序, 接下来令 p=q+1, 其同样可以对右子数组进行排序.

故由归纳法,TRQUICKSORT() 可以对 A 进行排序.

b

由 (a) 知, 每次递归调用都把相关信息压入至栈中,$TRQUICKSORT(A,p,r)$ 的第三个参数会经过小于 r-p=n 次递归调用就能满足 p=r, 达到返回条件. 所以对自身递归调用不超过 n 次. 每次调用过程值需要 O(1) 的空间, 所以不超过 n 次调用就是 $\Theta(n)$ 的空间. 即此时 $\Theta(n)$ 就为栈深度.

c

Algorithm 2 IM-TRQUICKSORT

$IM-TRQUICKSORT(A,p,r)$
while $p<r$ **do**
  $q=PARTITION(A,p,r)$
  if $q-p<r-p$ **then**
    $IM-TRQUICKSORT(A,p,q-1)$
    $p\leftarrow q+1$
  else
    $IM-TRQUICKSORT(A,q+1,r)$
    $r\leftarrow q+1$
  end if
end while

Problem 5

a

每次调用 SqrtSort() 函数可以对 $\sqrt{n}$(为整数) 个元素进行排序. 可以考虑通过从 k=1 开始遍历, 调整最后的几个元素为最大的元素. 这样每次可以减少对 $\sqrt{n}-1$ 个元素再次进行排序, 以此类推, 最终可以实现对整个数组的排序过程.

伪代码如下:

设原数组中有 n 个元素, 每两次进行 SqrtSort() 操作时有 $\sqrt{n}-1$ 个元素时重复的, 以此类推, 当遍历过第一遍之后, 原数组的最后 $\sqrt{n}-1$ 个元素肯定是最大的几个且按照大小顺序排列好的, 那么下一次循环时就可以不用管这几个数字, 以此类推, 直到最后需要遍历的几个元素与 $\sqrt{n}$ 比较接近时, 无需进行下一次遍历. 这样最后得到的数组一定是排好序的.

最坏情况下: 第一次遍历需要调用 $n-\sqrt{n}+1$ 次, 第二次循环时, 需要调用 $n-2\sqrt{n}+2$ 次, 以此类推, 最后一次需要调用 1 次. 总数为 n 时, 每次循环调用次数符

Algorithm 3 SqrtSort

Require: $SqrtSort(k), A[1...n]$
  $length = A.size, looptime = 0$
  while $length >= \sqrt{n}$ **do**
    for $k = 0 \to n - \sqrt{n} - looptime * (\sqrt{n} - 1)$ **do**
      $SqrtSort(k)$
    end for
    $looptime \leftarrow looptime + 1$
    $length \leftarrow length - \sqrt{n} + 1$
  end while
Ensure: $sortedA[1...n]$

合数列 $a_k = (1-\sqrt{n})(k-1) + 1 + \sqrt{n}(\sqrt{n}-1) = n - k(\sqrt{n}-1)$.k 的取值范围是 1 到 $\sqrt{n}+1.sum = \frac{(n-1)\sqrt{n}}{2} + \sqrt{n} + 1$.

b

Problrm 6

a

第一次调用该函数, 如果 FairCoin 返回的值是 0, 那么该函数返回 0, 否则返回 1-OneInThree(). 再次调用该函数, 如果 FairCoin 返回的值是 0, 则退出递归, 返回 1, 否则再次递归..., 通过分析可知, 如果想要让返回值是 1, 那么第二次调用 OneInThree() 时需要返回 0, 分为直接返回和再次递归两种, 直接返回的概率是 1/4, 再次递归时, 如果要返回 0, 那么第三次调用需要返回 1, 即 (1-(1-1))=0, 即第四次需要返回 0, 概率为 1/16..., 以此类推, 如果想要返回值为 1, 则任意一次偶数次递归时需要返回 0, 则转变为求 $1/4 + 1/16 + 1/64 + ... + 1/4^n$, 由级数可知, 其和为 1/3.

b

当第一次直接返回 0 时, 调用一次 FairCoin, 概率为 1/2. 当第一次为 1, 第二次为 0 时, 调用两次, 概率为 1/4..., 也就是说, 当调用该函数返回值为 0 时, 便退出该程序. 所以总的期望是 $1 \times 1/2 + 2 \times 1/4 + 3 \times 1/8 + ... + n \times 1/2^n = \sum_{n=1}^{\infty} n/2^n = 2$

c

由于不知道概率 p, 可以考虑连续两次调用 BiasedCoin 函数, 若第一次为 0, 第二次为 1, 则返回 0, 第一次为 1, 第二次为 0 则返回 1, 否则再次调用, 这样返回 1 和 0 的概率都是 P(1-p).

伪代码如下:

Algorithm 4 OneInTwo

```
while True do
    x ← BiasedCoin()
    y ← BiasedCoin()
    if x ≠ y then return x
    end if
end while
```

d

在第一次循环以内就可以返回 0 或 1 的概率是 2p(1-p), 此时需要调用 Biased-Coin() 两次, 第二次循环内可以返回的概率是 (1-2p(1-p))(2p(1-p)), 需要调用四次, 令 m=2p(1-p), 则:

期望 $=m\times 2+m(1-m)\times 4+m(1-m)^2\times 6+...+m(1-m)^{n-1}\times 2n$

$=\sum_{n=1}^{\infty} m(1-m)^{n-1}\times 2n$

Problem 7

只需要 20 步操作且必须需要 20 步操作便可以问出 Eve 所想的数字.

具体原理是利用二分法, 考虑到 $2^{20}=1048576$, 与 1,000,000 较接近. 所以可以通过 1048576 来进行二分操作, 每次确定该数字在原区间的一半区间, 只需要 20 次便可以确定, 在询问 20 次之后, 必然可以得到一个大小为 1 的区间, 则该数字就在这个区间内, 以此确定该数的值. 算法上界和下界都为 20.

以下为一个例子: 设该数为 654321.

1. Q: 它比 524288 大吗?(2^{19})

   A:Yes.

2. Q: 它比 786432 大吗?($524288+2^{18}$)

   A:No.

3. Q: 它比 655360 大吗?($786432-2^{17}$)

   A:No.

4. Q: 它比 589824 大吗?($655360-2^{16}$)

   A:Yes.

5. Q: 它比 622592 大吗?($589824+2^{15}$)

   A:Yes.

6. Q: 它比 638976 大吗?($622592+2^{14}$)

   A:Yes.

7. Q: 它比 647168 大吗?($638976+2^{13}$)

   A:Yes.

8. Q: 它比 651264 大吗?($647168+2^{12}$)

   A:Yes.

9. Q: 它比 653312 大吗?($651264+2^{11}$)

   A:Yes.

10. Q: 它比 654336 大吗?($653312+2^{10}$)

    A:No.

11. Q: 它比 653824 大吗?($654336-2^{9}$)

    A:Yes.

12. Q: 它比 654080 大吗?($653824+2^{8}$)

    A:Yes.

13. Q: 它比 654208 大吗?($654080+2^{7}$)

    A:Yes.

14. Q: 它比 654272 大吗?($654208+2^{6}$)

    A:Yes.

15. Q: 它比 654304 大吗?($654272+2^{5}$)

    A:Yes.

16. Q: 它比 654320 大吗?$(654304+2^4)$

    A:Yes.

17. Q: 它比 654328 大吗?$(654320+2^3)$

    A:Np.

18. Q: 它比 654324 大吗?$(654328-2^2)$

    A:No.

19. Q: 它比 654322 大吗?$(654324-2^1)$

    A:No.

20. Q: 它比 654321 大吗?$(654322-2^0)$

    A:No.

到此为止, 通过第 16 行和第 20 行, 这个数比 654320 大, 但是不比 654321 小, 故这个数只能是 654321. 其他情况同理, 最多只需要 20 次即可得到结果.

Problem 8

由定理 8.1, 在最坏情况下, 任何比较排序算法都需要做 $\Omega(n\lg n)$ 次比较, 即其时间复杂度为 $\Omega(n\lg n)$. 堆排序显然属于比较排序. 另由第 6 章内容, 堆排序的时间复杂度为 O(nlgn), 综上所述,HEAPSORT 的时间复杂度为 $\Theta(n\lg n)$.