Week 9

软73 沈冠霖 2017013569

April 23, 2019

1 T1

证明：假设删除一个元素没有导致size改变，那么$c_i = 1$,$\hat{c}_i = c_i + \phi_i - \phi_{i-1} = 1 + |2T.num_i - T.size_i| - |2T.num_{i-1} - T.size_{i-1}|$，而$T.size_i = T.size_{i-1}, T.num_i = T.num_{i-1} - 1$，因此原式可化为
$\hat{c}_i = 1 + |2T.num_i - T.size_i| - |2T.num_i - T.size_i + 2| \leq 3$
而如果删除一个元素导致size改变了，那么有$c_i = 1 + T.numi, T.size_i = \frac{2}{3}T.size_{i-1}, T.num_i = T.num_{i-1} - 1, T.num_{i-1} = \frac{1}{3}T.size_{i-1}$，原式可化为
$\hat{c}_i = 1 + T.numi - 2T.num_i + T.size_i + 2T.num_{i-1} - T.size_{i-1}$
$= 3 + \frac{1}{3}T.size_{i-1} - 1 + \frac{2}{3}T.size_{i-1} - T.size_{i-1} = 2$

2 T2

2.1 1

最坏情况是每个数组都是满的，搜索没个数组的时间复杂度是$O(i) = O(lgn)$，一共$\lceil lg(n+1) \rceil = O(lgn)$个数组，最坏时间复杂度是$O((lgn)^2)$

2.2 2

算法 从m-1个数插入一个数使得数组变为m个数的时候，找到最小的二进制位号k使得二进制代表2^{k-1}的一位从0变1，那么前面k-1位一定都是从1变0。
把前面k-1个数组和新的数一起做归并锦标排序（每个数组待更新的第一个数构成一个败者树，每次更新把败者树根节点推入第k个数组中，之后更新败者树。直到第k个数组被排满为止

复杂度 因为败者树一共m个叶子节点，初始建树代价O(m)，每次更新代价O(lgm)，一共需要更新2^m次，因此时间复杂度$O(lgm2^m)$。而最坏情况是第$\lceil lg(n+1) \rceil$位从0变成1，时间复杂度就是O(nlglgn)
而采用聚集法进行均摊分析，令$k = \lceil lg(n+1) \rceil$，则$2^{k-1} \leq n < 2^k$，那么$m = k-1$最多有1次，$m = k-2$最多有2次，以此类推，$m = i$最多有$2^{k-i-1}$次。对前m-1个数组和新加的数做归并排序进入第m个数组的时间复

杂度是$O(lgm2^m)$，因此对长度为0到n-1的数组插入1个数的总共时间复杂度求和，总复杂度为$\sum_{i=0}^{k-1}2^{k-1}lgi = k2^{k-1}lg(k-1)! = O(2^{k-1}k^2lgk) = O(n(lgn)^2lglgn)$，均摊后每次插入时间复杂度是$O((lgn)^2lglgn)$

2.3 3

算法 从长为m的数组删除一个数的时候，也必定可以找到最小的二进制位号k使得二进制代表2^k的一位从1变0，那么前面k-1位一定都是从0变1。
先把待删除的数a插入第k个数组中，也就是用a替换大于等于a的最小数b。之后把b插入a原来所在的数组。（如果a原来就在第k个数组中，则不必操作）
之后把第k个数组按照顺序依次写入前k-1个数组里。

复杂度 先考虑对于任意一个长度m删除一个数的最坏情况：这些数组满足一个条件：每个数组内部由小到大排序，但是标号小的数组中任何一个数都比标号大的数组中任一个数大。而且每次都是删除标号最大的满数组中的一个数，设这个最大标号是i_{max}，而二进制由1变0的最小标号是i_{min}。把待删除的数m放到标号i_{min}的数组里需要$O(i_{min})$的时间，而把替换得到的数换到标号为i_{max}的数组则需要$O(2^{i_{max}})$的时间。而把标号为i_{min}的数组分到前i_{min}个数组中需要时间$O(2^{i_{min}})$。
对于长度1到n的最坏情况是$i_{max} = \lceil lg(n+1) \rceil, i_{min} = \lceil lg(n+1) \rceil - 1$，此时最坏时间复杂度是O(n)。
先均摊i_{min}和把标号为i_{min}的数组分到前i_{min}个数组中的时间，使用聚集法进行均摊。长度为1到n中，$k = \lceil lg(n+1) \rceil$，则$i_{min} = j$最多出现$2^{k-j-1}$次，总共的时间是$\sum_{j=0}^{k-1}2^{k-j-1}2^j = O(nlgn)$，平均每次时间$O(lgn)$。
而再均摊插入数组的时间，对于长度m，其最坏情况就是标号为$t = \lceil lg(m+1) \rceil - 1$的数组中删除数字了。对于长度1到n，$k = \lceil lg(n+1) \rceil$，$t = j$最多出现$2^j$次。总共时间就是$\sum_{j=0}^{k-1}2^j2^j = O(4^k) = O(n^2)$,平均时间为O(n)。
两者时间相加，可以得出删除的均摊时间是O(n)，当然，这里每次都是考虑的最坏情况，其期望的时间复杂度应该远小于这个数字。

3 T3

3.1 1

算法 如果$k \leq x$，则执行把x缩小到k的函数
否则先把x缩小到$-\infty$，然后提取最小的元素，最后插入新元素k

复杂度 $k \leq x$的时候，复杂度和把x缩小到k一样，均摊下来是O(1)。
$k > x$的时候，把x缩小到$-\infty$均摊复杂度O(1),提取最小的元素均摊复杂度O(lgn),插入新元素均摊复杂度O(1),因此总共复杂度O(lgn)

3.2 2

算法 多一个双向循环链表H.leaf存储H的所有叶子节点，同时每个节点要有两个指针，分别指向其在树链表，叶子链表里的位置。
执行流程：每次删除H.leaf的第一个节点，如果这个节点的父亲是空，则在H.rootlist中也删除这个节点，同时更新A等信息。否则，这个节点的父亲度数-1，并且这个节点父亲的孩子链表也移除这个节点，如果这个节点父亲的度数变为0，则将其加入H.leaf中。
删除完q个节点后，再更新H.n,H.min等信息。

复杂度 先不考虑之后更新H.n,H.min等信息的时间。
定义势函数$\phi = k(tree(H)+degree(H))$，其中k为任意正常数，tree(H),degree(H)分别代表堆中树的个数和每个节点度数之和。在堆为空的时候，两者都是0，势函数也是0。无论如何这两者一定非负，势函数一定非负。因此堆的定义正确。
先计算均摊时间$c = O(q)$。因为每个删除操作都是O(1)的时间，总共删除是O(q)时间。
之后计算$\Delta\phi$。每次删除的节点如果父亲是空，那么tree(H)-=1。但是只删除了一个叶子节点，degree(H)不变。如果被删除的节点父亲不为空，那么tree(H)不变，而被删除的节点的父亲度数-1，degree(H)-=1。因此，每删除一个节点，ϕ都会少k，因此$\delta\phi = -qk$。
而最后$\hat{c} = c + \delta\phi = O(q) - kq = O(1)$，得出如果不含之后更新信息的时间，则均摊时间是O(1)。
而更新H.min的时候需要遍历剩余的所有树，时间是O(lgn)。而对于任意的$q \leq n - D(n)$，都能构造出一种情况，使得没有任何树被完全删去。而$q > n - D(n)$的时候，可以保证有一种最坏情况还需要遍历$n - q$个树，因此用聚集法进行均摊分析，总时间是$(n - D(n))D(n) + \sum_{i=1}^{D(n)-1}$，因为$D(n) = O(lgn)$，因此总时间是$O(nD(n))$，均摊时间为O(D(n))。
综上，总共均摊时间为O(D(n))，其中删除均摊时间是O(1),更新其余信息时间O(D(n))。