算法作业 12

孟妍廷 2015202009

2017 年 12 月 16 日

17.1-3

解：假设第 i 个操作的代价是 c_i，则 c_i 可表示为：

$$c_i = \begin{cases} i, lgi\text{是整数} \\ 1, Otherwise \end{cases}$$

因此，利用聚合分析计算这个操作序列的总代价如下;

$$cost = \sum_{i=1}^{n} c_i$$

$$= \sum_{lgi\text{非整数}} 1 + \sum_{j=0}^{\lfloor lgn \rfloor} 2^j \qquad (*)$$

由于聚合分析分析的是最坏情况，且 $2^i \geq 1$，$i \in N$，故 n 为 2 的幂时达到最坏情况，此时：

$$(*) = \sum_{lgi\text{非整数}} 1 + \sum_{j=0}^{lgn} 2^j$$

$$\leq n + \sum_{j=0}^{lgn} 2^j$$

$$= n + 1 + 2 + \cdots + 2^{lgn}$$

$$= n + 2 \cdot 2^{lgn} - 1$$

$$= n + 2n - 1 = O(3n)$$

故每个操作的摊还代价为 $\frac{O(3n)}{n} = O(1)$

17.2-2

解：由于直接分析抽象的操作的摊还代价不太容易，而题目中的操作序列的代价类似于动态表中扩表的操作的代价，因此我们利用修改后的扩表操作来进行分析。

修改动态表的操作由“表填满之后的下一个元素插入时进行扩表”修改为“填入表中的最后一个元素时进行扩表”，这样题目中描述的操作序列符合扩表操作的代价。

利用核算分析，对于第 i 次插入，缴纳 $\hat{c}_i = \$3.$，其中：

\$1 用于当前插入，\$2 用来预存，未来表大小翻倍时使用。当填入最后一个表项时表的大小翻倍，\$1 用于移动一个新元素，\$1 用于移动一个旧元素。因此 n 个操作的总摊还代价为 O(3n)，为总实际代价的上界。

17.3-2

解：假设 $2^{\lfloor lg0 \rfloor + 1} = 0$，将第 i 次插入后的表的势能定义为 $\Phi(D_i) = 2i - 2^{\lfloor lgi \rfloor + 1}$，满足 $\Phi(D_0) = 0$且$\Phi(D_i) \geq 0, \forall i$，故第 i 次插入的摊还代价为:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$$= \begin{cases} i, lgi\text{是整数} \\ 1, Otherwise \end{cases} + 2 - 2^{\lfloor lgi \rfloor + 1} + 2^{\lfloor lg(i-1) \rfloor + 1}$$

(1)lgi 为整数：

$$\hat{c}_i = i + 2 - 2i + i = 2$$

(2)i 不是 2 的次幂:

$$\hat{c}_i = 1 + 2 + 0 = 3$$

故单个操作的摊还代价是 O(1) 的，则 n 个操作序列的摊还代价为 O(n).

17.2
分析：在动态二分查找中，有一个非常有用的性质：k 个数组中每个数组都是有序的。
a.SEARCH 操作如下:
$SEARCH(x):$
  $for\ i\ =\ 0\ to\ k-1$
    $if\ n_i\ ==\ 1$
      对有序数组A_i做二分查找
      直到找到目标元素为止
对于 SEARCH 操作，其最坏情况是被查找的元素 x 并不在这个 n 元集合中，此时运行时间为：

$$\begin{aligned} time &= \sum_{i=0}^{k-1} n_i lg 2^i \\ &\le \sum_{i=0}^{k-1} lg 2^i \\ &= \sum_{i=0}^{k-1} i \\ &= \frac{k(k-1)}{2} = \frac{\lceil lg(n+1)\rceil(\lceil lg(n+1)\rceil - 1)}{2} \\ &= (lg^2 n) \end{aligned}$$

b.INSERT 操作的核心是保持 k 个有序数组不被破坏，算法如下：
$INSERT(x)$
  $flag = true, i = 0$
  $if\ n_0\ ==\ 0$ //只有大小为1的数组为空，直接插入即可
    $A_0[0] = x$
    $n_0 = 1$
    $return$
  //建立一个长度为1的数组$temp$，放入x
  $while\ flag == true$ //如果不能直接插入就要把数组放大
    $i++$
    $if\ n_i\ ==\ 0$
      $merge(temp, A_{n_{i-1}})$ //用$temp$代替A_{n_i}
      $n_i = 1$
      $flag = false$

由于对于 m 个待合并元素，merge 操作的时间复杂度是 O(m) 的，因此在最坏情况下，INSERT 操作的运行时间为

$$\begin{aligned} time &= O(1) + O(2) + \cdots + O(2^i) \\ &= O(1) + \cdots + O(n) = O(n) \end{aligned}$$

接下来利用聚合分析求 INSERT 操作的摊还代价：由于 n 的二进制表示为 $< n_{k-1}.......n_0 >$，所以类似于二进制计数器，n_0 每 1 次插入操作翻转，n_1 每 2 次插入操作翻转 $\cdots n_i$ 每 i+1 次插入翻转一次。而 n_i 翻转意味着执行了 merge 操作，代价为 2^i，故 n 次操作的摊还代价为:

$$cost = \sum_{i=0}^{k-1} \lceil \frac{n}{2^i} \rceil 2^i$$

$$= kn = \lceil lg(n+1) \rceil n = O(nlgn)$$

故一次操作的摊还代价为 $\frac{O(nlgn)}{n}$=O(lgn)

c. 对于 DELETE 操作，首先应该利用 SELECT 操作找到要删除的元素所在的位置，然后把这个元素删除，假设该元素在数组 A_i 中，删除该元素后还剩 $2^i - 1$ 个元素，则若 $n_0 = 1$，此时把 A_0 中的元素插入 A_i 中，并把 n_0 置为 0；否则把 A_i 中的剩余元素拆分放去 A_i 前空着的数组中。