数据结构与算法 II 作业 (11.24)

中国人民大学　信息学院　崔冠宇　2018202147

T17-2　(动态二分查找) 有序数组上的二分查找花费对数时间，但插入一个新元素的时间与数组规模呈线性关系。我们可以通过维护多个有序数组来提高插入性能。

具体地，假定我们希望支持 n 元集合上的 SEARCH 和 INSERT 操作。令 $k = \lceil \lg(n+1) \rceil$，令 n 的二进制表示为 $< n_{k-1}, n_{k-2}, \cdots, n_0 >$。我们维护 k 个有序数组 A_0，A_1，$\cdots$，A_{k-1}，对 $i = 0, 1, \cdots, k-1$，数组 A_i 的长度为 2^i。每个数组或满或空，取决于 $n_i = 1$ 还是 $n_i = 0$。因此，所有 k 个数组中保存的元素总数为 $\sum\limits_{i=0}^{k-1} n_i 2^i = n$。虽然单独每个数组都是有序的，但不同数组中的元素之间不存在特定的大小关系。

a. 设计算法，实现这种数据结构上的 SEARCH 操作，分析其最坏情况运行时间。

b. 设计 INSERT 算法。分析最坏情况运行时间和摊还时间。

c. 讨论如何实现 DELETE。

解:

a. 由于每个数组是有序的，但各数组之间无特定大小关系，所以可以分别对各数组进行二分查找。给出 SEARCH 算法的伪代码:

```
1  // n 元集合的查找操作
2  // 参数:
3  //       A[]: 题目中各有序数组形成的数组
4  //       n: 元素个数
5  //       x: 待查元素
6  SEARCH(A[], n, x):
7      k = ceil(log2(n + 1))
8      // BIN-SEARCH(arr, x) 是在数组 arr 中二分查找 x 的函数
9      // 当搜索成功，返回元素地址；不成功返回 NULL
10     for i = 0 to k - 1:
11         addr = BIN-SEARCH(A[i], x)
12         if addr != NULL:
13             return addr
14     return NULL
```

最坏情况运行时间分析:

第 11 行二分查找一个有序数组的最坏情况运行时间是 $O(\log|A[i]|)$，最多查找次数为 $(\lfloor \log|A[i]| \rfloor + 1)$ 次（其中 $|A[i]|$ 指数组 $A[i]$ 中元素个数）。SEARCH 算法的最坏情况是 $n = 2^j - 1$ 时各 $A[i]$ 均满，而且算法直到最后一次二分

查找才成功/失败，此时总查找次数为

$$
\begin{aligned}
\sum_{i=0}^{k-1}(\lfloor \log |A[i]| \rfloor + 1) &= \sum_{i=0}^{k-1}(\lfloor \log 2^i \rfloor + 1) \\
&= \sum_{i=0}^{\lceil \log(n+1) \rceil - 1}(i+1) \\
&= \Theta(\log^2 n)
\end{aligned}
$$

所以最坏情况运行时间是 $\Theta(\log^2 n)$。

b. 类似于二进制计数器递增问题中的置位操作。首先将待插入元素构成一个长度为 1 的数组，然后先检查 $A[0]$ 是否为空，若为空，放置在此；否则将两者合成一个数组，向更高位挪动。给出 INSERT 算法的伪代码：

```
1  // n 元集合的插入操作
2  // 参数:
3  //       A[]: 题目中各有序数组形成的数组
4  //       n: 元素个数
5  //       x: 插入元素
6  INSERT(A[], n, x):
7      // 增加一个元素，并计算 k
8      k = ceil(log2(n + 1 + 1))
9      // 把 x 做成一个数组
10     arr = MAKE-ARRAY(x)
11     for i = 0 to k - 1:
12         // 如果某 A[i] 为空，直接放
13         if A[i].empty:
14             A[i] = arr
15             break
16         // 否则合并两个数组，往高位走
17         else:
18             arr = MERGE-SORTED-ARRAY(arr, A[i])
```

最坏情况运行时间分析：

第 18 行合并有序数组的时间复杂度是 $\Theta(|arr| + |A[i]|)$，挪动元素次数为 $|arr| + |A[i]|$。最坏情况是 $n = 2^j - 1$ 时，增加一个元素需要合并 k 次有序数组，此时总操作次数为

$$
\begin{aligned}
2(\sum_{i=0}^{k-1} 2^i) &= 2 \times (2^k - 1) \\
&= 2^{k+1} - 2 \\
&= \Theta(n)
\end{aligned}
$$

摊还代价分析：

可以用聚合分析的方法，先计算 n 次插入步骤的总代价。假定第 i 次操作中，在 n 的二进制表示中，最右侧的 0 位于 n_r，此时插入的代价是 $2(\sum\limits_{i=0}^{r-1} 2^i) = O(2^r)$。类似于二进制计数器：有 $1/2$ 的情况 $r = 0$，有 $1/4$ 的情况 $r = 1$ $\cdots$，所以 n 次操作的总代价是 $O(\sum\limits_{r=0}^{k-1} \lceil \frac{n}{2^{r+1}} \rceil 2^r) = \sum\limits_{r=0}^{\lceil \log(n+1) \rceil - 1} O(n) = O(n \log n)$。因此，摊还代价是 $O(\log n)$。

c. 删除的策略如下：

- 找到最小的 j 使得 A_j 是满数组，设 y 是 A_j 的最后一个元素；
- 使用 $SEARCH$ 定位到 x 所在数组 A_i；
- 删除 x 并将 y 放到 A_i 的正确位置；
- 将 A_j 拆分：第一个元素放到 A_0，第二三个元素放到 A_1，$\cdots$

简要分析最坏情况复杂度：

最坏情况是 $i = k - 1$ 且 $j = k - 2$。定位 A_j 所需时间是 $\Theta(k) = \Theta(\log n)$；定位 A_i 与 x 的时间根据上面的分析是 $\Theta(\log^2 n)$；将 y 放置到 A_i 中的时间是 $\Theta(2^i) = \Theta(n)$；拆分 A_j 的时间是 $\Theta(2^j) = \Theta(n)$。因此最坏情况总复杂度是 $\Theta(n)$。