数据结构与算法 I 作业 23

2019201409 于倬浩

2020 年 12 月 26 日

20.2-4

只需修改 $u=2$ 时的情况即可，将原先的布尔数组改为整数数组，每次插入时，将对应位置的值加一，每次删除时，将对应位置的值减一。对于其他情况，数组含义保持不变，只要有叶子的值大于 0 即为 1，否则为 0，维护时依旧使用或运算。

这样，对于每个元素，即可正确处理被多次插入/删除的情况。

20-1

- a.

  对于每个 $vEB(u)$，需要维护一个 `summary` 结构，本质上是一个 $vEB(\sqrt{u})$；需要维护 $\sqrt{n}$ 个 `cluster`，本质上是 $\sqrt{n}$ 个 $vEB(\sqrt{u})$，同时还需 $\Theta(\sqrt{n})$ 个指针，维护指向所有子树的根，因此，每个 $vEB(u)$ 需要的空间为：

$$
\begin{aligned}
P(u) &= P(\sqrt{u}) + \sqrt{u}P(\sqrt{u}) + \Theta(\sqrt{u}) \\
&= (\sqrt{u}+1)P(\sqrt{u}) + \Theta(\sqrt{u})
\end{aligned}
$$

- b.

$$
\begin{aligned}
P(u) &= (\sqrt{u}+1)P(\sqrt{u}) + \Theta(\sqrt{u}) \\
&= \prod_{i=1}^{\lfloor lg(lg(u)) \rfloor} (2^{2^i}+1)P(2) + \sum_{i=1}^{\lfloor lg(lg(u)) \rfloor} 2^{2^i} \prod_{j=i}^{\lfloor lg(lg(u)) \rfloor} (2^{2^j}+1)
\end{aligned}
$$

$$
\begin{aligned}
&\because \prod_{i=1}^{\lfloor lg(lg(u)) \rfloor} (2^{2^i}+1) \le 2 \times 2^{2^{lg(lg(u))}} \le 2 \times u \\
&\therefore \prod_{i=1}^{\lfloor lg(lg(u)) \rfloor} (2^{2^i}+1)P(2) \le 4 \times u \\
&\because \sum_{i=1}^{\lfloor lg(lg(u)) \rfloor} 2^{2^i} \prod_{j=i}^{\lfloor lg(lg(u)) \rfloor} (2^{2^j}+1) \le 2 \times u \\
&\therefore P(u) \le 4 \times u + 2 \times u \le 6u \\
&\therefore P(u) = O(u)
\end{aligned}
$$

- c、d.

  实际上这两问解决的是同一个问题：当我们访问到的节点并没有真的被分配空间，因此访问到了空指针。解决方案也很简单，只需要在每个访问子树指针的地方，判断该指针指向的位置是否被分配了空间即可。如果发现没有被分配空间，调用 `Create-New-RS-vEB-Tree(child)` 为该指针分配空间，接下来一切照旧即可。

- e.

  如果哈希表采用简单均匀的随机策略，将哈希表的创建、插入、删除、访问等各项操作的运行时间均视为 $O(1)$，那么实际上使用哈希表替换原来的数组只是将 vEB 树的各项操作的运行时间中的各项都乘上了 $O(1)$，因此并不会破坏原有运行时间递归式的上界，例如插入和查找前驱的操作的运行时间依旧是 $O(lglgn)$。

- f.

  根据勘误，原题 $O(n)$ 的界是错的，因为有可能构造出需要 $O(nlglgn)$ 空间的数据。

  实际上，分析后者就容易得多了。考虑单次插入带来的最大空间增量。

  显然，每次插入时，最坏情况是每层调用都访问到未分配的空间，而又因为每次的空间增量都是 $O(1)$ 的，vEB 树的深度为 $O(lgn)$，因此单次插入所带来的最大空间增量为 $O(lgn)$，那么 n 次操作后的空间开销即为 $O(nlgn)$。

- g.

  由于修改后的版本不再需要直接分配 $O(\sqrt{u})$ 的指针数组，新建空树的时候只需要初始化动态表维护的哈希表以及 min、max、summary 等指针即可，因此时间复杂度为 $O(1)$。