Design and Analysis of Algorithms: Lecture 4

Ben Chaplin

Contents

1 Background

1.1 Operations

We want to maintain n elements from the set $\{0, 1, \ldots, u-1\}$, and support the following operations:

- INSERT(V, x): insert x into V
- DELETE(V, x): delete x from V
- SUCCESSOR(V, x): return the smallest element in V which is larger than x

1.2 Problem

Balanced binary search trees support all three of the above operations in $O(\log n)$ time. The goal of **van Emde Boas trees** is to support operations in $O(\log \log n)$ time.

Let n and u be defined as they were above. If $u = n^{O(1)}$, then $\log \log u = O(\log \log n)$.

1.3 Recurrences

Recall the binary search recurrence:

$$T(k) = T\left(\frac{k}{2}\right) + O(1) \tag{1}$$
$$= O(\log k) \tag{2}$$

So we're seeking:

$$T(\log u) = T\left(\frac{\log u}{2}\right) + O(1) \tag{3}$$
$$= O(\log \log u) \tag{4}$$

Which can be written:

$$T'(u) = T'(\sqrt{u}) + O(1) \tag{5}$$
$$= O(\log \log u) \tag{6}$$

2 Van Emde Boas Trees

2.1 Data structure

At their core, **van Emde Boas Trees** are bit vectors of size u where:

$$V[i] = \begin{cases} 0 & \text{if } i \text{ is not present} \\ 1 & \text{if } i \text{ is present} \end{cases}$$

The vector is split into **clusters** of size $\sqrt{u}$.

Above each cluster is a tree structure. Consider the elements of the cluster as the leaves of the tree, and store $(l_0 \vee l_1)$ in the parent node for each two neighboring leaves l_0, l_1. The **summary** vector contains the root from each tree above each cluster.

Both clusters and summary vectors store their minimum and maximum values.

2.2 Definitions

To make writing algorithms quicker:

Definition. Given $x \in \{1, \ldots, u-1\} \subseteq \mathbb{N}$, $\textbf{high}(x) = \lfloor \frac{x}{\sqrt{u}} \rfloor$.

Definition. Given $x \in \{1, \ldots, u-1\} \subseteq \mathbb{N}$, $\textbf{low}(x) = x \mod \sqrt{u}$.

Definition. Given $i, j \in \{1, \ldots, u-1\} \subseteq \mathbb{N}$, $\textbf{index}(i, j) = i\sqrt{u} + j$.

2.3 Algorithms

```
1: procedure SUCCESSOR(V, x)
2:     i = high(x)
3:     if low(x) < V.cluster[i].max then                ▷ successor is in cluster i
4:         j = SUCCESSOR(V.cluster[i], low(x))
5:     else                ▷ successor is not in cluster i, check next in summary
6:         i = SUCCESSOR(V.summary, high(x))
7:         j = V.cluster[i].min
8:     end if
9:     return index(i, j)
10: end procedure
```

Runtime analysis: SUCCESSOR runs in $O(\log \log u)$ time. Because each