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 $x \in \{1, \dots, u-1\} \subseteq \mathbb{N}$, **high**$(x) = \lfloor \frac{x}{\sqrt{u}} \rfloor$ (this effectively gives us the index of the parent of x in the summary vector).

Definition. Given $x \in \{1, \dots, u-1\} \subseteq \mathbb{N}$, **low**$(x) = x \mod \sqrt{u}$ (this effectively gives us the index of x in its cluster).

Definition. Given $i, j \in \{1, \dots, u-1\} \subseteq \mathbb{N}$, **index**$(i, j) = i\sqrt{u} + j$ (index(high(x), low(x)) = x).

2.3 Algorithms

```
procedure SUCCESSOR(V, x)
    if |V| ≤ 4 then
        return index of successor                                   ▷ base case
    end if
    if x < V.min then
        return V.min
    end if
    i = high(x)
    if low(x) < V.cluster[i].max then                   ▷ successor is in cluster i
        j = SUCCESSOR(V.cluster[i], low(x))
    else                       ▷ successor is not in cluster i, check next in summary
        i = SUCCESSOR(V.summary, high(x))
        j = V.cluster[i].min
    end if
    return index(i, j)
end procedure
```

Runtime analysis: SUCCESSOR runs in $O(\log \log u)$ time where $|V| = u$.

- Each line 7 recursive call reduces the size of V by $\sqrt{u}$, so we will have at most $O(\log \log u)$ recursive calls.
- If we hit a line 9 recursive call, we will finish in $O(1)$ time, as we need only find the max of V.summary, then the max of the cluster it summarizes.

1: **procedure** INSERT(V, x)
2:     **if** V.min $=$ *null* **then**
3:         V.min $=$ V.max $= x$, **return**
4:     **end if**
5:     **if** $x <$ V.min **then**
6:         swap$(x, V$.min$)$
7:     **end if**
8:     **if** $x >$ V.max **then**
9:         V.max $= x$
10:     **end if**
11:     **if** V.cluster[high(x)].min $=$ null **then**
12:         INSERT$(V$.summary, high$(x))$
13:     **end if**
14:     INSERT$(V$.cluster[high(x)], low$(x))$
15: **end procedure**

Runtime analysis: INSERT runs in $O(\log \log u)$ time where $|V| = u$.

- Each line 14 recursive call reduces the size of V by $\sqrt{u}$, so we will have at most $O(\log \log u)$ recursive calls.
- If we hit a line 12 recursive call, we will finish in $O(\log \log u)$ time, as we only need to insert into the summaries, of which there are $O(\log \log u)$.