Van Emde Boas Tree

Lecture Notes

1 Introduction

Goal: Maintain n elements among $\{0, 1, \ldots, U-1\}$ subject to:

Operation Definitions:

- **Insert**(x): $S \leftarrow S \cup \{x\}$
- **Successor**(x): Return $\min\{y \in S : y \geq x\}$, or ∞ if $\{y \in S : y \geq x\} = \emptyset$
- **Delete**(x): $S \leftarrow S \setminus \{x\}$

Space: $O(U)$
Time: $O(\lg \lg U)$

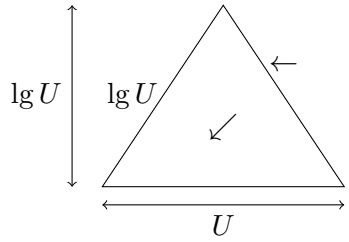
1.1 When is $O(\lg \lg U)$ useful?

If $U = n^{O(1)}$ or $U = n^{\lg^{O(1)} n}$, then $\lg \lg U = O(\lg \lg n)$.

Application: Networking — IP address ranges.

1.2 Where might $O(\lg \lg n)$ come from?

Binary search on the levels of the tree.



For binary search on k levels:

$$T(k) = T(k/2) + O(1) = O(\lg k)$$

For the Van Emde Boas recurrence:

$$T(U) = T(\sqrt{U}) + O(1) = O(\lg \lg U)$$

Example: $\lg \frac{U}{2} = \sqrt{U}$. If $U = 2^{16}$, then $\lg 2^{16} = \frac{16}{2} = 8$, and $(2^{16})^{1/2} = 2^{16/2} = 2^8$.

2 Version 1: Bit Vector

Insert/Delete: $O(1)$
Successor: $O(U)$
Bit vector = array of size U

- 0 = absent
- 1 = present

Example: $U = 16$, $n = 4$

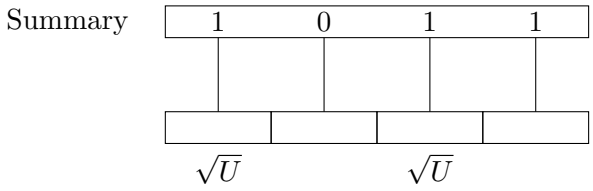
0	1	0	0	0	0	0	0	1	1	0	0	0	0	1

3 Version 2: Split Universe into Clusters

Split the universe into $\sqrt{U}$ clusters, each of size $\sqrt{U}$.

3.1 Structure

Summary vector: A bit vector of size $\sqrt{U}$ where each bit indicates whether the corresponding cluster is non-empty.



Operations:

- Insert: $O(1)$
- Delete: ??
- Successor(x):
  1. Look in x's cluster
  2. Look for next 1 bit in summary
  3. Look for first 1 in that cluster
- Time: $O(\sqrt{U})$

4 Version 3: Recursive Structure

If $x = i\sqrt{U} + j$ where $0 \leq j < \sqrt{U}$:
If $x = 9$, then $x = 2\sqrt{U} + 1$ (for $U = 16$, $\sqrt{U} = 4$).

Definitions:

$$\begin{aligned} \text{high}(x) &= \lfloor x/\sqrt{U} \rfloor \\ \text{low}(x) &= x \mod \sqrt{U} \\ \text{index}(i,j) &= i\sqrt{U} + j \end{aligned}$$

Example: $9 = \boxed{1001}$ where high bits give cluster index, low bits give position within cluster.

4.1 VEB Structure

V = size U VEB tree

- V.cluster$[i]$ = size $\sqrt{U}$ VEB tree, for $0 \le i < \sqrt{U}$
- V.summary = size $\sqrt{U}$ VEB tree

4.2 Insert(V, x)

$$T(U) = 2T(\sqrt{U}) + O(1) = O(\lg U)$$

```
1: procedure INSERT(V, x)
2:     Insert(V.cluster[high(x)], low(x))
3:     Insert(V.summary, high(x))
4: end procedure
```

4.3 Successor(V, x)

$$T(U) = O((\lg U)^3)$$

```
1: procedure SUCCESSOR(V, x)
2:     i ← high(x)
3:     j ← Succ(V.cluster[i], low(x))
4:     if j = ∞ then
5:         i ← Succ(V.summary, i)
6:         j ← Succ(V.cluster[i], −∞)
7:     end if
8:     return index(i, j)
9: end procedure
```

5 Version 4: Store Min and Max

5.1 Insert(V, x)

Time: $O(\lg U)$

```
1: procedure INSERT(V, x)
2:     if x < V.min then
3:         V.min ← x
4:     end if
5:     if x > V.max then
6:         V.max ← x
```

```
    end if
    Insert(V.cluster[high(x)], low(x))
    Insert(V.summary, high(x))
end procedure
```

5.2 Successor(V, x)

Time: $O(\lg \lg U)$

```
procedure SUCC(V, x)
    if x < V.min then
        return V.min
    end if
    i ← high(x)
    if low(x) < V.cluster[i].max then
        j ← Succ(V.cluster[i], low(x))
    else
        i ← Succ(V.summary, i)
        j ← V.cluster[i].min
    end if
    return index(i, j)
end procedure
```

6 Version 5: Don't Store Min Recursively

6.1 Insert(V, x)

Time: $O(\lg \lg U)$

```
procedure INSERT(V, x)
    if V.min = ∅ then
        V.min ← V.max ← x
        return
    end if
    if x < V.min then
        swap(x, V.min)
    end if
    if x > V.max then
        V.max ← x
    end if
    if V.cluster[high(x)].min = ∅ then
        Insert(V.summary, high(x))
    end if
    Insert(V.cluster[high(x)], low(x))
end procedure
```

6.2 Delete(V, x)

```
procedure DELETE(V, x)
    if x = V.min then
```

```
        i ← V.summary.min
        if i = ∅ then
            V.min ← V.max ← ∅
            return
        end if
        x ← V.min ← index(i, V.cluster[i].min)
    end if
    Delete(V.cluster[high(x)], low(x))
    if V.cluster[high(x)].min = ∅ then
        Delete(V.summary, high(x))
    end if
    if x = V.max then
        if V.summary.max = ∅ then
            V.max ← V.min
        else
            i ← V.summary.max
            V.max ← index(i, V.cluster[i].max)
        end if
    end if
end procedure
```

7 Lower Bound

$$\Omega(\lg \lg U)$$

for $U = n^{\lg^{O(1)} n}$ and space $= O(n \cdot \text{poly} \lg n)$.

8 Space Optimization

How to get $O(n)$ space:

- Only store non-empty clusters
- Make V.cluster a hash table
- Space: $O(n \cdot \lg \lg U)$