X-fast Tree and Y-fast Tree

Lecture Notes

1 Models for Integer Data Structures

- **Word** = w-bit integer, where $U = \{0, 1, \ldots, 2^w - 1\}$
- Original vEB = Stratified trees
  - Each node stores a pointer to 2^i ancestor, for $i = 0, 1, \ldots, \lg U$
  - $O(U \lg w)$ space
  - Every node stores min/max

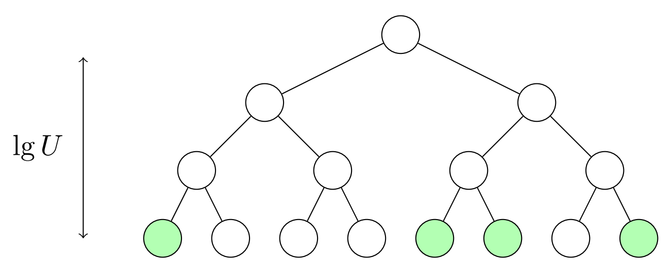
2 Trie

A **trie** is, in effect, a tree in which every node is labeled according to the path which goes from root to that node.

3 X-fast Tree

3.1 Simple Tree View

The X-fast tree is a binary trie of height $\lg U$.



Leaves correspond to bit vector: $0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1$
Key Idea: Any root-to-leaf path is monotone. We can do binary search for the 0-1 transition.

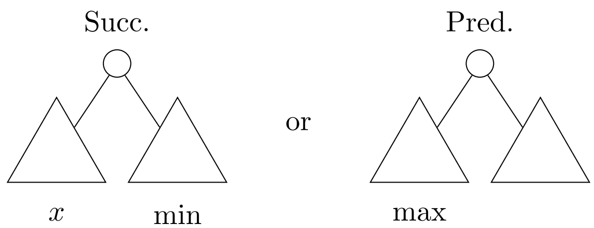
3.2 Complexity

$$\boxed{O(\lg \lg U) \text{ — Pred./Succ.}}$$

3.3 Auxiliary Pointers

Let N be some internal node such that N has exactly one child. Then N contains an auxiliary pointer to a leaf of the trie.

- If $N \to$ right child $= \emptyset$, then $N \to$ right child $=$ Auxiliary Pointer to rightmost descendant
- If $N \to$ left child $= \emptyset$, then $N \to$ left child $=$ Auxiliary Pointer to leftmost descendant



3.4 Hash Tables

Each level of nodes in the trie is also entered into a hash table of the same level. We can query the existence of any node in constant time.

3.5 Search (Membership)

To check for the presence of an element, we can just check the lowest (and largest) hash table.

$$\boxed{\text{Find}(x) = O(1)}$$

3.6 Successor Algorithm

```
procedure Succ(x, Tree)
    N ← lowest node on search path to x in tree
    if N is a left child then
        SuccNode ← N.AuxPointer
    else
        PredNode ← N.AuxPointer
        SuccNode ← PredNode.Next
    end if
    return SuccNode.value
end procedure
```

To find the lowest node, we do a binary search on the full search path of y. We can determine the existence of a node in constant time using hash tables.

3.7 X-fast Tree Complexity Summary

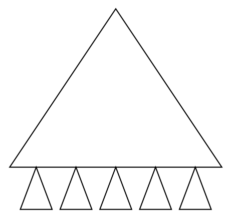
- **Insert:** $O(\lg U)$
- **Space:** $O(n \lg U)$

4 Y-fast Tree

4.1 Structure

The Y-fast tree combines:

- **Big-Universe:** X-fast tree of size $O\left(\frac{n}{\lg U}\right)$
- **Little-Universe:** Binary search trees of size $O(\lg U)$ each



4.2 Y-fast Tree Complexity

- **Succ/Pred/Search:** $O(\lg \lg U)$ worst-case
- **Insert/Delete:** $O(\lg \lg n)$ amortized
- **Space:** $O(n)$

4.3 Design Details

- Divide n items into $O\left(\frac{n}{\lg U}\right)$ pieces, each of size $O(\lg U)$
- In practice, each piece will be between two sizes: $\frac{\lg U}{4}$ and $4 \lg U$
- Little-Universe: Binary search tree with $O(\lg \lg U)$ time and $O(\lg U)$ space
- Big-Universe: X-fast tree