CS 4810: Advanced Algorithms Spring 2026

Lecture 3 — January 14, 2026

Prof. Prashant Pandey *Scribe: Michael Pimble*

1 Summary and Review

In the last lecture, we discussed the **Van Emde Boas** (vEB) tree, which allowed for all operations in $O(\log \log U)$ time while taking up $O(U)$ space. For further context, see [1]

In this lecture, we discuss **X-fast** and **Y-fast** tries. Our goal is to maintain n elements in universe U, completing all operations in $O(\log \log U)$ time while taking up $O(n)$ space. This use of X-fast and Y-fast tries originates from [2].

1.1 Recap From Lecture 2

The **vEB tree** is a recursive data structure which stores n elements from U, as well as the following attributes:

- $\sqrt{U}$ clusters, which are all vEB trees
- A summary vEB tree of length $\sqrt{U}$, which stores the minimum and maximum values of each cluster.
  - Note: Only the maximum value is stored recursively in a vEB tree.

The vEB tree supports two types of operations, which X-fast and Y-fast tries will also support:

1. **Insert** and **Delete** operations
2. **Find**, **Predecessor** (find the largest value that is at most n), and **Successor** (find the small value that is at least n) operations

For time complexity purposes, these operations can be grouped together for vEB, X-fast, and Y-fast trees.

2 X-fast Tries

2.1 Preliminary Terms

Word – A w-bit integer within $U = \{0, 1, 2, \ldots, 2^{w-1}\}$
Trie – A tree in which every node is labeled according to the path which goes from the root to that node.

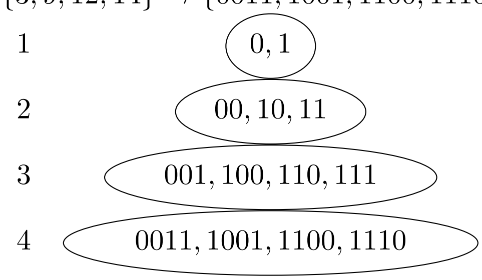
- For U items, a binary trie has $\log U$ layers with a width of U on the bottom layer.

2.2 The Idea of an X-fast Trie

- For each layer of the trie, we make a hashtable
  - This allows for constant lookup on each layer while using $O(n \log U)$ space total.
- Since any root-to-leaf path is monotone, we can do a binary search for 0-1 translation, achieving $O(\log \log U)$ time for the successor operation.

2.3 X-fast Trie Example

$U = 16$, $n = 4$
$\{3, 9, 12, 14\} \rightarrow \{0011, 1001, 1100, 1110\}$



Each oval represents the hashmap in each layer

In summary, we make a hashtable for each level, binary search each hashtable to perform successor in $O(\log \log U)$ time.
Insert takes $O(\log U)$ time, and the tree takes up $O(n \log U)$ space.

2.4 Auxiliary Pointer

Let N be some internal node such that N contains an auxiliary pointer to a leaf of the tree.

if $N \rightarrow$ rightchild $= \varnothing$:
    $N \rightarrow$ rightchild = Aux Points
then $N \rightarrow$ rightchild = Rightmost descendant

2.5 Search Path

Let T be the trie of an X-fast tree such that T has $l + 1$ levels
For any $y \in U$, there is in T some ordered list of nodes which would be followed if we were doing a tree search for element y
Each level of nodes in the trie is also entered in the hashtableof the same level. We can query the existence of any node in $O(1)$ time

Succ(x, Trie):
    N = Lowest node on the search path to x
    if N.leftchild:
        SuccNode = N.AuxPointer
    else:

PredNode = N.AuxPointer
Return SuccNode.value

To find the lowest node on the search path, we will do a binary search on the full search path of element y
At each level, we query in $O(1)$ time using a hashmap

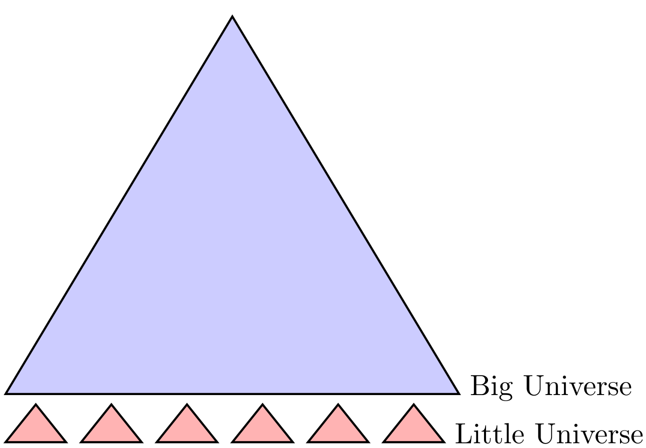
3 Y-fast Trie

3.1 Time and Space Comparison

	Insert	Pred/Succ	Space
vEB	$\log\log U$	$\log\log U$	U
X-fast	$\log U$	$\log\log U$	$n\log U$
Y-fast	$\log\log U$	$\log\log U$	n

3.2 Idea of a Y-fast Trie: Big/Little Universe

- Divide n items into $O(\frac{n}{\log U})$ pieces, each of size $O(\log U)$
- In practice, each piece will be between size $\log U$ and $4\log U$
- Little Universe – Balanced BST
  - Time: $O(\log\log U)$
  - Space: $O(\log U)$
- Big Universe – X-fast trie
  - Time: $O(\log\log U)$
  - Space: $O(\frac{n}{\log U}\log U) = O(n)$



References

[1] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 75–84. IEEE Computer Society, 1975.

[2] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space theta(n). *Inf. Process. Lett.*, 17(2):81–84, 1983.