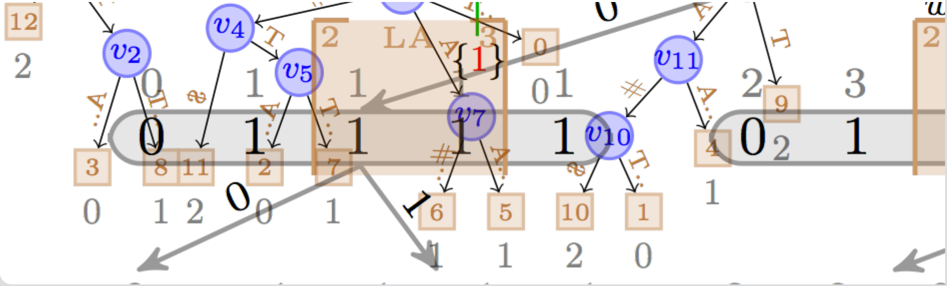
Karlsruhe Institute of Technology

Advanced Data Structures

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



Predecessor data structures

We want to support the following operations on a set of integers from the domain $U = [u]$.

insert(x)	Add x to S. I.e. $S' = S \cup \{x\}$.
delete(x)	Delete x from S. I.e. $S' = S \setminus \{x\}$.
member(x)	$= \lvert\{x \in S\}\rvert$
predecessor(x)	$= \max\{y \mid y \leq x \wedge y \in S\}$
successor(x)	$= \min\{y \mid y \geq x \wedge y \in S\}$

where x is an integer in U and S the set of integers of size n stored in the data structure.

- $\min\{S\} = \text{successor}(0)$
- $\max\{S\} = \text{predecessor}(u-1)$

Solution know from „Algo I": Balanced search trees. E.g. red-black trees. In all comparison based approaches at least one operation takes $\Omega(\log n)$ time. Why?

Predecessor data structures

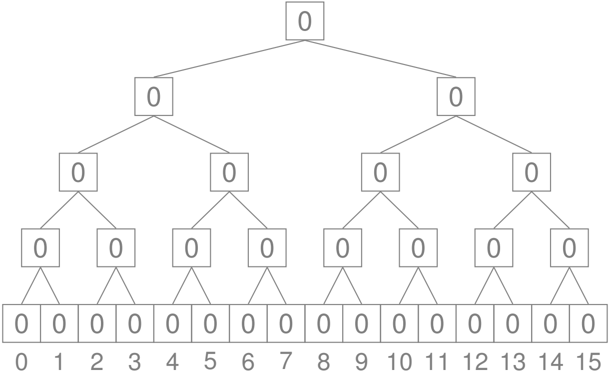
$\Omega(\log n)$ bound can be beaten in the *word RAM* model:

- Memory is organized in words of $b \in O(\log u)$ bits
- A word can be accessed in constant time
- We can address all data using one word
- Standard arithmetic operations take constant time on words (i.e. addition, subtraction, division, shifts ...)

We first concentrate on the static case: The set S is fixed. I.e. no insert and delete operations.

x-fast trie (Willard, 1982)

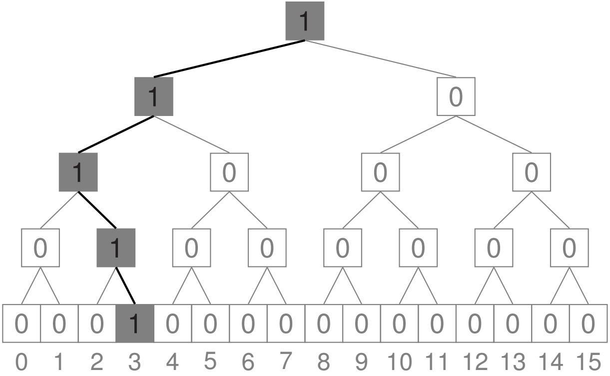
- Conceptional: Complete binary tree of hight $w = \lceil \log u \rceil$



- Operations member/successor/prodecessor can be answered in $O(w)$ time by traversing the tree

x-fast trie (Willard, 1982)

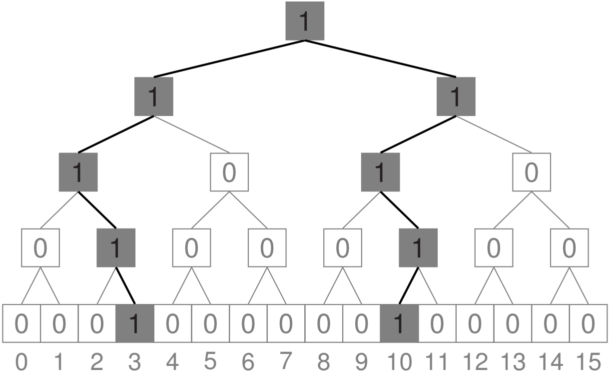
- Conceptional: Complete binary tree of hight $w = \lceil \log u \rceil$



- Operations member/successor/prodecessor can be answered in $O(w)$ time by traversing the tree

x-fast trie (Willard, 1982)

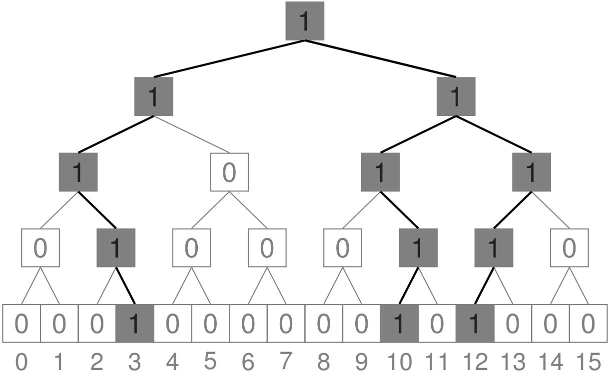
- Conceptional: Complete binary tree of hight $w = \lceil \log u \rceil$



- Operations member/successor/prodecessor can be answered in $O(w)$ time by traversing the tree

x-fast trie (Willard, 1982)

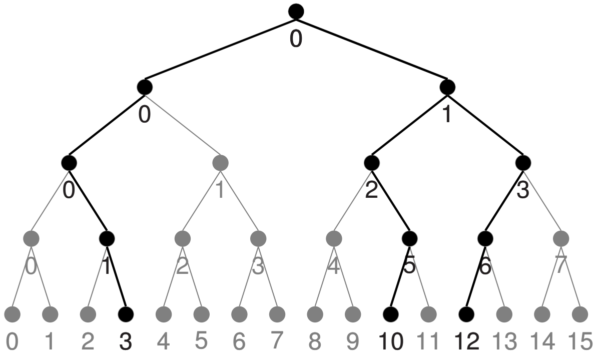
- Conceptional: Complete binary tree of hight $w = \lceil \log u \rceil$



- Operations member/successor/prodecessor can be answered in $O(w)$ time by traversing the tree

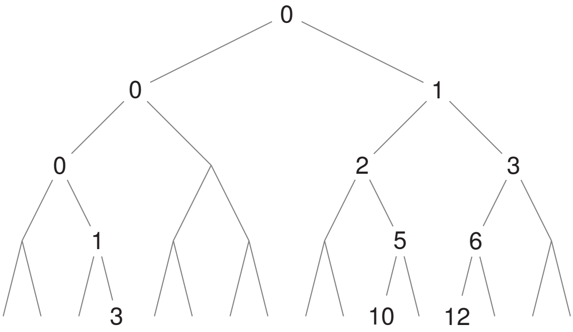
x-fast trie

From $O(\log u)$ to $O(\log \log u)$...



x-fast trie

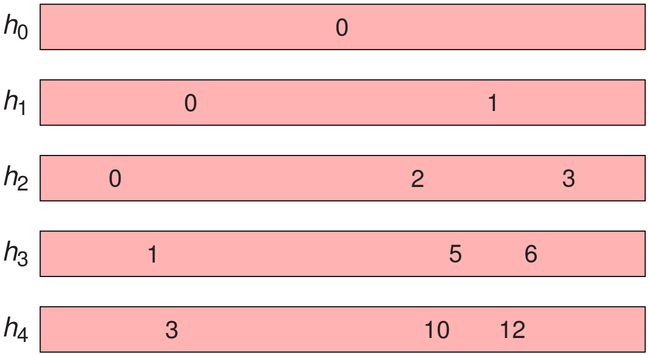
From $O(\log u)$ to $O(\log\log u)$...



- Generate a perfect hash table h_i for each level i (keys are the present nodes)
- Note: Node numbers (represented in binary) are prefixes of keys in S

x-fast trie

From $O(\log u)$ to $O(\log\log u)$...



- Generate a perfect hash table h_i for each level i (keys are the present nodes)
- Note: Node numbers (represented in binary) are prefixes of keys in S

x-fast trie

Query time from $O(\log u)$ to $O(\log \log u)$...

- Member queries can be answered by a binary search using prefixes of the searched key.
- There are w prefixes, i.e. binary search takes $O(\log w)$ or $O(\log \log u)$ time
- Space: $w \cdot O(n)$ words, i.e. $O(n \log u)$ bits

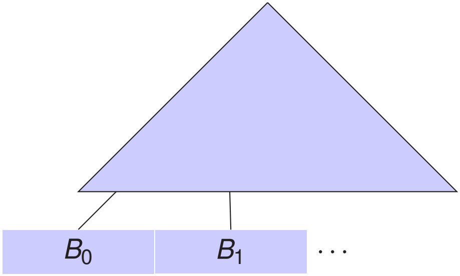
Predecessor/Successor queries

- For each node store a pointer to the maximal/minimal leaf in its subtree
- Use a double linked list to represent leaf nodes.
- Solving predecessor: Search for a node v which represents the longest prefix of x with any key in S. Two cases:
  - Minimum in subtree of v is larger x: Return element to the left of the leaf.
  - Maximum in the subtree of v is smaller than x. Return maximum.

y-fast trie (Willard, 1982)

Space from $O(n \log u)$ to $O(n)$ words...

- Split S into $\frac{n}{w}$ blocks of $O(\log u)$ elements $B_0, B_1, \ldots, B_{\lceil \frac{n}{w} \rceil - 1}$.
- $\max\{B_i\} < \min\{B_{i+1}\}$ for $0 \leq i < \lceil \frac{n}{w} \rceil - 1$
- Let $r_i = \max\{B_i\}$ be a *representative* of block B_i.
- Build x-fast trie over representatives.



- Total space: $\frac{n}{w} \cdot O(w) + O(n) = O(n)$ words

y-fast trie

Space from $O(n \log u)$ to $O(n)$ words...

- Use sorted array to represent B_i
- A member query is answered as follows
  - Search for successor of x in x-trie of r_i's
  - Let B_k be the block of the successor of x
  - Search in $O(\log w) = O(\log \log u)$ time for x in B_k
- How does predecessor/successor work?

y-fast trie

Changes to make structure dynamic

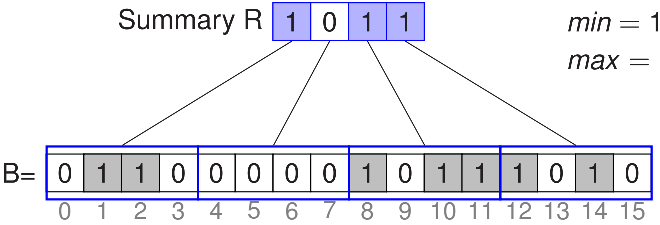
- use cuckoo hashing for x-fast trie
- use balanced search trees of size between $\frac{1}{2}w$ and $2w$ for B_is
- representative is not the maximum, but any element separating two consecutive groups

Summary:

Operation	static y-fast trie	dynamic y-fast trie
pred(x)/succ(x)	$O(\log\log u)$ w.c.	$O(\log\log u)$
insert(x)/delete(x)		$O(\log\log u)$ exp. & am.
construction	$O(n)$ exp.	

Van Emde Boas Trees

- Conceptual bitvector B of length u with $B[i] = 1$ for all $i \in S$
- Split B into $u/\sqrt{u}$ blocks (blue blocks) $B_0, B_1, \ldots$
- Set bit in $R[i]$ if there is at least one bit set in B_i
- Also store the minimum/maximum of S



$min = 1$

$max = 14$

- Here: $u = 16$

Van Emde Boas Trees

- Define van Emde Boas tree (vEB tree) recursively
- I.e. use vEB to represent B_i's and R
- vEB(u) denotes the vEB tree on a universe of size u
- Base case: $u = 2$. Only one node and variables min, max.

Technicalities

- $\sqrt[\uparrow]{u} = 2^{\lceil (\log u)/2 \rceil}$
- $\sqrt[\downarrow]{u} = 2^{\lfloor (\log u)/2 \rfloor}$
- $high(x) = \left\lfloor \frac{x}{\sqrt[\downarrow]{u}} \right\rfloor$ (block that contains x)
- $low(x) = x \bmod \sqrt[\downarrow]{u}$ (relative position of x in $B_{high(x)}$)

Van Emde Boas Trees

Predecessor(x, B) (first attempt)

- Let $y = hight(x)$ and $z = Predecessor(low(x), B_y)$
- If $z \neq \bot$ return $z + y \cdot \sqrt[\downarrow]{u}$
- Let $b = Predecessor(high(x), R)$
- If $b \neq \bot$ return $max(B_b) + b \cdot \sqrt[\downarrow]{u}$
- Return $\bot$

Problem

- Recurrence for time complexity: $T(u) = 2T(\sqrt{u}) + O(1)$
- Solution (Master Theorem or drawing recursion tree): $T(u) = \Theta(\log u)$
- Now: avoid one of the recursive calls

Van Emde Boas Trees

Predecessor(x, B) (second attempt)

- If $x > max$ return max
- Let $y = high(x)$, if $\min(B_y) < x$ return $Predecessor(low(x), B_y)$
- Let $b = Predecessor(high(x), R)$
- If $b \neq \perp$ return $max(B_b) + b \cdot \sqrt[\downarrow]{u}$
- Return $\perp$

- Recurrence for time complexity: $T(u) = T(\sqrt{u}) + O(1)$
- Solution (Master Theorem or drawing recursion tree):
  $T(u) = \Theta(\log\log u)$