Advanced Data Structures
Simon Gog – gog@kit.edu
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($x$)** $= |\{x \in S\}|$
- **predecessor($x$)** $= \max\{y | y \leq x \land y \in S\}$
- **successor($x$)** $= \min\{y | y \geq x \land y \in S\}$

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 height \( w = \lceil \log u \rceil \)

- 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 height \( w = \lceil \log u \rceil \)

- 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 height \( w = \lceil \log u \rceil \)

- Operations member/successor/predecessor can be answered in \( O(w) \) time by traversing the tree
x-fast trie (Willard, 1982)

- Conceptual: Complete binary tree of height \( w = \lceil \log u \rceil \)

- Operations member/successor/predecessor 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 $O(\log \log u)$...*

<table>
<thead>
<tr>
<th>$h_0$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>0 1</td>
</tr>
<tr>
<td>$h_2$</td>
<td>0 2 3</td>
</tr>
<tr>
<td>$h_3$</td>
<td>1 5 6</td>
</tr>
<tr>
<td>$h_4$</td>
<td>3 10 12</td>
</tr>
</tbody>
</table>

- 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 than $x$: 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 $B_i$s
- representative is not the maximum, but any element separating two consecutive groups

Summary:

<table>
<thead>
<tr>
<th>Operation</th>
<th>static y-fast trie</th>
<th>dynamic y-fast trie</th>
</tr>
</thead>
<tbody>
<tr>
<td>pred($x$)/succ($x$)</td>
<td>$O(\log \log u)$ w.c.</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td>insert($x$)/delete($x$)</td>
<td>$O(n)$ exp.</td>
<td>$O(\log \log u)$ exp. &amp; am.</td>
</tr>
<tr>
<td>construction</td>
<td>$O(n)$ exp.</td>
<td></td>
</tr>
</tbody>
</table>
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) $B_0, B_1, ...$
- Set bit in $R[i]$ if there is at least one bit set in $B_i$
- Also store the minimum/maximum of $S$

Here: $u = 16$

Summary $R$

$$\begin{array}{cccc}
1 & 0 & 1 & 1 \\
\end{array}$$

$$\begin{array}{cccccccccccc}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
\end{array}$$

$max = 14$

$min = 1$
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 $x$)

$low(x) = x \mod \sqrt[\downarrow]{u}$ (relative position of $x$ in $B_{high(x)}$)
Van Emde Boas Trees

Predecessor($x$, $B$) (first attempt)

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

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