Finger Search Trees
with Constant Insertion Time

Gerth Stølting Brodal

Max-Planck-Institut für Informatik
Saarbrücken

Supported by the Carlsberg Foundation
The Problem

Maintain a sorted list of elements under

- **Insert \((f, x)\)**, insert \(x\) to the right of \(f\)
  
  Ex. \([5, 7, 9, 11] \rightarrow [5, 7, 8, 9, 11]\)

- **Search \((f, x)\)**, search for \(x\) in the list starting at \(f\).
  
  Ex. \([4, 5, 6, 8, 9, 11, 13, 15, 17, 19]\)

"finger"
A Simple Finger Search Tree

Brown, Taijan 1980

(2, 3)-tree

Insert: Amortized $O(1)$
A Simple Finger Search Tree
Brown, Taijan 1980

Level linked (2,3)-tree

Insert: Amortized $O(1)$.
Search: Worst-case $O(\log \delta)$. 
Search Trees With Constant Insertion Time

Levcopoulos, Overmars 1988

Idea
For every $\log n$-th insertion split the largest bucket and update the $(2,3)$-tree over the next $\log n$ insertions.

(2,3)-tree

Buckets of size $O(\log^2 n)$

Insert: $O(1)$
Search: $O(\log n)$
Search Trees With Constant Insertion Time

Levcopoulos, Overmars 1988

Idea
For every $\log n$-th insertion split the largest bucket and update the (2,3)-tree over the next $\log n$ insertions.

<table>
<thead>
<tr>
<th>Recursive</th>
<th>$\log^* n$ Recursive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>$O(\log^* n)$</td>
</tr>
<tr>
<td>$O(\log \log n + \log 3)$</td>
<td>$O(\log 3)$</td>
</tr>
</tbody>
</table>

Insert: $O(1)$
Search: $O(\log n)$

Buckets of size $O(\log^2 n)$
### Previous Work

<table>
<thead>
<tr>
<th>Search Trees</th>
<th>Insert</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Levcopoulos, Overmars 1988</td>
<td>1</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Brown, Tarjan 1980</td>
<td>1</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Dietz, Raman 1994</td>
<td>( \log^k n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Harel, Lueker 1979</td>
<td>1</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Guibas et al. 1977</td>
<td>1</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Brodal 1997</td>
<td>1</td>
<td>( \log n )</td>
</tr>
</tbody>
</table>

- Amortized insertions
- Requires the RAM
- \( O(1) \) fixed fingers
- Pointer Machine
A New Algorithm for Splitting Nodes in a Search Tree

- To each leaf $e$ is associated a binary counter $c_e^2$.
- All nodes of height $i$ have degree at least $\Delta_l^i$. 

$\Delta_3$  $\Delta_2$  $\Delta_1$  

$C_2 = 1001$
A New Algorithm for Splitting Nodes in a Search Tree

- To each leaf $l$ is associated a binary counter $c_l$
- All nodes of height $d$ have degree at least $\Delta_d$

**Insert($l, l'$)**

1. $c_l := c_l + 1$, $c_{l'} := c_l$
2. Find $d$ such that $c_l \mod 2^d = 2^{d-1}$
3. Split the $d$'th ancestor of $l$
Main Lemma

Lemma. All nodes of height $d$ have degree $\leq 2^d \cdot \Delta_d$

Proof: Define $\Phi^d_l = 2^d - 1 - ((c_l + 2^{d-1}) \mod 2^d)$

Define $\Phi^d_v = \sum_{l \in T^d_v} \Phi^d_l$

Invariant $\Phi^d_v \leq 2^d \cdot 2^{d \sum \Delta_i}$

provided $\Delta_d \geq 2^d - 1$

The lemma follows from $|T^d_v| \geq \frac{d}{\prod_{i=1}^d \Delta_i}$. \hfill \blacksquare
Minor Problems to Solve

1) How to split a node of non-constant degree in worst-case constant time?

- Do the splitting incrementally in advance.

2) How to perform finger searches?

- a) Level link the tree
- b) Represent the children of each node by the search tree of Levcopoulos, Overmars 1988

Finger searches require $O(\log n)$ time.
A Major Problem to Solve

3) How to find the level $d$ ancestor of a node?

(a) Represent each counter $S_e$ by a stack $S_e$ of intervals of 1's
(b) With each interval $(i,j)$ in $S_e$ store a pointer to the $j+1$st ancestor of $e$

$S_e$ can be updated ($c_e := e+1$) and the $d$'th ancestor of $e$ can be found in worst-case constant time.

$S_e = (0,0,0)\to (2,2,2)\to (5,5,5)$
More Minor Problems To Solve

4) How to copy $S_L' : = S_L$ in worst-case constant time?
   Let the stacks be functional implemented.

5) Splitting a node can make $S_L$ stacks contain pointers to wrong subtrees!
   Don't worry! A height $d$ node has degree $\leq 2^{3 \cdot 2^d \Delta_d}$.
Conclusion and Open Problems

Thm  A pointer based implementation of finger search trees exists, supporting:

- Insert in worst-case $O(1)$ time, and
- Search $\sim O(\log S)$ time.

The data structure requires linear space.

Delete can be supported in worst-case $O(\log^* n)$ time.

Open problems

- Delete in $O(1)$ too.
- Make other splitting based data structures worst-case.
  - Ex. Full Persistence.
  - Dynamic Fractional Cascading.