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. $\boxed{5}-\boxed{7}-\boxed{9}-\boxed{11} \longrightarrow \boxed{5}-\boxed{7}-\boxed{8}-\boxed{9}-\boxed{11}$

  $\uparrow$ f $\quad x=8$

- Search(f, x), search for x in the list starting at f.

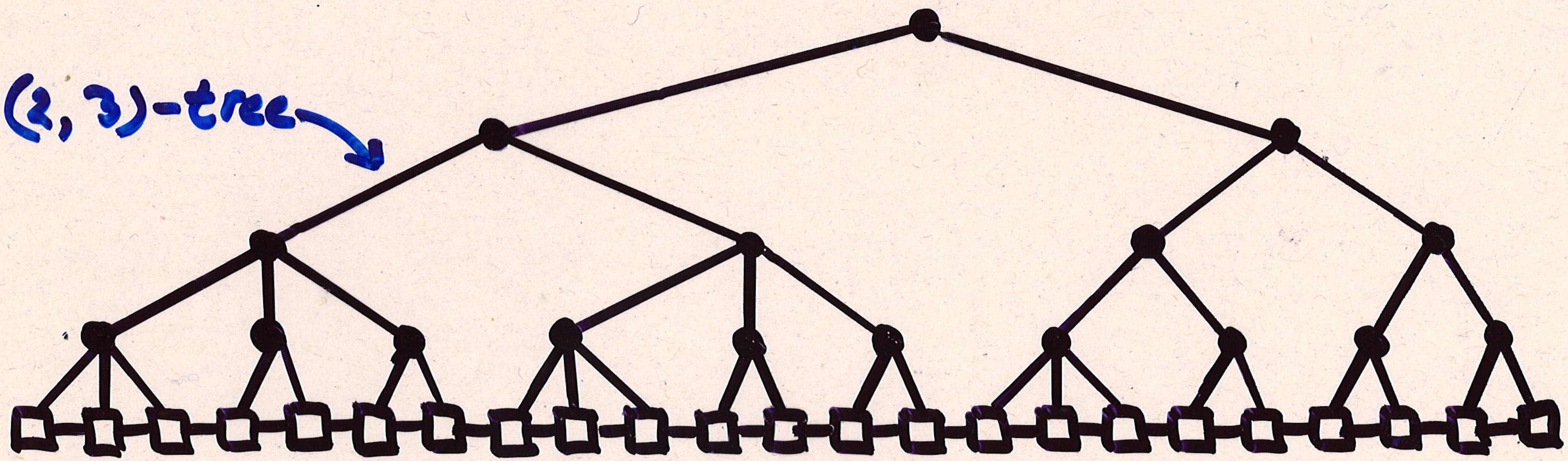
  Ex. $\boxed{4}-\boxed{5}-\boxed{6}-\boxed{8}-\boxed{9}-\boxed{11}-\boxed{13}-\boxed{15}-\boxed{17}-\boxed{19}$

  $\uparrow$ f = finger $\quad \xleftrightarrow{\ \delta\ } \quad$ = x

A Simple Finger Search Tree

Brown, Tarjan 1980

(2, 3)-tree

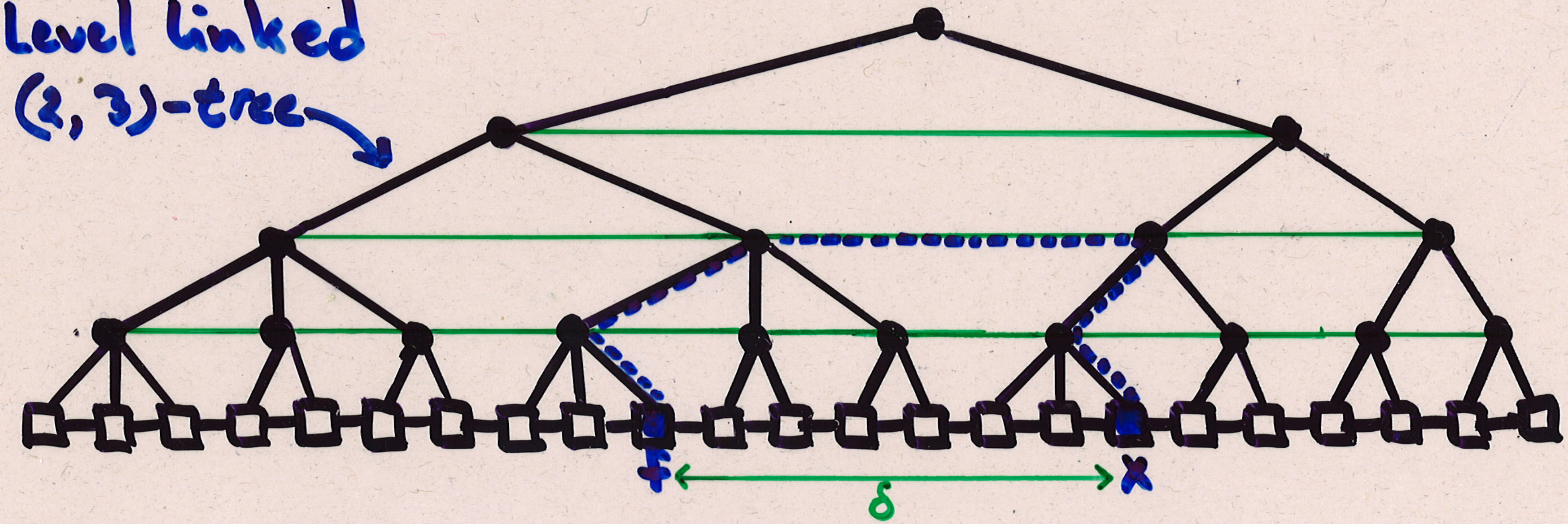


Insert : Amortized O(1)

A Simple Finger Search Tree

Brown, Tarjan 1980

Level linked (2,3)-tree

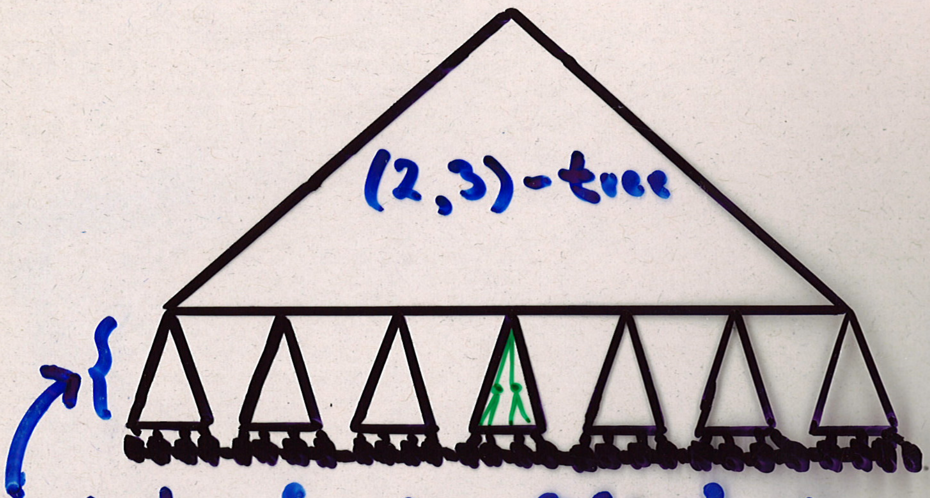


Insert : Amortized $O(1)$.

Search : Worst-case $O(\log \delta)$.

Search Trees With Constant Insertion Time

Levcopoulos, Overmars 1988



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

Idea

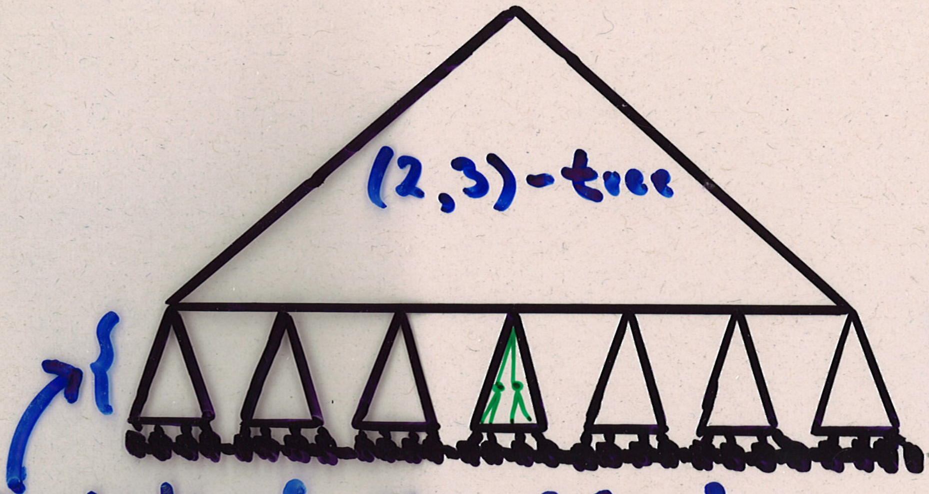
For every $\log n$'th insertion split the largest bucket and update the (2,3)-tree over the next $\log n$ insertions.

Insert : $O(1)$

Search : $O(\log n)$

Search Trees With Constant Insertion Time

Levcopoulos, Overmars 1988



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

Idea

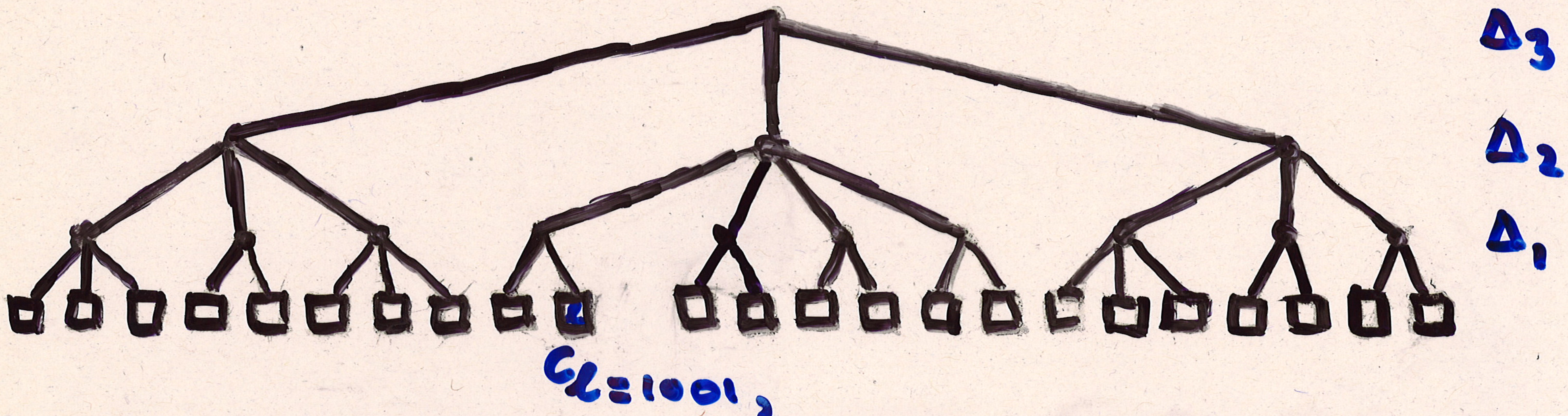
For every $\log n$'th insertion split the largest bucket and update the (2,3)-tree over the next $\log n$ insertions.

		1 recursive	$\log^* n$ recursive
Insert	$O(1)$	$O(1)$	$O(\log^* n)$
Search	$O(\log n)$	$O(\log\log n + \log \delta)$	$O(\log \delta)$

Previous Work

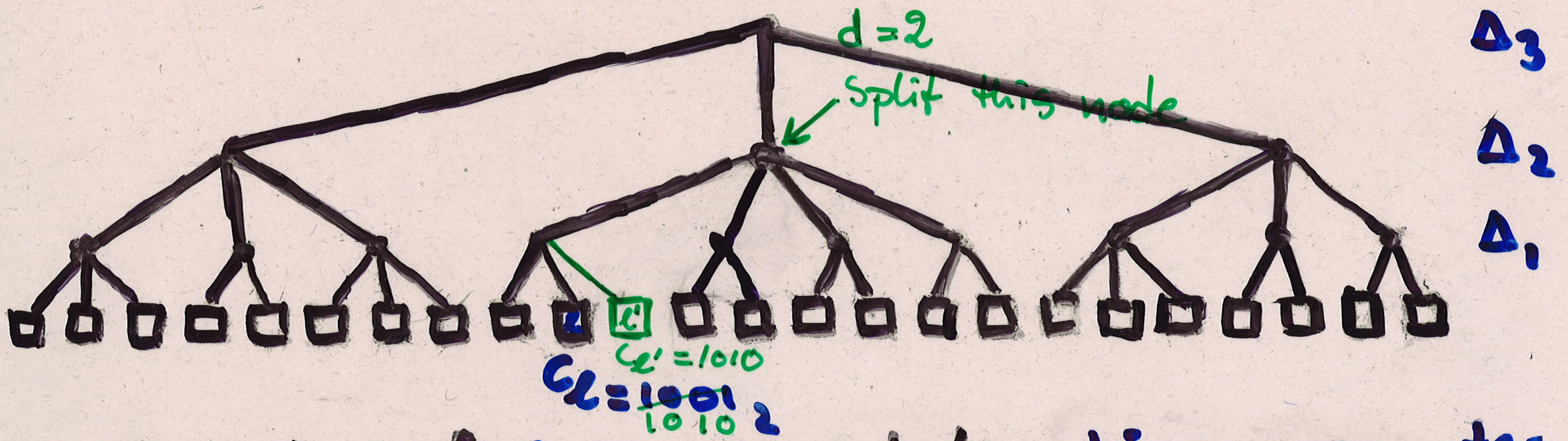
	Insert	Search	
Search Trees	$\log n$	$\log n$	
Levcopoulos, Overmars 1988	1	$\log n$	
Brown, Tarjan 1980	1	$\log \delta$	Amortized insertions
Dietz, Raman 1994	1	$\log \delta$	Requires the RAM
Harel, Lueker 1979	$\log^* n$	$\log \delta$	
Guibas et al. 1977	1	$\log \delta$	$O(1)$ fixed fingers
→ Brodal 1997	1	$\log \delta$	Pointer Machine

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 Δ_d.

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 Δ_d.

Insert(ℓ, ℓ')

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

Main Lemma

Lemma All nodes of height d have degree $\le 2^{2\cdot 2^d}\Delta_d$.

Proof: Define $\Phi_\ell^d = 2^{2^d} - 1 - ((c_\ell + 2^{d-1}) \bmod 2^d)$

Define $\Phi_v^d = \sum_{\ell \in T_v^d} \Phi_\ell^d$

Invariant $\Phi_v^d \le 2^{2\cdot 2^d} \prod_{i=1}^{d} \Delta_i$

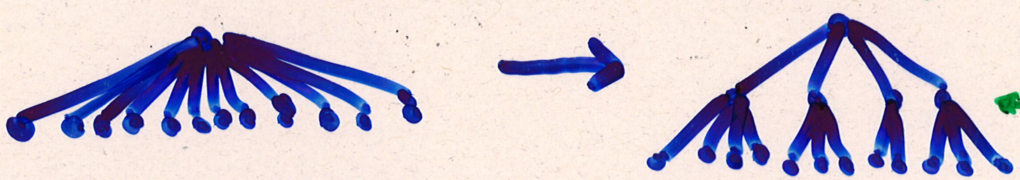
provided $\Delta_d \ge 2^{2^d} - 1$

The lemma follows from $|T_v^d| \ge \prod_{i=1}^{d} \Delta_i$. $\square$

Minor Problems to Solve

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

Do the splitting incrementally in advance.



← Intermediate nodes of degree $\Delta_d .. 2\Delta_d - 1$

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**

$\Downarrow$ Finger searches require $O(\log \delta)$ time.

A Major Problem to Solve

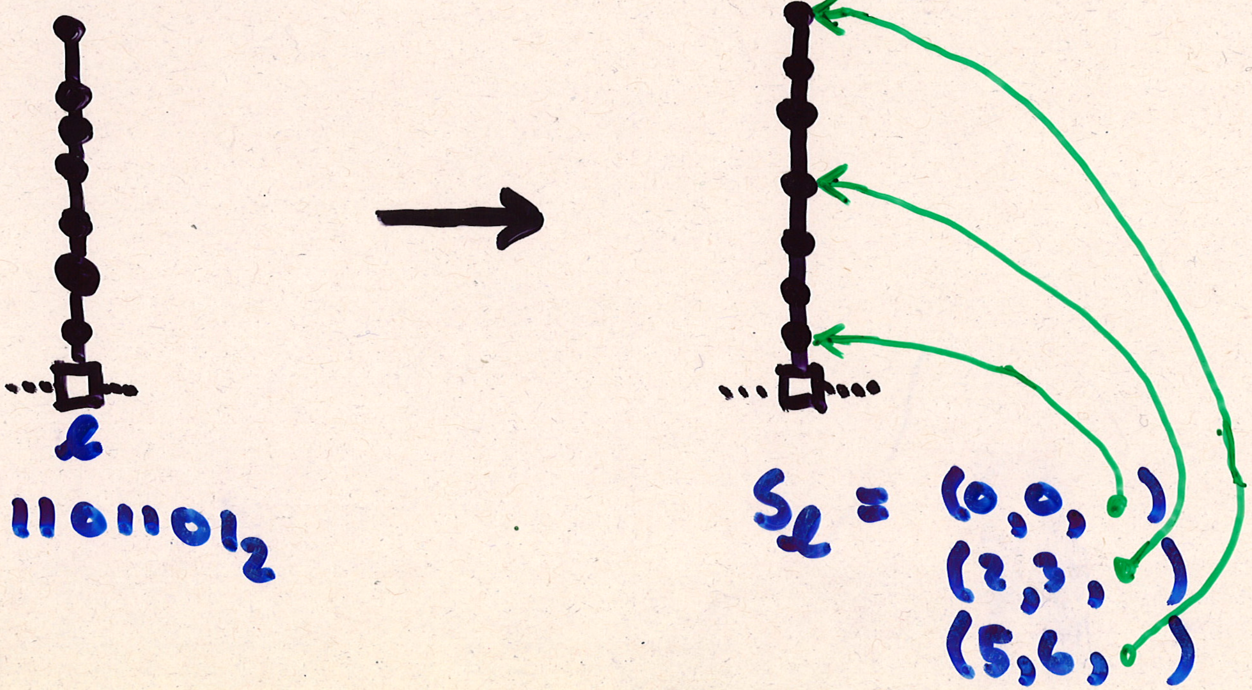
3) How to find the level d ancestor of a node?

a) Represent each counter c_ℓ by a stack S_ℓ of intervals of 1's

b) With each interval (i,j) in S_ℓ store a pointer to the $j+1$'st ancestor of ℓ.

$\Downarrow$

S_ℓ can be updated ($c_\ell := c_\ell + 1$) and the d'th ancestor of ℓ can be found in worst-case constant time.



More Minor Problems To Solve

4) How to copy $S_{\ell'} := S_\ell$ 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 —"— $O(\log \delta)$ 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.