Finger Search
Searching in a sorted array

Exponential-search(13)

Binary-search(13)

Finger

time $O(\log n)$

time $O(\log d)$

Bently Yao 1976

$\sum_{i=1}^{\log^* n} \log(i) x + O(\log^* n)$
O(1) Insertions


- Buckets $O(\log n)$ $\Rightarrow$ Amortized $O(1)$ insertions (also by 2-4-trees)
- 2-level buckets $O(\log^2 n)$ size
- Incremental splitting of buckets $\Rightarrow$ Worst-case $O(1)$ insertions
- Split largest bucket

$n/\log^2 n$ leafs

degree $\Theta(\log n)$
Zeroing Game


- Variables \(x_1, \ldots, x_n \geq 0\) (initially \(x_i = 0\))
- Players Z and A alternate to take turns
  - Z: Select \(j\) where \(a_j = \max_i x_i : x_j := 0\)
  - A: Select \(a_1, \ldots, a_n \geq 0\) and \(\sum_i a_i = 1 : x_i += a_i\)

**Theorem** \(\forall i : x_i \leq H_{n-1} + 1 \leq \ln n + 2\)

**Proof**
- Consider a vector \(x^{(m)}\) after \(m \geq n\) rounds
- \(S_k \overset{\text{def}}{=} \text{sum of } k \text{ largest } x_i \text{ of } x^{(m+1-k)}\)
- \(S_n \leq n\) (induction)
- \(S_i \leq 1 + S_{i+1} \cdot i / (i+1)\)
- \(S_1 \leq 1 + S_2 / 2 \leq 1 + 1/2 + S_2 / 3 \leq 1 + 1/2 + \cdots + 1/(n-1) + S_n / n \leq H_{n-1} + 1\)

**Corollary**
For the halving game, \(Z : x_i := x_i / 2\)
For the splitting game, \(Z : x_i, x_i' := x_i / 2\)
\[\forall i : x_i \leq 2 \cdot (H_{n-1} + 1)\]
## Dynamic Finger Search

<table>
<thead>
<tr>
<th>Search without fingers</th>
<th>Search</th>
<th>Insert/Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red-black, AVL, 2-4-trees, ...</td>
<td>$O(\log n)$</td>
<td>[ \begin{align*} O(\log n) \ O(1) \end{align*} ]</td>
</tr>
<tr>
<td>Levopolous, Overmars 1978</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>O(1) fixed fingers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Guibas et al. 1977, ...</td>
<td>$O(\log d)$</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Each node a finger</th>
<th>Search</th>
<th>Insert/Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level-linked (2,4)-trees</td>
<td>$O(\log d)$</td>
<td>[ \begin{align*} O(\log n) \ O(1) \text{ am.} \end{align*} ]</td>
</tr>
<tr>
<td>Randomized Skip lists</td>
<td>$O(\log d)$ exp.</td>
<td>O(1) exp.</td>
</tr>
<tr>
<td>Treaps</td>
<td>$O(\log d)$ exp.</td>
<td>O(1) exp.</td>
</tr>
<tr>
<td>Brodal, Lagogiannis, Makris, Tsakalidis, Tsichlas 2003</td>
<td>$O(\log d)$</td>
<td>O(1)</td>
</tr>
<tr>
<td>Dietz, Raman 1994 (RAM)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Level-Linked (2,4)-trees


Potential $\Phi = 2 \cdot \# \text{ degree-4} + \# \text{ degree-2}$

Updates  Split nodes of degree $>4$, fusion nodes of degree $<2$

Search   Search up + top-down search
Randomized Skip Lists


**Insertion**  
Increase pile to next level with pr. = 1/2

**Height**  
$O(\log n)$ expected with high probability

**Pointer**  
Horizontally spans $O(1)$ exp. piles one level below

**Finger**  
Remember nodes on search path
Treaps – Randomized Binary Search Trees


- Each element random priority
- Search tree wrt element
- Heap order wrt priority
- Height $O(\log n)$ expected
- Insert & deletion rotations $O(1)$ expected time
- Search: Go up to LCA, and search down – concurrently follow excess path to find next LCA candidate
  Search path $O(\log d)$ expected
Application: Binary Merging


- Merging sorted lists $L_1$ and $L_2$ / finger search trees
  
  \[ \sum \log(d_i) = |L_1| \log \left( \frac{|L_2| + |L_1|}{|L_1|} \right) \]

- Merging leaf lists in an **arbitrary** binary tree $O(n \cdot \log n)$

**Proof** Induction $O(\log n!)$

\[ O(\log n_1! + \log n_2! + n_1 \cdot \log ((n_1+n_2)/n_1)) \]
\[ \quad = O(\log n_1! + \log n_2! + \log (\frac{n_1+n_2}{n_1})) \]
\[ \quad = O(\log (n_1! \cdot n_2! \cdot (\frac{n_1+n_2}{n_1}))) = O(\log (n_1+n_2)!) \]
Maximal Pairs with Bounded Gap


Build suffix tree (ST) & make it binary
Create leaf lists at each node
Right-maximal pairs = ST nodes
Find maximal pairs = finger search at ST nodes

O(n⋅log n+k)