Balanced Trees

Prof. Clarkson
Fall 2018

Today’s music: Get the Balance Right by Depeche Mode
Finding an element in a linked list is $O(n)$. How efficient is finding an element in a binary search tree?

A. $O(1)$  
B. $O(\log n)$  
C. $O(n)$  
D. $O(n \log n)$  
E. $O(n^2)$
Review

Previously in 3110:
• Streams

Today:
• Balanced trees
Running example: Sets

module type Set = sig
  type 'a t
  val empty : 'a t
  val insert : 'a -> 'a t -> 'a t
  val mem : 'a -> 'a t -> bool
  ...
end
## Set implementations: performance

<table>
<thead>
<tr>
<th>Workload 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>insert</td>
<td>mem</td>
</tr>
<tr>
<td><strong>ListSet</strong></td>
<td>35s</td>
<td>106s</td>
</tr>
</tbody>
</table>

MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs
# Set implementations: performance

<table>
<thead>
<tr>
<th>Workload 1</th>
<th>insert</th>
<th>mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListSet</td>
<td>35s</td>
<td>106s</td>
</tr>
<tr>
<td>BstSet</td>
<td>130s</td>
<td>149s</td>
</tr>
</tbody>
</table>

MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs
## Set implementations: performance

<table>
<thead>
<tr>
<th></th>
<th>Workload 1</th>
<th></th>
<th>Workload 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>insert</td>
<td>mem</td>
<td>insert</td>
<td>mem</td>
</tr>
<tr>
<td>ListSet</td>
<td>35s</td>
<td>106s</td>
<td>35s</td>
<td>106s</td>
</tr>
<tr>
<td>BstSet</td>
<td>130s</td>
<td>149s</td>
<td>0.07s</td>
<td>0.07s</td>
</tr>
</tbody>
</table>

MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs
## Set implementations: performance

<table>
<thead>
<tr>
<th></th>
<th>Workload 1</th>
<th>Workload 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>insert</td>
<td>mem</td>
</tr>
<tr>
<td>ListSet</td>
<td>35s</td>
<td>106s</td>
</tr>
<tr>
<td>BstSet</td>
<td>130s</td>
<td>149s</td>
</tr>
<tr>
<td>RbSet</td>
<td>0.12s</td>
<td>0.07s</td>
</tr>
</tbody>
</table>

*MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs*
Sir Tony Hoare

b. 1934

Turing Award Winner 1980

For his fundamental contributions to the definition and design of programming languages.

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil."
BST SET
Binary search tree (BST)

• Binary tree: every node has two subtrees
• BST invariant:
  – all values in l are less than v
  – all values in r are greater than v
## Back to performance

<table>
<thead>
<tr>
<th></th>
<th>Workload 1</th>
<th></th>
<th>Workload 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>insert</td>
<td>mem</td>
<td>insert</td>
<td>mem</td>
</tr>
<tr>
<td>ListSet</td>
<td>35s</td>
<td>106s</td>
<td>35s</td>
<td>106s</td>
</tr>
<tr>
<td>BstSet</td>
<td>130s</td>
<td>149s</td>
<td>0.07s</td>
<td>0.07s</td>
</tr>
</tbody>
</table>
Workloads

• Workload 1:
  – **insert**: 50,000 elements in **ascending** order
  – **mem**: 100,000 elements, half of which not in set

• Workload 2:
  – **insert**: 50,000 elements in **random** order
  – **mem**: 100,000 elements, half of which not in set
Insert in random order

- Resulting tree depends on exact order
- One possibility for inserting 1..4 in random order 3, 2, 4, 1:
Insert in linear order

Only one possibility for inserting 1..4 in linear order 1, 2, 3, 4:

unbalanced: leaning toward the right
Finding an element in a linked list is $O(n)$. How efficient is finding an element in a binary search tree?

A. $O(1)$  
B. $O(\log n)$  
C. $O(n)$  
D. $O(n \log n)$  
E. $O(n^2)$
When trees get big

• Inserting next element in linear tree always takes \( n \) operations where \( n \) is number of elements in tree already
• Inserting next element in randomly-built tree might take far fewer...
all paths through *perfect binary tree* have same length: $\log_2 (n+1)$, where $n$ is the number of nodes, recalling there are implicitly leafs below each node at bottom level
Performance of BST

- `insert` and `mem` are both $O(n)$
- But if trees always had short paths instead of long paths, could be better: $O(\log n)$
- How could we ensure short paths?
  - i.e., *balance* trees so they don't lean
Strategies for achieving balance

• In general:
  – Strengthen the RI to require balance
  – And modify insert to guarantee it

• Well known data structures:
  – 2-3 trees: all paths have same length
  – AVL trees: length of shortest and longest path from any node differ at most by one
  – Red-black trees: length of shortest and longest path from any node differ at most by factor of two

• All of these achieve $O(\log(n))$ insert and mem
RED-BLACK TREES
Red-black trees

• [Guibas and Sedgewick 1978], [Okasaki 1998]

• Binary search tree with:
  – Each node colored red or black
  – Leafs colored black

• RI: BST +
  – Local invariant: No red node has a red child
  – Global invariant: Every path from the root to a leaf has the same number of black nodes
Question

Is this a valid rep?

A. Yes
B. No
Path length

• Invariants:
  – No red node has a red child
  – Every path from the root to a leaf has the same number of black nodes

• Together imply: length of longest path is at most twice length of shortest path
  – e.g., B-R-B-R-B-R-B vs. B-B-B-B
RED-BLACK SET
RB rotate (1 of 4)

rotates to

eliminates y-x violation
but maybe y has a red parent: new violation
keep recursing up tree
RB rotate (2 of 4)

rotates to
RB rotate (3 of 4)

Original:

```
    x
   / \  
  y   z
 /     
`b`     `d`
```

Rotates to:

```
    y
   / \  
  x   z
 /     
`a`   `c`

    b
   / \  
  c   d
```

rotates to

```
    y
   / \  
  x   z
 /     
`a`   `c`

    b
   / \  
  c   d
```
RB rotate (4 of 4)

rotates to
RB balance

let balance = function
  | (Blk, Node (Red, Node (Red, a, x, b), y, c), z, d) (* 1 *)
  | (Blk, Node (Red, a, x, Node (Red, b, y, c)), z, d) (* 2 *)
  | (Blk, a, x, Node (Red, Node (Red, b, y, c), z, d)) (* 4 *)
  | (Blk, a, x, Node (Red, b, y, Node (Red, c, z, d))) (* 3 *)
    -> Node (Red, Node (Blk, a, x, b), y, Node (Blk, c, z, d))
  | t -> Node t
Upcoming events

• [next Thur] Prelim

This is blissfully balanced.

WE ARE GROOT
Upcoming events

• [next Thur] Prelim

This is blissfully balanced.

WE ARE 3110
Upcoming events

• [next Thur] Prelim

This is blissfully balanced.

THIS IS 3110