CSE 250
Lecture 27
Red-Black Trees
## BST Operation Costs

<table>
<thead>
<tr>
<th>Operation</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>remove</td>
<td>$O(d)$</td>
</tr>
</tbody>
</table>
Red-Black Trees

• Color each node red or black
  1) # of black nodes from each empty to root must be identical
  2) Parent of a red node must be black

• On Insertion (or deletion)
  – Inserted node is red (won’t change # of black nodes)
  – “Repair” violations of rule 2 by rotating or recoloring
    • Repairs guarantee rule 1 is preserved
Red-Black Trees

• # of black nodes on a path from root to leaf is the same
  – Call this number (for a given tree) B

• Each red node must have a black parent
  – What’s the longest possible path from the root to a leaf?  
    2B (Black, Red, Black, Red, Black, Red, ...)
  – What’s the shortest possible path from the root to a leaf?  
    B (Black, Black, Black, ...)

Balancing Empty Node Depth

\[ d/2 = \log(n) \]
\[ d = 2\log(n) = O(\log(n)) \]

Must be full \((2^{\lceil d/2 \rceil} \text{ nodes})\)
Red-Black Trees
Red-Black Trees

![Red-Black Tree Diagram]
Red-Black Trees

A

B

C

D

E

F

G

H

I

3

3

3

3

3

3

3

3

3

3

3
Red-Black Trees
Red-Black Trees

All Valid R-B Tree Fragments

Repair A
Red-Black Trees

Case 1: All Good!
Red-Black Trees

Case 1b: All Good!
Red-Black Trees

Case 1b: All Good!
Red-Black Trees

Problem!
Red-Black Trees

Case 2: Split Black Node
Red-Black Trees

Case 2: Split Black Node

- C's parent may be red (repeat the repair process)
- # of black nodes on each path didn’t change
Red-Black Trees

Case 2: Split Black Node

Also works if A is right-child of B (or B is right-child of C)
Red-Black Trees

Case 3: Rotate B, C
Red-Black Trees

Case 3: Rotate B, C

-1 black node to the root

Same # of black nodes to the root
Red-Black Trees

Case 3: Rotate B, C

Root of subtree under consideration is black (repair is all done)

-1 black node to the root

Same # of black nodes to the root
Red-Black Trees

Case 4: Rotate A, B → B, C
Case 4: Rotate A, B → B, C

Now identical to case 3
## Red/Black Trees

- **Case 1** (Parent of Red is Black) \( O(1) \)
  - Done!
- **Case 1.a** (Root is Red) \( O(1) \)
  - Recolorparent Black \( O(1) + O(\log(n) \cdot O(1)) \)
- **Case 2** (Parent is Red; Aunt is Red) \( O(1) \) \( + \) fix grandparent
  - Recolor Grandparent Red, Recolor parent and aunt Black
  - Grandparent is now red; Repeat check there
- **Case 3** (Left child of Red Parent; Aunt is Black) \( O(1) \)
  - Rotate Grantparent Right; Swap rotated node colors
- **Case 4** (Right child of Red Parent; Aunt is Black) \( O(1) \)
  - Rotate Parent Left; Continue with Case 3
Insertion

- Find the insertion point (as in a BST) \( O(d) = O(\log(n)) \)
- Insert the node as red \( O(1) \)
  - Preserves the black depth
- Fix colors (if needed) \( O(\log(n)) \)
  - Preserves the black depth (or adds 1 at root)
Hash Tables
Finding Items: Sequences

- Is it element 1?
  - If so, return, else...
- Is it element 2?
  - If so, return, else...
- Is it element 3?
  - If so, return, else...
- etc...
Finding Items: Sorted Sequences

- How does it compare to element $\frac{1}{2} n$?
  - If equal, return
  - If lesser, how does it compare to element $\frac{1}{4} n$?
    - If equal return
    - If lesser, etc...
    - If greater, etc...
  - If greater, how does it compare to element $\frac{3}{4} n$?
    - etc...
Finding Items: Trees

- How does it compare to root?
  - If equal, return
  - If lesser, how does it compare to left child?
    - If equal return
    - If lesser, etc...
    - If greater, etc...
  - If greater, how does it compare to right child?
    - etc...
Finding Items

The most expensive part of finding records is finding them. (i.e., where is the record located?)

So... skip the search
Finding the item
Alternative Idea: Assign Bins

- Create an array of size $N$
- Pick an $O(1)$ function to assign each record a number in $[0, N)$
  - First letter of name $\rightarrow [0, 26)$
Alternative Idea: Assign Bins

Athos  B  C  D'Artagnan  ...  Porthos  ...  Y  Z
Alternative Idea: Assign Bins

- **Pros**
  - $O(1)$ Insert
  - $O(1)$ Find
  - $O(1)$ Remove
- **Cons**
  - Wasted Space (Only 3/26 slots used)
  - Duplication (What about Aramis?)
Other Functions

- Identity Function: \((x: \text{Int}) \Rightarrow x\)
  - **Problem**: Can return values over \(N\)
  - **Solution**: Cap return value by Modulus with \(N\)
    - \((x: \text{Int}) \Rightarrow x \% N\)
### Other Functions

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>
Other Functions

- Identity Function: $(x: \text{Int}) \Rightarrow x \mod N$
- Linear Function: $(x: \text{Int}) \Rightarrow (x \times a + b) \mod N$ (for some $a,b$)
- .. or Quadratic: $(x: \text{Int}) \Rightarrow ((x \times a + b) \times x + c) \mod N$ (for $a,b,c$)
Other Functions

• **Ideal**: Function assigns every record to a unique position
  - If \( n = N \) records, every array position is used
  - No conflicted assignment

• Examples
  - Unique Record IDs from \([0, N)\) (like UBIT #s)
    • ... no deletions
  - Cumulative Distribution Functions (CDFs)
    • ... hard to encode
Almost Ideal...

- A function $a$ that evenly distributes records
  - $O(1)$ means we can’t compare against other records.
  - **Not random**: Same input = same output
  - **Pseudorandom**: Every position has the same probability
    - (for a given record)
- For $n$ records, the chance of first conflict is $n/N$
  - Expect $\sqrt{N}$ insertions before the first conflict