CMPSC250
Lecture 9: Binary Search Trees
Prof. John Wenskovitch
02/10/2015
Last Time

- Symbol tables – a data structure associating a key with some value info
- Sequential search
- Binary search
Can We Do Better?

<table>
<thead>
<tr>
<th>Algorithm (Data Structure)</th>
<th>Worst-Case Cost (after N inserts)</th>
<th>Average-Case Cost (after N random inserts)</th>
<th>Efficiently Support Ordered Operations?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential search (unordered linked list)</td>
<td>N</td>
<td>N/2</td>
<td>no</td>
</tr>
<tr>
<td>Binary search (ordered array)</td>
<td>lg N</td>
<td>lg N</td>
<td>yes</td>
</tr>
</tbody>
</table>
Goals

- Improve insertion speed beyond the current $N$
- Keep search speed in the $\log(N)$ complexity class
- Keep the implementation as simple as possible
Binary Tree

- Binary Tree
- Node
- Link
- Root
- Child
- Parent
- Subtree
Binary Search Tree

- **Binary Search Tree** – A binary tree where each node has a Comparable key (and an associated value), and satisfies the restriction that the key in any node is:
  - ...**larger** than the keys in all nodes in that node’s **left** subtree, and
  - ...**smaller** than the keys in all nodes in that node’s **right** subtree.

- (how is this different from our heap implementation?)
Binary Search Tree – Insertion

```
put(Key key, Value val) {
    root = put(root, key, val);
} //put

put(Node x, Key key, Value val) {
    if (x == null) {
        return new Node(key, val, 1);
    } //if
    int cmp = key.compareTo(x.key);
    if (cmp < 0) {
        x.left = put(x.left, key, val);
    } else if (cmp > 0) {
        x.right = put(x.right, key, val);
    } else {
        x.val = val;
    } //if-else
    x.N = size(x.left) + size(x.right) + 1;
    return x;
} //put
```
get(Node x, Key key) {
    if (x == null) {
        return null;
    } //if
    int cmp = key.compareTo(x.key);
    if (cmp < 0) {
        return get(x.left, key);
    } else if (cmp > 0) {
        return get(x.right, key);
    } else {
        return x.val;
    } //if-else
} //get
Binary Search Tree – Insertion Visual

1. Inserting L:
   - Search for L ends at this null link.

2. Create new node:
   - Create a new node (L).

3. Reset links and increment counts on the way up:
   - Reset links and increment counts on the way up.
Binary Search Tree – Retrieval Visual

Successful search for R:
- "black nodes could match the search key"
- "R is less than S so look to the left"
- "found R (search hit) so return value"

Unsuccessful search for T:
- "T is greater than S so look to the right"
- "T is less than X so look to the left"
- "link is null so T is not in tree (search miss)"
Binary Search Tree – Analysis

• Is the shape of a BST independent of input?
  – Best case – tree is balanced
  – Average case – tree is sort of balanced, sort of unbalanced
  – Worst case – tree is a linked list
Binary Search Tree – Analysis

• **Theorem:** Search hits in a BST built from $N$ random keys require $\sim 2 \log(N)$ compares in the average case.

• **Proof:** Sort of the same proof from Mergesort, based on an extra observation.

• **Corollary:** Search misses and insertions also require $\sim 2 \log(N)$ compares in the average case, because they take 1 more compare, which does not change the complexity class.
Binary Search Tree – Analysis

• Let $C(N)$ be the \textbf{internal path length} of a BST for a search operation. Thus, the average cost of each search hit is $1 + \frac{C(N)}{N}$. We know that $C(0) = C(1) = 1$.

• For $N > 1$, $C(N) = (N - 1) + \frac{C(0)+C(N-1)}{N} + \frac{C(1)+C(N-2)}{N} + \cdots + \frac{C(N-1)+C(0)}{N}$.

• Because all heights are equally likely, the average case will be the same as the best case. This lets us simplify: $C(N) = 2(C(N - 1)) + (N - 1)$.

• We know that this pattern continues: $C(N - 1) = 2(C(N - 2)) + (N - 2)$.

• Thus, $C(N) = 2(2(C(N - 2)) + (N - 1)) + (N - 1)$.

• Simplify: $C(N) = 4(C(N - 2)) + 2(N - 2) + (N - 1)$.

• We know that this pattern continues: $C(N - 2) = 2(C(N - 3)) + (N - 3)$.

• Thus, $C(N) = 4(C(N - 3) + (N - 3)) + 2(N - 2) + (N - 1)$.

• Simplify: $C(N) = 8(C(N - 3)) + 4(N - 3) + 2(N - 2) + (N - 1)$.

• A pattern emerges: $C(N) = 2^k(C(N - k)) + \sum_k (2^{k-1})(N - k)$.

• Now, consider the case where $k = N$...

• Here, $C(N) = 2^N(C(0)) + [(2^{N-1})(0) + (2^{N-2})(1) + \cdots + (2^0)(N)]$.

• Simplify: $C(N) = 2^N + (2^N)N$.

• Log: $C(N) = 2^{\log(N)} + \log((2^N)N) = N + N \times \log(N) = O(N \times \log(N))$. ■
Binary Search Tree – Analysis

- Sequential Search – 2246 compares
- Binary Search – 484 compares
- Binary Search Tree – 13.9 compares
Binary Search Tree – Analysis

- Sequential Search – 2246 compares
- Binary Search – 484 compares
- Binary Search Tree – 13.9 compares

<table>
<thead>
<tr>
<th></th>
<th>tale.txt</th>
<th>leipzig1M.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>words</td>
<td>distinct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>all words</td>
<td>135,635</td>
<td>10,679</td>
</tr>
<tr>
<td>8+ letters</td>
<td>14,350</td>
<td>5,737</td>
</tr>
<tr>
<td>10+ letters</td>
<td>4,582</td>
<td>2,260</td>
</tr>
</tbody>
</table>
Order-Based Methods

• How can we find the min/max in a BST?

\[
\text{min()} \{ \\
\quad \text{return min(root).key; } \\
\} \quad //\text{min}
\]

\[
\text{min(Node x) } \{ \\
\quad \text{if (x.left == null)} \{ \\
\quad \quad \text{return x; } \\
\quad \} \quad //\text{if} \\
\quad \text{return min(x.left); } \\
\} \quad //\text{min}
\]
Order-Based Methods

• How about floor/ceiling?

```java
floor(Key key) {
    Node x = floor(root, key);
    if (x == null) {
        return null;
    } //if
    return x.key;
} //floor

floor() {
    if (x == null) {
        return null;
    } //if
    int cmp = key.compareTo(x.key);
    if (cmp == 0) {
        return x;
    } if (cmp < 0) {
        return floor(x.left, key);
    } //if-else
    Node t = floor(x.right, key);
    if (t != null) {
        return t;
    } else {
        return x;
    } //if-else
} //min
```
Order-Based Methods

• Selection?

```java
select(int k) {
    return select(root, k).key;
} //select

select(Node x, int k) {
    if (x == null) {
        return null;
    } //if

    int t = size(x.left);
    if (t > k) {
        return select(x.left, k);
    } else if (t < k) {
        return select(x.right, k-t-1);
    } else {
        return x;
    } //if-else
} //select
```
Order-Based Methods

• Rank?

```java
rank(Key key) {
    return rank(key, root);
} //rank

rank (Key key, Node x) {
    if (x == null) {
        return 0;
    } //if

    int cmp = key.compareTo(x.key);
    if (cmp < 0) {
        return rank(key, x.left);
    } else if (cmp > 0) {
        return 1+size(x.left)+rank(key, x.right);
    } else {
        return size(x.left);
    } //if-else
} //rank
```
The Tough One – Delete

delete(Key key) {
    root = delete(root, key);
} //delete

delete(Node x, Key key) {
    if (x == null) { return null; }
    int cmp = key.compareTo(x.key);
    if (cmp < 0) { x.left = delete(x.left, key); }
    if (cmp > 0) { x.right = delete(x.right, key); }
    else {
        if (x.right == null) { return x.left; }
        if (x.left == null) { return x.right; }
        Node t = x;
        x = min(t.right);
        x.right = deleteMin(t.right);
        x.left = t.left;
    } //if-else
    x.N = size(x.left) + size(x.right) + 1;
    return x;
} //delete
The Tough One – Delete

deleteMin(Node x) {
    if (x.left == null) {
        return x.right;
    } //if
    x.left = deleteMin(x.left);
    x.N = size(x.left) + size(x.right) + 1;
    return x;
} //delete
## Conclusions

<table>
<thead>
<tr>
<th>Algorithm (Data Structure)</th>
<th>Worst-Case Cost (After N Inserts)</th>
<th>Average-Case Cost (After N Random Inserts)</th>
<th>Efficiently Support Ordered Operations?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert</td>
<td>Search Hit</td>
</tr>
<tr>
<td>Sequential Search</td>
<td>$N$</td>
<td>$N$</td>
<td>$N/2$</td>
</tr>
<tr>
<td>(Unordered Linked List)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Search</td>
<td>$\lg N$</td>
<td>$N$</td>
<td>$\lg N$</td>
</tr>
<tr>
<td>(Ordered Array)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Tree Search</td>
<td>$N$</td>
<td>$N$</td>
<td>$1.39 \lg N$</td>
</tr>
<tr>
<td>(BST)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Any Questions?