Hash Tables implement the Dictionary ADT, namely:

- Insert($x$) - $O(1)$ expected time, $\Theta(n)$ worst case
- Lookup($x$) - $O(1)$ expected time, $\Theta(n)$ worst case
- Delete($x$) - $O(1)$ expected time, $\Theta(n)$ worst case

Red-Black trees (a kind of binary tree) also implement the Dictionary ADT, namely:

- Insert($x$) - $O(\log n)$ time
- Lookup($x$) - $O(\log n)$ time
- Delete($x$) - $O(\log n)$ time
Why BST?

• Q: When would you use a Search Tree?
• A1: When need a hard guarantee on the worst case run times (e.g. “mission critical” code)
• A2: When want something more dynamic than a hash table (e.g. don’t want to have to enlarge a hash table when the load factor gets too large)
• A3: Search trees can implement some other important operations...

What is a BST?

• It’s a binary tree
• Each node holds a key and record field, and a pointer to left and right children
• Binary Search Tree Property is maintained

Search Tree Operations

• Insert
• Lookup
• Delete
• Minimum/Maximum
• Predecessor/Successor

Binary Search Tree Property

• Let \( x \) be a node in a binary search tree. If \( y \) is a node in the left subtree of \( x \), then \( \text{key}(y) \leq \text{key}(x) \). If \( y \) is a node in the right subtree of \( x \) then \( \text{key}(x) \leq \text{key}(y) \)
Inorder Tree-Walk

Inorder-TW(x)
{
    if (x is not nil)
    {
        Inorder-TW(left(x));
        print key(x);
        Inorder-TW(right(x));
    }
}

- BSTs are arranged in such a way that we can print out the elements in sorted order in $\Theta(n)$ time
- Inorder Tree-Walk does this
Analysis

• Correctness?
• Run time?

Analysis

• Let \( h \) be the height of the tree
• The run time is \( O(h) \)
• Correctness???

Search in BT

Tree-Search(x,k){
    if (x=nil) or (k = key(x)){
        return x;
    }
    if (k<key(x)){
        return Tree-Search(left(x),k);
    }else{  
        return Tree-Search(right(x),k);
    }
}

In-Class Exercise

• Q1: What is the loop invariant for Tree-Search?
• Q2: What is Initialization?
• Q3: Maintenance?
• Q4: Termination?
Tree Min/Max

- Tree Minimum(x): Return the leftmost child in the tree rooted at x
- Tree Maximum(x): Return the rightmost child in the tree rooted at x

Successor Intuition

- Case 1: If right subtree of x is non-empty, successor(x) is just the leftmost node in the right subtree
- Case 2: If the right subtree of x is empty and x has a successor, then successor(x) is the lowest ancestor of x whose left child is also an ancestor of x.

Tree-Successor

```
Tree-Successor(x){
    if (right(x) != null){
        return Tree-Minimum(right(x));
    }
    y = parent(x);
    while (y!=null and x!=left(y)){
        x = y;
        y = parent(y);
    }
    return y;
}
```

Insertion

```
Insert(T,x)
1. Let r be the root of T.
2. Do Tree-Search(r,key(x)) and let p be the last node processed in that search
3. If p is nil (there is no tree), make x the root of a new tree
4. Else if key(x) ≤ p, make x the left child of p, else make x the right child of p
```
Deletion

- Code is in book, basically there are three cases, two are easy and one is tricky
- Case 1: The node to delete has no children. Then we just delete the node
- Case 2: The node to delete has one child. Then we delete the node and “splice” together the two resulting trees

Case 3: The node, $x$ to be deleted has two children

1. Swap $x$ with Successor($x$) (Successor($x$) has no more than 1 child (why?))
2. Remove $x$, using the procedure for case 1 or case 2.

Analysis

- All of these operations take $O(h)$ time where $h$ is the height of the tree
- If $n$ is the number of nodes in the tree, in the worst case, $h$ is $O(n)$
- However, if we can keep the tree balanced, we can ensure that $h = O(\log n)$
- Red-Black trees can maintain a balanced BST

Randomly Built BST

- What if we build a binary search tree by inserting a bunch of elements at random?
- Q: What will be the average depth of a node in such a randomly built tree? We’ll show that it’s $O(\log n)$
- For a tree $T$ and node $x$, let $d(x, T)$ be the depth of node $x$ in $T$
- Define the total path length, $P(T)$, to be the sum over all nodes $x$ in $T$ of $d(x, T)$
Analysis

"Shut up brain or I'll poke you with a Q-Tip" - Homer Simpson

- Note that the average depth of a node in $T$ is
  \[ \frac{1}{n} \sum_{x \in T} d(x, T) = \frac{1}{n} P(T) \]
- Thus we want to show that $P(T) = O(n \log n)$

- Let $P(n)$ be the expected total depth of all nodes in a randomly built binary tree with $n$ nodes
- Note that for all $i$, $0 \leq i \leq n - 1$, the probability that $T_l$ has $i$ nodes and $T_r$ has $n - i - 1$ nodes is $1/n$.
- Thus $P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n - i - 1) + n - 1)$

Let $T_l$, $T_r$ be the left and right subtrees of $T$ respectively. Let $n$ be the number of nodes in $T$

Then $P(T) = P(T_l) + P(T_r) + n - 1$. Why?

\[
P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n - i - 1) + n - 1) \]
\[
= \frac{1}{n} \left( \sum_{i=0}^{n-1} P(i) + P(n - i - 1) + \frac{1}{n} \sum_{i=0}^{n-1} n - 1 \right) \]
\[
= \frac{1}{n} \left( \sum_{i=0}^{n-1} P(i) + P(n - i - 1) + \Theta(n) \right) \]
\[
= 2 \frac{1}{n} \left( \sum_{k=1}^{n-1} P(k) + \Theta(n) \right) \]
We have \( P(n) = \frac{2}{n} \left( \sum_{k=1}^{n-1} P(k) \right) + \Theta(n) \).

This is the same recurrence for randomized Quicksort.

In your hw (problem 7-2), you showed that the solution to this recurrence is \( P(n) = O(n \log n) \).

The expected average depth of a node in a randomly built binary tree is \( O(\log n) \).

This implies that operations like search, insert, delete take expected time \( O(\log n) \) for a randomly built binary tree.

\( P(n) \) is the expected total depth of all nodes in a randomly built binary tree with \( n \) nodes.

We've shown that \( P(n) = O(n \log n) \).

There are \( n \) nodes total.

Thus the expected average depth of a node is \( O(\log n) \).

In many cases, data is not inserted randomly into a binary search tree.

I.e. many binary search trees are not “randomly built”.

For example, data might be inserted into the binary search tree in almost sorted order.

Then the BST would not be randomly built, and so the expected average depth of the nodes would not be \( O(\log n) \).