These notes are hand-written, unedited and sketchy. They are primarily used for, and based on my lectures.

If you find any bug, impreciseness, or a rare poor-/mis-interpretation of facts, please let me know. I will be grateful for any additional comments you have that are intended to make the quality of the notes better.

Please note that I will provide my hand-written lecture notes only for a subset of my lectures, not for all lectures. Therefore, it is your responsibility to attend all the lectures, take notes regularly, and ask me and/or the TAs if you have any questions.

Thank you!

--Chittu
**Binary Search Trees**

- Data structure to store information for efficient access: minimum, maximum, other queries
- Manipulations: insert, delete, modify key
- Supports dynamic set operations
- Used as a dictionary or priority queue
- Often augmented to provide new functionality

**BST Property:**
- \( y \in \text{LeftSubtree}(x) \)  
  \( \Rightarrow y \cdot \text{key} \leq x \cdot \text{key} \)
- \( y \in \text{RightSubtree}(x) \)  
  \( \Rightarrow y \cdot \text{key} \geq x \cdot \text{key} \)

**Tree Traversals:**
- **Preorder:** root-left-right
- **Inorder:** left-root-right
- **Postorder:** left-right-root

**Inorder** \( (x) \)
1. If \( x \neq \text{null} \)
2. **Inorder** \( (x \cdot \text{left}) \)
3. Print \( x \)
4. **Inorder** \( (x \cdot \text{right}) \)

**Time Complexity:**
<table>
<thead>
<tr>
<th>Type</th>
<th>Pre/in/post-order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each edge</td>
<td>Traversed exactly twice, ( n ) nodes, ( n ) edges, ( O(n) )</td>
</tr>
<tr>
<td>Alternatively</td>
<td>Solve ( T(n) = T(k) + T(n-k-1) + d = \Omega(n) ) (Substitution)</td>
</tr>
</tbody>
</table>
Querying a BST:

- **SEARCH**(x, k)
  - if x = NULL or x.key = k
    - return x
  - if k < x.key
    - return SEARCH(x.left, k)
  - else return SEARCH(x.right, k)

Exercise: Give iterative version of the algorithm.

let h = height of the BST

- **MIN**(x)
  - Time: O(h)

- **MAX**(x)
  - Time: O(h)

- **SUCCESSOR**(x)
  - Time: O(h)

- **PREDECESSOR**(x)
  - Time: O(h)

**INSERTION:**

```
10
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
RED-BLACK TREES

* Balanced Search Tree:
  - Suppose you build a tree by inserting \(1, 2, 3, 4, 5, 6, 7\) in order.

  Here is the tree:

  \[
  \begin{array}{c}
  1 \\
  2 \\
  3 \\
  4 \\
  5 \\
  6 \\
  7 \\
  \end{array}
  \]

  Height \(h = n-1\)

  *NOT \(O(\log n)\)*

  How can we guarantee \(O(\log n)\) operations?
  
  Idea: Can we make the tree balanced, while doing operations (e.g., insert/delete/access) on it?

  *BECAUSE, a balanced BST will guarantee \(O(\log n)\) operations, e.g., insert, delete and access.*

- Here is a balanced BST with the above set of nodes.

Examples of Balanced Search Trees:

- AVL trees
- B-trees
- 2-3 trees
- Red-Black trees
- Splay trees (self-adjusting)
Red-Black Trees:

- Balanced BST with additional COLOR field in each node.

Red-Black Properties:

1. Every node is either RED or BLACK.
2. The root is BLACK.
3. Every leaf is BLACK.
4. If a node is RED, then both its children are BLACK.
5. A node $x$, path to leaf, has same #BLACK nodes $= \text{Black Height}(x) = bh(x)$.

Example:

Height Lemma: The height of a RB-tree with $n$ external nodes: $h \leq \lceil \log_b(n+1) \rceil$.

Proof:
- Claim 1: Any node of height $h$ has black height $\geq \frac{h}{2}$.
  - Proof: By property 4 above
    - Claim 2: #internal nodes of subtree rooted at $x \geq \frac{bh(x)-1}{2}$.
      - Proof: Induction on height of $x$.
        - $y$ RED $\Rightarrow bh(y) = b$, $y$ BLACK $\Rightarrow bh(y) = b-1$.
        - $\#\text{internal Nodes}(y) \geq \frac{bh(y)}{2} - 1$ (by IH) $\Rightarrow \frac{bh(x)}{2} - 1$.
          $\Rightarrow \#\text{internal Nodes}(x) \geq 2(\frac{bh(x)}{2} - 1) + 1 = 2 - 1$. □

Proof of lemma: $bh(\text{Root}) = h(\text{Root})/2$ (Claim 1)

$\Rightarrow \#\text{internal Nodes} \geq \frac{bh(\text{root})}{2} - 1 \Rightarrow \frac{h(\text{root})}{2} - 1$.

$\Rightarrow h = h(\text{root}) \leq 2 \log(n+1)$. □
Operations on Red-Black Trees:

• **MIN, MAX, SUCCESSOR, PREDECESSOR, SEARCH** takes \( O(h) \) time i.e. \( O(\log n) \) time for red-black trees.

• **INSERT**: What color the new node will have?
  - **RED**: Might violate property 4
  - **BLACK**: Might violate property 5

• **DELETE**:
  - **RED**: We are good.
  - **BLACK**: Can cause the following:
    1. Two RED nodes in a row (property 4 violated)
    2. Violation of property 5
    3. Violation of property 2, when the removed node was the root, and its red child now become the new root.

  - Good news: All these problems can be fixed!
  How? Rotation and Color manipulation.

Rotations:

\[
\begin{align*}
&\text{Zig} \\
&\text{RIGHT-ROTATE}(b) \quad \Theta(1) \text{ time} \\
&\text{LEFT-ROTATE}(c) \\
&\text{Zag}
\end{align*}
\]

Rotation preserves inorder ordering.
Insertion:
- Do usual BST insertion. Let the inserted node be \( x \). Set \( x \text{-color} = \text{RED} \).
- Fix up the color violation by rebalancing (i.e., rotation) on ancestor nodes recursively.

Case I: Uncle node \( D \) is \( \text{RED} \).

If \( C \)'s parent is \( \text{RED} \), then recurse on \( C \) to fix up color violations.

Case II: Uncle node \( D \) is \( \text{BLACK} \) and \( x \) is a right child.

Case III: Uncle node \( D \) is \( \text{BLACK} \) and \( x \) is a left child.

Time: \( O(\log n) \). \( O(1) \) rotations.
Example of INSERT:

![Diagram of a binary search tree with nodes labeled 23, 27, 29, 15, 13, 19, 8, 6, 11, 11, 7, 3, 2, 1, and 0.]

**Case I:**

- Inserted node

**Case II:**

**Case III:**

Deletion:
- Similar to INSERT.
- Time: $O(n)$. $O(1)$ rotations.