3.2 Binary Search Trees

- BSTs
- ordered operations
- deletion
3.2 Binary Search Trees

- BSTs
- ordered operations
- deletion
**Definition.** A BST is a binary tree in symmetric order.

A binary tree is either:
- Empty.
- Two disjoint binary trees (left and right).

**Symmetric order.** Each node has a key, and every node’s key is:
- Larger than all keys in its left subtree.
- Smaller than all keys in its right subtree.
- **leaves** of tree: the nodes with no child nodes
- **height** of tree: the maximum number of **links** from the root to a leaf
- **levels** of tree: the maximum number of **nodes** from the root to a leaf (incl. root and leaf)
- **size** of tree: the number of nodes in the tree
- **depth** of a node: the number of **links** from the root to this node.
→ **leaves** of tree: the nodes with no child nodes
→ **height** of tree: the maximum number of **links** from the root to a leaf
→ **levels** of tree: the maximum number of **nodes** from the root to a leaf (incl. root and leaf)
→ **size** of tree: the number of nodes in the tree
→ **depth** of a node: the number of **links** from the root to this node.

Q: how many leaves in this tree?
Q: what is the height of this tree?
Q: how many levels in this tree?
Q: what is the size of this tree?
Q: what is the depth of ‘H’?
→ **leaves** of tree: the nodes with no child nodes
→ **height** of tree: the maximum number of **links** from the root to a leaf
→ **levels** of tree: the maximum number of **nodes** from the root to a leaf (inl. root and leaf)
→ **size** of tree: the number of nodes in the tree
→ **depth** of a node: the number of **links** from the root to this node.

Q: how many leaves in this tree? 4
Q: what is the height of this tree? 4
Q: how many levels in this tree? 5
Q: what is the size of this tree? 9
Q: what is the depth of ‘H’? 3
Binary search tree demo

**Search.** If less, go left; if greater, go right; if equal, search hit.

成功的搜索结果为 H
Binary search tree demo

Insert. If less, go left; if greater, go right; if null, insert.

insert G
**Java definition.** A BST is a reference to a root Node.

A Node is composed of four fields:
- A Key and a Value.
- A reference to the left and right subtree.

```
private class Node {
    private Key key;
    private Value val;
    private Node left, right;
    public Node(Key key, Value val) {
        this.key = key;
        this.val = val;
    }
}
```

Key and Value are generic types; Key is Comparable.
BST implementation (skeleton)

```java
public class BST<Key extends Comparable<Key>, Value> {
    private Node root;

    private class Node
    { /* see previous slide */  }

    public void put(Key key, Value val)
    { /* see next slides */  }

    public Value get(Key key)
    { /* see next slides */  }

    public void delete(Key key)
    { /* see next slides */  }

    public Iterable<Key> iterator()
    { /* see next slides */  }
}
```
**BST search: Java implementation**

**Get.** Return value corresponding to given key, or null if no such key.

```java
public Value get(Key key) {
    Node x = root;
    while (x != null) {
        int cmp = key.compareTo(x.key);
        if (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
        else if (cmp == 0) return x.val;
    }
    return null;
}
```

**Cost.** Number of compares is equal to 1 + depth of node.
Put. Associate value with key.

Search for key, then two cases:
- Key in tree ⇒ reset value.
- Key not in tree ⇒ add new node.
BST insert: Java implementation

**Put.** Associate value with key.

```java
public void put(Key key, Value val) {
    root = put(root, key, val); }

private Node put(Node x, Key key, Value val) {
    if (x == null) return new Node(key, val);
    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 if (cmp == 0)
        x.val = val;
    return x;
}
```

**Cost.** Number of compares is equal to 1 + depth of node.
Tree shape

- Many BSTs correspond to same set of keys.
- Number of compares for search/insert is equal to $1 + \text{depth of node}$.

Bottom line. Tree shape depends on order of insertion.
BST insertion: random order visualization

Ex. Insert keys in random order.
BSTs: mathematical analysis

Proposition. If \( N \) distinct keys are inserted into a BST in random order, the expected number of compares for a search/insert is \( \sim 2 \ln N \).

Pf. 1–1 correspondence with quicksort partitioning.

Proposition. [Reed, 2003] If \( N \) distinct keys are inserted in random order, expected height of tree is \( \sim 4.311 \ln N \).

How Tall is a Tree?

Bruce Reed
CNRS, Paris, France
reed@moka.ccr.jussieu.fr

ABSTRACT

Let \( H_n \) be the height of a random binary search tree on \( n \) nodes. We show that there exists constants \( \alpha = 4.31107 \ldots \) and \( \beta = 1.95 \ldots \) such that \( \mathbb{E}(H_n) = \alpha \log n - \beta \log \log n + O(1) \). We also show that \( \text{Var}(H_n) = O(1) \).

But... Worst-case height is \( N \).

[ exponentially small chance when keys are inserted in random order ]
### ST implementations: summary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Guarantee</th>
<th>Average Case</th>
<th>Operations on keys</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>(\frac{1}{2}N)</td>
</tr>
<tr>
<td>(unordered 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>BST</td>
<td>(N)</td>
<td>(N)</td>
<td>1.39 (\lg N)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why not shuffle to ensure a (probabilistic) guarantee of 4.311 \(\ln N\)?
Q: Which of the following are Binary Search Trees? Why?