19: Binary Search Trees

Alexandra Papoutsaki
she/her/hers
Binary Search Tree: A binary tree in symmetric order.

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.

Our textbook uses BSTs to implement dictionaries, therefore each node holds a key-value pair. Other implementations hold only a key.
## Differences between heaps and BSTs

<table>
<thead>
<tr>
<th></th>
<th>Heap</th>
<th>BST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used to implement</td>
<td>Priority queues</td>
<td>Dictionaries</td>
</tr>
<tr>
<td>Supported operations</td>
<td>Insert, delete max</td>
<td>insert, search, delete, ordered ops</td>
</tr>
<tr>
<td>What is inserted</td>
<td>Keys</td>
<td>Key-value pairs</td>
</tr>
<tr>
<td>Underlying data str.</td>
<td>(Resizing) array</td>
<td>Linked nodes</td>
</tr>
<tr>
<td>Tree shape</td>
<td>Complete binary tree</td>
<td>Depends on data</td>
</tr>
<tr>
<td>Ordering of keys</td>
<td>Heap-ordered</td>
<td>Symmetrically-ordered</td>
</tr>
<tr>
<td>Duplicate keys</td>
<td>Yes</td>
<td>No*</td>
</tr>
</tbody>
</table>

*: when are BSTs used to implement dictionaries.
BST representation of dictionaries

- We will use an inner class `Node` that is composed by:
  - A Key that is comparable and a Value
  - A reference to the root nodes of the left (smaller keys) and right (larger keys) subtrees.
  - Potentially, the total number of nodes in the subtree that has root this node.
- A BST has a reference to a Node `root`. 
BST and Node implementation

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

    private class Node {
        private Key key;  // sorted by key
        private Value val;  // associated value
        private Node left, right;  // roots of left and right subtrees
        private int size;  // #nodes in subtree rooted at this

        public Node(Key key, Value val, int size) {
            this.key = key;
            this.val = val;
            this.size = size;
        }
    }
}
```
Search for a key

- If less than key in node go to left subtree.
- If greater than key in node go to right subtree.
- If given key and key at examined node are equal, search hit.
- Return value corresponding to given key, or null if no such key.
  - In other implementations, you return the last node you reached.
- Number of compares is equal to the depth of the node + 1.
Search example

Successful (left) and unsuccessful (right) search in a BST
Search - iterative implementation

```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;
}
```
Search - recursive implementation

```java
public Value get(Key key) {
    return get(root, key);
}

private Value get(Node x, Key key) {
    if (x == null)
        return null;
    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;
}
```
Insert

- If less than key in node go left.
- If greater than key in node go right.
- If null, insert.
- If already exists, update value.
- Number of compares is equal to the depth of the node + 1.
Insert example
Insert

- **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, 1);
  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;
  x.size = 1 + size(x.left) + size(x.right);
  return x;
}
3.2 Binary Search Tree Demo
Tree shape

- The same set of keys can result to different BSTs based on their order of insertion.
- Number of compares for search/insert is equal to depth of node +1.
BSTs mathematical analysis

- If $n$ distinct keys are inserted into a BST in random order, the expected number of compares of search/insert is $O(\log n)$.
  - If $n$ distinct keys are inserted into a BST in random order, the expected height of tree is $O(\log n)$. [Reed, 2003].
- Worst case height is $n$ but highly unlikely.
  - Keys would have to come (reversely) sorted!
- All ordered operations in a dictionary implemented with a BST depend on the height of the BST.
Hibbard deletion: Delete node which is a leaf

- Simply delete node.
- Example: delete 52 locates a node which is a leaf and removes it.
Hibbard deletion: Delete node with one child

- Delete node and replace it with its child.
- Example: delete 70 locates a node which has one child and replaces it with the child.
Hibbard deletion: Delete node with two children

- Delete node and replace it with successor (node with smallest of the larger keys). Move successor’s child (if any) where successor was.
- Example: delete 50 locates a node which has two children. Successor is 51.

https://visualgo.net/en/bst
/*
 * Removes the smallest key and associated value from the symbol table.
 *
 * @throws NoSuchElementException if the symbol table is empty
 */

public void delete(Key key) {
    root = delete(root, key);
}

private Node 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);
    else 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; //replace with successor
        x = min(t.right);
        x.right = deleteMin(t.right);
        x.left = t.left;

    }

    x.size = size(x.left) + size(x.right) + 1;
    return x;
}

private Node deleteMin(Node x) {
    if (isEmpty()) throw new NoSuchElementException();
    root = deleteMin(root);
}

private Node deleteMin(Node x) {
    if (x.left == null) return x.right;
    x.left = deleteMin(x.left);
    x.size = size(x.left) + size(x.right) + 1;
    return x;
}

private Node min(Node x) {
    if (x.left == null) return x;
    else return min(x.left);
}
Practice Time

- Delete the node 21 following Hibbard's deletion
Answer

- Delete the node 21 following Hibbard’s deletion
Hibbard’s deletion

- Unsatisfactory solution. If we were to perform many insertions and deletions the BST ends up being not symmetric and skewed to the left.
  - Extremely complicated analysis, but average cost of deletion ends up being $\sqrt{n}$. Let’s simplify things by saying it stays $O(\log n)$.
  - No one has proven that alternating between the predecessor and successor will fix this.

- Hibbard devised the algorithm in 1962. Still no algorithm for efficient deletion in Binary Search Trees!

- Overall, BSTs can have $O(n)$ worst-case for search, insert, and delete. We want to do better (see future lectures).
Lecture 19: Binary Search Trees

- Binary Search Trees
The story so far

- The symbol table/dictionary is a fundamental data type.
- Naive implementations (arrays/linked lists sorted or unsorted) are way too slow.
- Binary search trees work well in the average case, but can grow too tall and imbalanced in the worst case.
- **Question of the day**: How to balance search trees?
Order of growth for symbol table/dictionary operations

<table>
<thead>
<tr>
<th></th>
<th>Worst case</th>
<th></th>
<th></th>
<th>Average case</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
</tr>
<tr>
<td>BST</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\sqrt n$</td>
</tr>
<tr>
<td>Goal</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
</tbody>
</table>
Readings:

- **Recommended Textbook:** Chapters 3.2 (Pages 396-414)
- **Website:**
  - [https://algs4.cs.princeton.edu/32bst/](https://algs4.cs.princeton.edu/32bst/)
- **Visualization:**
  - [https://visualgo.net/en/bst](https://visualgo.net/en/bst)

Practice Problems:

- **In-class worksheet**