CSC148H

Lecture 6

Binary Search Trees
Motivating Binary Search Trees

• Last week we saw examples of where a tree is a more appropriate data structure than a linear structure.
• Sometimes we may use a tree structure even if a linear structure will do.
• Why?
Motivating Binary Search Trees

- For certain tasks, trees can be more efficient.
- One such task is searching.
- Suppose you have a linear structure, and you want to find an element inside this structure.
  - This means going through each element in the structure one at a time until you find the desired element.
Motivating Binary Search Trees

• We could alternatively store elements in a *Binary Search Tree (BST)*.

• A BST is a binary tree in which
  
  – every node has a label (or “key”)
  
  – every node label is
    
    • greater than the labels of all nodes in its left subtree
    
    • less than the labels of all nodes in its right subtree

• (examples on board)
Binary Search Trees

- How do we search for an element in a BST?
Binary Search Trees

- Why is searching for an element in a BST more efficient than (linearly) searching for an element in a list?
- Are there any cases where searching for an element in a BST is no more efficient than (linearly) searching for an element in a list?
Height of a Binary Tree

• What is the maximum height of a binary tree with n nodes?

• What is the minimum height?

• What does a tree with minimum height (on n nodes) look like?
Minimum Height of a Binary Tree

- A **minimum-height binary tree** with \( n \) nodes is a binary tree whose height is no greater than any other binary tree with \( n \) nodes.
Complete Binary Tree

- A **complete binary tree** with $n$ nodes is a binary tree such that every level is full, except possibly the bottom level which is filled in left to right.

- We say that a level $k$ is full if $k = 0$ and the tree is non-empty; or if $k > 0$, level $k-1$ is full, and every node in level $k-1$ has two children.

- Terminology alert: Some sources define a complete binary tree to be one in which all levels are full, and refer to the definition above as an “almost” complete binary tree.
Determining Minimum Height

- A binary tree with height $h$ has at most $2^{h+1} - 1$ nodes. (Prove by induction.) In fact, there exists a binary tree of height $h$ having exactly $2^{h+1} - 1$ nodes.
- A minimum-height binary tree with height $h$ has at least $2^h$ nodes. (Follows from the result above.)
- Let $T$ be a minimum-height binary tree with $n$ nodes and height $h$. By the two points above, $2^h \leq n \leq 2^{h+1} - 1$. Thus $\text{floor}(\log_2 n) = h$. 
Determining Minimum Height

- Previous slide implies that the minimum height of any binary tree on n nodes is floor($\log_2 n$)
Binary Search Trees

- If we can always ensure that a binary search tree is roughly in the shape of a minimal-height binary tree, then searching a binary tree will be much more efficient than linearly searching a list.

- You'll see more on this in later courses: “AVL Trees”, “Red-Black Trees” are BSTs that are balanced.
BST Operations

- So far we've only discussed searching for an element in a BST.
- What do we do once we found it?
- Right now, we can only really report whether it's found or not, but in many applications we may want to store some data with the node.
- Your textbook calls the label of a node its **key**, and the data associated with the node its **value**.
BST Operations

- In general, a BST can be used for mapping keys to data values. (This is much like a Python dictionary).

- Useful operations for such a structure include:
  - `has_key(key)` – test if a node with the given key is present in the tree
  - `get(key)` – get data associated with the key
  - `put(key, val)` – associate `val` with the given key
  - `delete(key)` – remove a node
BST Representation

- How are we going to represent a BST?
- We can use the nodes and references representation that we discussed last week.
- But what if a BST is empty? How do we keep track of this?
BST Representation

- We use a `TreeNode` class to represent a node in the BST.

- We use a `BinarySearchTree` class to represent the tree itself. This class has an attribute that points to the root `TreeNode` of the tree.

- We'll define operations on the `BinarySearchTree` class, but most of them will just delegate to operations in the `TreeNode` class.
BST Representation

• Your textbook keeps track of the parent node of each node in order to implement their version of the delete method.

• There's a way to implement the delete method without requiring the parent node explicitly be kept track of.
BST Operations

- How do we implement get(key)?
- Implementing has_key(key) can be done by delegating to get(key) and checking if the returned value is None
BST Operations

- How do we implement put(key) (i.e., insert a node into the tree)?
- Keep in mind, we want to ensure the BST property is maintained after the node is inserted
- Put operation is similar to the get operation
  - the put operation follows the same path through the tree as a get operation
  - node is added at the end of the path
BST Operations - Deletion

- Removing a node (the delete(key)) operation is the most complex
- Deleting a node with 0 or 1 children is easy, but a node with 2 children is more difficult
- With 0 children, the node is just deleted.
- With 1 child, the child node can just be promoted to the position of the deleted node.
BST Operations - Deletion

- If the node being deleted has two children, then we can't just arbitrarily promote one of the children, otherwise BST property may not be satisfied.

- We can replace the node being deleted with its **successor** to guarantee the BST property still holds.

- Why does replacing the node with its successor ensure the BST property holds?
BST Operations - Deletion

- FACT 1: The successor to a node with two children is guaranteed to have 0 or 1 children.
- FACT 2: The successor of a node with a right subtree is the node with the smallest key in the right subtree.
- FACT 3: The node with the smallest key in a tree is the leftmost node in the tree that does not have a left child (i.e., the leftmost child node).
BST Operations - Deletion

- We now know how to find the successor node of a node with two children (it's the leftmost child in the right subtree).
- Since the successor has 0 or 1 children, we can easily delete it from its current position by promoting its right child if necessary.