Binary Trees/Binary Search Trees

Jonathan Windle

University of East Anglia

J.Windle@uea.ac.uk

May 23, 2017

Overview I

1. Binary Trees
   - Intro
   - Terminology
   - Traversing a Tree

2. Binary Search Trees
   - Intro
   - Insertion
   - Deletion
   - Balanced Binary Search Trees
     - Balance example
     - Case +1 node becomes unbalanced
     - Transform unbalanced to balanced

Intro

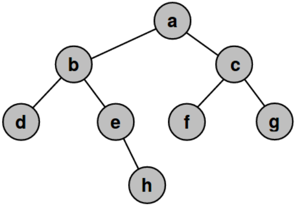
- Tree structures are Non-Linear.
- A Binary Tree T, on a set of elements E is either:
  - empty, or
  - consists of a finite collection of nodes, each containing an element of E, and which contains a particular node called the root of T, with the remaining nodes of T partitioned into to binary tress, called left sub-tree and right sub-tree respectively.

Terminology

- nodes/vertices - contain elements of T.
- parent - Every node except for the root has a unique parent node.
- child - if p is the parent of c thee c is a child of p.
- siblings - two nodes are siblings if they have the same parent node.
- ancestor - Node a is an ancestor of node d if either a is the parent of d or a is the parent of an ancestor of d.
- descendant - Node d is a descendant of a if a is an ancestor of d.
- leaf - a node with no child.
- external - another name for a leaf node. internal - a non-leaf node, i.e. a nde with at least one child.
- level - if n is the root node, then $level(n) = 0$, otherwise $level(n) = level(parent(n)) + 1$.
- height - $height(T) = max_{n \in T} level(n)$ (Height T is also called the level of the tree).

Traversing a Tree

- Preorder:
  - Visit root
  - Visit left sub-tree in preorder
  - Visit right sub-tree in preorder
- Inorder:
  - Visit left sub-tree inorder
  - Visit root
  - Visit righ sub-tree inorder.
- Postorder:
  - Visit left sub-tree in postorder.
  - Visit right sub-tree in postorder.
  - Visit root.



- a,b,d,e,h,c,f,g
- d,b,e,h,a,f,c,g
- d,h,e,b,f,g,c,a

Intro

- A Binary Search Tree, t, is a binary tree, either it is empty or each node in the tree has an associated identifier, or key from a totally ordered set of keys, such that:
    - The keys of all nodes in the left sub-tree of t are less than the key of the root of t
    - The keys of all nodes in the right sub-tree of t are greater than the key of the root t.
    - The left and right sub-trees of t are themselves binary search trees.

Insertion

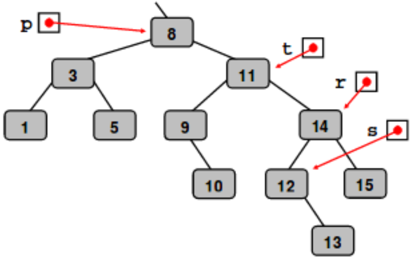
1. Search for the given key:
    - Use two reference variables, t and p.
    - t references the current `TreeNode`
    - p references the parent node of t.
2. If the given key is found, do nothing.
3. Otherwise, p references the `TreeNode` which will be the parent of the incoming node:
    - update p.left or p.right as appropriate.

Deletion

1. Locate node t if present nd the parent node p like in insertion.
2. If found, three cases need to be considered:
   1. Node contains no children:
      - Just set the left or right reference field in p as appropriate to null.
   2. Node has a single child:
      - Set the left or right reference field in p to reference `t.left` or `t.right` as appropriate.
   3. Node to be deleted has two children:
      - The inorder successor of t must take its place.
      - The inorder successor cannot have a left child, right child can be moved up to take its place.
      - See example on next page:

Deletion Example

- Suppose the element 11 is to be deleted from the following bst:
  - t denotes the node (i.e. references the node) containing 11
  - p denotes the parent of t
- r denotes the parent of the node, s, containing the inorder successor of the element to be deleted.

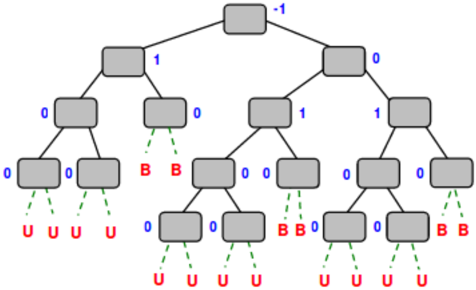


Balanced Binary Search Trees

- Given a BST is balanced, then searching a BST is $O(logn)$.
- A BST, t is height-balanced if either:
  - $t = 0$
  - $t \neq 0$ and:
    - $left(t)$ and $right(t)$ are height balanced or,
    - $|height(left(t)) - height(right(t))| \leq 1$.
- AVL trees are height-balanced BST.
- $balance(t) = height(left(t)) - height(right(t))$

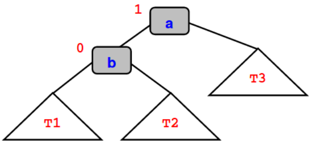
Balance example

- Dashed lines represent where insertions can take place.
- B = Balanced insertion.
- U = Unbalanced insertion.



Case $+1$ node becomes unbalanced

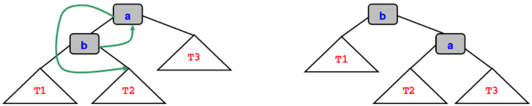
- The balance of a node, a is $+1$.
- Suppose a is the youngest ancestor to become unbalanced when a new node is inserted into the bst.
  - $balance(a) = 1 \implies left(a) \neq 0$.
- Let b be the left child of a.
- Deduce that $balance(b) = 0$.
  - $balance(b) = 0 \implies height(left(b)) = height(right(b)) = n$
  - $(height(b) = n + 1) \wedge (balance(a) = 1) \implies height(right(a)) = n$



T1, T2 and T3 are each trees of height n

Transform unbalanced to balanced

- Right rotation:



- Left rotation:

- Any previously balances BST made unbalanced by a single insertion can be re-balanced by performing either a single rotation or a double rotation using the above.

The End