CS 209
Data Structures and Mathematical Foundations

04 / 19 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions?/Comments?
- Balanced Binary Search Trees

Balanced BSTs

- What does it mean to be balanced
- Why would we like them balanced
- What can we do to balance them?

Balanced BSTs

- The definition of height of a node n_i
  - it is the length of the longest path from n_i to a leaf.
  - length is in terms of number of edges
- Definition of balanced
  - ensure the height of the tree is O(logN) – in other words, height of a balanced tree is NO WORSE than a constant times logN
  - require that for any node that
    - the height of its left subtree is no more than one different than
    - the height of its right subtree
  - the height of an empty (sub)tree is -1
  - Why not require (for all nodes) that the height of left and right subtrees (of any node) to be equal?

Balanced BSTs

- Why not require (for all nodes) that the height of left and right subtrees (of any node) to be equal?
  - because for an arbitrary size n (number of nodes) there may not exist a tree that for all nodes, that its LST and RST have equal heights.
  - Only n's of the form $n = 2^k - 1$ could satisfy a requirement of LST and RST at all nodes having equal heights.
    - for instance, n=4 nodes could never satisfy that
    - neither could n=5,6,9,10,11,12,13,14,17, etc.

Balanced BSTs

- Recall what a BST is. Anyone?

Balanced BSTs

- Is any kind of balancing (size of left subtree vs. size of right subtree) required for BSTs?

Balanced BSTs

- Is any kind of balancing (size of left subtree vs. size of right subtree) required for BSTs?
  - NO
- Example unbalanced tree on board.
- What are the (negative) effects of an unbalanced tree?

Balanced BSTs

- An AVL tree is a BST with the added restriction that
  - for all nodes, the height of its left subtree is at most 1 different than the height of its right subtree. The height of an empty subtree is -1.
  - A node in an AVL tree contains
    - the data item,
    - reference to left child node
    - reference to right child node
    - height of the node

Balanced BSTs

- When we insert a node into an AVL tree we first insert as we would in a BST and update the height information in all the nodes on its path back to the root

- but there are several situations that we can encounter:

  a) the tree retains the properties of an AVL tree (that is, no node has left and right subtrees that differ in height by more than 1)

  b) or some node, alpha, along the path back to the root has subtrees that differ in height by 2 (stop there with the updating of height information) --- there are 4 cases

    - 1) node inserted into LST of the LC of alpha
    - 2) node inserted into RST of the LC of alpha
    - 3) node inserted into LST of the RC of alpha
    - 4) node inserted into RST of the RC of alpha

  Note: only nodes on the path back to the root could possibly be affected by the insertion.

- LST = left subtree, RST = right subtree, LC = left child, RC = right child

Balanced BSTs

- Let's look at b) case 1)

  b) some node, alpha, along the path back to the root has subtrees that differ in height by 2

  - 1) node inserted into LST (left subtree) of the LC (left child) of alpha

- To rebalance the AVL tree in this situation, we do what is called a single rotation.

- Two concrete examples and then the example general situation for case 1 on the board.
  - note: the triangles used to denote subtrees can either all be empty or they have relative heights as shown.

Balanced BSTs

- Let's look at b) case 4)

  b) some node, alpha, along the path back to the root has subtrees that differ in height by 2

  - 4) node inserted into RST (right subtree) of the RC (right child) of alpha

- To rebalance the AVL tree in this situation, again, we perform a single rotation.
- Example general situation for case 4 on the board.

- Case 1 and Case 4 are mirror images of each other. As you are about to see, Case 2 and Case 3 are mirror images of each other.

Balanced BSTs

- Let's look at b) case 2)

  b) some node, alpha, along the path back to the root has subtrees that differ in height by 2

  - 2) node inserted into RST (right subtree) of the LC (left child) of alpha

- To rebalance the AVL tree in this situation, we perform what is called a double rotation.
- Example general situation for case 2 on the board.
- Note: either A,B,C and D are all empty OR
  - One of B or C is 2 levels deeper than D

Balanced BSTs

- Let's look at b) case 3)

  b) some node, alpha, along the path back to the root has subtrees that differ in height by 2

  - 3) node inserted into LST (left subtree) of the RC (right child) of alpha

- To rebalance the AVL tree in this situation, we again perform a double rotation.

Balanced BSTs

- Now, let's go through an example.
- Insert integer items into an AVL tree (initially empty) one at a time in order from 1 through 7. Then 16 through 10 and then 8 then 9.
- Recall that we insert like in a BST, but after that we need to check if it is still AVL or not by travelling up towards the root and checking the heights of the LST and RST along the way. If we get to a position where a node's LST and RST differ in height by more than 1, then that is alpha.
- We'll mark down the following after each insertion (and perform rotations if necessary)
  - we'll note when no rotation is necessary, still AVL
  - and we'll note when a rotation is necessary and state what node alpha is and which case it is (1, 2, 3 or 4)

Efficiency

- AVL trees cause the height of the tree to be no more than 1.44*log(n)
- So, search is O(log(n))
- Insert initially then also takes at worst log(n) and then we need to go up the tree setting heights and verifying heights of LST vs. RST. If we go all the way up the tree that is another log(n) amount of work. The rebalancing takes a constant amount of time.
- So, insert is also O(log(n))

Other Balanced Trees

- Red/Black trees are another form of balanced binary search tree, that we will not discuss.
- There are others too.