COMP 352

Tutorial 9: Binary Search and AVL Trees

Outline

- Binary Search Trees
  - Definitions
  - Search and Update Algorithms
  - Performance
- AVL Trees
  - Definitions
  - Update Operations / Rotations
- Problem Solving

Binary Search Trees - Definitions (1)

- A ***binary search tree*** is a data structure for storing the entries of a dictionary.
- A ***binary search tree*** is a binary tree T such that each internal node v of T stores an entry (k,x) such that:
  - Keys stored at nodes in the left subtree of v are less than or equal to k.
  - Keys stored at nodes in the right subtree of v are greater than or equal to k.

Binary Search Trees - Definitions (2)

- Entries in a ***binary search tree*** are stored in internal nodes; empty external nodes are added to form a **proper** binary tree.
- An **inorder traversal** of nodes in a ***binary search tree*** lists the keys in increasing order.

BINARY SEARCH TREES - SEARCHING ALGORITHM

```
Algorithm TreeSearch(p, k):
    if p is external then
        return p                                  {unsuccessful search}
    else if k == key(p) then
        return p                                  {successful search}
    else if k < key(p) then
        return TreeSearch(left(p), k)             {recur on left subtree}
    else {we know that k > key(p)}
        return TreeSearch(right(p), k)            {recur on right subtree}
```

- This algorithm has *O(h)* complexity, where *h* is the height of the binary search tree.
- In the worst case, it is linear because the height will be equal to *n*.

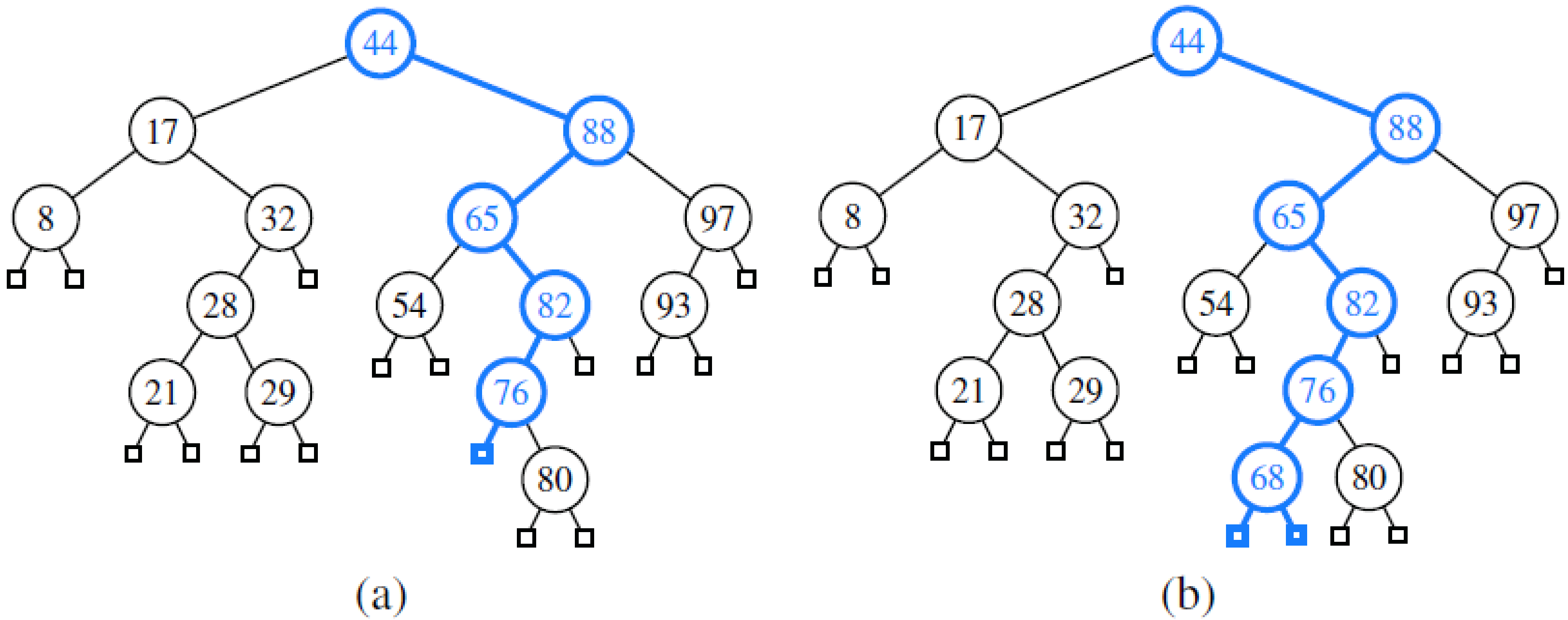
Binary Search Trees - Update Operations (1)

- *insertAtExternal(v,e)*: Insert the element e at the external node v, and expand v to be internal, having new (empty) external node children; an error occurs if v is an internal node.
- Recursive Algorithm for insertion:

Algorithm TreeInsert(k, v):
  Input: A search key k to be associated with value v
  p = TreeSearch(root(), k)
  if k == key(p) **then**
    Change p's value to (v)
  else
    expandExternal(p, (k, v))

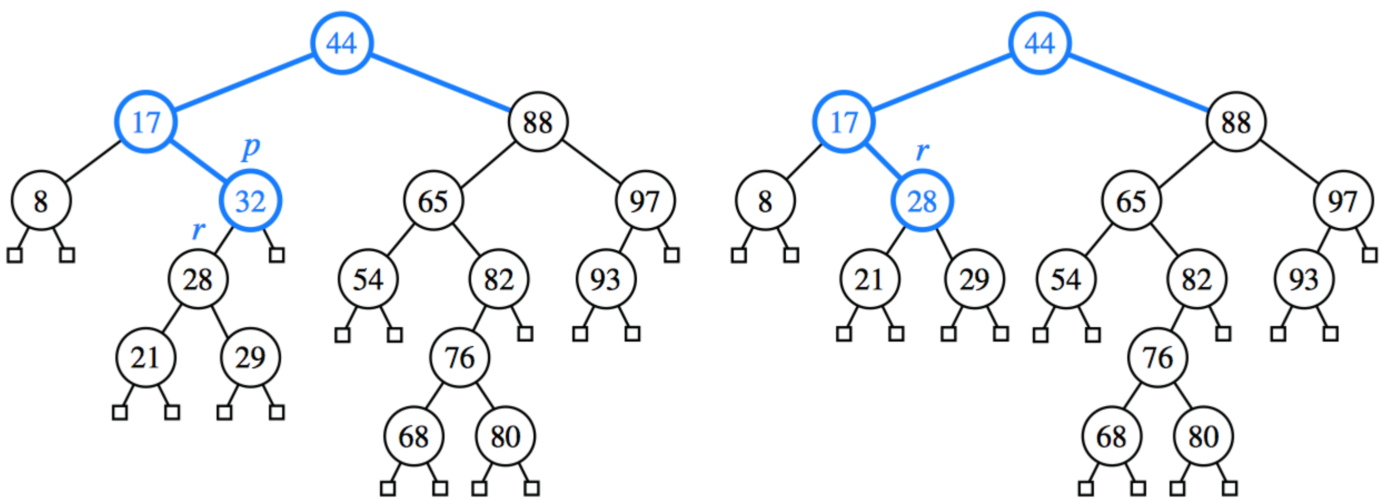
Binary Search Trees - Update Operations (2)

- Insertion of an entry with key 68:



Binary Search Trees - Update Operations (3)

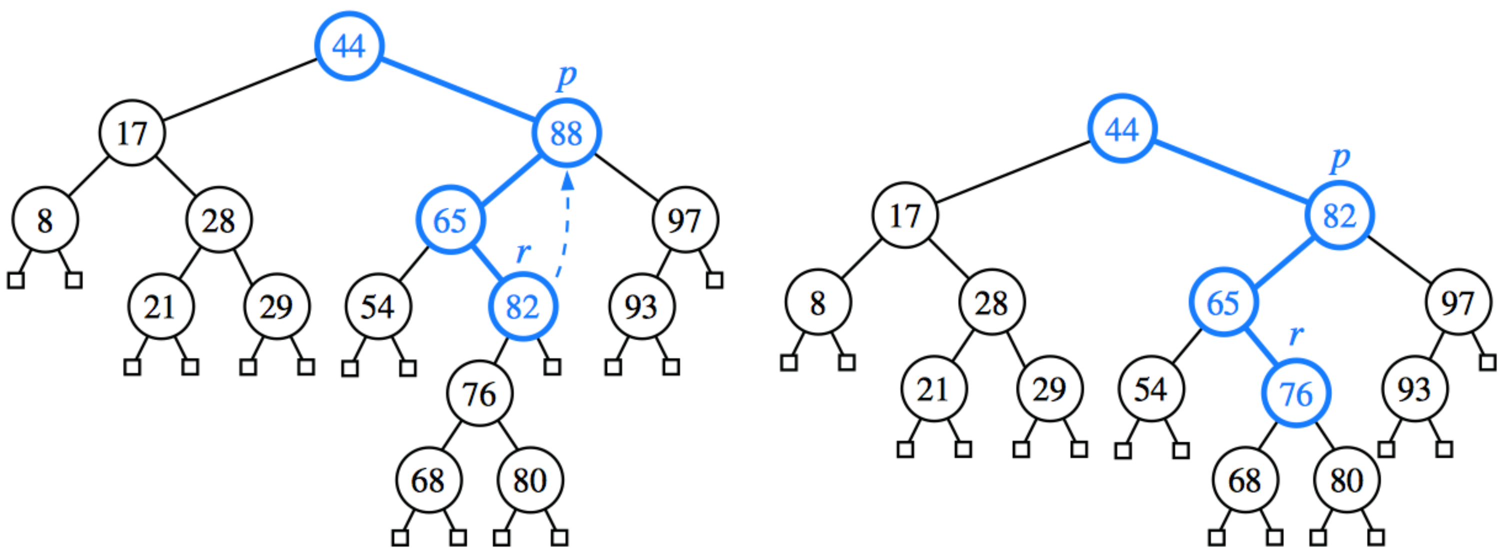
- *removeExternal(v)*: Remove an external node v and its parent, replacing v's parent with v's sibling; an error occurs if v is not external.
- Removal with entry to be removed having an external child: *remove(32)*



(a) (b)

Binary Search Trees - Update Operations (4)

Removal with entry to be removed having both its children internal: *remove(88)*



(a) (b)

Binary Search Trees - Performance

- The find, insert, and remove methods run in $O(h)$ time, where h is the height of T.
- A binary search tree T is an efficient implementation of a dictionary with n entries only if the height of T is small
- In the worst case, T has height n

AVL Trees - Definitions

- An ***AVL Tree*** presents a more efficient way to implement a dictionary.

- It maintains a logarithmic-time performance, due to the ***Height-Balance Property***:
  - For every internal node v of the tree, the heights of the children of v differ by at most 1.

- If this property is violated after insertion/removal from an AVL tree, a re-struction is needed.

AVL Trees - Rotations (1)

- Rotation Algorithm:
  - z is the first unbalanced node encountered while going upwards, y is the child of z with a higher height, and x the child of y with a higher height

Algorithm restructure(x):

Input: A position x of a binary search tree T that has both a parent y and a grandparent z

Output: Tree T after a trinode restructuring (which corresponds to a single or double rotation) involving positions x, y, and z

1: Let (a, b, c) be a left-to-right (inorder) listing of the positions x, y, and z, and let (T_1, T_2, T_3, T_4) be a left-to-right (inorder) listing of the four subtrees of x, y, and z not rooted at x, y, or z.

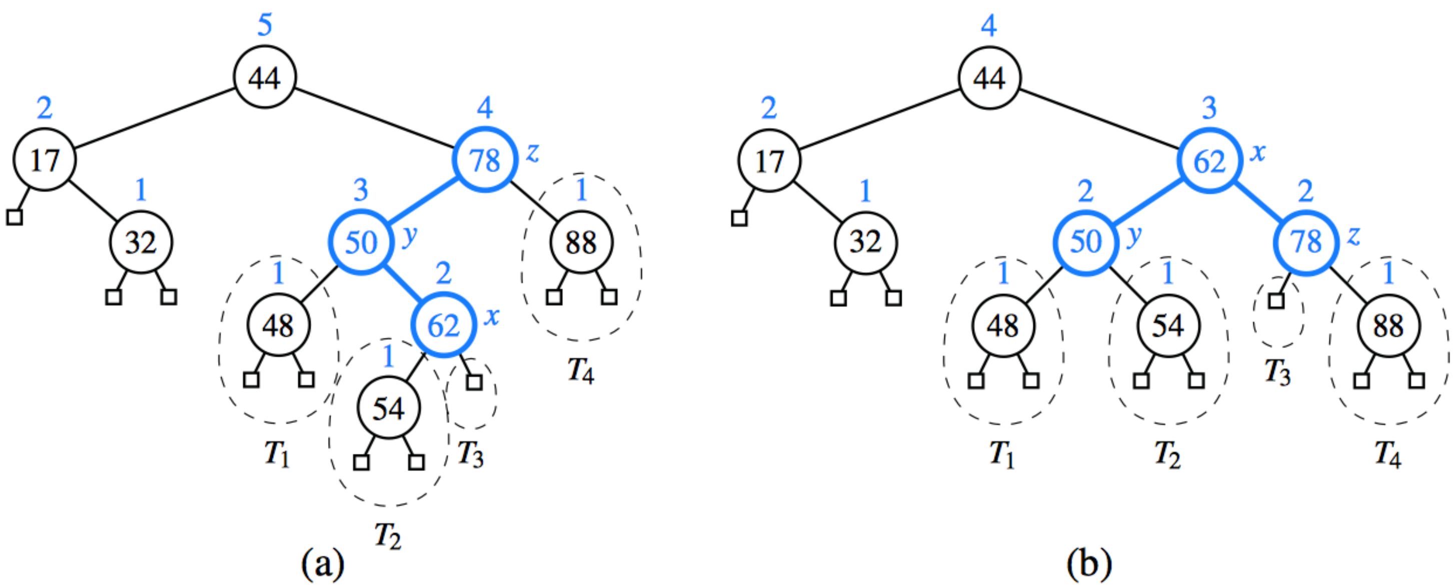
2: Replace the subtree rooted at z with a new subtree rooted at b.

3: Let a be the left child of b and let T_1 and T_2 be the left and right subtrees of a, respectively.

4: Let c be the right child of b and let T_3 and T_4 be the left and right subtrees of c, respectively.

AVL Trees – Rotations (2)

- Representation of a tree before and after a rotation:



  - If $b = y$, the trinode restructuring method is called a single rotation.
  - If $b = x$, the trinode restructuring method is called a double rotation.

Problem Solving - R – 10.1 Question 1

- We defined a BST so that keys equal to a node's key can be in either the left or right subtree of the node. Suppose we change the definition so that we restrict equal keys to the right subtree.
- What must a subtree of a binary search tree containing only equal keys look like in this case?

Problem Solving - R – 10.6 Question 2

- Dr. Amongus claims that the order in which a fixed set of entries is inserted into a binary search tree does not matter— the same tree results every time. Give a small example that proves he is wrong.

Problem Solving - R – 10.3 Question 3

- How many different BST's can store the keys *{1,2,3}*?

Problem Solving – Question 4

- Draw a Binary Search Tree that initially is empty and shows the result of the tree after inserting the following keys (from left to right):
  - key ={ 30, 40, 24, 58, 48, 26, 11, 13 }

Problem Solving - Question 5

- Build an AVL tree with the following keys:

  {3, 2, 1, 4 ,5 ,6 ,7 ,16 ,15}

Problem Solving - Question 6

- Using the AVL tree obtained in Question 5, delete nodes with values 1 and 3, and draw the final orientation of the tree.