Balancing Search Trees

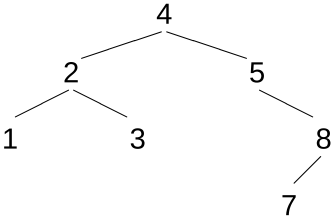
MCS 360 Lecture 33
Introduction to Data Structures
Jan Verschelde, 8 November 2010

Balancing Search Trees

Binary Search Trees

Consider 4, 5, 2, 3, 8, 1, 7 (recall lecture 24).

Insert the numbers in a tree:



Rules to insert x at node N:

- if N is empty, then put x in N
- if $x < N$, insert x to the left of N
- if $x \geq N$, insert x to the right of N

Recursive printing: left, node, right sorts the sequence.

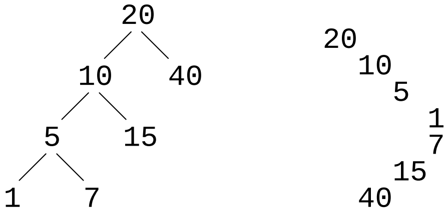
an unbalanced tree

Inserting $0, 1, 2, \ldots, 9$.

```
depth of tree : 9
0
  1
    2
      3
        4
          5
            6
              7
                8
                  9
```

shaping binary search trees

To make a binary search tree with given shape:



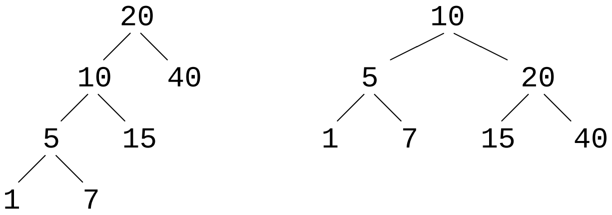
Insert numbers in a particular order: 20, 40, 10, 5, 15, 1, 7.

The tree is unbalanced because the depth of the left tree is two, while the depth of the right three is zero.

Balancing Search Trees

Right Rotation

To balance the binary search tree tree,
we do a right rotate around the root:

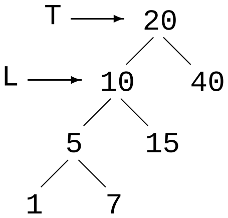


Observe the effects of a right rotation:

- left tree has become the new root;
- old root is now at the right of new root;
- left tree of old root is now the right tree of the left tree of old root.

Right Rotation in 3 Steps

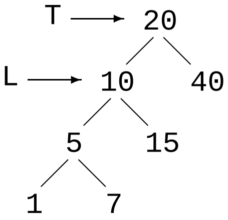
Tree with root node T:

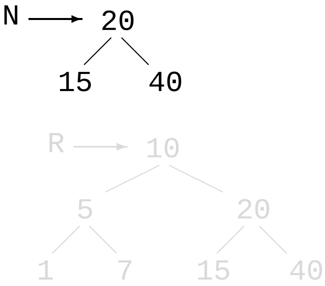


1. Label left of T with L.
2. New tree N has right of T as right and as left the right of L.
3. Result R has L as root, the tree N as right, and the left of L as left.

Right Rotation in 3 Steps

Tree with root node T:

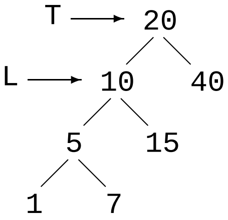


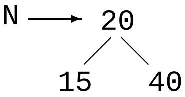


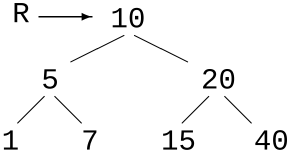
1. Label left of T with L.
2. New tree N has right of T as right and as left the right of L.
3. Result R has L as root, the tree N as right, and the left of L as left.

Right Rotation in 3 Steps

Tree with root node T:



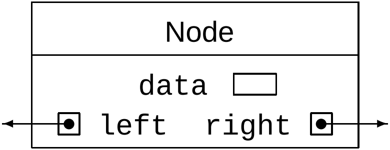




1. Label left of T with L.
2. New tree N has right of T as right and as left the right of L.
3. Result R has L as root, the tree N as right, and the left of L as left.

Balancing Search Trees

a node struct




```
struct Node
{
   int data;    // numbers stored at node in tree
   Node *left;  // pointer to left branch of tree
   Node *right; // pointer to right branch of tree

   Node(const int& item, Node* left_ptr = NULL,
                         Node* right_ptr = NULL) :
      data(item),
      left(left_ptr), right(right_ptr) {}
```

a class Tree

```
#include "mcs360_integer_tree_node.h"

namespace mcs360_integer_tree
{
  class Tree
  {
     private:
       Node *root;  // data member

     public:
       Tree(const int& item,
            const Tree& left = Tree(),
            const Tree& right = Tree() ) :
     root(new Node(item,left.root,right.root)) {}
       Tree get_left() const;
       Tree get_right() const;
       void insert(int item);
```

function `rotate_right`

Prototype of function in client of class Tree:

```
Tree rotate_right ( Tree t );

// Returns the tree rotated to the right
// around its root.

// Precondition: left of t is not null.
```

definition of rotate_right

```
Tree rotate_right ( Tree t )
{
   Tree left = t.get_left();

   Tree new_t = Tree(t.get_data(),
      left.get_right(),t.get_right());

   Tree R = Tree(left.get_data(),
                 left.get_left(),new_t);

   return R;
}
```

Balancing Search Trees

AVL Trees

Define the balance of a tree as

$$\text{balance} = \text{depth(right tree)} - \text{depth(left tree)}.$$

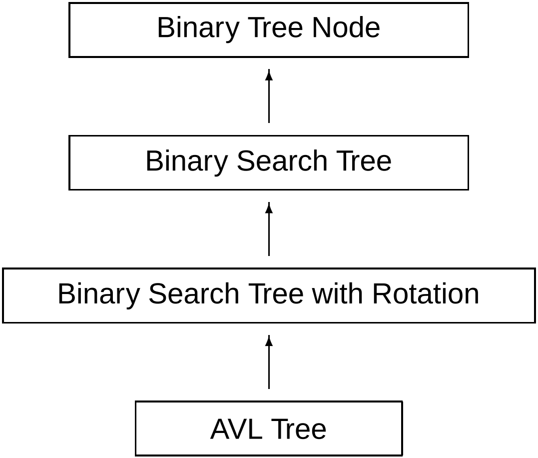
Note: depth (chapter 8) = height (chapter 11).

G.M. Adel'son-Vel'skiî and E.M Landis published an algorithm to maintain the balance of a binary search tree.

If balance gets out of range $-1 \ldots +1$,
the subtree is rotated to bring into balance.

Their approach is known as *AVL trees*.

a Class Hierarchy

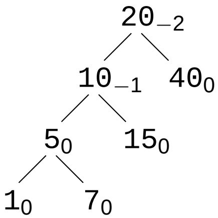


computing the balance

Recall the definition:

$$\text{balance} = \text{depth(right tree)} - \text{depth(left tree)}.$$

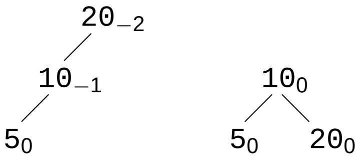
At every node we compute the balance, displayed as subscript:



balancing a left-left tree

The tree below is *left heavy* as the balance is -2.

We also say that this is a *left-left tree*.



Executing a right rotation balances the tree.

Balancing Search Trees

critically unbalanced trees

A tree is *critically unbalanced* if its balance is -2 or $+2$.

20_{-2}
10_{-1}
5_0

a left-left tree

20_{-2}
5_{+1}
10_0

a left-right tree

5_{+2}
10_{+1}
20_0

a right-right tree

5_{+2}
20_{-1}
10_0

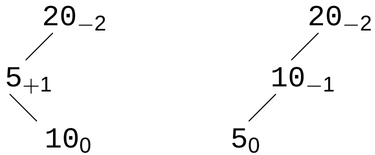
a right-left tree

balancing trees of mixed kind

A right rotation balances a left-left tree
and a left rotation balances a right-right tree.

Balancing a left-right tree happens in two stages:

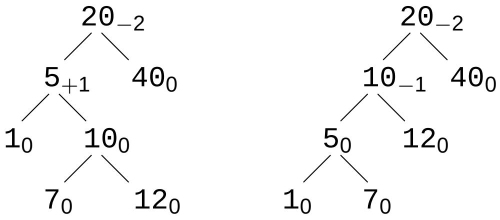
1. rotate left-right tree to left-left tree:



2. apply right rotation to left-left tree:

rotating a left-right tree

We rotate the left-right tree to a left-left tree:

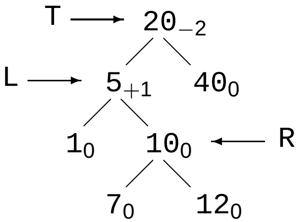


Observe the effects of the rotation:

- the data at the left node of the new tree (10) is swapped with the data of the left of the old tree (5);
- the right of the left of the new tree (12) is the right of the right of the left of the old tree;
- the right of the left of the left of the new tree (7) is the left of the right of the left of the old tree.

rotating to left-left tree in 4 steps

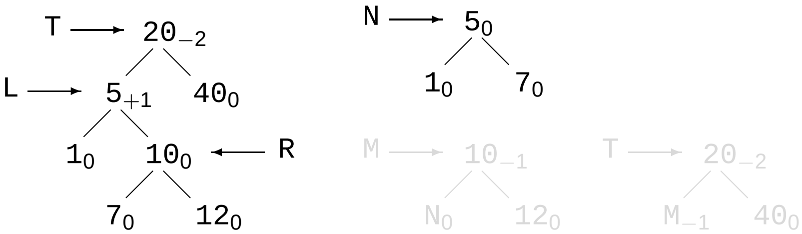
Tree with root node T:



1. Label left of T with L and right of L with R.
2. Tree N has as its left the left of L, as its right the left of R.
3. Tree M has as its left N, as its right the right of R.
4. Return the tree with its left M and its right the right of T.

rotating to left-left tree in 4 steps

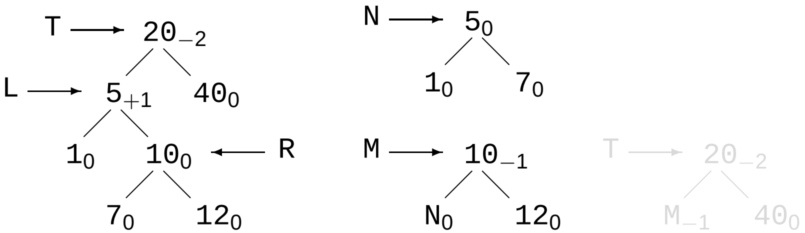
Tree with root node T:



1. Label left of T with L and right of L with R.
2. Tree N has as its left the left of L, as its right the left of R.
3. Tree M has as its left N, as its right the right of R.
4. Return the tree with its left M and its right the right of T.

rotating to left-left tree in 4 steps

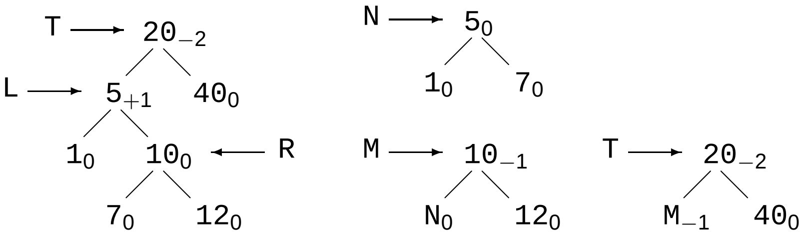
Tree with root node T:



1. Label left of T with L and right of L with R.
2. Tree N has as its left the left of L, as its right the left of R.
3. Tree M has as its left N, as its right the right of R.
4. Return the tree with its left M and its right the right of T.

rotating to left-left tree in 4 steps

Tree with root node `T`:



1. Label left of `T` with `L` and right of `L` with `R`.
2. Tree `N` has as its left the left of `L`, as its right the left of `R`.
3. Tree `M` has as its left `N`, as its right the right of `R`.
4. Return the tree with its left `M` and its right the right of `T`.

Balancing Search Trees

prototype of the function

```
Tree rotate_to_left_left ( Tree t );

// Returns the tree rotated to a left-left tree.

// Preconditions:
//   (1) left of t is not null; and
//   (2) right of left of t is not null.
```

Test: insert 20, 5, 1, 10, 7, 12 to binary search tree.

definition of the function

```
Tree rotate_to_left_left ( Tree t )
{
   Tree left = t.get_left();
   Tree right = left.get_right();

   Tree new_left = Tree(left.get_data(),
      left.get_left(),right.get_left());

   Tree new_right = Tree(right.get_data(),
      new_left,right.get_right());

   Tree R = Tree(t.get_data(),
      new_right,t.get_right());

   return R;
}
```

rebalancing search trees

After each insert (or removal):

- check the balance of the tree,
- and if critically unbalanced, rebalance.

Performance of the AVL tree:

- worst case: $1.44 \times \log_2(n)$,
- on average: $\log_2(n) + 0.25$ comparisons needed.

$\rightarrow$ close to complete binary search tree.

Summary + Assignments

Started chapter 11 on balancing binary search trees.

Assignments:

1. Formulate the algorithm for left rotation and illustrate with an example.
2. Write code for left rotation around the root and give the output of a test to show that it works.
3. Formulate the algorithm to rotate a right-left tree to a right-right tree and illustrate with an example.
4. Write code for the rotation of the previous exercise and give the output of a test to show that it works.

Homework due Monday 15 November, at noon:
#2, 3 of L-27; #1, 2 of L-28; and #2 of L-29.
Lab session of tomorrow, Tue 9 Nov, in EPASWL270!