Balancing Search Trees

1. Tree Balance and Rotation
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. AVL Trees
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree

MCS 360 Lecture 33
Introduction to Data Structures
Jan Verschelde, 11 November 2019
Balancing Search Trees

1. Tree Balance and Rotation
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. AVL Trees
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree
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, . . . , 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:

```
    20
   / \
  10  40
 /     \
5      15
/   \
1    7
```

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

\[
\text{depth}(T) = \begin{cases} 
0, & \text{if } T \text{ is empty,} \\
1 + \max(\text{depth(left}(T)), \text{depth(right}(T))), & \text{otherwise.}
\end{cases}
\]

The tree is unbalanced because the depth of the left tree is two, while the depth of the right tree is zero.
Balancing Search Trees

1. **Tree Balance and Rotation**
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. **AVL Trees**
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree
Right Rotation

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

```
          20
         /  \
        10   40
       /    /\    \\
      5    15 15
     /  \     \  \
    1    7     7
```

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.
Balancing Search Trees

1. Tree Balance and Rotation
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. AVL Trees
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree
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:

```c
Tree rotate_right ( Tree t );
```

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

// Precondition: left of t is not null.
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

1. Tree Balance and Rotation
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. AVL Trees
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. Code for rotation
   - from left-right to left-left tree
Define the balance of a tree as

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

G.M. Adel’son-Vel’skiî and E.M Landis published in 1962 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

Binary Tree Node

Binary Search Tree

Binary Search Tree with Rotation

AVL Tree
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*.

```
  20
 /   \
10   10
 /   / \
5    5   20
```

Executing a right rotation balances the tree.
Balancing Search Trees

1. **Tree Balance and Rotation**
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. **AVL Trees**
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree
A tree is critically unbalanced if its balance is $-2$ or $+2$.

\[ \begin{align*}
20 & \quad -2 \\
10 & \quad -1 \\
5 & \\
\end{align*} \quad \text{a left-left tree} \]

\[ \begin{align*}
20 & \quad -2 \\
5 & \quad +1 \\
10 & \\
\end{align*} \quad \text{a left-right tree} \]

\[ \begin{align*}
5 & \quad +2 \\
10 & \quad +1 \\
20 & \\
\end{align*} \quad \text{a right-right tree} \]

\[ \begin{align*}
5 & \quad +2 \\
20 & \quad -1 \\
10 & \\
\end{align*} \quad \text{a right-left tree} \]
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:

```
  20_2
 /     /
5_1   10_0
```

2. apply right rotation to left-left tree:

```
  20_2
 /     /
10_1  5_0
```

   20_2
 /     /
5_0   10_0
```

   5_0
```
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.
Balancing Search Trees

1. Tree Balance and Rotation
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. AVL Trees
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree
Insert the numbers 50, 68, 98, 25, 15, 43, 47, 45 into an AVL tree.

After inserting 50, 68, and 98, we get a right-right tree:

```
   50
   |
   68
   |
   98
```

We rotate the right-right tree:

```
   50
   |
   68
   |
  98
```

```
   50
   |
   68
   |
  98
```

```
   50
   |
   68
   |
  98
```
insert 25, 15

We insert 25 and 15 into

\[
\begin{array}{c}
68_0 \\
\quad \downarrow \\
50_0 \quad 98_0 \\
\end{array}
\]

and we get a left-left tree in the middle:

\[
\begin{array}{c}
68_{-1} \\
\quad \downarrow \\
50_{-1} \quad 98_0 \\
\end{array}
\begin{array}{c}
68_{-2} \\
\quad \downarrow \\
50_{-2} \quad 98_0 \\
\end{array}
\begin{array}{c}
50_0 \\
\quad \downarrow \\
25_{-1} \quad 68_{+1} \\
\end{array}
\]

\[
\begin{array}{c}
25_0 \\
\quad \downarrow \\
15_0 \\
\end{array}
\]

and the left-left tree in the middle is rotated.
insert 43 and 47

```
      50₀
   25₋₁  68₊₁
  15₀  98₀
```

```
      50₀
   25₊₁  68₊₁
  15₀  43₀  98₀
```

```
      50₋₁
   25₊₁  68₊₁
  15₀  43₊₁  98₀
```

```
insert 45

The right-left tree is first rotated to a right-right tree. Then the right-right tree is rotated.
Balancing Search Trees

1. Tree Balance and Rotation
   - binary search trees
   - right rotation of a tree around a node
   - code for right rotation

2. AVL Trees
   - self-balancing search trees
   - four kinds of critically unbalanced trees
   - insert a sequence of numbers into an AVL tree

3. code for rotation
   - from left-right to left-left tree
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.
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.
Started chapter 11 on balancing binary search trees.

Exercises:

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.