MERRIMACK COLLEGE

CSC 6013
Week 8

Transform-and-Conquer Algorithms

Algorithms and Discrete Structures - Dr. Paulo Fernandes

Presentation Agenda

Week 8

- Transform-and-Conquer
  a. Meet the family
  b. Basic Principles, Recursion and Iteration
- Examples
  a. Element Uniqueness in an Array
  b. Computing the Mode
  c. Balancing Binary Search Trees
- Course wrap-up
  a. The Final Exam
  b. Next Steps

MERRIMACK COLLEGE

Transform-and-Conquer Algorithms

Solve the problem by changing the problem size.

Transform-and-conquer are usually very effective, but the less known among the algorithms of the ...-and-conquer family.

- **Meet the family**
  - Decrease-and-Conquer
  - Divide-and-Conquer
  - Transform-and-Conquer
- Transform-and-Conquer
  - Basic Principles
  - Recursion and Iteration

Meet the ...-and-conquer family

While in military, politics, management, etc., the usage of divide-and-conquer became extremely popular, and effective, in algorithms we have a subdivision of the algorithm techniques:

- **Divide-and-conquer**
  - When the original problem is broken into complementary parts;
- **Decrease-and-conquer**
  - When at each time the problem becomes smaller, but not into complementary problems;

- **Transform-and-conquer**
  - When the problem not always becomes smaller.

This division is acknowledged by great authors (as our textbook ones: Cormen, Leiserson, Rivest, and Stein), but it is not unanimous.

Transform-and-conquer

- Modifying the problem to a more amenable form to a particular solution:
  - The **transform part**!
  - It can be done by:
    - Instance Simplification
    - Representation Change
    - Problem Reduction

- The transformed problem is solved yielding a solution to the original problem
  - The **conquer part**!

Video: [Introduction to Transform and Conquer].

Transform-and-conquer

This is a representation change example.

Converting **12,345** into **54,321**

A simple example, reversing the digits of an Integer.

- The problem to reverse an Integer with a numerical representation is hard;
- The solution transforms the numerical into a string representation

```python
def reverse_integer(num):
    # 1. Transform: Convert the integer to a string
    num_str = str(num)
    # 2. Conquer: Reverse the string
    reversed_str = num_str[::-1]
    # 3. Transform: Convert the reversed string back to an integer
    reversed_num = int(reversed_str)
    return reversed_num

number = 12345
reversed_number = reverse_integer(number)
print(reversed_number)  # Output: 54321
```

Note that there are two transform parts (#1 and #3) and one conquer part (#2).

Transform-and-Conquer Algorithms

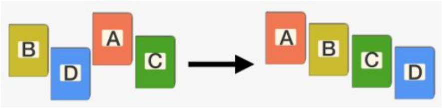
Solve the problem by changing the problem size.

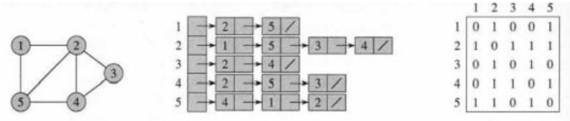
Transform-and-conquer are usually very effective, but the less known among the algorithms of the ...-and-conquer family.

- Meet the family
  - Decrease-and-Conquer
  - Divide-and-Conquer
  - Transform-and-Conquer
- **Transform-and-Conquer**
  - Basic Principles
  - Recursion and Iteration

Transform-and-conquer variants

- Instance Simplification
  - Transform to a simpler or more convenient instance of the same problem;
  - For example, the search on an array can be easier if the array is sorted first;





- Representation Change
  - It remains the same instance, but in a different representation;
  - For example, a graph can change from an adjacency matrix representation into an adjacency list representation;
- Problem Reduction
  - Transform to an instance of a different problem for which a solution is simpler or already available.

Let $n \in \mathbb{Z}_{>0}, k \in \mathbb{Z}$.

Then:

$$\binom{n}{k} = \binom{n}{n-k}$$

where $\binom{n}{k}$ is a binomial coefficient.

MERRIMACK COLLEGE

Text: Transform-and-Conquer.

Transform-and-Conquer

Transform-and-Conquer, unlike Decrease and Divide-and-Conquer, is not particularly related to recursion paradigm.

```python
1  def decimal_to_binary_iterative(n):
2      binary_string = ""
3      while n > 0:
4          remainder = n % 2
5          binary_string = str(remainder) + binary_string
6          n //= 2
7      return binary_string
8
9  decimal_num = 13
10 binary_num = decimal_to_binary_iterative(decimal_num)
11 print(binary_num)  # Output: 1101
```

- Transform: compute the remainder of the decimal number by 2 (#4);
- Transform: concatenate the remainder to the binary string (#5);
- Conquer: divide the decimal number by 2 (#6).

```python
1  def decimal_to_binary_recursive(n):
2      if n == 0:
3          return "0"
4      else:
5          remainder = n % 2
6          binary_string = decimal_to_binary_recursive(n // 2)
7          return binary_string + str(remainder)
8
9  decimal_num = 13
10 binary_num = decimal_to_binary_recursive(decimal_num)
11 print(binary_num)  # Output: 01101
```

- Transform: compute the remainder of the decimal number by 2 (#5);
- Conquer: Recursively convert the quotient to binary (#6);
- Transform: Concatenate the result of the recursive call to the remainder (#7).

This is a representation change example.

Note that the output is not exactly the same for both codes.

Converting
13 to **01101**
13 to **1101**

Transform-and-Conquer Algorithms Examples

The examples that will be seen are:

- **Element Uniqueness in an Array**
- Computing the Mode
- Balancing Binary Search Trees

Example 1 - Element Uniqueness in an Array

- Given a set of elements stored in an unsorted array ***A***;
- It returns True if every element in ***A*** is unique:
  - a_i is different of a_j for all i different of j;
- A brute force solution has $O(n^2)$:

```python
def unique(a):
    ans = True
    for i in range(len(a)):
        for j in range(len(a)):
            if (i != j) and (a[i] == a[j]):
                ans = False
                break
        if (not ans):
            break
    return ans
```

- What if the array ***A*** is sorted?
  - The non unique elements will be contiguous;
- How much it takes to sort ***A***?
  - Mergesort and Quicksort are usually:
    - $O(n \log n)$

Wikipedia: Element distinctness problem.

Example 1 - Element Uniqueness in an Array

- What if the array ***A*** is sorted?
    - The non unique elements will be contiguous.

O(n log n) — `a.sort()`

O(n) — `for` loop

```python
def uniqueSorted(a):
    a.sort()
    for i in range(len(a)):
        if (a[i-1] == a[i]):
            return False
    return True
```

O(n log n) + ***O(n)*** = ***O(n log n)***

which is better than ***O(n²)*** (brute force)

This is an instance simplification example.

- How much it takes to sort ***A***?
    - Mergesort and Quicksort are usually:
        - ***O(n log n)***
    - Actually, Python list method ***sort*** implements a version of Quicksort.

O(n log n) - Log Linear Algorithm

Transform-and-Conquer Algorithms Examples

The examples that will be seen are:

- Element Uniqueness in an Array
- **Computing the Mode**
- Balancing Binary Search Trees

Example 2 - Computing the Mode

This is an instance simplification example.

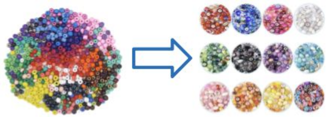
- Given a set of elements stored in an array ***A***;
- It returns the Mode (the more frequent value).
- A brute force solution has $\boldsymbol{O(n^2)}$:

```python
def mode(a):
    maxValue, maxCount = 0, 0
    for i in range(len(a)):
        count = 0
        for j in range(len(a)):
            if (a[i] == a[j]):
                count += 1
        if (count > maxCount):
            maxCount = count
            maxValue = a[i]
    return maxValue
```

- What if the array ***A*** is sorted?
    - Search for the longest sequence of the same number to compute the Mode;
- How much it takes to sort ***A***?
    - $\boldsymbol{O(n \log n)}$

Wikipedia: Mode (statistics).

Example 2 - Computing the Mode



Search for the longest sequence of the same number to compute the Mode in a sorted array ***A***:

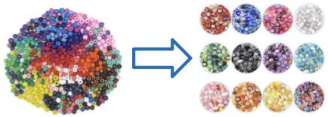
- Start from the first element towards the last (keep track of ***mode_count*** and ***mode*** value);
  - If the next element has the same value:
    - increase the counting;
  - Else:
    - check if the ***current_count*** is larger;
- By the end get the value with the highest count (comparing the last sequence too).

```python
def computeMode(A):
    A.sort()
    mode = None
    mode_count = 0
    current_value = A[0]
    current_count = 1
    for i in range(1, len(A)):
        if A[i] == current_value:
            current_count += 1
        else:
            if current_count > mode_count:
                mode = current_value
                mode_count = current_count
            current_value = A[i]
            current_count = 1
    if current_count > mode_count:
        mode = current_value
    return mode

A = [1, 2, 3, 2, 2, 4, 5, 5, 5]
print(computeMode(A))
```


Several Transform-and-Conquer using instance simplification perform pre-sorting.

Example 2 - Computing the Mode



Similarly to the Element Uniqueness in an Array example, here we have the costs as:

- An efficient sorting algorithm:
  - ***O(n log n)***
- The passage by all elements of A doing the counting of unique elements:
  - ***O(n)***
- The resulting complexity in time will be:
  - ***O(n log n)*** + ***O(n)*** = ***O(n log n)***
    - which is better than ***O(n^2)*** (brute force)

O(n log n)

O(n)

```python
def computeMode(A):
    A.sort()
    mode = None
    mode_count = 0
    current_value = A[0]
    current_count = 1
    for i in range(1, len(A)):
        if A[i] == current_value:
            current_count += 1
        else:
            if current_count > mode_count:
                mode = current_value
                mode_count = current_count
            current_value = A[i]
            current_count = 1
    if current_count > mode_count:
        mode = current_value
    return mode

A = [1, 2, 3, 2, 2, 4, 5, 5, 5]
print(computeMode(A))
```


O(n log n) - Log Linear Algorithm

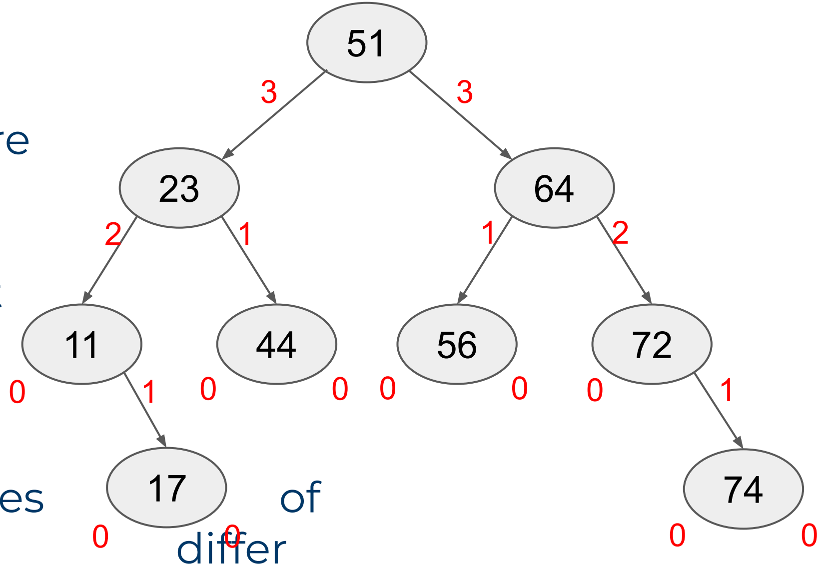
Transform-and-Conquer Algorithms Examples

The examples that will be seen are:

- Element Uniqueness in an Array
- Computing the Mode
- **Balancing Binary Search Trees**

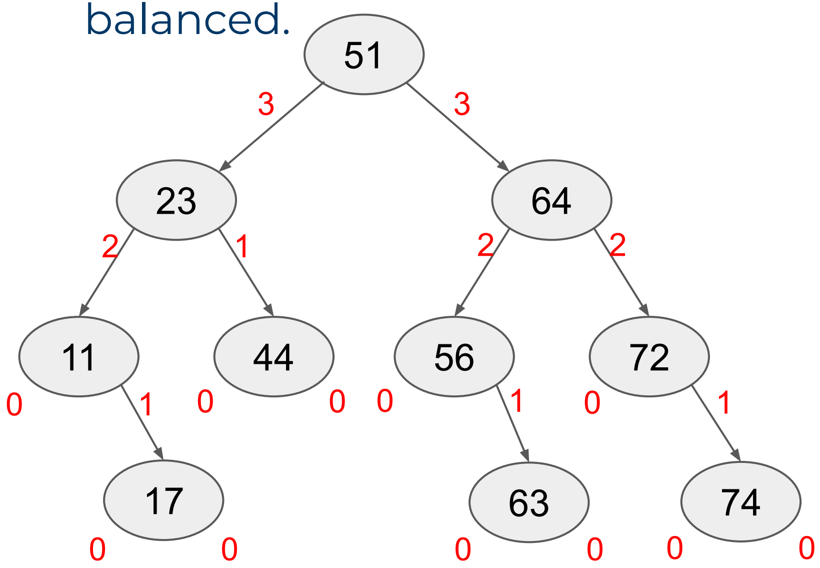
Example 3 - Balancing Binary Search Trees

- A **Binary Search** tree is a tree where:
  - the nodes have 0, 1, or **2** children (hence, **Binary**);
  - the values stored in each node are greater than the values stored in its left child subtree and smaller than the values stored in its right child subtree (hence, **Search**).
- A **Balanced Binary Search** tree is a tree where:
  - for all nodes the height of subtrees of its left and right children may differ by at most 1 (hence, Balanced).

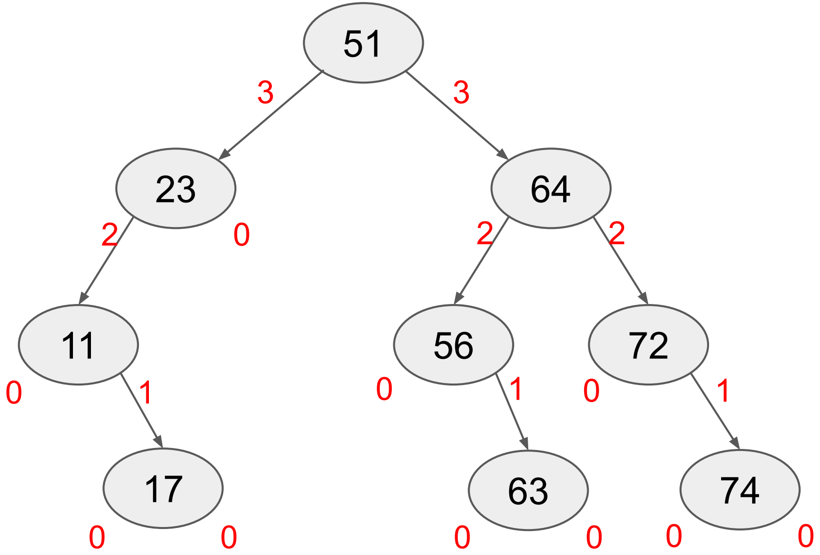


Example 3 - Balancing Binary Search Trees

- If we insert 63 the tree remains balanced.



- If we remove 44, after, then the tree becomes unbalanced.

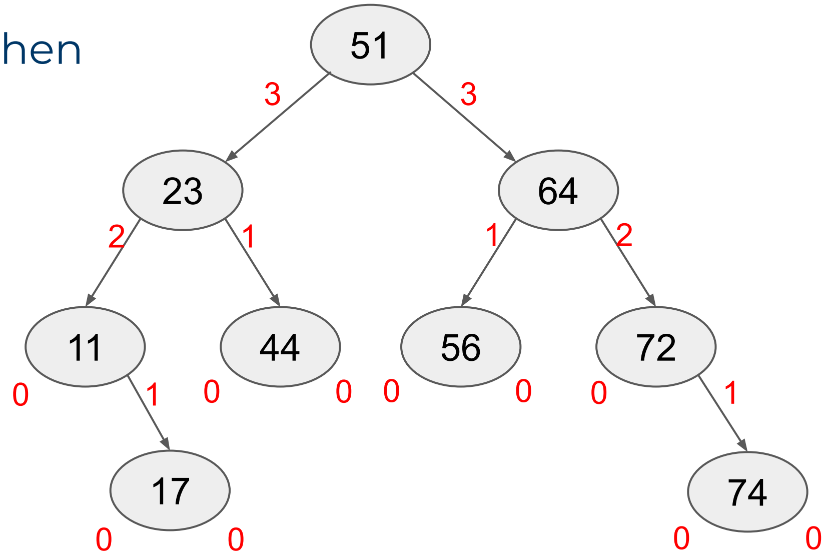


MERRIMACK COLLEGE

Text: [Balanced Binary Tree]

Example 3 - Balancing Binary Search Trees

- Given a binary search tree, balance it when performing an insertion or removal
- The brute force solution is based on:
  - Detect unbalancing
  - For every node compute the height of left and right subtrees, then balancing it
    - High complexity - ***$O(n^2 \log n)$***
  - An instance simplification solution make sure insertion and removal preserves the balance
  - Self-balanced trees - e.g., **AVL Trees** - named after Adelson-Velskii and Landis (1962).



Text: AVL Tree Data Structure.

Example 3 - Balancing Binary Search Trees

- AVL trees performs a check after each insertion or deletion;
- If the tree becomes unbalanced a correction procedure, called rotation is performed:
    - Simple rotation
        - Left rotation or Right rotation:
            - With or without children;
    - Double rotation
        - Right-left rotation or Left-right rotation:
            - With or without children.

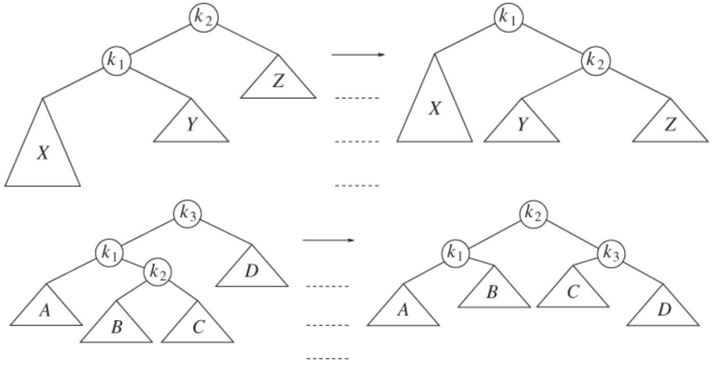
This is an instance simplification example.

Another option would be a representation change solution. In this case, the binary search tree can be changed into a structure allowing more than two childs for a node - a N-ary tree - e.g., **B-trees**.

Wikipedia: AVL Tree.

Example 3 - Balancing Binary Search Trees

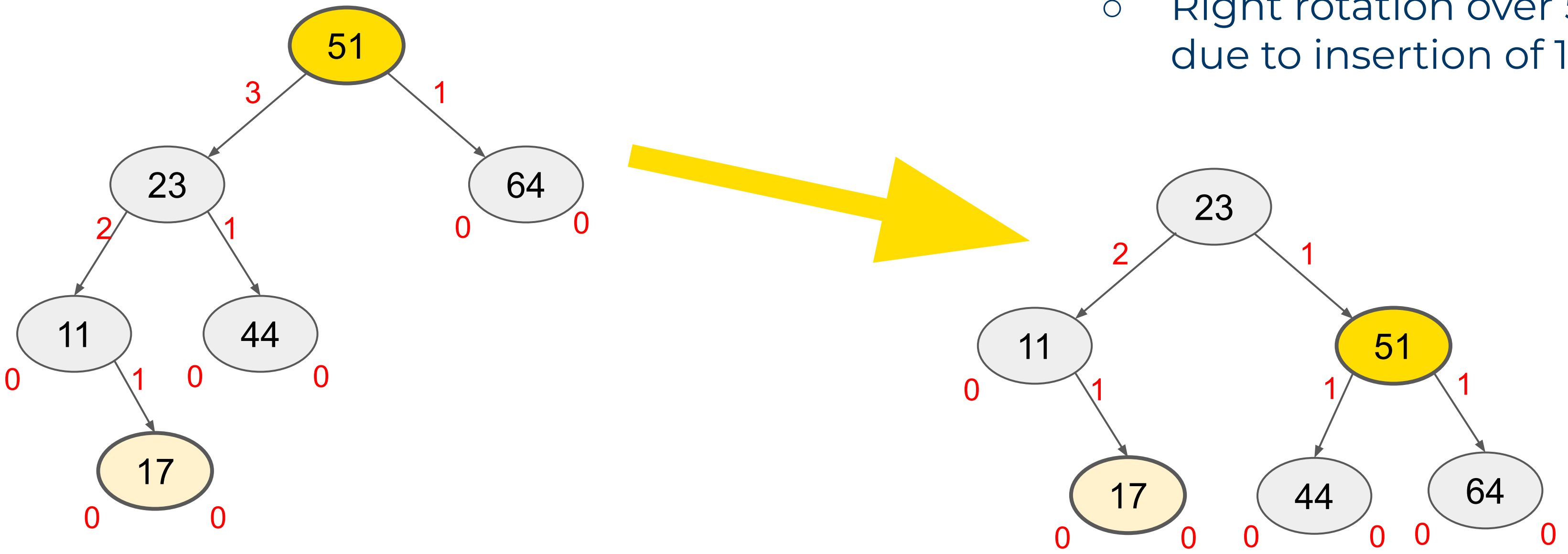
- AVL trees performs a check after each insertion or deletion;
- If the tree becomes unbalanced a correction procedure, called rotation is performed:
  - Simple rotation
    - **Left rotation** (push to left) or **Right rotation** (push to right);
  - Double rotation
    - **Right-left rotation** (push to right, then to left) or **Left-right** rotation (push to left then to right).



Implementations may vary.

Example 3 - Balancing Binary Search Trees

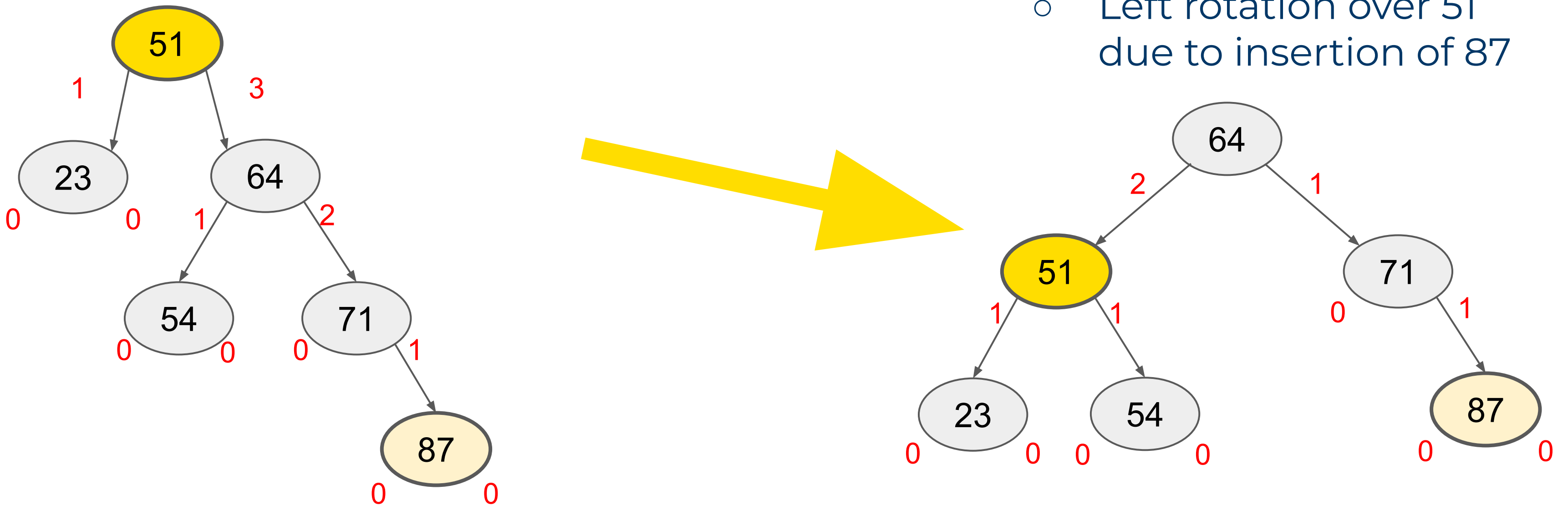
- Simple rotation
  - Right rotation over 51 due to insertion of 17



Simple right rotation.

Example 3 - Balancing Binary Search Trees

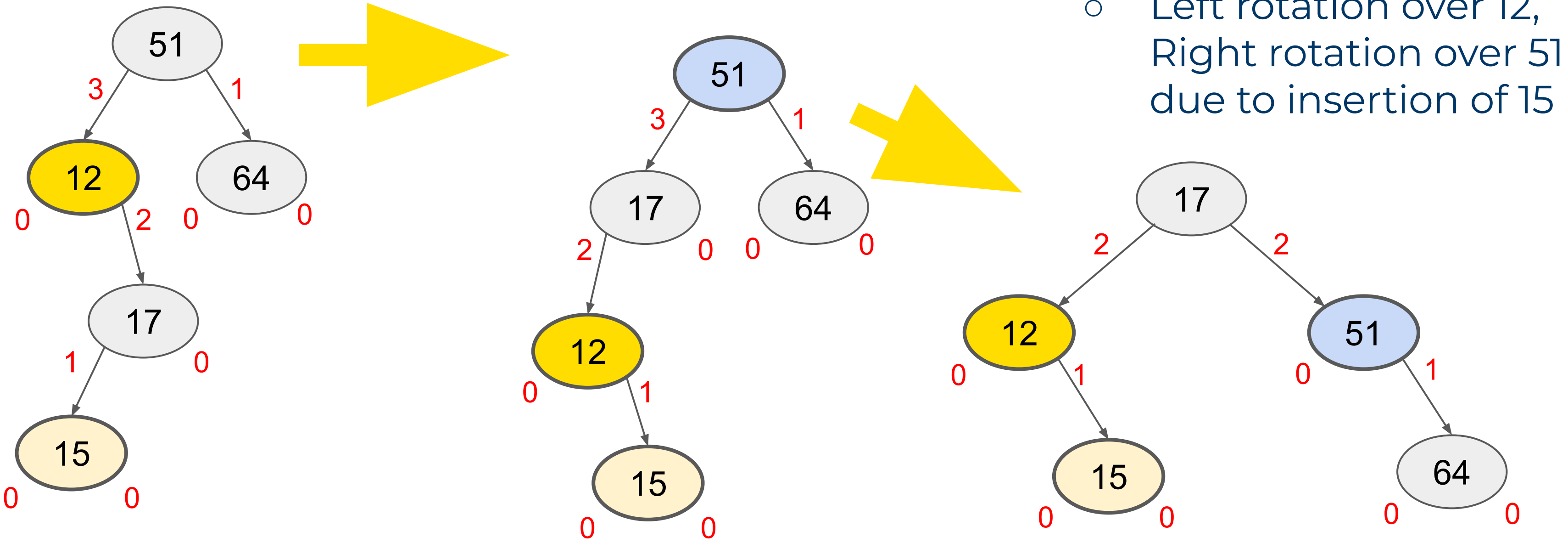
- Simple rotation
  - Left rotation over 51 due to insertion of 87



Simple left rotation.

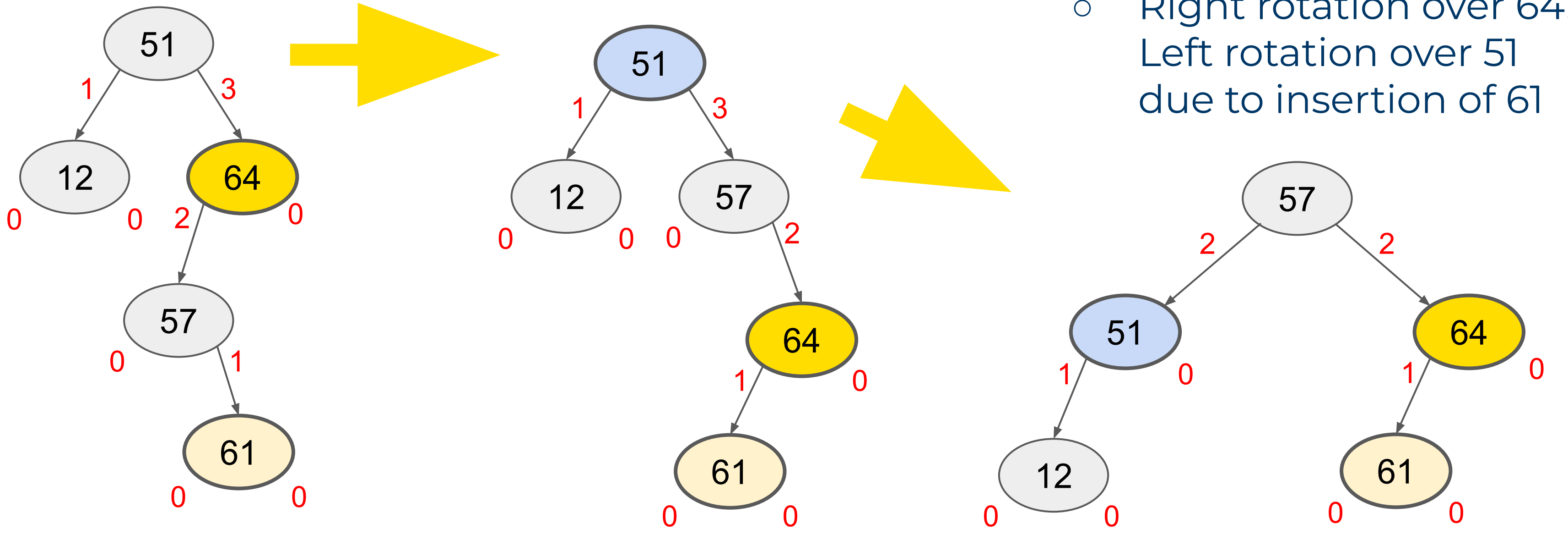
Example 3 - Balancing Binary Search Trees

- Double rotation
  - Left rotation over 12, Right rotation over 51 due to insertion of 15



Double left-right rotation.

Example 3 - Balancing Binary Search Trees



- Double rotation
  - Right rotation over 64, Left rotation over 51 due to insertion of 61

Double right-left rotation.

Example 3 - Balancing Binary Search Trees

```python
1 class TreeNode(object):
2     def __init__(self, data):
3         self.data = data
4         self.left = None
5         self.right = None
6         self.height = 0
```

A tree is defined by a root node (and its children).

Insertion starts in the root.

TreeNode with data, pointer to children, and the current height.

AVLtree with useful methods (lines 8, 12, and 61), rotations (lines 16 and 26), and the insertion (line 36).

```python
7  class AVLtree(object):
8 >    def getHeight(self, node):…
12 >   def getBalance(self, node):…
16 >   def leftRotate(self, z):…
26 >   def rightRotate(self, z):…
36 >   def insert_node(self, node, data):…
61 >   def printTree(self, node, level=0):…
```


Text: AVL Tree (implementation).

Example 3 - Balancing Binary Search Trees

```python
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 0
```

```python
    def printTree(self, node, level=0):
        if node is None:
            return
        print("---"*(level+1), node.data, "({})".format(node.height))
        self.printTree(node.left, level+1)
        self.printTree(node.right, level+1)
```

```python
    def getHeight(self, node):
        if not node:
            return 0
        return node.height
```

```python
    def getBalance(self, node):
        if not node:
            return 0
        return self.getHeight(node.left) - self.getHeight(node.right)
```

Useful methods

- to print a tree,
- to get the height of a node,
- to compute the balance (a positive value if left is the tallest subtree, a negative value if right is the tallest subtree).

MERRIMACK COLLEGE

Full code available here.

Example 3 - Balancing Binary Search Trees

```python
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 0
```

```python
36      def insert_node(self, node, data):
37          # find the location to insert
38          if not node:        # if the node does not exist
39              return TreeNode(data)
40          elif data < node.data:
41              node.left = self.insert_node(node.left, data)
42          else:
43              node.right = self.insert_node(node.right, data)
44          node.height = 1 + max(self.getHeight(node.left),
45                                self.getHeight(node.right))
46          # update the balance factor
47          balanceFactor = self.getBalance(node)
48          if balanceFactor > 1:   # unbalanced to the left
49              if data < node.left.data:  # simple right rotation
50                  return self.rightRotate(node)
51              else:                      # double left-right rotation
52                  node.left = self.leftRotate(node.left)
53                  return self.rightRotate(node)
54          if balanceFactor < -1:   # unbalanced to the right
55              if data > node.right.data: # simple left rotation
56                  return self.leftRotate(node)
57              else:                      # double right-left rotation
58                  node.right = self.rightRotate(node.right)
59                  return self.leftRotate(node)
60          return node
```

insert_node - recursively look for an empty spot. If ***node*** is empty create a new node (line 39). Otherwise try left or right depending on ***data*** and ***node.data*** (lines 41 or 43).

After inserting, check the balance, and if unbalanced, call a rotation (lines 47 to 59).

Full code available [here].

Example 3 - Balancing Binary Search Trees

left_rotate - push the node ***z*** to left (lines 17 to 20), then adjust the height of ***z*** and ***y*** (lines 21 and 23).

```python
16    def leftRotate(self, z):
17        y = z.right
18        w = y.left
19        y.left = z
20        z.right = w
21        z.height = 1 + max(self.getHeight(z.left),
22                           self.getHeight(z.right))
23        y.height = 1 + max(self.getHeight(y.left),
24                           self.getHeight(y.right))
25        return y
```

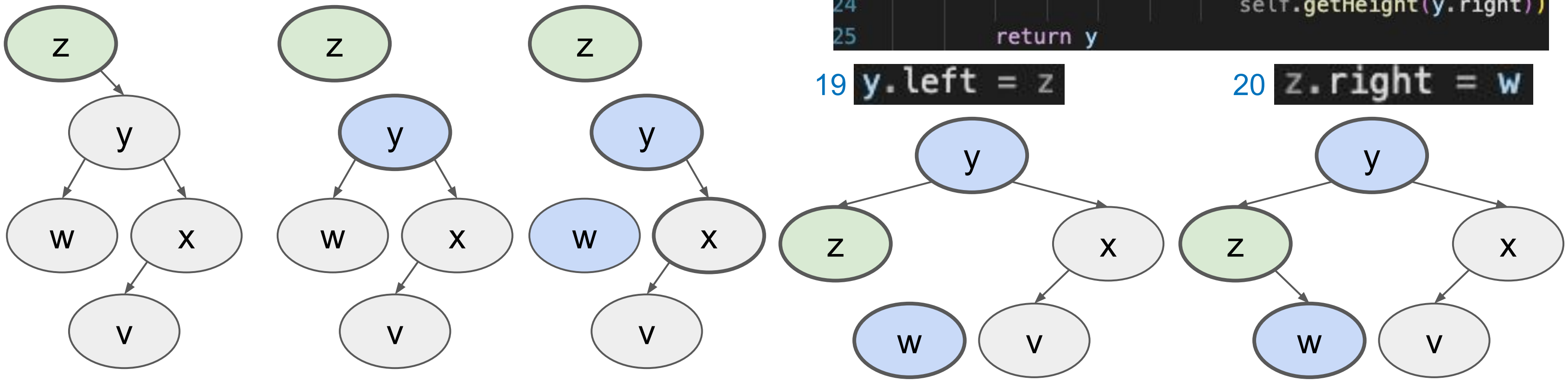
16 `leftRotate(self, z)`

17 `y = z.right`

18 `w = y.left`

19 `y.left = z`

20 `z.right = w`



This operation has constant complexity - ***O(1)***.

Example 3 - Balancing Binary Search Trees

right_rotate - push the node ***z*** to right (lines 27 to 30), then adjust the height of ***z*** and ***y*** (lines 31 and 33).

```
26    def rightRotate(self, z):
27        y = z.left
28        w = y.right
29        y.right = z
30        z.left = w
31        z.height = 1 + max(self.getHeight(z.left),
32                           self.getHeight(z.right))
33        y.height = 1 + max(self.getHeight(y.left),
34                           self.getHeight(y.right))
35        return y
```

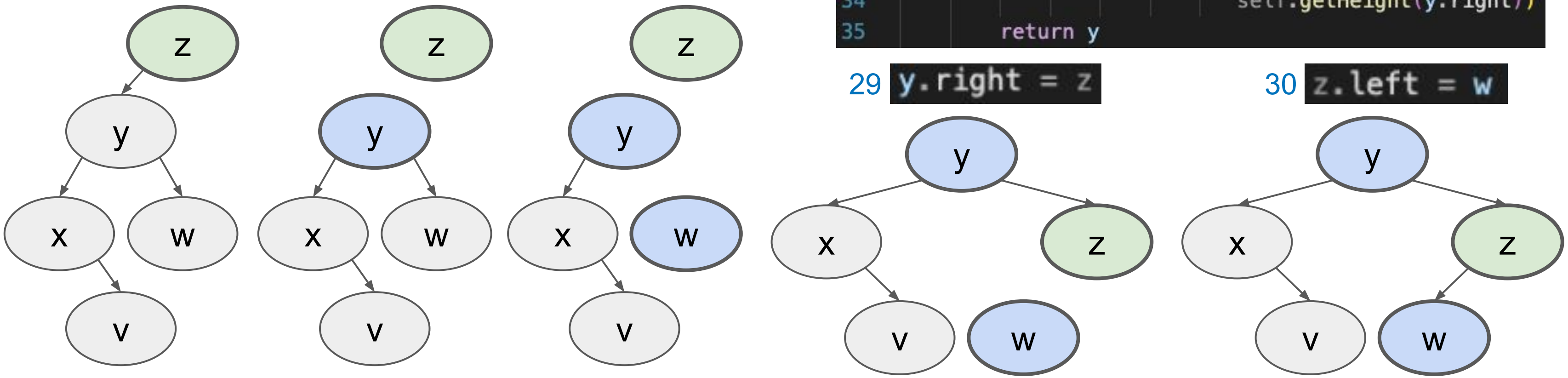
26 `rightRotate(self, z)`

27 `y = z.left`

28 `w = y.right`

29 `y.right = z`

30 `z.left = w`



This operation has constant complexity - ***O(1)***.

Example 3 - Balancing Binary Search Trees

The insertion procedure is just as complex as the search in binary trees, thus ***O(log n)***.

The cost of simple or double rotation is constant ***O(1)***.

```python
class AVLtree(object):
    def getHeight(self, node): ...
    def getBalance(self, node): ...
    def leftRotate(self, z): ...
    def rightRotate(self, z): ...
    def insert_node(self, node, data): ...
    def printTree(self, node, level=0): ...
```

```python
    def insert_node(self, node, data):
        # find the location to insert
        if not node:        # if the node does not exist
            return TreeNode(data)
        elif data < node.data:
            node.left = self.insert_node(node.left, data)
        else:
            node.right = self.insert_node(node.right, data)
        node.height = 1 + max(self.getHeight(node.left),
                              self.getHeight(node.right))
        # update the balance factor
        balanceFactor = self.getBalance(node)
        if balanceFactor > 1:   # unbalanced to the left
            if data < node.left.data:  # simple right rotation
                return self.rightRotate(node)
            else:                      # double left-right rotation
                node.left = self.leftRotate(node.left)
                return self.rightRotate(node)
        if balanceFactor < -1:   # unbalanced to the right
            if data > node.right.data: # simple left rotation
                return self.leftRotate(node)
            else:                      # double right-left rotation
                node.right = self.rightRotate(node.right)
                return self.leftRotate(node)
        return node
```


O(log n) - Logarithmic Algorithm

This Week's task

Hard deadline
Saturday 11:59PM EST

You can start the exam anytime from Monday 9PM EST to Saturday 8PM EST.

Final Exam

- Go to Module 8 on Canvas and take the Final Exam.
  - The exam is composed by 8 questions;
  - You have 4 hours once you start to take the exam;
  - You will download a ***.pdf*** with the 8 questions, and within 4 hours you need to upload a single ***.pdf*** with your answers;
- There is about one question about each week topic.
- You can consult all class materials, but not general access to the Internet.

Final Exam

- The exam is composed by 8 questions and you have 4 hours once you start to take the exam:
    - It is probably a good idea to spend 15 minutes at each question, if you got stuck, go to the next one;
- You will download a ***.pdf*** with the 8 questions, and within 4 hours you need to upload a single ***.pdf*** with your answers:
    - Other formats (***docx***, ***jpg***, etc) are not acceptable, practice generating ***pdf*** files before the exam;
- There is about one question about each week topic and you can consult all class materials, but not general access to the Internet:
    - Study the quizzes, and all other materials before taking the exam and make sure you have four non-interrupted hours to take the exam.

If something exceptional happens during the exam, send me an email as soon as possible, preferably with your answers.

Feel free to type it or hand write it, but you have to submit a single ***.pdf*** with your answers.

Next Steps

Grades

- Likely published next Tuesday on Canvas and MyMack.

Next Course

- CSC6023 - Advanced Algorithms, ideally next term.

After CSC6023

- CSC6033 (ideally just after CSC6023)
- CSC6301, CSC6302, CSC6303, and CSC6304 (in any order).

Thank you for taking CSC6013

- **Don't forget to take the exam;**
  - **Make sure your .** ***pdf*** **is ok (even after submitting - if something is amiss, send me an email immediately with the correct version).**

It was my honor and privilege to be your instructor at CSC6013.

MERRIMACK COLLEGE