APS106

UNIVERSITY OF TORONTO

Tutorial 12 – Week 13

We'll be starting at the 10 minute mark

if nothing else, write #cleancode

Agenda

- Linked Lists review
- Lecture review
  - Binary Trees
- Practice questions

Learning Objectives

After this tutorial, learners should be able to:

- recognize / describe / create Python classes
  - recognize / describe / create data attributes
    - recognize / describe / create class data attributes
    - recognize / describe / create instance data attributes
  - recognize / describe / create methods
    - recognize / describe / create class initializers (__init__)
    - recognize / describe / create non-initializer methods
- recognize / describe / create Python objects
- call methods on class / class instance objects
- recognize / describe / create linked data structures:
  - recognize / describe / create linked binary trees

APS106

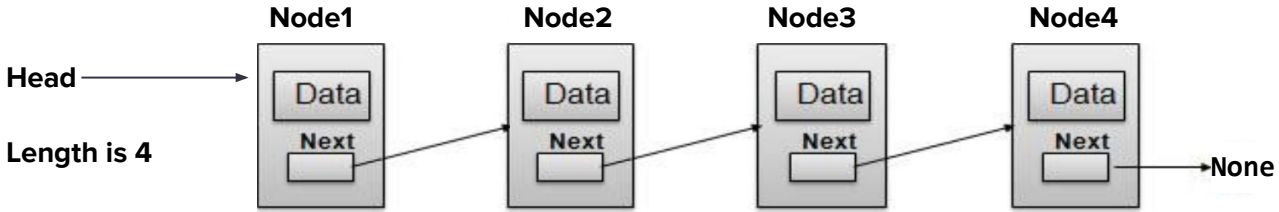
UNIVERSITY OF TORONTO

Review

Linked Data structures: ***Linked Lists***

Linked Lists review

- Linear collection of elements, known as of **nodes**
  - Each node contains data (called **cargo** in lecture) and a **link** to the next element in the list
- Linked lists are typically implemented using a container class (called **LinkedList** in lecture) endowed with various operations.
  - A typical class for representing **linked lists** would have (1) an attribute to keeps track of the first element of the linked list and an attribute to store the length of the list (called **head** and **length,** respectively, in the lecture examples) and (2) methods for various operations, such as adding a first element, adding a last element and removing the first element (called **add_first**, **add_last** and **remove_item** in the lecture examples)**.**



APS106

UNIVERSITY OF TORONTO

Review of Lecture

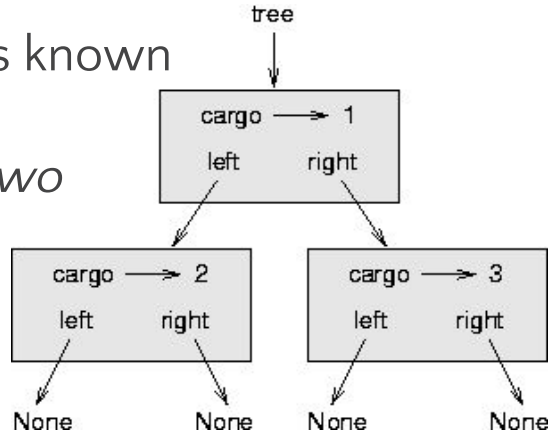
Linked Data structures: ***Binary Trees***

if nothing else, write #cleancode

Binary Trees

Instead of organizing elements in a "chain", aka a linked list, we can organize them in a tree !

- Like linked lists, binary trees are collections of elements known as **Nodes.**
  - Each node of a binary tree contains references to *two* nodes, referred to as the left and right children.



- A typical binary tree Node contains data attributes
  - to store data (called **cargo** in the lectures)
  - to store the address of the left child (called **left** in lectures)
  - to store the address of the right child (called **right** in lectures)

NOTE: These are the attribute naming conventions used in this course.
You can use other attribute names, if you wish.

Operations on Binary Trees

- **Insert** a new node (at the root, at a specific position, etc.)
- **Delete/Remove** a node (the root, with a given value, etc.)
- **Update** a node (modify a node's value and/or links to child nodes)
- **Search for/Retrieve** a node with a given value
- **Display/Print** the tree

How do you set-up a (linked) binary tree?

Option 1: use a class (for tree elements) – 1

We use a class for the elements of the collection and link the elements to each other.

- We start by creating a **Node** class
  - A node has three data attributes: **cargo** (i.e., the data/value of the node), **left** and **right** (i.e., the links to the left child node and the right child node)

```
class Node:

    def __init__(self, cargo = None, left = None, right = None):
        self.cargo = cargo
        self.left = left
        self.right = right
```

Option 1: use a class (for tree elements) – 2

We also implement the __str__ method to return a string representation of each node

```
def __str__(self):

    return str(self.cargo)
```

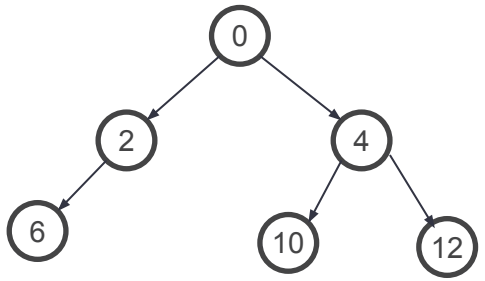
Operations on Binary Trees

- **Insert** a new node (at the root, at a specific position, etc.)
- **Delete/Remove** a node (from the root, from a specific position, etc.)
- **Update** a node (modify a node's value and/or link to the left/right child node)
- **Search for/Retrieve** a node with a given value
- **Display/Print** the tree

Option 1: Creating a Tree and Inserting Elements

Now, let's create some nodes and link them together to create a binary tree

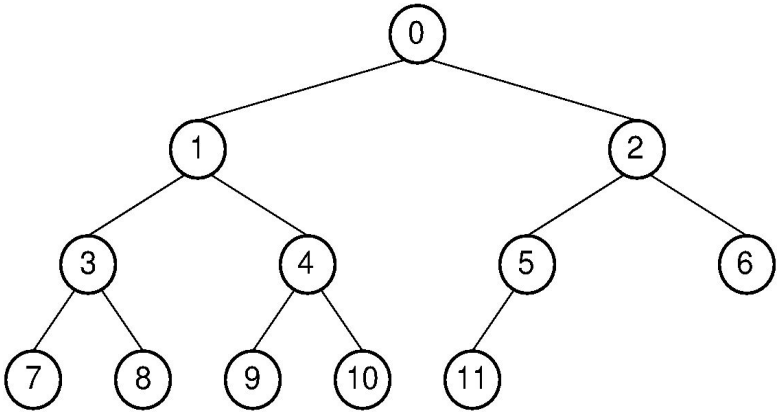
```
# create two " independent" nodes, i.e., left and right attributes are set to None
left_node = Node(2)
right_node = Node(4)

# create a binary tree
# tree_root is the node at the root of the tree
tree_root = Node(0, left_node, right_node)

# add three more nodes to the tree
# a node with cargo 6 as the left child of the left child of the root
left_node.left = Node(6)

# a node with cargo 10 as the left child of the right child of the root
# a node with cargo 12 as the right child of the right child of the root
right_node.left = Node(10)
right_node.right = Node(12)
```



The nodes with cargo values 6, 10 and 12 are referred to as **leaf** nodes.

Code-along

Let's work together to create the following binary tree in Python. We will use the **Node** class defined in the previous slides, including the **print_tree** method to visualize the tree.



Option 2: use a container Class

While a binary tree can be implemented using just the Node class, you can also create a class to represent binary trees. For example:

```
class BinaryTree:

    def __init__ (self):

        self.size = 0  #attribute to keep track of the size of the tree

        self.root = None #attribute to keep track of root node of the tree
```

Operations on Binary Trees

- **Insert** a new node (at the root, at a specific position, etc.)
- **Delete/Remove** a node (from the root, from a specific position, etc.)
- **Update** a node (modify a node's value and/or link to the left/right child node)
- **Search for/Retrieve** a node with a given value
- **Display/Print** the tree

Displaying/Printing binary trees

Method to visualize the tree:

```
def print_tree(self):

    ...
```

Practice Problem

Operations on Binary Trees

- **Insert** a new node (at the root, at a specific position, etc.)
- **Delete/Remove** a node (from the root, from a specific position, etc.)
- **Update** a node (modify a node's value and/or link to the left/right child node)
- **Search for/Retrieve** a node with a given value
- **Display/Print** the tree

Coding Question

Let's extend the BinaryTree class we created. Let's write a method min() for the BinaryTree class that returns the element in the tree with the smallest cargo. We assume that the tree is not empty.

- If there are multiple nodes that contain the lowest value,
  - **return the one that is closest to the root.**
- If there are multiple nodes that have the same minimal cargo value and are at the same distance from the root.
  - **return the left-most among them**.

Note: the distance of a node n to the root r of a tree is calculated as the number of parent nodes of n on the path from the root to n, including the root itself. Nodes that are the same distance from the root of the tree are said to be on the the same level in the tree.

THANK YOU FOR A GREAT SEMESTER!!

[Python Program to Draw Smiley Face Emoji Using Turtle]

slido

Any questions!?

ⓘ Start presenting to display the poll results on this slide.