CSC148H

Lecture 4

Trees

Motivating Trees

- A **data structure** is a way of organizing data
- Up until now all we have seen are **linear data structures**
  - stacks
  - queues
  - python lists (used in the implementation of stack and queue ADT)
  - naïve implementation of the priority queue ADT.

Motivating Trees

- A linear data structure organizes data in a linear (“one after another”) fashion
- This tends to be a natural way to organize data in certain types of applications:
  - A queue can be used for maintaining a line-up of pizza deliverers (assignment 1)
  - A stack can be used for keeping track of function calls made during a program's execution.

Motivating Trees

- It doesn't make sense to organize certain types of data into a linear structure.
- For example:
  - directories in a file system
  - structure of an HTML document

Tree - Definition

- A tree consists of (i) a set of **nodes** and (ii) a set of **edges,** each of which connects two nodes. In order for a set of nodes and a set of edges to define a tree, they have to satisfy a number of properties.
- Before we describe these properties, lets define some terms that we'll be using.

Trees - Terminology

- Node: Intuitively, think of a node as a point in the tree. It usually contains some piece of data, known as the “key” (or label).
- Edge: Connects two nodes. The connection is **directed** in the sense that the edge is “outgoing” from one node and “incoming” into the other node.
- Root node: The only node in the tree that has no incoming edge.
- Children: Set of nodes that have incoming edges from the same node.

Trees - Terminology

- Parent node: A node is the parent of all nodes to which it has outgoing edges.
- Siblings: Set of nodes that share a common parent.
- Leaf: A node that has no children (i.e., no outgoing edges).
- Internal node: A non-leaf node.
- Path: An ordered list of nodes that are connected by edges. (The traversal of edges only goes from parent to child.)

Trees - Terminology

- Descendant: A node *n* is a descendant of some other node *p* if there is a path from *p* to *n*
- Subtree: A subtree of some tree *T* is a tree whose root node *r* is a node in *T*, and which consists of all the descendants of *r* in *T* and the edges among them.
- Length of a path: Number of edges on a path
- Branching Factor: Maximum number of children for any node

Trees - Terminology

- Level (Depth): The level (or depth) of node n is the number of edges on the path from the root node to n
- Height: The maximum level of all nodes in the tree.

Tree - Definition

- A tree consists of (i) a set of **nodes** and (ii) a set of **edges,** each of which connects two nodes. In order for a set of nodes and a set of edges to define a tree, they have to satisfy a number of properties:
  - One node in the tree is designated as the root node
  - Each node, except the root, has exactly one parent
  - There is a unique path from the root to every node
  - There are no cycles – i.e, no paths that form “loops”

Tree – Definition

- A tree with a maximum branching factor of 2 (i.e., each node has a maximum of two children) is a **binary tree**.

Trees

- Common operations:
  - insert a new node
  - remove a node
  - traverse a tree: visit the nodes in some order and apply operations to each
  - attach a subtree at a node
  - remove a subtree

Representing Binary Trees

- Using list of lists:
  - First element of the list contains the label of the root node.
  - Second element is the list that represents the left subtree.
  - Third element is the list that represents the right subtree.

Representing Binary Trees

- Using nodes and references

```
class BinaryTree:
   def __init__(self, rootObj):
      self.key = rootObj
      self.left = None
      self.right = None

   def insertLeft(self, newNode):
      self.left = BinaryTree(newNode)
      # but what if self.left already exists?
...
```

Examples

- Lets see some examples in Wing!

Parse Trees

- Used for representing constructions like sentences and mathematical expressions.

Tree Traversal

- If we “traverse a list” we access each element in the list, possibly performing some operation with that element.
- List traversals are simple – you either go through a list from the first element to the last element, or vice-versa.
- There are different ways of traversing the nodes in a tree.

Tree Traversal

- We say that we have **visited** a node when we have done “something” with it (e.g. printed its key).
- The standard ways of traversing a binary tree are as follows:
  - preorder traversal
  - inorder traversal
  - postorder traversal

Tree Traversals

- Preorder: Visit the root node, do a preorder traversal of the left subtree, and do a preorder traversal of the right subtree
- Inorder: Do an inorder traversal of the left subtree, visit the root node, and then do an inorder traversal of the right subtree.
- Postorder: ???

Examples

- Lets see some examples of what the different traversals look like