Programming Fundamentals Using Python

2018

Problem Set 10

Most recent updated: July 14, 2018

Objectives

1. Heap
2. Binary Search Tree

Note: Solve the programming problems listed using your favorite text editor. Make sure you save your programs in files with suitably chosen names, **and try as much as possible to write your code with good style (see the style guide for python code)**. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems: Cohort sessions

1. *Trees* See Figure 1 and answer the following:

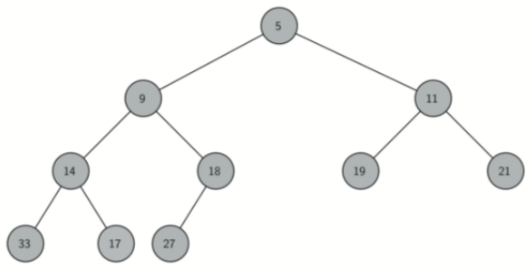

Figure 1: Example of a Tree

   (a) Which is the root node?

   (b) How many nodes does the tree have?

   (c) How many edges does the tree have?

   (d) What is the path from the root to the node with the largest value?

   (e) Which is(are) the child(ren) of the node with a value of 14?

   (f) Which is the parent of the node with a value of 19?

   (g) Which is(are) the sibling(s) of the node with a value 21?

   (h) How many leaves does the tree have?

   (i) What is the height of the tree?

   (j) What is the level of the node with the value 18?

   (k) Identify the largest subtree in the tree.

2. *Binary Tree: heapq* Implement several functions to modify Python's built-in list for heap operations. We are not going to create a separate class but simply use the existing Python's List to represent heap. So we will only implement those operations as functions.

   (a) `heappush(heap, item)` : to insert item into the heap.

   (b) `heappop(item)` : to remove the smallest item from the heap and return this item. The function should preserve the heap property invariant.

   (c) `heapify(heap)` : to create a heap data structure from an existing list. The function does not return anything and it should modify the argument in place.

   Design your own test cases and compare with Python's built-in `heapq` module.

3. *Binary Search Tree* We will implement Binary Search Tree in order to implement Map ADT. To do this we will build two classes. The first class is called `TreeNode` and the other class is called `BinarySearchTree`. The `TreeNode` class has the following attributes:

   (a) `key` : which is the key of the current node.

   (b) `val` : which is the value of the current node.

   (c) `left_child` : which is the reference to the left child node.

   (d) `right_chid` : which is the reference to the right child node.

   (e) `parent` : which is the reference to the parent of the current node.

   You will need to add the necessary methods for the class `TreeNode`. The class `BinarySearchTree` has the following attributes:

   (a) `root` which is the root node of the tree.

   (b) `size` which gives the number of nodes in the tree.

   Note that the root node is of class `TreeNode`. You need to define the following methods:

   (a) `put(key, val)`: to add an item into the Map with its key and value.

   (b) `get(key)`: to retrieve the value of an item given the key.

   (c) `__len__()`: to give the size of the tree. This allows the function `len()`: to work on this Map data type.

   (d) `__iter__()`: to iterate over the nodes using inorder traversal. This means starting from the left subtree, root node, and then the right subtree.

   (e) `__delitem__(key)`: to delete an item given the key. This allows one to use the `del` operator.

   (f) `__contains__(key)`: to find if an item of a given key is in the Map. This allows one to use the `in` operator.

   Design your own test cases to test the Map ADT.

End of Problem Set 10.