Assignment 10: AVL Trees and Priority Queues

CS 301

March 25, 2024

You have now seen how an AVL tree works and how a binary heap works. Now, let's implement them in Python. Implementations for both exist in the textbook, but it is best that you implement them yourself.

Implement your code in a `BalancedTree.py` Python code file.

1. Implement a `AVLTree` class that represents an `AVLTree`. You may use your prior BST implementation from last week as a base (through object-oriented inheritance). You may implement any helper methods that you need to maintain the AVL tree's properties. As with the prior BST implementation, your `AVLTree` class must contain the following methods, with an additional method that you did not implement in the prior assignment:
    - `insert(item)` inserts the item into the AVL tree, and ensures that the tree remains balanced.
    - `search(item)` looks for the item in the AVL tree. If it exists, return True, otherwise return False.
    - `delete(item)` deletes the item from the AVL tree if it exists inside the list. After deletion, the resulting AVL tree should maintain the balanced property. If the item exists in the list and is removed, return True. Otherwise, if the item does not exist return False.
    - `sortedList()` returns the list of items in ascending sorted order in a list.
    - `reverseSortedList()` returns the list of items in descending sorted order in a list.

    For each method, provide the average and worst-case running time in comments in your code.

2. Implement a `PriorityQueue` class using a binary heap, specifically a **max heap** (you have seen a **min heap** in class). Your priority queue implementation stores items along with their priority keys in the binary heap, and as such should be ordered by the **priority key**, not the item being stored. Your `PriorityQueue` implementation must contain the following methods:
    - `insert(item, key)` inserts the item with its associated key into the priority queue. After insertion, ensure that the internal binary heap satisfies the heap order property.
    - `pop()` returns a tuple `(key, item)` that is the item associated with the **largest key** in the queue. After popping, the priority queue should ensure that the internal binary heap maintains the heap order property.
    - `returnQueue()` returns a list representation of the priority queue that is ordered by how the contents of the queue will be returned by successive pop() operations. Note that this method does not return a list that if modified, modifies the original queue.

    For each method, provide the average and worst-case running time in comments in your code.