Midterm 2: Worksheet

CSCI2270-202: Data Structures

TA: Sanskar Katiyar

Binary Tree

1. Perform preorder, inorder, postorder traversal on a binary tree.
2. Write a function `findProduct`, which returns the product of values in the tree nodes.
3. Write a function `countNodes`, which returns the number of nodes in the binary tree.
4. Perform a level-order traversal on a binary tree.
5. Write a function `isMirror`, which determines whether two binary trees are mirrors.
6. Write a function `printLevelNodes`, which prints all nodes at parameter level *k*.
7. Write a function to calculate the height of a binary tree

Binary Search Tree

1. Write a function `isBST`, which determines whether a given binary tree is a BST or not.
2. Write a function `searchBST`, which searches for a given key *x* in the BST.
3. Which node will contain the (i) maximum node (ii) minimum node in a BST?
4. Write a function `insertBST`, which inserts an item *x* into the BST.
5. Write a function `deleteBST`, which deletes an item *x* from the BST. *What if nodes had parent pointers*?
6. Which tree traversal will result in a sorted (ascending) order over node keys?
7. How can you find the k^{th} smallest element in the tree without using extra memory?
8. What is the *average* time complexity of BST operations: insert, delete, search?
9. Review `leftRotate` function from Assignment 7, and implement `rightRotate`. How can you identify which child a node is (left or right)?

Graph

1. Write all supplementary functions for Graph class: `addVertex`, `addEdge`, `removeVertex`, `removeEdge`.
2. Write an iterative version of Breadth First Traversal.
3. Write a recursive version of Breadth First Traversal.
4. Write an iterative version of Depth First Traversal.
5. Write a recursive version of Depth First Traversal.
6. Write the implementation for Dijkstra's single-source shortest path algorithm. Also, print the path found to some destination.
7. Write a function to find the number of components in a given Graph.

Hash Table

1. Compose two distinct hash functions as examples.
2. Simulate Linear Probing for collision resolution on any example.
3. Simulate Quadratic Probing for collision resolution on any example.
4. Simulate Direct Chaining for collision resolution on any example.
5. What is the difference between Open Addressing and Separate Chaining.
6. How can you determine the efficiency of some given hashing technique?
7. What is the time complexity of inserting/searching an item x in a hash table, on average?

Hints, Solutions

[BT: 1] Core concept
[BT: 2] This is similar to the sumNodes function from Recitation 7 exercise.
[BT: 3] This is a minor modification on *any* tree traversal algorithm
[BT: 4] Perform BFS on the root node of a tree; adjust neighbors to use left, right children
[BT: 5] The key is to compare Tree 1's node's left child to Tree 2's node's right child, and vice versa.
[BT: 6] Page #6: Assignment 6
[BT: 7] Make sure you get the max heights from your returns.

[BST: 1] Recitation 8 Exercise
[BST: 2] Core concept
[BST: 3] Max: (Leftmost node in the tree); Min: (Rightmost node in the tree)
[BST: 4] Core concept
[BST: 5] With parent pointers, Core concept
[BST: 6] Inorder traversal. *Why?*
[BST: 7] Initialize a counter = 0. Perform inorder traversal, increment counter per node. If counter == k, return.
[BST: 8] O(log N). Note that the questions mentions average since it's not necessarily a balanced BST.
[BST: 9] Page #6: Check a node's value with its parent's. Smaller value implies left, else right.

[Graph: 1] Can you convert an adjacency matrix representation into an adjacency list?
[Graph: 2] [Graph: 3] Core: Also simulate an example on paper.
[Graph: 4] [Graph: 5] Core: Also simulate an example on paper. In the iterative version, create your own stack. The implementation will be similar to the iterative version of a BFS, except that you use a stack instead of a queue.
[Graph: 6] Careful, there are multiple objectives in this question. Break it down to smaller chunks.
[Graph: 7] Recitation 11 Exercise

[Hash: 1] Refer to textbook for examples
[Hash: 2] Slide #23
[Hash: 3] Slide #34
[Hash: 4] Slide #37
[Hash: 5] Slide #46
[Hash: 6] Slide# 19: Load Factor
[Hash: 7] O(1) in the average case. O(N) in the worst case.