CSCD320 Homework3, Winter 2024, Eastern Washington University, Spokane, Washington.

Name: **EWU ID:**

Please follow these rules strictly:

1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own solutions, without referring to anybody else's solution.
2. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.
3. Submission must be computer typeset in the **PDF** format and sent to the Canvas system. I encourage you all to use the LaTeX system for the typesetting, as what I am doing for this homework as well as the class slides. LaTeX is a free software used by publishers for professional typesetting and are also used by nearly all the computer science and math professionals for paper writing.
4. Your submission PDF file must be named as: **firstname_lastname_EWUID_cscd320_hw3.pdf**
   (1) We use the underline '_' not the dash '-'.
   (2) All letters are in the lower case including your name and the filename's extend.
   (3) If you have middle name(s), you don't have to put them into the submission's filename.
5. Sharing any content of this homework and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Problem 1 (20 points)**.** *Given the root of an existing binary search tree (BST) which is quite balanced and contains n distinct keys, you are asked to design a $\Theta(n \log n)$-time algorithm that will create a copy of the given BST. By "copy" we mean (1) the two trees have the exactly same topological structure; (2) between the two trees, for each pair of two nodes that occupy the same topological position, they contain the same key.*

1. *Clearly and concisely describe your algorithmic idea. Note: You do not have to write every detail of the code in a real programming language, but must be precise and clear about the main algorithmic idea, and reason why your algorithm works.*
2. *Give the pseudocode of your algorithm.*

Problem 2 (20 points)**.** *Given an arbitrary node* `x` *of a binary search tree (BST), show how to re-structure the BST using $O(n)$ time, so that* `x` *becomes the global root of the BST after the re-structuring. Describe and show the pseudocode of your algorithmic idea. Explain why your algorithm's time cost is $O(n)$. You can assume each node has the* `LeftChild`*,* `RightChild`*, and* `Parent` *links.* **Hint***: You may want to use tree rotations. If you use tree rotations, you don't have to give the detailed code of rotations, but instead just clarify what rotation on which node is being used.*

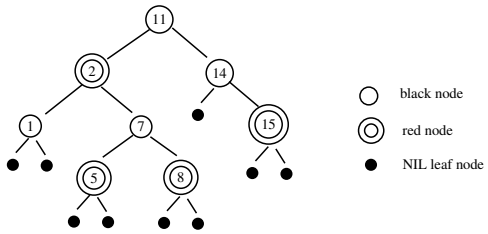
Problem 3 (20 points)**.** *We know both binary search tree (BST) and hash table can be used for data indexing for fast search. Each has pros and cons in the time and space efficiency. Do an extensive research to learn and compare these two data structures. Explain in your own language the pros and cons of each of them, and in what scenarios which works better. For each idea/opinion, provide the source of them if they are not your owns. For ex., the url of the webpages, the title and page of a book, the title/author/year of an article, etc.*

CSCD320 Homework3, Winter 2024, Eastern Washington University, Spokane, Washington.

Name: **EWU ID:**

Problem 4 (15 points)**.** *Show the trace of the construction of the Red-Black tree for the sequence* $19, 21, 12, 17, 15, 16$*. That is, you need to draw the state of the tree after inserting each number.*

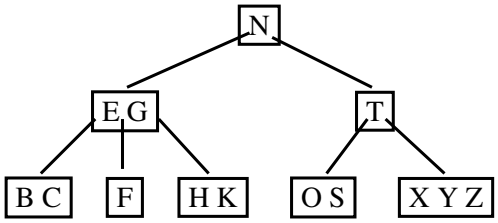
Problem 5 (15 points)**.** *Show the trace of deleting the nodes from the Red-Black tree below in the order of* $8, 15, 7, 11, 2, 5, 1, 14$*. That is, show the state of the tree after deleting each node.*



Problem 6 (15 points)**.** *Suppose that a node* x *is deleted from a red-black tree and then is immediately inserted back. Is the resulting red-black tree the same as the initial red-black tree ? If yes, prove it; If no, give an example to justify your answer.*

Problem 7 (15 points)**.** *Trace the ONE-PASS construction of the B-tree for the sequence* $\{S, U, O, A, W, X, G, H, P, C, M\}$*. Draw the configuration of the B-tree after inserting each letter. We use* $t = 2$ *as the branching degree threshold of the B-tree, so that: (1) all the non-leaf node must have at least* $t - 1 = 1$ *key and at most* $2t - 1 = 3$ *keys; and (2) The root node of an non-empty B-tree must have at least one key and at most* $2t - 1 = 3$ *keys.*

Problem 8 (15 points)**.** *Trace the deletion the sequence of keys* $\{T, G, F, O\}$ *from the B-tree below. Draw the configuration of the B-tree after each deletion. We use* $t = 2$ *as the branching degree threshold of the B-tree, so that: (1) all the non-leaf node must have at least* $t - 1 = 1$ *key and at most* $2t - 1 = 3$ *keys; and (2) The root node of an non-empty B-tree must have at least one key and at most* $2t - 1 = 3$ *keys.*



Problem 9 (25 points)**.** *We know binary search trees support the operations of finding (1) the minimum and maximum node of a given subtree; and (2) the successor and predecessor of a given node in the tree.* **Now you are asked to present the algorithmic idea and the pseudocode of the operations below for B-trees. Give the time cost of your algorithms in the big-oh notation and make the bound as tight as possible.** *(Note: You can assume all the keys in the B-tree are distinct; Don't forget the DISK_READ operations.)*

- `B_tree_max(root)`
  - *input:* `root`.
    "`root`*" is the reference to the root of the B-tree.*
  - *output:* `(node, i)` *or* `NULL`.
    If the tree is not empty, then the "`i`*"th key in the node "*`node`*" is the maximum key; Otherwise return* `NULL`.

Name: **EWU ID:**

- `B_tree_predecessor(node, i)`
  - *input:* `node, i`
    A node referenced by "`node`*" and its "*`i`*"th key.*
  - *output:* `(predecessor_node, j)` *or* `NULL`.
    If the predecessor of the "`i`*"th key of "*`node`*" exists, return the node referenced by "*`predecessor_node`*" whose "*`j`*"th key is the predecessor of the "*`i`*"th key of "*`node`*"; Otherwise, return* `NULL`.
  - *Notes: (1) You can use the* `B_tree_max` *as a subroutine for* `B_tree_predecessor` *if needed; (2) You can assume that each node has a parent link that points to the node's parent, and the parent link at the B-tree's root points to NULL; (3) you can assume the given "*`node`*" and "*`i`*" are valid, meaning that they exist in the tree. (4) Make sure you consider all the possibilities.*

`B_tree_min` *and* `B_tree_successor` *are asymmetry to* `B_tree_max` *and* `B_tree_predecessor` *respectively, so you don't have to work on them.*