**CSC 228 Data Structures and Algorithms, Fall 2019**
**Instructor: Dr. Natarajan Meghanathan**

**Exam 2 (Take Home)**

**Submission (in Canvas; See instructions in the last page)** **Due: Oct. 29th, by 11.59 PM**

**Q1 - 23 pts)** In this question, you will explore the tradeoff between the average number of comparisons for a successful search of an element in a hash table vs. the load imbalance index. It is logical to expect that as the hash table size (the size of the array of linked lists representing the hash table) grows, the length of each of the linked lists reduces: as a result, the number of comparisons that would be needed to search for any element is likely to reduce. On the other hand, as we fix the number of elements to store in a hash table, the load imbalance index (the ratio of the sum of the maximum and minimum lengths of the linked lists and the difference between the maximum and minimum lengths of the linked lists; see the slides for the formulation and details) is expected to increase with increase in the hash table size. Thus, for a fixed number of elements to store in a hash table, as we increase the hash table size, the average number of comparisons for a successful search is expected to decrease and the load imbalance index is expected to increase. As part of your solution, you will explore/quantify the above tradeoff.

You are given the code featuring the implementation of a hash table as an array of singly linked lists. The main function is already coded to create an array of size 100,000 with the maximum value per element being 50,000. You will try 20 different values for the hash table size ranging from 11 to 2,287 as given in the code (note the array of hash table size is already created for you in the main function). For each hash table size, you will run 25 trials.

You are required to implement the functions FindAvgNumComparisonsSuccessfulSearch( ) and ComputeLoadImbalanceIndex( ) in the Hashtable class. The main function is coded in such a way that these two functions are called for each of the trials for a certain hash table size and the average of the average number of comparisons for a successful search and the average load imbalance index are computed and printed for each value of the hash table size.

Take a screenshot of the output displaying the average number of comparisons for a successful search and the load imbalance index for different values of the hash table size.

Plot two Excel charts (X-Y plots) that features the values for the hash table size in X-axis and the values for the average number of comparisons for a successful search and the average load imbalance index in the Y-axes. Use the *trend line* option in Excel for each of these plots and determine a power function-based relation between the metric on the Y-axis and the hash table size in the X-axis. Display the power function-relation as well as the $R^2$ value for the fit in the Excel plots.

**Q2 - 25 pts)** Design and implement an algorithm to determine the next greater element of an element in an array in $\Theta(n)$ time, where 'n' is the number of elements in the array. You could use the Stack ADT for this purpose.

The next greater element (NGE) for an element at index i in an array A is the element that occurs at index j ($i < j$) such that $A[i] < A[j]$ and $A[i] \geq A[k]$ for all $k \in [i+1..., j-1]$. That is, index j ($j > i$) is the first index for which $A[j] > A[i]$. If no such index j exists for an element at index i, then the NGE for the element A[i] is considered to be -1.

For example: if an array is {1, 15, 26, 5, 20, 17, 36, 28}, then the next greater element (NGE) of the elements in the array are as follows:

| | |
|---|---|
| 1 | 15 |
| 15 | 26 |
| 26 | 36 |
| 5 | 20 |
| 20 | 36 |
| 17 | 36 |
| 36 | -1 |
| 28 | -1 |

Note: Your code need not determine and print the elements and their NGE values in the same order of the appearance of the elements in the array. An out of order print is also acceptable as long as the correct NGE for an element is printed out. For example, the following out of order print for the above array is also acceptable.

| | |
|---|---|
| 1 | 15 |
| 15 | 26 |
| 5 | 20 |
| 17 | 36 |
| 20 | 36 |
| 26 | 36 |
| 28 | -1 |
| 36 | -1 |

You are given a code file to run the algorithm on a smaller array (of size 10 integers) and print the output (i.e., the NGE for each element in the array, in the order determined by your algorithm). You should implement the algorithm in the main function itself and make use of the **Stack stack** object as part of the processing needed by your algorithm.

You are given the doubly linked list-based implementation code for Stack and there is no need to make any changes in the Stack class.

**Q3 - 12 pts)** You are given the code (including the main function) for evaluating an expression in post-fix format using a Stack. Your task is to modify the code in the main function to input an expression in pre-fix format, reverse it (as discussed in class) and then evaluate the value of the reversed expression (scanned from left to right) using a Stack.

You should also test your code with an expression string in pre-fix format that comprises of all the four operators (at least once) in a randomly chosen order with randomly chosen values in the range 1 to 9. For example, a sample expression string (pre-fix notation) could be input as **-, +, *, /, 5, 2, 3, 4, 8** for the expression (infix-notation) 5 / 2 * 3 + 4 - 8.

**Q4 - 20 pts)** A binary tree is a complete binary tree if all the internal nodes (including the root node) have exactly two child nodes and all the leaf nodes are at level 'h' corresponding to the height of the tree.

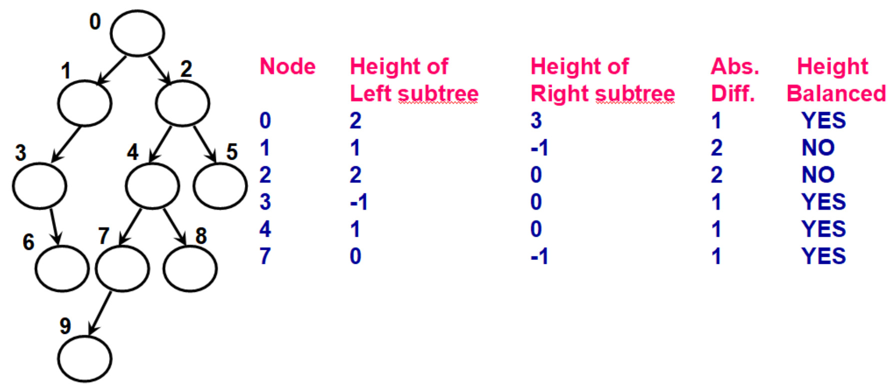
Consider the code for the binary tree given to you for this question. Add code in the blank space provided for the member function checkCompleteBinaryTree( ) in the BinaryTree class. This member function should check whether the binary tree input by the user (in the form of the edge information stored in a file) is a complete binary tree.

To test your code, come up with two binary trees of at least 12 vertices: one, a complete binary tree and another, a binary tree that is not a complete tree.

Prepare the input file for the two binary trees and input them to the code for this question. Capture the screenshots of the outputs.

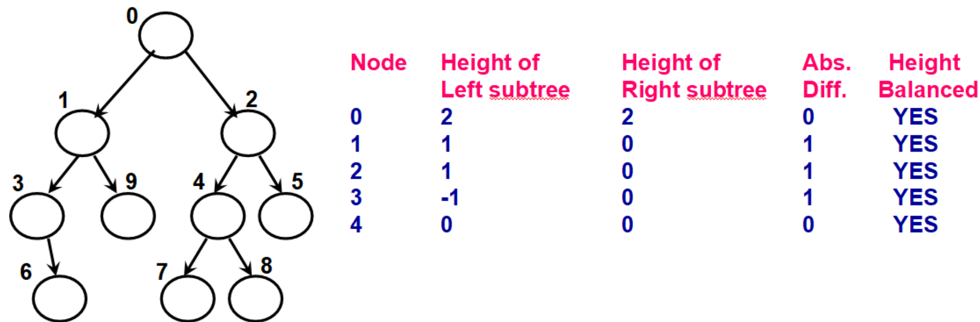
**Q5 -20 pts)**

In this question, you will write the code to determine whether a binary tree input by the user (in the form of an edge file, as discussed in the slides/class) is height-balanced or not. A binary tree is said to be height-balanced if each internal node (including the root) in the tree is height-balanced. A node is said to be height-balanced if the absolute difference in the heights of its left sub tree and right sub tree is at most 1. The binary tree (a) below is not height-balanced (as nodes 1 and 2 are not balanced), whereas the binary tree (b) below is height-balanced.



| Node | Height of Left subtree | Height of Right subtree | Abs. Diff. | Height Balanced |
|---|---|---|---|---|
| 0 | 2 | 3 | 1 | YES |
| 1 | 1 | -1 | 2 | NO |
| 2 | 2 | 0 | 2 | NO |
| 3 | -1 | 0 | 1 | YES |
| 4 | 1 | 0 | 1 | YES |
| 7 | 0 | -1 | 1 | YES |

(a) A Binary Tree that is "not" Height-Balanced



| Node | Height of Left subtree | Height of Right subtree | Abs. Diff. | Height Balanced |
|---|---|---|---|---|
| 0 | 2 | 2 | 0 | YES |
| 1 | 1 | 0 | 1 | YES |
| 2 | 1 | 0 | 1 | YES |
| 3 | -1 | 0 | 1 | YES |
| 4 | 0 | 0 | 0 | YES |

(b) A Binary Tree that is Height-Balanced

Note that the height of a leaf node is 0 and the height of a non-existing tree (or sub tree) is -1.

You are given the code for the binary tree implementation discussed in class. You could first find the height of each node in the tree and then test for each internal node: whether the difference in the height of its left child and right child is at most 1. If this property is true for every internal node, then we exit the program by printing the binary tree is indeed height-balanced.

Come up with files for storing the edge information of the two binary trees (a) and (b), and demonstrate the execution of your code to determine whether a tree is height-balanced or not.

Add any member variable (an array) to keep track of the height of the individual nodes in the tree as well as use the information in this array to determine whether the binary tree is height-balanced or not.

**Submission (in Canvas)**

**Submit separate C++ files for the codes as follows (so, a total of FIVE .cpp files as mentioned below):**

Question 1 (15 pts) The entire code (that includes your implementations for the FindAvgNumComparisonsSuccessfulSearch( ) and ComputeLoadImbalanceIndex( ) functions in the Hashtable class).

Question 2 (15 pts) The entire code, including the main function that has the implementation of the algorithm that you come up with to determine the next greater element (NGE) for an element in an array.

Question 3 (10 pts) The entire code (including the modification to evaluate an expression in pre-fix format) of the Stack class, Node class and the main function.

Question 4 (14 pts) The entire code (including the implementation of the function to check whether a binary tree is complete)

Question 5 (17 pts) The entire code of the binary tree program, extended to determine if the tree is height-balanced or not.

**Submit a single PDF file that includes your screenshots and analysis for all the five questions put together as mentioned below (clearly label your screenshots/responses for each question):**

Question 1 (8 pts)

(i) Screenshot of the output displaying the hash table size, the average number of comparisons for a successful search and the load imbalance index.

(ii) Screenshots of the Excel plots (drawn as mentioned earlier in the question description) that also display the power function-based relation and the $R^2$ value for each.

Question 2 (10 pts)
(i) Pseudo code for the NGE algorithm designed and implemented.
(ii) The time complexity of the NGE algorithm and show that it is $\Theta(n)$.
(iii) Screenshot of the execution of your algorithm for an array of 10 elements whose maximum value could be 50.

Question 3 (2.5 pts) Screenshot of the execution of the code for a sample pre-fix expression, like the one given in the question description.

Question 4-(6 pts) Draw the two binary trees (along with their node ids) that you have come up with, include the contents of the text files (corresponding to these two binary trees) that are input to the program as well as screenshots of the outputs.

Question 5-(3 pts) Screenshot of the outputs when the program is run for the two binary trees (a) and (b) shown in the question description.