Fall 2022 Algorithms
Take-home MWE-2 and Homework
Due: 27/12/2022

This document contains two questions corresponding to different assignments. The first question is to be added as a bonus of maximum of 40 points to the already held MWE-2. The second question is your homework.

1. (40 points) *MWE-2 Bonus Take-home Question*

   We are given a sorted array $keys[0 \dots n-1]$ of n search keys to be stored in a binay search tree (BST) and an array $freq[0 \dots n-1]$ of frequency counts where $freq[i]$ is the expected number of times of search for $keys[i]$. In order to do an efficient search for these keys, we want to construct an optimal binary search tree of all keys that minimizes the total cost of all the searches (i.e. $Total\ cost = \sum_i (depth(i)+1) \times freq(i)$).

   (a) (10 points) A Java implementation for optimal BST problem using Dynamic Programming is given here https://www.geeksforgeeks.org/optimal-binary-search-tree-dp-24/. Given n keys along with their frequencies, the **optimalSearchTree** method returns the total cost of the optimal BST. In that solution, only the optimal cost when the is computed. However, we also want to construct the BST from n keys and their frequencies.

   Modify the code so that the **optimalSearchTree** method returns, in the SAME table, the key of the root node as well as the cost of the optimal tree for every subproblem. You may add new classes/methods.

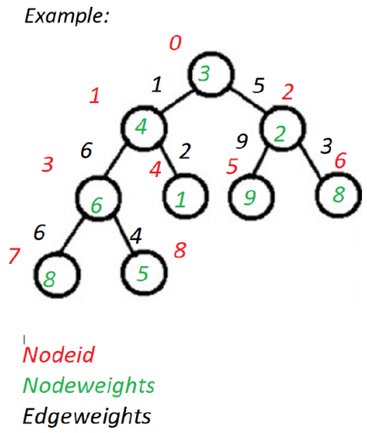
   Hint: Consider a table holding objects instead of numbers.

   (b) (10 points) You will use the **BinarySearchTree** class here https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/. Your goal is to write a method **constructOptimalSearchTree** which constructs an optimal BST given a table holding root node information, as returned by the above **optimalSearchTree** method.

   (c) (10 points) Write a separate test class. First, create one-hundred-thousand distinct integer keys between 0 and 99999 and randomly assign their frequency counts as integers between 1 and 100. Call **optimalSearchTree** method to return a table holding the root node information for an optimal BST. Then, create two instances of the **BinarySearchTree** class to construct two different BSTs of those one-hundred-thousand keys by calling **insert** method for each key in a random order. You will also create a one last **BinarySearchTree** by calling the **constructOptimalSearchTree** method. Finally, randomly select one-hundred keys and in each BST, sequentially call **search** method to search for each of these keys a number of times equal to its frequency. Compute and compare the average physical running times of **search** over all keys for each tree.

2. (100 points) *Homework Question*

   We are given a COMPLETE BINARY TREE where nodes and edges have positive weights. Node weights are stored in a 1-dimensional array WN. Edge weights are stored in a 2-dimensional array WE where 0 denotes no edge. Starting at the root of the tree and moving to either one of the children from the current node, the goal is to find the minimum total weight (i.e. sum of node and edge weights) path from the root to any one of the leaves.



Node weights array: ***WN*** = [3 4 2 6 1 9 8 8 5]

Edge weights array: ***WE*** = [
0 1 5 0 0 0 0 0 0
0 0 0 6 2 0 0 0 0
0 0 0 0 0 9 3 0 0
0 0 0 0 0 0 0 6 4
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
]

Output: Min total weight path includes nodes 1-2-5 with total weight 9.

   (a) (10 points) Implement a function that generates complete binary tree (i.e. two arrays) with given size as input where the node and edge weights are set randomly between 1 and 20 inclusive.

   (b) (20 points) Implement the greedy strategy (i.e. write a function) of choosing the child with the smallest sum of edge and node weights each time.

   (c) (20 points) Implement a recursive algorithm (i.e. write a function) to find the minimum total weight. You must determine the input parameters. Also, give the time complexity of a recursive algorithm that implements this formulation? Show your work.

   (d) (25 points) Implement a dynamic programming algorithm to solve the problem. You must determine the input parameters. Also, give the time complexity of your dynamic programming solution? Show your work.

   (e) (25 points) In your main function:

   - Show that the greedy algorithm does not solve this problem optimally.

- Run each of the recursive and dynamic functions with three different input sizes and compute the actual running times (in milliseconds or seconds) of these three algorithms. You will need to calculate the time passed before and after making a call to each function. Provide a 2x3 table involving the actual running times.