Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2022 – Assignment 3

Due date and time: Monday December 5th, 2022
by 10:00 AM (Morning)
The due date is sharp and strict. NO EXTENSION WILL BE ALLOWED BEYOND THIS TIME!

Part 1: Written Questions (50 marks):

Your submission must be typed (no hand-written text is allowed), and the quality of your submission matters. A better mark will be given for high-quality submissions.

Question 1

Assume the utilization of *linear probing* for hash-tables. To enhance the complexity of the operations performed on the table, a special *AVAILABLE* object is used. Assuming that all keys are positive integers, the following two techniques were suggested in order to enhance complexity:

i) In case an entry is removed, instead of marking its location as AVAILABLE, indicate the key as the negative value of the removed key (i.e. if the removed key was 16, indicate the key as -16). Searching for an entry with the removed key would then terminate once a negative value of the key is found (instead of continuing to search if AVAILABLE is used).

ii) Instead of using AVAILABLE, find a key in the table that should have been placed in the location of the removed entry, then place that key (the entire entry of course) in that location (instead of setting the location as AVAILABLE). The motive is to find the key faster since it now in its hashed location. This would also avoid the dependence on the AVAILABLE object.

Will either of these proposal have an advantage of the achieved complexity? You should analyze both time-complexity and space-complexity. Additionally, will any of these approaches result in misbehaviors (in terms of functionalities)? If so, explain clearly through illustrative examples.

Question 2

i) Draw the min-heap that results from the **bottom-up** heap construction algorithm on the following list of values:

20, 12, 35, 19, 7, 10, 15, 24, 16, 39, 5, 19, 11, 3, 27.

Starting from the bottom layer, use the values from left to right as specified above. Show immediate steps and the final tree representing the min-heap. Afterwards perform the operation `removeMin` 6 times and show the resulting min-heap after each step.

ii) Create again a min-heap using the list of values from the above part (i) of this question but this time you have to insert these values step by step (i.e. one by one) using the order from left to right (i.e. insert 20, then insert 12, then 35, etc.) as shown in the above question. Show the tree after each step and the final tree representing the min-heap.

Question 3

Assume a hash table utilizes an array of 13 elements and that collisions are handled by separate chaining. Considering the hash function is defined as: *h(k)=k mod 13*.

i) Draw the contents of the table after inserting elements with the following keys:
32, 147, 265, 195, 207, 180, 21, 16, 189, 202, 91, 94, 162, 75, 37, 77, 81, 48.

ii) What is the maximum number of collisions caused by the above insertions?

Question 4

To reduce the maximum number of collisions in the hash table described in Question 3 above, someone proposed the use of a larger array of 15 elements (that is roughly 15% bigger) and of course modifying the hash function to: *h(k)=k mod 15*. The idea is to reduce the *load factor* and hence the number of collisions.

Does this proposal hold any validity to it? If yes, indicate why such modifications would actually reduce the number of collisions. If no, indicate clearly the reasons you believe/think that such proposal is senseless.

Question 5

Draw a (single) ordered tree T, such that:

- Each node of T stores a single character;
- A preorder traversal of T yields: E K D M J G I A C F H B L T Z;
- A postorder traversal of T yields: D J I G A M K F L T Z B H C E.

Question 6

Assume an *open addressing* hash table implementation, where the size of the array is $N = 19$, and that *double hashing* is performed for collision handling. The second hash function is defined as: $d(k) = q - k \bmod q$,
where k is the key being inserted in the table and the prime number q is = 7. Use simple modular operation ($k \bmod N$) for the first hash function.
i) Show the content of the table after performing the following operations, in order:

put(25), put(12), put(42), put(31), put(35), put(39), remove(31), put(48), remove(25), put(18), put(29), put(29), put(35).

ii) What is the size of the longest cluster caused by the above insertions?
iii) What is the number of occurred collisions as a result of the above operations?
iv) What is the current value of the table's *load factor*?

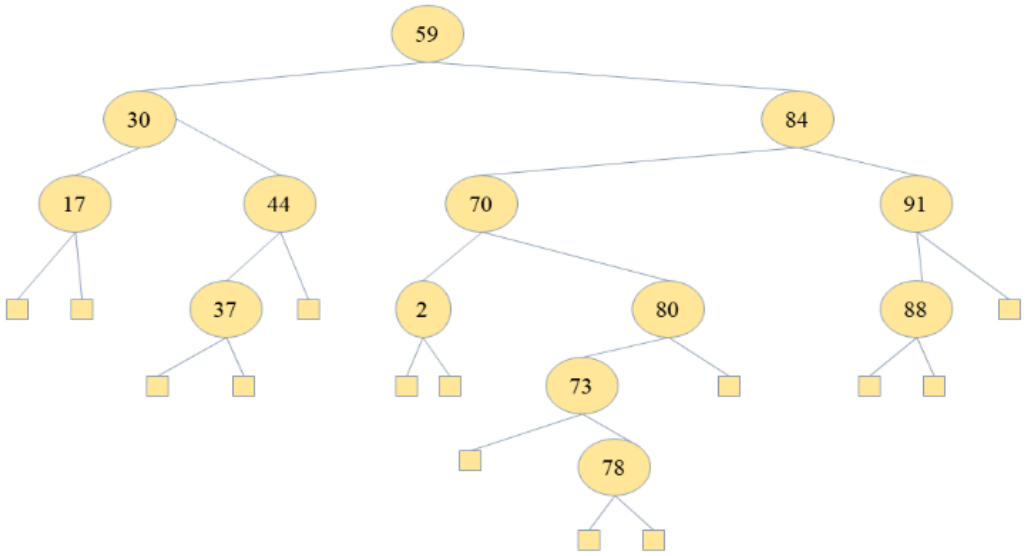
Question 7

Given 2 trees *T1* and *T2*, where T1 has more nodes than T2, and where each of the nodes in both trees holds an integer value. Further, the values in any of the trees are assumed to be unique (no duplicate values within the same tree).

i) Give an algorithm that determines whether or not T2 represents a subtree of T1.
ii) What is the time and space complexity of your algorithm in i)?
iii) Will your algorithm still work if duplicate values are permitted in these trees?
   a. If your algorithm still works when duplicate values are allowed, explain clearly why this change could not have affected your solution.
   b. If your algorithm will fail if duplicate values are allowed, provide an example of the failure and propose some modification to fix this failure.

Question 8

Given the following tree, which is assumed to be an AVL tree!



i) Are there any errors with the tree as shown? If so, indicate what the error(s) are, correct these error(s), show the corrected AVL tree, then proceed to the following questions (Questions ii to iv) and start with the tree that you have just corrected. If no errors are there in the above tree, indicate why the tree is correctly an AVL tree, then proceed to the following questions (Questions ii to iv) and continue working on the tree as shown above.

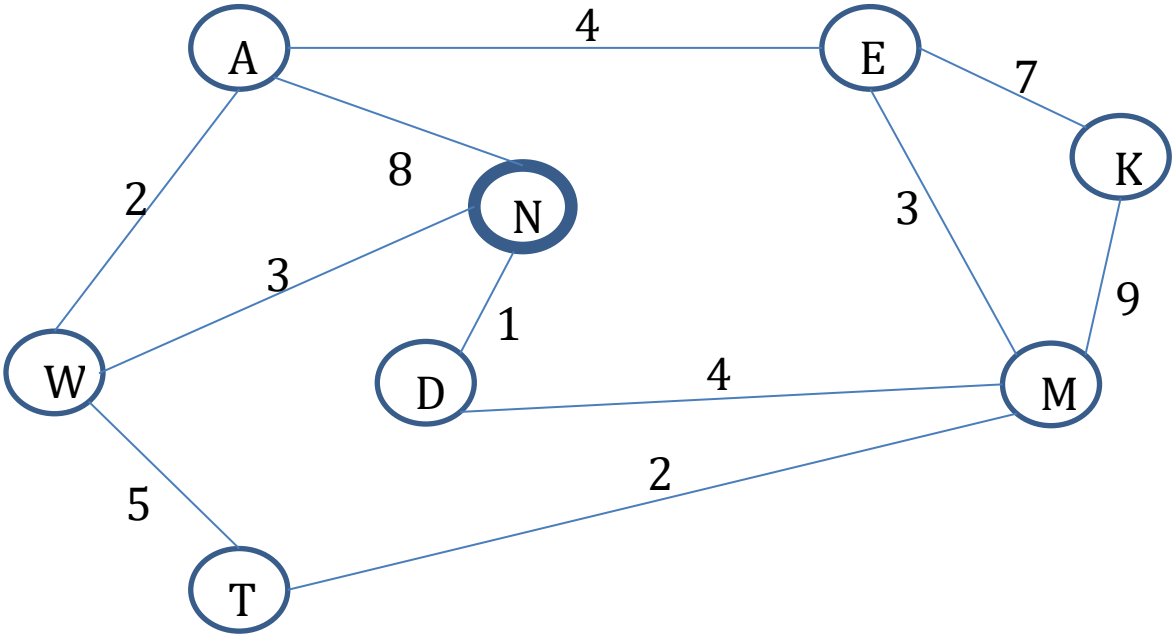
ii) Show the AVL tree after ***put(74)*** operation is performed. What is the complexity of this operation?

iii) Show the AVL tree after ***remove(70)*** is performed. What is the complexity of this operation?

iv) Show the AVL tree after ***remove(91)*** is performed. Show the progress of your work step-by-step. What is the complexity of this operation?

Question 9

Using ***Dijksra's Alogrithm***, *find the shortest path of the following graph form node N to each of the other nodes.*



Question 10

Show the steps that a radix sort takes when sorting the entries that have the following keys (notice that each entry has 3 keys)

8,3,2 9,1,0 4,1,1 1,7,2 2,4,3 5,7,3 3,2,6 2,9,2 6,8,2 4,8,9 9,6,7

Programming Questions (50 marks):

EHITS, is an inventory tracking system that supplies information about hospital inventory to prevent loss of valuable equipment. It keeps records of all Equipment Identification Number (EIN). Where each EIN is unique and consists of 8-digit code (e.g. # EIN: 89700035). EHITS has different types of lists that are local for small city hospitals where the number of equipment counts a few hundreds. Other lists are for equipment at the provincial/state level hospitals, that is *n* counts tens of thousands or more, or even at country levels, that is *n* counts millions or more. Furthermore, to track the hospital inventory status, it is important to have access to the equipment history. The historical record for the Equipment Identification Number should be kept in reverse chronological order (i.e. from most recent). EHITS, requires a design for a smart and flexible ADT for "Equipment history Report Listing", called ***ElasticERL***. Keys of ElasticERL entries are equipment identification numbers, that are 8-digit codes, and one can retrieve the key of a ElasticERL or access a single element by its key. Additionally, similar to sequences, given an equipment identification number in a ElasticERL, one can access its predecessor or successor (if it exists).

In general, the ElasticERL should adapt to its usage and the dynamic content that it operates on (hence the name *Elastic*), and it must keep a balance between memory and runtime requirements. For instance, if an ElasticERL contains only a small number of entries (e.g., few hundreds), it might use less memory overhead but slower (sorting) algorithms. On the other hand, if the number of contained entries is large (greater than 1000 or even in the range of tens of thousands of elements or more), it might have a higher memory requirement but faster (sorting) algorithms. ElasticERL might be almost constant in size or might grow and/or shrink dynamically. Ideally, operations applicable to a single ElasticERL entry should be between $O(1)$ and $O(\log n)$ but never worse than $O(n)$. Operations applicable to a complete ElasticERL should not exceed $O(n^2)$.

You are asked to **design** and **implement** the ElasticERL ADT. The ADT should accept the size (total number of Equipment Identification Number) as a parameter and uses an appropriate set of data types and data structures from the ones studied in class in order to perform the operations below efficiently[1]. **You are NOT allowed however to use any of the built-in data types** (that is, you must implement whatever you need, for instance, linked lists, expandable arrays, hash tables, etc. yourself).

The ElasticERL must implement the following methods:

- **SetEINThreshold (Size)**: where 100 ≤ Size ≤ ~500,000 is an integer number that defines the size of the list. This size is very important as it will determine what data types or data structures will be used (i.e. a Tree, Hash Table, AVL tree, binary tree, sequence, etc.);
- **generate()**: randomly generates new non-existing key of 8 digits;
- **allKeys(ElasticERL)**: return all keys in ElasticERL as a **sorted sequence;**
- **add(ElasticERL,key,value[2])**: add an entry for the given key and value;
- **remove(ElasticERL,key)**: remove the entry for the given key;

[1] The lower the memory and runtime requirements of the ADT and its operations, the better will be your marks.

[2] Value here could be any info of the equipment. You can use a single string composed of Equipment name, brand, model, and warranty expiry date.

- **getValues(ElasticERL,key)**: return the values of the given key;
- **nextKey(ElasticERL,key)**: return the key for the successor of key;
- **prevKey(ElasticERL,key)**: return the key for the predecessor of key;
- **rangeKey(key1, key2)**: returns the number of keys that are within the specified range of the two keys *key1* and *key2*.

1. Write the pseudo code for at least 4 of the above methods.
2. Write the java code that implements all the above methods.
3. Discuss how both the *time* and *space* complexity change for each of the above methods depending on the underlying structure of your ElasticERL (i.e. whether it is an array, linked list, etc.)?

You have to submit the following deliverables:

a) A detailed report about your design decisions and specification of your ElasticERL ADT including a rationale and comments about assumptions and semantics.

b) Well-formatted and documented Java source code with the implemented algorithms.

c) Demonstrate the functionality of your ElasticERL by documenting at least 5 different, but representative, data sets. These examples should demonstrate all cases of your ElasticERL ADT functionality (e.**g., all operations of your ADT for different sizes).** You must additionally test your implementation with benchmark files that are posted along with the assignment.

Submissions

Part I:

- **Part I must be submitted individually (No groups are allowed) under the submission folder: Assignment 3 – Part I. Submit all your answers in PDF or text formats only (no scans of handwriting except for images as indicated above; violating that will result in your answer being discarded). Please be concise and brief (less than ¼ of a page for each question) in your answers.**

Part II:

- **For Part II, a group of 2 (maximum) is allowed. No additional marks are given for working alone.**
- **If working alone, you need to zip (see below) and submit your zipped file under the submission folder: Assignment 3 - Part II.**
- **If working in a group, only one submission is to be made by either of the two members (do not submit twice). You need to zip (see below) and submit your zipped file, under the submission folder: Assignment 3 - Part II.**

Submission format: All assignment-related submissions that have more than one document must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The

submission itself must also contain your name(s) and student ID(s). Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names.

IMPORTANT: For Part II of the assignment, a demo for about 5 to 10 minutes will take place with the marker. You (or **both** members if working in a group) **must** attend the demo and be able to explain their program to the marker. Different marks may be assigned to teammates based on this demo. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time-slot per group; or zero mark is given).

Now, please read very carefully:

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**