Department of Computer Science and Software Engineering
Concordia University
COMP 352: Data Structure and Algorithms
Fall 2017
Assignment 4
Due date and time: Monday November 27th, 2017 by 6am

Important Note:
Since solutions of this assignment will be discussed during the tutorials/PODs of the last week of classes (week of November 27 – December 1), no deadline extension will be granted and no late assignments will be accepted (not even with a penalty!)

Part 1: Written Questions (40 marks):

Q.1

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:
5, 38, 127, 21, 16, 210, 28, 19, 150, 201, 332, 95, 123, 10, 30.

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

Q.2

To reduce the maximum number of collisions in the hash table described in the above question, someone proposed the use of a larger array of 14 elements and of course modifying the hash function to: *h(k)=k mod 14*. The idea was 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.

Q.3

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*?

Q.4

Assume the utilization of *linear probing* instead of *double hashing* for the above implementation given in the above question. Still, the size of the array is $N = 19$, and that simple modular operation (k mod N) is used for the hash function. Additionally, you must use a special *AVAILABLE* object to enhance the complexity of the operations performed on the table.

i) Show the contents of the table after performing the following operations, in order: **put(29), put(53), put(14), put(95), remove(53), put(32), put(19), put(30), remove(19), put(12), remove(32), put(72), put(32).**

ii) What is the size of the longest cluster caused by the above insertions? Using Big-O notation, indicate the complexity of the above operations.

iii) What is the number of occurred collisions as a result of the above operations?

Q.5

Consider a hash table where the size of the array is M = 5. If items with keys k = 22; 15; 25; 18 are inserted in that order, draw the resulting hash table if we resolve collisions using:

a) Separate chaining with $h(k) = (k + 2) \bmod 5$.
b) Linear probing with $h(k) = (k + 2) \bmod 5$.
c) Double hashing with $h1(k) = (k + 2) \bmod 5$ and $h2(k) = 1 + (k \bmod 4)$.

Part 2: Programming Question (60 marks):

The General Automobile Registration Office (GARO) keeps records of car registrations. It operates on multiple lists of *n* car registrations, where each registered car is identified by its unique license plate that consists of alphanumeric characters (e.g. R4G5OO54TE). The composition of license plates can be different for provinces and countries but their maximum length is restricted to 12 alphanumeric characters. Some of the lists are local for cities and areas, where *n* counts a few hundred properties. Others are at the provincial level, that is *n* counts tens of thousands or more, or even at country levels, that is *n* counts millions or more. Furthermore, it is important to have access to the history cars of that have been registered with the same license plate. Such a historical record for a license plate should be kept in reverse chronological order.

GARO needs your help to design a smart "automobile registration listing" data structure called SmartAR. Keys of SmartAR entries are strings composed of 6-12 alphanumeric characters, and one can retrieve the key of a SmartAR or access a single element by its key. Furthermore, similar to sequences, given a SmartAR element one can access its predecessor or successor (if it exists). SmartAR adapts to its usage and keeps the balance between memory and runtime requirements. For instance, if a SmartAR 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. SmartAR might be almost constant in size or might grow and/or shrink dynamically. Ideally, operations applicable to a single SmartAR entry should be between O(1) and O(log n) but never worse than O(n). Operations applicable to a complete SmartAR should not exceed $O(n^2)$.

You are asked to **design** and **implement** SmartAR, a smart data structure which automatically adapts to the dynamic content that it operates on. In other words, it accepts the size (total number of *n* car registrations identified by their key, i.e., license plate) as a parameter and uses an appropriate (set of) data structure(s) from the one(s) studied in class in order to perform the operations below efficiently[1].

The SmartAR must implement the following methods:

- **setThreshold(Threshold)**: where 100 ≤ Threshold ≤ ~500,000 is an integer number that defines when a listing should be implemented with a data structure such as a Tree, Hash Table, AVL tree, binary tree, if its size is greater than or equal to value of Threshold. Otherwise it is implemented as a Sequence.
- **setKeyLength(Length)**: where 6 ≤ Length ≤ 12 is an integer number that defines the fixed string length of keys.
- **generate(n)**: randomly generates a sequence containing n new non-existing keys of alphanumeric characters
- **allKeys()**: return all keys as a **sorted sequence (lexicographic order)**

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

- **add(key,value[2])**: add an entry for the given key and value
- **remove(key)**: remove the entry for the given key
- **getValues(key)**: return the values of the given key
- **nextKey(key)**: return the key for the successor of key
- **prevKey(key)**: return the key for the predecessor of key
- **previousCars(key)**: returns a sequence (sorted in reverse chronological order) of cars (previously) registered with the given key (license plate).

1. Write the pseudo code for each of the methods above.
2. Write the Java code that implements the methods above.
3. Discuss how both the time and space complexity change for each of the methods above if the underlying structure of your SmartAR is an array or a linked list?

You have to submit the following deliverables:

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

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

c) Demonstrate the functionality of your SmartAR by documenting at least 10 different but representative data sets. These examples should demonstrate all cases of your SmartAR ADT functionality (e.**g., all operations of your ADT for different sizes).** You have to additionally test your implementation with benchmark files containing a sequence of keys (one key per line without values) that will be posted soon here:

http://users.encs.concordia.ca/~haarslev/teaching/comp352/

Both the written part and the programming part must be done individually or by a team of max 2 students. Submit all your answers to written questions in PDF (no scans of handwriting) or text formats only. Please be concise and brief (less than ¼ of a page for each question) in your answers. For the Java programs, you must submit the source files together with the compiled executables. The solutions to all the questions should be zipped together into one .zip or .tar.gz file and submitted via EAS. You may upload at most one file to EAS.

For the programming component, you must make sure that you upload the assignment to the correct directory of Assignment 4 using EAS. Assignments uploaded to the wrong directory will be discarded and no resubmission will be allowed.

You will need to submit both the pseudo code and the Java program, together with your test data and experimental results. Keep in mind that Java code is **not** pseudo code.

[2] Value here could be any feature of the car or its owner.