American University of Armenia, CSE
CS121 Data Structures B
Spring 2022

Homework Assignment 7

Due Date: Tuesday, May 10 by 23:59 electronically on Moodle

Please solve the programming tasks using Java, following good coding practices (details are on Moodle). Don't forget to submit the corresponding assignment report.

1. **(25 points)** Write a class `UnsortedQueueMap` that extends the `AbstractMap` class using a circular queue as the underlying data structure.

   Your class should support all of the following functionality:

   (a) a no-arg constructor that creates an empty map;
   (b) `size` method (note that `isEmpty` is inherited from `AbstractMap`);
   (c) the fundamental methods `get`, `put`, and `remove` from the Map ADT;
   (d) the `entrySet` method from the Map ADT (note that `keySet` and `values` methods are inherited from `AbstractMap`).

   You may want to add utility method(s) to your class to simplify the implementation of the above functionality. Specify and justify the running times of all the methods. Test your implementation in a program.

2. **(35 points)** Write a class `BST` that extends the `AbstractBinaryTree` class and stores comparable elements using the same `Node` structure as the `LinkedBinaryTree`. All the external nodes should be sentinel nodes (they store `null`). An empty tree should contain only a single sentinel node at the root.

   In addition to overriding the necessary inherited methods, your class should support all of the following functionality:

   (a) a no-arg constructor that creates an empty BST;
   (b) a method for validating a position;
   (c) a method `expand` for assigning a given value to a sentinel node and adding two sentinel children to it;
   (d) a method `shrink` for turning the current node into a sentinel (setting element, right and left to null) and returning the value stored previously
   (e) a method `find` for finding and returning a position that stores the specified element or the position in which the element can be stored
   (f) a method `insert` for inserting a given value in the correct position in the tree
   (g) a method `remove` for removing a node with the given value from the tree and returning its element
   (h) overridden traversal methods that traverse the tree without the sentinel nodes.

   Specify and justify the running times of all the methods. Test your implementation in a program.

3. (35 points) This task investigates the implementation of several hashing approaches for Strings to be used for a hash map.

First, create a generic interface `HashFunction` with a single method `value`, that will be responsible for taking in an object and returning a hash value. Next, create four concrete implementations of `HashFunction`;

(a) Generic `DefaultHashFunction` where `value` returns the default hash value for an object

(b) String `SummingHashFunction` where `value` sums up the ascii values of the input string's characters and returns it as the hash value

(c) String `ProductHashFunction` where `value` multiplies the ascii values of the input string's characters and returns the product as the hash value

(d) String `PolynomialHashFunction` where the constructor of the class accepts two positive constant arguments c and h that are going to be used when computing the hash value. The `value` should use Horner's rule to compute the hash code based on the ascii values of the characters.

Create a class `CompressionFunction` that is going to be responsible for mapping the hash code to range $[0, N-1]$.

(a) The method `value` should take two arguments v and N and return the appropriately mapped value.

Now create class `StringHashMap` which extends `AbstractMap` with String Keys and represents a `Hash Map` with separate chaining. The underlying container should be an array of Array lists. In addition to overriding the necessary inherited methods, your class should support all of the following functionality:

(a) Three-argument constructor that takes as an input the size of the underlying container, the hash function and the compression function.

(b) `remove`, `put` and `get` values should invoke the `value` methods of the hash and mapping functions to find out the bucket for the input key.

(c) Method `NumberOfCollisions` that calculates the total number of collisions along all the buckets.

(d) Method `maxCollisions` that calculates the maximum number of collisions among all the buckets.

Create a program for testing your implementation. Create four string hash maps of size 61729. Each one should use one of the defined hash functions. Use the *.txt* file attached to the assignment (containing 58K English words) to fill those maps. For each of the four maps, print out the total and maximum number of collisions. Discuss the results. Experiment with different values for c, h and try to achieve less than 8 collisions for the `PolynomialHashFunction`. Discuss the results.