Programming Assignment 1
Points: 500
Due: Feb 25, 11:59PM
Late Submission Due: Feb 26, 11:59PM (25% penalty)

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistants for any questions/clarifications regarding the assignment. Your programs must be in Java, preferably Java 8.1.

In this programming assignment you will

- Implement a hash table that can store $\langle key, value \rangle$ pairs.
- Learn about a hashing technique known as *neighbor preserving hashing*, use it to solve *nearest points problem in 1-dimension*, and
- use it an application—rudimentary recommendation system.

You will design following classes:

- Tuple
- HashFunction
- HashTable
- NearestPoints
- RecSys

All your classes must be in the **default package** (even though it is not a good programming practice).

1 Tuple

Design a class name `Tuple` to represent tuples of form $\langle key, value \rangle$, where *key* is of type `int` and `value` is of type double. This class will have following constructor and public methods

`Tuple(int keyP, float valueP)` Creates `Tuple` object with $keyP$ as *key* and $valueP$ as *value*.

`getKey()` Returns *key*

`getValue()` Returns *value*

`equals(Tuple t)` returns *true* if *this* tuple equals t; otherwise returns false.

2 HashFunction

This class will represent a *random hash function* that can be used in a hash table. This class will have following instance variables: `a, b,` and `p` and should have following constructors and public methods.

`HashFunction(int range).` Picks the first (positive) prime integer whose value is at least `range`, and sets the value of p to that prime integer. Then picks two random integers x and y from $\{0, 1, \cdots, p-1\}$ and sets a as x and b as y.

`hash(int x)` Returns the value of the hash function on x; i.e, returns $(ax + b)\%p$.

Accessor methods named `getA, getB, getP` that respectively return a, b and p.

Modifier methods named `setA(int x), setB(int y)` that change the value of a (resp. b) to $x\%p$ (resp. $y\%p$). Modifier method named `setP(int x)`. This method will pick the first (positive) prime whose value is at least x and sets the value of p to that integer.

3 HashTable

This class will implement a hash table. The hash table will hold data items whose type is `tuple`. This class will have following public methods and constructor.

`HashTable(int size)` Finds the smallest prime integer p whose value is at least `size`. Creates a hash table of size p where each cell initially is NULL. It will determine the hash function to be used in the hash table by creating the object `new HashFunction(p)`.

`maxLoad()` Returns the maximum load of the hash table

`averageLoad()` Returns the average load of the hash table

`size()` returns the current size of the hash table.

`numElements()` returns the number of `Tuple`s that are currently stored in the hash table.

`loadFactor()` return the load factor which is numElements()/size()

`add(Tuple t)` Adds the tuple t to the hash table; places t in the list pointed by the cell $h(t.getKey())$ where h is the hash function that is being used. When the load factors becomes bigger than 0.7,

then it (approximately) doubles the size of the hash table and rehashes all the elements (tuples) to the new hash table. The size of the new hash table must be: Smallest prime integer whose value is at least twice the current size.

`search(int k)` returns an array list of Tuples (in the hash table) whose key equals k. If no such Tuples exist, returns an empty list. Note that the type of this method must be `ArrayList<Tuple>`

`remove(Tuple t)` Removes the Tuple t from the hash table.

4 Nearest Point Problem

Let S be a set of non-negative floating point numbers. Let n denote the number of points in S. We can view each member of S as a point on the x-axis (a point in the 1-dimensional space). We say that two points/numbers p and q are *close* if $abs(p-q) \leq 1$. Here *abs* denotes the absolute value. Below are the two most common queries that we would like to perform. These queries are called *nearest point queries*.

1. Given a point p (that may or may not be S) identify all the points that are close to p.
2. For every point $p \in S$, identify all the points that are *close* to p.

A naive approach for the first task the following: Given p, cycle through all points from S and identify the points that are close to p. Clearly this task takes $O(n)$ time where n is the number of points in S. Using this, the second task can be done in $O(n^2)$ time.

Can we build a data structure that can speed up these operations? We will use a technique known as *neighbor preserving hashing* together with hash tables to build the desired data structure. We say that a function g from floating point numbers to integers is a *neighbor preserving hash function* if the following holds for every pair of points p and q:

$$abs(p-q) \leq 1 \Rightarrow abs(g(p)-g(q)) \leq 1,$$

and

$$abs(p-q) \geq 2 \Rightarrow abs(g(p)-g(q)) > 1,$$

that is if p and q are close, then their hash values differ by at most 1 and if p and q are far apart, then their hash values differ by more than 1.

What could be a candidate for g? A simple candidate is $g(p) = floor(p)$. It is easy to verify that g is *neighbor preserving hash function*. Using g, we can build a data structure that stores S as follows. Build a hash table of size $m > 1.5n$ (where n is number if points in S). For each $p \in S$, add the tuple $\langle g(p), p\rangle$ at $h(g(p))$, where h is the hash function used to create the hash table.

Now, I claim that we can solve the first task in expected/average time $O(N(p))$, where $N(p)$ is the number of points that are close to p. This is an improvement over $O(n)$. Similarly the second task can be done in average/expected time $O(n+\sum_{p\in S} N(p))$ which is an improvement over $O(n^2)$. Your task is to design the algorithms with these running times and implement them.

4.1 NearestPoints

Design a class that contains methods to solve the nearest point problem using both approaches—-Naive and neighbor preserving hash functions. This class will have following methods/constructor.

`nearestPoints(String dataFile)` The variable `dataFile` holds the absolute path of the file that contains the set of points S.

`nearestPoints(ArrayList<float> pointSet)` The array list `pointSet` contains the set of points S.

`naiveNearestPoints(float p)` Returns an array list of points (from the set S) that are close to p. This method **must** implement the naive approach. Note that the type of this method must be `ArrayList<float>`

`buildDataStructure()` Builds the data structure that enables to quickly answer *nearest point queries*. Your data structure **must** use the notion of *neighbor preserving hashing* and along with the class `HashTable`. Otherwise, you will receive **zero** credit.

`npHashNearestPoints(float p)` Returns an array list of points (from the S) that are close to p. This method must use the data structure that was built. The expected run time of this method *must* be $O(N(p))$; otherwise you will receive **zero** credit.

`allNearestPointsNaive()` For every point $p \in S$, compute the list of all points from S that are close to p by calling the method `NaiveNearestPoints(p)`. Write the results to a file named `NaiveSolution.txt`

`allNearestPointsHash()` For every point $p \in S$, compute the list of all points from S that are close to p by calling the method `NPHashNearestPoints(p)`. Write the results to a file named `HashSolution.txt`. The expected time of this method **must** be $O(n + \sum_{p \in S} N(p))$; otherwise you will receive **zero** credit.

5 Recommendation Systems

Computing nearest points has numerous applications especially when the set of points reside in a high-dimensional space. In this PA, you will learn about an application to recommendation systems and implement (part of) a rudimentary recommendation system. Imagine a set of users and a set of movies. Some users have watched certain movies and rated them using a scale of [1, 2, 3, 4, 5]. Suppose u is a user and m is a movie, and u has not rated the movie m. Can you predict the rating of user u to movie m ? This is the core problem in recommendation systems. For this, we start with a ratings matrix. A *ratings matrix* format is described below. Assume that we have 5 users (with user ID's 1, 2, 3, 4, 5) and 4 movies (named M1, M2, M3, M4).

```
5 4
2 0 5 1
```

```
0 0 3 4
2 3 5 0
0 0 0 3
5 0 0 0
```

The above data is interpreted as follows. The first line of tells number of users (5) and number of movies (4). Each line below tells the ratings given by the user. Here 0 denotes that the user *has not* given a rating. Ratings given by User 1 are: 2, 0 , 5 and 1. Thus User 1's rating for M1 is 2, rating for M3 is 5, rating for M4 is 1. And User 1 did not rate M2.

Note that User 3 has not rated M4. Can we predict his/her rating for M4 from this data? One approach to solve this is via "nearest/similar users" approach. Compute users who are "similar/nearest" to User 3. Consider the ratings given by these similar users to the movie M4. Take the average of these ratings. That is the predicted rating of User 3 to M4. Suppose that we have determined that Users 2, 4 and 5 are nearest/similar to User 3. User 2's rating for M4 is 4, User 4's rating for M4 is 3 and User 5 has not rated M4. Thus we take the average of 4 and 2 which is 3.5. Thus our predicted rating for User 3 to M4 is 3.5.

How can we determine whether an user is "similar/close/near" to another user? An approach is to map each user to a point in high dimensional space. Now two users are "close/similar", if the distance between the corresponding points is at most 1. For example, lets say User 1 is mapped to the 3-dimensional point $\langle 2.3, 4, 3.6\rangle$, User 2 is mapped to 3-dimensional point $\langle 2, 4, 3.5\rangle$ and User 3 is mapped to 3-dimensional point $\langle 2, 6, 5\rangle$. Now, we can conclude that User 1 is close to User 2, but not to User 3. Techniques to map users to high dimensional points are beyond the scope of this course.

For this assignment, you will work with a mapped ratings matrix. Each user has been mapped to a 1-dimensional point, and the *mapped ratings matrix* looks like the following.

```
5 4
2.1 2 0 5 1
3.0 0 0 3 4
2.8 2 3 5 0
3.1 0 0 0 2
4.2 5 0 0 0
```

The first line tells that there are 5 users and 4 movies. The first column of the matrix lists the points to which user's has been mapped. Thus User 1 has been mapped to the point 2.1, User 2 has been mapped to 3.0, User 3 has been mapped to 2.8, User 4 has been mapped to 3.1 and User 5 has been mapped to 4.2. The rest of the information is as before.

5.1 RecSys

You will implement a class named `RecSys` that operates on a *mapped ratings matrix* (where each user is mapped to a 1-Dimensional point). This class **must** use appropriate (efficient) methods from the class `NearestPoints`; otherwise you will receive **zero** credit. This class must have following methods and constructor.

`RecSys(String mrMatrix)` The string `mrMatrix` is contains the absolute path name of the file that contains the mapped ratings matrix.

`ratingOf(int u, int m)` If the user u has rated movie m, then it returns that rating; otherwise it will predict the rating based on the approach described above, and returns the predicted rating. The type of this method must be float.

6 Report

Write a brief report that includes the following.

- Algorithm for the method `buildDataStructure()`
- Algorithm for the method `npHashNearestPoints(float p)`
- Create an instance of the class `NearestPoints` using the data from the file `points.txt`. Run the methods `allNearestPointsNaive()` and`allNearestPointsHash()` (note that you must build the data structure) before running `allNearestPointsHash()`. Report the run times of both methods. Use `System.currentTimeMillis()` to measure run times.

7 Guidelines

For this PA, you **may work in teams of 2**. It is your responsibility to find a team member. If you can not find a team member, then you must work on your own.

You are not allowed to use any external libraries. You are not allowed use Java inbuilt classes such as `HashMap`, `HashTable` etc.

Your code **must strictly adhere to the specifications**. The names of methods and classes must be exactly as specified. The return types of the methods must be exactly as specified. For each method/constructor the types and order of parameters must be exactly as specified. Otherwise, you will lose a significant portion points (even if your program is correct).

Your grade will depend on *adherence to specifications*, *correctness of the methods/programs*, and *efficiency of the methods/programs*. If your programs do not compile, you will receive **zero** credit.

8 What to Submit

Your submission must have following files.

- Tuple
- HashFunction
- HashTable
- NearestPoints
- RecSys
- report.pdf (must be in pdf format)
- ReadMe.txt (list the name(s) of team member(s)).

You must include any additional helper classed that you designed. Please include only source .java files, do not include any .class files. Please remember to use default package for all the java classes. Place all the files that need to be submitted in a folder (without any sub-folders) and `zip` that folder. Submit the `.zip` file. Only `zip` files please. Please include all team members names as a JavaDoc comment in each of the Java files. Only **one** submission per team please.