CSC 385
Homework Assignment, Module 7 (25 points)

This assignment consists of two problem. The first problem does not require coding. All you have to do for this problem is to answer the questions and submit a document containing your answers. The second problem requires coding. You must submit a java file containing your solution for this problem.

Problem 1: (10 points)

Instructions

For all of the questions assume you have a HashMap class that uses a threshold of 0.75 (regardless of the collision resolution mechanism used), and has an initial array size of 13. Assume the array is resized, when the current item to be added will make the load factor greater than or equal to the threshold value. Recall that the load factor is the fraction of a hash map that is full.

Table 1 contains a list of items, along with associated hash codes that were computed using some hypothetical hash function. Assume the items are added to a newly created instance of the HashMap class in the same order in which they are listed in the table. Based on this information, answer the following questions.

1. (5 points) Show what the array of the `HashMap` would look like after all the items below have been added, assuming the `HashMap` class uses **separate chaining hashing** to resolve collisions.

2. (5 points) Show what the array of the `HashMap` would look like after all the items below have been added, assuming the `HashMap` class uses **quadratic probing** to resolve collisions instead of separate chaining hashing.

Note that for both questions, you do not need to compute the hashcodes, you only need to determine where the items would be placed into the underlying array of the HashMap.

Table 1-entry-hashcode pairs for problems 1 & 2

Item	Hashcode
Ellie	1342415383
John	700056533
Sarah	330628742
David	532139483
Karen	217142585
Ronald	2112979549
Tom	207265348
Tim	1631149803

Problem 2 (coding problem) (15 points)

Problem: Given a collection of lists, use a HashMap to find the elements that are common in all lists. If there are duplicate items in the list then your program must return ALL occurrences that are common between all the lists for each item. For example, if an element occurs at least twice in each list, then your output must include two copies of this element.

For example, consider the following collection of lists of integers:

A1=[3 4 9 8 12 15 3 7 13 3]
A2=[15 3 24 3 50 12 3 9 3]
A3=[78 9 3 65 3 24 13 9 3 12]
A4=[3 15 78 3 14 3 2 9 3 44 12 3]

Then the common elements in this collection would be: [3 3 3 9 12] (Note that element 3 occurred at least three times in all the lists so the output must include three copies of 3)

An example of a real world application of this problem is in the National Football League, when teams are selected for the playoffs, there is always the possibility that two or more teams may have the same won-lost-tied record. There is an ordered, step-by-step, tiebreaker process by which such ties are broken. If the first step in the process fails to break the tie, the next step is applied, and so on, until the tie is broken. In the third step of this process the won-lost-tied percentages in common games between the two teams are compared. Two games are considered common between two teams if each of those teams faced the same opponent. So the problem will come down to find the common opponents for the teams.

There are several solutions to this problem. The most straightforward (but inefficient) solution is to designate one list as a query list and then set up a nested loop structure that for every element of the query list performs a sequential search on all other lists in the collection to find that element. If the element is found in all other lists as well, then it is considered a common element. The efficiency of this poor solution is quadratic.

The most efficient solution to this problem uses a HashMap and has a linear running time in the average case scenario.

What you need to do:

1. Create a class called `CommonElements`, to contain your algorithm and associated methods and attributes.

2. In your `CommonElements` class, encapsulate your algorithm within a method called `findCommonElements`, that has the following signature:

```
public static<T> List<T> findCommonElements(List<List<T>> collections){
```

   Note: the input argument to this method is a nested list (i.e., a list of list). The value returned by your method is a list of common elements. Your algorithm for this method must use a HashMap **and should have an average running time of O(nk) where n is the maximum number of elements in each list and K is the number of lists in the collection.**

Attention: You may NOT use the retainAll method in java.utilHashSet. Using this method will result in Zero credit for this problem.

Below is an example of calling your method for a nested list of integers mentioned in page 1 of this assignment:

```
public static void main(String[] args)
    {
      List<List<Integer>>collections =new ArrayList<List<Integer>>();

      collections.add(0,
      new ArrayList<Integer>(Arrays.asList(3, 4, 9, 8, 12, 15, 7, 13)));

      collections.add(1,
      new ArrayList<Integer>(Arrays.asList(15,24,50,12,3,9)));

      collections.add(2,
      new ArrayList<Integer>(Arrays.asList(78,65,24,13,9,3,12)));

      collections.add(3, new
      ArrayList<Integer>(Arrays.asList(15,78,14,3,2,9,44,12)));

      List<Integer> result =findCommonElements(collections);
}
```

What you need to turn in

First problem:

1. A document containing your solution for the first problem

Second problem

2. Your CommonElements.java file (This should contain your findCommonElements method, as stated above)
3. A document containing two things:
    a. **A table that shows your test cases.** This table should have three columns:
        i. The collection of lists you used for testing
        ii. The expected list of common elements
        iii. The return list of common elements by your program
    b. Briefly describe your algorithm and determine the time efficiency of it. Don't just say, for example, that my algorithm is O(kn) (We already know this) but explain how you got that time efficiency.

Grading Rubric for the second problem

Test Data: The test collections you used for your program	**1**
Analysis of time efficiency of your algorithm	4
Functionality: • Uses HashMap and Correctly returns common elements given a collection of K lists • All occurrences of duplicate items common to all lists in the collection are returned	5 5
Total	**15**