CS20B: Homework 6 (60 points)

prof. dr. Irma Ravkic

Questions (20 points)

1. (8 points) **(Hashing from Chapter 8)** Critique the following hash functions for a domain consisting of people with attributes *firstName*, *lastName*, *number* (used to resolve identical first and last names, e.g., "John Smith 0," "John Smith 1," etc.), and *age*. The names are of class String and the other two attributes are of type int.

   (a) hash function returns $(age)^2$
   (b) hash function returns $(age)^2$ + lastName.hashCode()
   (c) hash function returns lastName.hashCode() + firstName.hashCode()
   (d) hash function returns lastName.hashCode() + firstName.hashCode() + number

   Note: answer the question in terms of how good the hashcode is for avoiding collisions (different entries having the same hashcode) and uniquely identifyng the objects.

2. (4 points) **(Hashing from Chapter 8)** Critique the following code which is intended to print out whether or not a key value $k1$ is used in a map named relationships:

```
1 if (relationships.get(k1) != null)
2
3       System.out.println("yes it does");
4
5 else
6
7       System.out.println("no it does not");
```

   Now rewrite the code so that it works correctly

3. (4 points) **(Graphs)** Use the following description of an **undirected graph** for Questions a) and b) below:

   EmployeeGraph = (V, E)

   V(EmployeeGraph) = {Susan, Darlene, Mike, Fred, John, Sander, Lance, Jean, Brent, Fran}

   E(EmployeeGraph) = {(Susan, Darlene), (Fred, Brent), (Sander, Susan), (Lance, Fran),

   (Sander, Fran), (Fran, John), (Lance, Jean), (Jean, Susan), (Mike, Darlene), (Brent, Lance), (Susan, John)}

(a) Draw a picture of EmployeeGraph (draw it the way graphs look like - see the book or slides)

(b) Which one of the following phrases best describes the relationship represented by the edges between the vertices in EmployeeGraph and **why**?

i. "works for"

ii. "is the supervisor of "

iii. "is senior to"

iv. "works with"

4. (4 points) **(Graphs)** Draw the adjacency matrix for the graph in Figure 1. Store the vertices in alphabetical order.

Coding Exercises - 40 points

1. (20 points) **(Hash Maps)** Complete the following exercises in a single project:

   (a) **Implementation**: add and implement *toString()* method to *HMap* (from ch08 code shared with you). Hint: you can use the iterator of HMap to iterate through the entries by using a for each loop. (Needs around 4 lines in toString() method). Your output should look something like this:

```
key1 -> value1
key2 -> value2
...
keyN -> valueN
```

   (b) **Application/Client code**: Create an application that uses *HMap* (from ch08 code shared with you) and reads a list of words from file 'constitution.txt' (shared with you in the homework package) and outputs whether or not any word repeats. As soon as the application determines there is at least one repeated word it can display "<such and such word> repeats" and terminate. If no words repeat it displays "No words repeat" and terminates. Note: make sure your code works! Include the .txt file in your project folder, and use the approapriate path when reading the file, so when I compile/run it I don't have to copy-paste the document myself for every student. (Needs around 13 lines of code in main()). Print your dictionary and see if your toString method works as you wanted.

2. (20 points) **(Graphs)** Class *WeightedGraph* in this chapter is to be extended with **two** methods: 1) *int connects* operation, passed two arguments a and b of type T representing two vertices in the graph that returns a count of the number of vertices v such that v is connected to **both** a and b, and 2) *boolean countEdges* operation that given a node determines to how many vertices that node connects with an edge.

   (a) Implement the method.

(b) Write a driver class that creates a graph structure from Figure 1. Note, the adjacency matrix you did in the questions part might be a visual help for you to solve this exercise. Call your *connects* method on node with values B and C, and the result should be 2 (since nodes B and C connect to both A and D). Call your *countEdges* method on node A (should return 2).

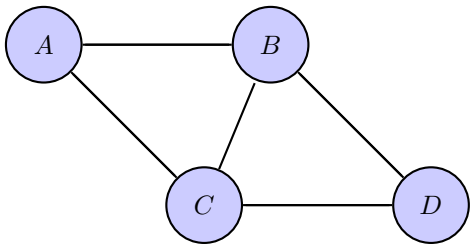

Figure 1: Graph example

3. **(Challenge)** For those of you who wish a slightly more challenging graph exercise here it is:
Class *WeightedGraph* is to be extended to include a *removeVertex* operation that removes a vertex from the graph.

   (a) Deleting a vertex is more complicated than deleting an edge from the graph. Discuss the reasons for this operation's greater complexity. Write the declaration of this method. Include adequate comments.

   (b) Implement the method and test it on the graph in Figure 1. To see if it works you probably should implement toString() method.