COMP 2002
Assignment 4
Fall 2019
Due: Nov. 18, 2019 @ 11:59pm

Notes to keep in mind for assignment questions:

- *You may use code from the labs, textbook website, etc. Where possible, start with already working code/classes and modify them accordingly.*
- *Sometimes it's easier to start coding from scratch; a good developer knows when to choose between using already existing code and writing their own code.*
- *For questions that ask for pseudo-code, you may also use Python instead. But you have to consider whether implementing the code is worth the extra time required.*
- *Read the Assignment Submission details thoroughly, and make sure to follow all instructions.*

1. (50 points) In Stella (Atari 2600 emulator), there is a built-in database containing information about all the games that it supports. This database is stored externally, and is read into the application when needed. The database contains fields as follows:

    - Cart_Name (name of the game)
    - Cart_MD5 (md5 hashcode of the game)
    - Cart_Manufacturer (self-explanatory)
    - Cart_ModelNo (model number of cart)
    - and so on ...

    There are actually many more fields, but we will look at only a simplified version with the fields mentioned above. All fields (and their associated data) are represented by strings. A database text file looks as follows:

```
"Cart_Manufacturer" "Activision, Carol Shaw"
"Cart_MD5" "393948436d1f4cc3192410bb918f9724"
"Cart_ModelNo" "AX-020, AX-020-04"
"Cart_Name" "River Raid (1982) (Activision)"

"Cart_Name" "Pitfall! (1982) (Activision)"
"Cart_Manufacturer" "Activision, David Crane"
"Cart_ModelNo" "AX-018, AX-018-04"
"Cart_MD5" "3e90cf23106f2e08b2781e41299de556"

...
```

    Note that each *record* represents information for one game, and records are separated by a blank line. Furthermore, note that the fields are not always in the same order within a record (ie, there is no guarantee that 'MD5' or 'Name' will be always be on line one, line two, etc; they can be in any order). As well, not all fields will be present in every record (although Cart_MD5 always will be).

    In this question, you will simulate the behaviour of this database by reading the data from a given database text file, and storing the information in a binary search tree, keyed by "**Cart_MD5**".

    Attached is a binary search tree class (**BST.py**) that partially implements the *BST* ADT for storing this data. Some of the methods are already present, but they will have to be modified to work with this data. Others need to be implemented completely. Carefully examine this code. Modify those methods that need to be changed, and implement the ones that have a comment indicating to do so. The comments will outline the expected behaviour of the method, what type of input it expects and what it should output/return. You should keep efficiency in mind when implementing all methods.

    Note that unless otherwise noted, the main method is not to be changed. Your completed class must work with the main method as it already exists. If you change the main method in any way other than what is allowed, you will receive 0% for this question.

    In addition to testing your knowledge of the *BST* and *Map* ADT's, this question also tests your ability to modify an already existing program to adhere to specifications that you cannot change. This is a very valuable skill to have in professional software development.

2. (50 points) Attached are several classes (from the textbook) implementing the standard *HashMap* ADT using linear probing as the collision resolution strategy, the standard hash function from Python, and the MAD compression function. For this question, you will make several modifications to the hashtable code, as follows:

(a) Consider the attached Python file **CyclicLeftShiftTest.py**. Read the comments in this file, and complete the included function as instructed within the code.

(b) Modify the classes to use the left-cyclic-shift function you completed in (a) for *_hash_function*. Convert the code from using the MAD method to the Division method. Make sure not to 'hard-code' the *shift* parameter for your function. That is, this parameter must be able to be changed during a particular program run.

(c) Modify the classes to start with a hashtable size of 2011.

(d) Modify the classes to do *both* linear probing (already present) and quadratic probing, including a way to *toggle* which one should be used for a particular program run.

(e) Implement a method that counts (and keeps track of) the number of collisions that occur each time an item is inserted. Note that you should only count collisions; if an item can be inserted directly into the location specified by its hash function value (and doesn't need to move elsewhere in the table), that does not count as a collision.

Once you've completed the above modifications, you will test your code by reading strings from the attached data file **words** and inserting them into the hashtable. You will do this on multiple runs, and calculate/display how many total collisions occurred, under the following scenarios:

Scenario 1: use linear probing, and shift values from 0 to 16.

- Set your class to use 'linear probing'
- Insert all strings from the data file for *shift* = 0, accumulating the total number of collisions
- Print the *shift* value, and the total number of collisions for that *shift* value
- Start with a new, empty hashtable
- Increment shift by one and then do the same process over again
- Repeat for all shift values from 0 to 16.

Scenario 2: use quadratic probing, and shift values from 0 to 16.

- Almost exactly the same as scenario 1, except set your class to use 'quadratic probing'

Each scenario should output a table similar to the following (note that the numbers here are made up):

```
Shift | Collisions
------------------
   0  |   643
   1  |   497
   2  |   430
   3  |   486
    .
    .
    .
  16  |   492
```

Briefly discuss how the various collision resolution strategies and shift values affect the number of collisions. Which one performs the best?

Submission Details

For Question #	Submit the following (all in one ZIP file):
1	*BST.py* file containing your code, any input files required for program to run
2	All files from the original ZIP download (whether you've modified them or not)