CS3114 (Fall 2014)
PROGRAMMING ASSIGNMENT #4

Due Tuesday, December 9 @ 11:00 PM for 150 points
Due Monday, December 8 @ 11:00 PM for 15 point bonus

Overview:

Note the due date! This is a Tuesday.

For this project, you will reproduce Project 1 with the following modifications:

1. Instead of using an array for your memory manager, its memory will be a disk file mediated by a buffer pool.
2. You will maintain a graph data structure.
3. There is a new command that will compute and print some statistics on the graph.

Assignment:

You will process a series of commands (similar to the commands of Project 1) to store a collection of artist and song names. You will use a memory manager to manage storing the names, with the names themselves stored in a file on disk. This memory manager will be identical to Project 1, except that instead of using an array, you will use the disk file managed by a buffer pool. Be careful that you write out all of the buffers at the end of the program, because your correctness score will depend in part on your memory file exactly matching the reference test's memory file.

Just as in Project 1, you will use two **closed hash tables**, one for accessing artist names and the other for accessing song titles.

The main addition to Project 1 is that you will maintain a graph data structure. The graph will be implemented using the **adjacency list** data structure (see Chapter 14 in OpenDSA). The graph will have a node for each song and each artist, and will link a song node with an artist node if there is a recording of that song by that artist. Specifically, when you process an insert command with artist ART and song SONG, you will do the following:

1. If this is a new artist name or new song name, add new nodes to the graph as necessary.
2. Add a link in the recording graph between the nodes for ART and SONG.

You will need to come up with a reasonable way for expanding the number of nodes in the graph as necessary. Keep in mind that nodes can be removed from the graph as well.

Invocation and I/O Files:

The program would be invoked from the command-line as:

```
java GraphProject {memFile} {numBuffs} {buffSize} {initHashSize} {commandFile} {statFile}
```

The name of the program is `GraphProject`. Parameter `{memFile}` is the name of the file that will store the memory pool. Parameter `{numBuffs}` determines the number of buffers allocated for the buffer pool. Parameter `{buffSize}` is the size for a block in the memory pool (and so also the size of a buffer in the buffer pool). Parameter `{initHashSize}` is the initial size of the hash table (in terms of slots). Your program will read from text file `{commandFile}` a series of commands, with one command per line. The program should terminate after reading the end of the file. The statFile should print out collected statistics just as in Project 3. You should time how long it takes

to process the command file, and then print the statistics (including time required) at the end of the program.

The formats for the commands are identical to that of Project 1, with the following addition:

print `{artist|song|blocks|graph}`

This is identical to Project 1, except for the addition of the `graph` option. When the `graph` option is given you will do the following:

1. Compute connected components on the graph. (You will use the Union/FIND algorithm for this purpose, see OpenDSA Module 13.1.) Print out the number of connected components, and the size of the largest connected component.

2. Compute (and print) the diameter for the largest connected component using Floyd's algorithm for computing all-pairs shortest paths (OpenDSA Module 14.8).

Programming Standards:

You must conform to good programming/documentation standards. Web-CAT will be used to assess a coding style score.

Do not expect a GTA or instructor to help you debug an implementation unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or the codebase provided by the instructor. Note that the textbook code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point.

Deliverables:

You will submit your project through the automated Web-CAT server. Links to the Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by Web-CAT's evaluation of code coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by test cases that are provided by the graders. Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will **not** be given a copy of grader's test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.

When structuring the source files of your project (be it in Eclipse as a "Managed Java Project," or in another environment), use a flat directory structure; that is, your source files will all be contained in the project root. Any subdirectories in the project will be ignored. If you used a makefile to compile your code, or otherwise did something that won't automatically compile in Eclipse, be sure to include any necessary files or instructions so that the TAs can compile it.

If submitting through Eclipse, the format of the submitted archive will be managed for you. If you choose not to develop in Eclipse, you will submit either a ZIP-compressed archive (compatible with Windows ZIP tools or the Unix `zip` command) or else a tar'ed and gzip'ed archive. Either way, your archive should contain all of the source code for the project, along with any files or

instructions necessary to compile the code. If you need to explain any pertinent information to aid the TA in the grading of your project, you may include an optional "readme" file in your submitted archive.

You are permitted (and encouraged) to work with a partner on this project. When you work with a partner, then **only one member of the pair** will make a submission. Be sure both names are included in the documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA.

Pledge:

Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement near the beginning of the file containing the function main() in your program. The text of the pledge will also be posted online.

```
// On my honor:
//
// - I have not used source code obtained from another student,
//   or any other unauthorized source, either modified or
//   unmodified.
//
// - All source code and documentation used in my program is
//   either my original work, or was derived by me from the
//   source code published in the textbook for this course.
//
// - I have not discussed coding details about this project with
//   anyone other than my partner (in the case of a joint
//   submission), instructor, ACM/UPE tutors or the TAs assigned
//   to this course. I understand that I may discuss the concepts
//   of this program with other students, and that another student
//   may help me debug my program so long as neither of us writes
//   anything during the discussion or modifies any computer file
//   during the discussion. I have violated neither the spirit nor
//   letter of this restriction.
```

Programs that do not contain this pledge will not be graded.