UNIVERSITY OF MICHIGAN

EECS 282

Assignment 5 – Using array-based lists for querying IMDB movies list

---

Due Date: Tuesday, Feb. 23rd, 2010, 11:55 PM.

Problem 1:

For each of these, indicate whether the statement is true or false, given the definition of big-Oh in the lecture notes. Remember that you want to look for the fastest growing term.

1. $n^2 + 100$ is $O(n^2)$
2. $n^3 + 1000\ n^2 + 4$ is $O(n^2)$
3. $n^3 + 1000\ n^2 + 4$ is $O(n^3)$
4. $n^3 + n^2 + 4$ is $O(n^4)$. Note that this is a bit tricky. Use the formal definition.
5. $2^n + n^3$ is $O(n^3)$
6. $2^n + n^3$ is $O(2^n)$
7. $n + \log n$ is $O(n)$

Problem 2:

In this assignment, you will be providing a way to find the movies given a particular keyword. The movies are available in movies.list file. You will be extending the assignment in subsequent assignments, so stick to the recommended design. The goal of the assignment is to make use of lists, do some I/O on files, basic string parsing, and write a Junit test. You will also instrument some tests to measure time, so as to get a sense of efficiency of programs.

Your program should consist of four classes:

- **Movie** class: this class stores information about one movie. The information includes the title and the year. You should look at the movies.list file carefully and decide the data type to use for each of these so that you do not end up excluding movies from the search. Keep the attributes of this class private. Provide appropriate constructor to initialize a movie object and the getters to read the attributes.
- **MovieList class:** This class maintains a list of movies. It internally should use an array of movies – you can reuse and modify the code for the list of monkeys in the previous assignment, replacing an array of monkeys with an array of movies, and making any other changes you need.
- **ManageMovies** class: this class uses your MovieList class and provides the following functions:
    - void importMovies(String filename): imports movies from the file with the given filename into the internal movie list (initialize list to empty when importing). If the file is missing, the method should be declared to throw the exception "FileNotFoundException", which is a standard Java exception.
    - int findMoviesByKeyword(String keyword, String outfile): find all movies that contain the given keyword in the title of the movie. Matching movies, along with the year, should be printed to the file with the name "outfile". Returns the number of matching movies.

- **MovieSearch** class**:** this class will only contain static function main, as well as any other helper functions or variables that you need. The main program should use the ManageMovies class to import all movies from movies.list and print out the time required to import the movies in seconds (see lecture notes on how to measure time). The main program should then go into a loop that asks the user for a keyword and then outputs two things: (1) the number of matching movies to the terminal ("Number of matches = …), and (2) the list of matching movies to the file movies.list_<keyword>.txt, where <keyword> is the keyword entered by the user. It must also print out the time required for the query only to the terminal as "Query time = …. seconds".

  The loop should terminate when the user hits "Control-D" (on Mac OS) or "Control-Z" (on Windows) to indicate the end of user input.

- **MovieSearchTest** class: This is a **JUnit** test that should contain at least two tests.
  - **1st test** loads in the movies.list file, invokes findMoviesByKeyword("casablanca", "outfile") and checks that the number of matching movies is correct.
  - **2nd test** attempts to load a missing movies file. You should confirm that FileNotFoundException is generated. Else, the test should fail.

  Note that on Mac OS and Linux, You can use

  ```
  grep –i casablanca <moviefile> | wc -l
  ```

  to get the number of matching lines – that should match the number of lines from your program (unless the word occurs in the first few header/comment lines in movies.list). `Grep` outputs lines that contain Casablanca (`–i` means case-insensitive match) and `wc –l` counts the number of lines in the line. On Windows, you can download TextUtils from Gnu to run the same commands.

Challenges in this assignment

1. You will find that there is sometimes incomplete information about movies. Some movies may have missing years, for example, or unknown years. So, think about how you will handle that.
2. The importMovies function needs to handle vagaries of the input file. For example, the initial lines up to and including "==========" should be ignored. You should also ignore any blank lines (or lines with only whitespace). If a line.trim() is same as an empty string, then the line can be skipped.
3. For keyword search, first do a simple string containment search for this assignment. Ignore case. This is equivalent to doing a partial match search. For example, if you search for "casa", it would match "Casablanca".
4. For 10% extra credit, you are welcome to explore the design of exact keyword search, where the keyword must be a whole word. In that case, your results should match the results of "grep –w -i <keyword> | wc –l", on any word that is not in the ignored lines at the beginning of the file. You should print out the results to a file movies.list_<keyword>_full.txt. Note: You still have to do the basic search. We expect to see a file movies.list_<keyword>.txt, as well.
5. For 10% off, you can use the Java's standard ArrayList class instead of your version of MovieList class.
6. It may help to work with a small version of movies.list file, initially, that contains, say, a few hundred movies or so.

What to submit

Submit the five Java class files, the text output from running the unit test (hopefully, showing that your program passed the test), and the text output that shows the results from the run of main, showing the number of matches and the execution times for three queries, one of which should be on the word "casablanca" (the other two words are your choice).