CSE250, Spring 2014 Project 2 (Assgt. 9) Due Fri. 5/9, 11:59pm*

Data Structures for the "Netflix Challenge"

`Groups of 2, online submission only---no hardcopy`

The Netflix Corporation ran a $1 million prize competition in 2006–2009 to devise a system for recommending video movie selections to particular users, based on what kinds of movies the user (and people of similar demographic profile) have rated highly in the past. This project concerns the object-oriented class-modeling and data-structure choices that would underlie a submission to such a competition. Accordingly it does not emphasize the sophisticated mathematical modeling that went into the winning entries—rather we implement a simple and straightforward averaging model. The task and model are as follows:

1. Read a list of reviewed movies, storing an object for each one that gives its index, title, year, and a set of kinds called *genres* that the movie belongs to.

2. Read a long file of user ratings of these movies. Each rating gives the (index of the) movie and a 1–5 star value. Each user u has an index and a set of ratings he/she has submitted.

3. For each movie m, compute r_m = the average star rating by users who rated that movie.

4. For each user u and *genre* g, compute u_g = the average rating $u(m)$ by user u of movies m that belong to *genre* g.

5. Most important, for each user u and genre g, compute the user's "preference factor" $p_{u,g}$ for the genre as u_g divided by the average star-rating for those same movies by users overall.

6. Then a preference factor $p_{u,g} > 1$ means that user u rated movies in genre g higher than users overall, while $p_{u,g} < 1$ means u gave overall lower ratings to movies of that kind.

7. Finally, for each user u, find the top 10 or 30 or 50 or so movies m that maximize $r_m \times p_{u,g}$, where g is the genre of movie m. (This is complicated somewhat by the fact that a movie m can fit more than one genre, such as being an action mystery film.) These are the movies that Netflix would choose to feature in ads on the user's online account page and/or flyers that come with mailed DVDs.

8. (Possible other questions / extra credits may require compiling other lists.)

To spell out the mathematical model more formally, let M_u denote the set of movies rated by user u. This need not be the set of all movies—indeed it will be much smaller but not tiny. Then for each genre g, let $M_{u,g}$ be the subset of M_u consisting of those movies rated by user u that belong to genre g. Then we can calculate

$$p_{u,g} = \frac{\sum_{m \in M_{u,g}} u(m)}{\sum_{m \in M_{u,g}} r_m}.$$

Note that if we divide both numerator and denominator by the size k of $M_{u,g}$, then the numerator would become the quantity u_g defined above, while the denominator would become the overall average star rating for those movies. Since these factors cancel, this equation gives an expedited way to compute the user's preference factor for that genre.

Task 7. changes the simulation from the competition scenario to how we think Netflix would behave when recommending movies overall, not just "new releases on video." In the competition,

there were separate sets of *training data* and *prediction data*. That could be done here if time and interest allow, but our setup is closer to projects that have been given in the recent past. We also condense the *genre* division from 18 to 6 categories as specified below.

1 Division of Labor and Ground Rules

The project is designed for teams of 2. Each team member works *individually* on one of the data structures—here the "moviebase" of all the movies or the "userbase" of all the users and their ratings. The code must be organized so that these bases are modeled by classes that have container objects, where most of the file-reading and most of the above calculations are carried out by methods of these classes. Then the team members may *collaborate freely* on the client file with `main` that drives the simulation. You are also encouraged to lighten the file-reading methods by extending the `StringWrap` class with some parsing functionality (as opposed to deriving from `string` to add methods, which is forbidden: the watchword is "use composition/delegation, not inheritance"), and if so, may work collaboratively on that. There are 105 individual points and 105 collaborative points, for 210 total points. [There will be one more hardcopy homework assignment, with one question about hash tables and one about red-black trees, plus possibly an extra question related to this project that would constitute an *individual* report question, and effectively bring the points up to 240. Whereas, the report questions given below are collaborative.]

Except where you might make a template class that is general enough to be re-usable outside this application (such as a heap or tree class, in which case having just a `.h` file is expected), your classes must follow the `.h, .cpp` convention. The files for the movie-related classes must have `Movie` somewhere in their names, and end with the movie-base person's initials (only), followed by `.h/.cpp`. The user-related classes must similarly have `User` somewhere and end with the user-base person's initials. The client file, makefile, and any utility-class files such as for an edited `StringWrap` class, must have both sets of initials, movie-person first, OK to just ram the initials together.

*Checkpoint (Mon. 5/5, 11:59pm): `.h` files of the movie-base and user-base related classes, with `/**...*/` style comments on each method explaining what you intend it to do, 15 individual points. Detailed feedback will be given only if we feel that changes need to be made, for instance if the desired functionality is not easily realized with the fields and declarations and data structures as given. Free use may be made of C++ standard-library data structures and routines; not just `vector` but possibly also `set`, `map`, `priority_queue`, and algorithms such as `sort` and `make_heap` from the `<algorithm>` header and others, for which documentation can be found on the course resources page as well as tables in the text itself.

2 The Data

The original Netflix data set was withdrawn on legal breach-of-privacy grounds when it was discovered that many users could be identified because they had posted the same star-ratings for the same movies on the *Internet Movie Database* (IMDB.org). We are using a publicly-downloadable substitute data set from the *GroupLens* Research Project at the University of Minnesota (`http://www.grouplens.org/`). We are using their "100k" data set, which is described at `http://www.grouplens.org/system/files/README_100K.txt` (the larger data sets there use slightly different formats).

The data sets are provided "as-is"—note that they are designed to minimize transmission size

rather than maximize user-friendliness. The "users" have only numbers, from 1 to 943. The 1,682 movies also are handled as numbers, though corresponding titles and years (which you should store in your class objects) are in the file `u.item`, along with the all-important *genre* information. The genres for each movie are given as bitstrings with each bit separated by a vertical bar, 1 if the movie belongs to the genre with the corresponding index, 0 if not. The index for each genre is in `u.genre`. We, however, will "condense" the genres into the following six named categories:

(a) Action: Action, Adventure, Thriller, Western.

(b) Noir: Crime, Film-Noir, Horror, Mystery.

(c) Light: Animation, Children's, Comedy, Musical.

(d) Serious: Drama, Romance.

(e) Fantasy: Sci-fi, Fantasy.

(f) History: War, Documentary.

There is also "unknown," but only one movie seems to have it, and we can ignore it.

Last and most, the file `u.data` has 100,000 user star-ratings. This is a tab-separated file with fields:

```
[user#]    [movie#]    [rating 1-5]    [UNIX timestamp]
```

We can ignore the last field, which gives the date and time as the number of seconds since 1/1/1970. This file is just under 2MB, with the other files adding about 300K, so there should be enough quota for your private use of these files. They, and some minor associated files and a `README`, are in the directory `~regan/cse250/PROJECTS/PROJ2/`. *As the readme-info states, there are some typographical errors in the files, which however have been judged not to mar projects like this one, so dealing with them is your normal responsibility.*

3 Report Section

In a separate file `Proj2reportMMMNNN.txt`, please answer the following two questions, for 15 pts. each:

(a) Describe at least four choices (can be two per person, but AOK if the reasons related to what your partner was doing) you made in designing the movie-base and user-base related classes, in terms of data structures and responsibility for carrying out certain functions. In each case, mention an alternative you considered, or could have considered, and why you chose your representation.

(b) Describe the efficiency of your simulation in terms of the numbers M of movies, U of users, and R of ratings. Use O, Θ notation, and explain why. You may suppose that the sizes of constants, such as the user and movie indexes, are $O(1)$ rather than $O(\log M)$ or $O(\log U)$, because they fit within the machine's wordsize.