Craig Lombardo

CS150 Data Structures and Algorithms

Professor Liew

Project 2 Write up

4/12/15

1. Introduction:

One of the biggest problems that chef's face today is managing all of the recipes that they have, it's rather difficult to keep track of all of the different recipes they may have to make. This project focuses specifically on creating a cookbook that will allow for efficient an accurate storage of recipes. The challenge here comes in processing customer needs, given dietary restrictions or preferences; one customer may not be able to eat the food that the prior customer could eat. When there are only a few recipes, this task is simple; however, most restaurants (most) do not have small menu sets, they offer a wide variety of options. For the purposes of this project we are trying to build a cookbook that can handle these needs. For this project, we will be designing, implementing and evaluating an electronic cookbook. The cookbook will be used to organize recipes and prepare meals. The cookbook will give the user a few abilities, to add a recipe, to delete a recipe, to find a recipe and finally, plan a meal. There are a few parts to each recipe, cuisine type (Italian, Chinese, Greek, Turkish, Indian, Pakistan, French, Korean), name, prepTime, cookTime, a type (salad, entrée, appetizer), and an ingredient list (main, addons and sides). The end goal of this program is to store recipes with all of this information and be able to process it and depending on the user's inputs, act accordingly

and correctly. The user may add a new recipe, delete an existing recipe, find a recipe or plan a meal that best matches their criteria of: total time, inclusions, exclusions, and cuisine type. For the purposes of this project I have assumed that the user may not create their own course or cuisine type (i.e. they may not extend to another region). I have assumed that the user may or may not be competent and in so I have made a small system of checks and balances in an attempt at early warnings for invalid user input. By this I mean I have made a simple method that attempts to notify the program that it is an impossible search.

2. Approach:

For the purposes of this project I decided to break things up into multiple parts, by this I mean my methods for the most part were broken up into smaller parts. There are a few methods that are a little lengthy; however, I tried to keep my methods small and private, as the user does not need access to certain things. I have restricted access to several methods because they are designed to take only certain inputs, and the methods before them ensure certain inputs are fed through, this ensures (hopefully) that there will not be unexpected or fatal errors. The project is broken up into two databases, one that is "smart" and one that is the "master". The master database is simply used if the user wants to delete something, as they will pass in a name and nothing else, since no two names can be alike it is a rather simple and easy search. The smart database however has to do a little more thinking, hence the name. The smart database is used for finding and planning a meal. The smart database will take the user input, analyze it and break it down into increments, narrowing the search based on the inputs it was given, if the field is blank

(excluding the time field as it is not allowed to be empty) then it will do it's best to figure it out given all possible forms of the unspecified category. In the worst case the program has to look through every element sorted by total time to make; however, that is only if no inputs other than time are passed in. I will now discuss the class breakdown. The CookBook class is the overarching class that brings everything together. The DataHandler class is used to read in and write out the recipes, the read in is from a .txt file, and the output is also a .txt file, but it is created based on the recipes found in the master database. The FoodRules class is used to impose a ruleset on the various classes, labeling what is and is not a valid type. Valid type in this context is variant, as it will be used for validity of everything from a certain food type to cuisine type. This rule set may be called and referenced by the other classes. The InputReader class handles the actions specified by the user, whether it be add, find, delete, plan or quit. It handles all of the prompts and prints out all of the results, if there are any, or prints error messages. The MasterDatabase is the database that contains one copy of each recipe and is sorted by name, it is used primarily for delete and writing out to the file (DataHandler uses this to ensure only one copy of each is written out). MyStack is a class that is a LinkedList implementing a LIFO ordering. The Recipe class acts as a recipe in our cookbook, storing pertinent information. RecipeTreeNameComparator is used for exactly what it's name implies, the comparison of two recipes names to determine which is greater. SmartDatabase is used for the more difficult searches in which the user doesn't know exactly what they want.

3. Methods:

For the purposes of this project I decided to run things with varying recipe file sizes and various searches for each size. I tested most of the ingredients from the provided ingredient list and all of the different options from the definitions list (I may have missed one in testing). I also did my best at putting these parameters in for both inclusions and exclusions. I experimented with different courses, and asserted that the program does not allow me to choose a course that was not on the list given. Time was also tested and I asserted that the input must be a valid integer, no strings are allowed. I decided to conduct my tests by starting with small sample sizes and testing different variations, and then applying those same options to the larger set of data.

4. Data and Analysis:

For the analysis of my data I have compared only a few things, as I was not sure what to compare other than compile and search time, and the pre-specified things. The compile time increases as the number of elements also increases; however, the 60,000 elements list does not work. As far as correctness goes, the program is not always accurate when it comes to finding and planning. The program can add and delete things correctly, however, the find is often off by a few minutes, it has consistently been correct in regards to the inclusions and exclusions as well as type; however, the time part of it is often off. When creating a meal plan it will be off even more as it is set to try and find a meal plan that will consist of 4 meals that attempt to divide the total time by 4. I am aware that this is not entirely correct and is a simplification, I simply ran out of time. Just like the find, the inclusions are all there and the exclusions are not. It is important to note

that for planning a meal (despite the prompt) you cannot choose meat, vegetarian etc. That being said, find has this functionality working, and both work for this case with exclusions. When finding a recipe with given inclusions, the returned recipe must have all of the ingredients the user specified as inclusions and none of the exclusions. For plan, all ingredients must be accounted for in some part of the recipe, but no parts of the recipe may have these exclusions. Search time is seemingly just as fast whether it is a small data set or the 40,000-element set and there is no noticeable difference for find or plan. Adding, deleting, and quitting follow this same pattern, size does not seem to make much of a difference.

5. Conclusion

Based on these results it is somewhat tough to say whether or not the program does a good job. It is always fast; however, it is not always accurate. I honestly do not know where things went awry but nonetheless I will take a few minutes off for total time over giving someone a meal that they did not ask for. In my opinion even though the functionality is not 100%, having the accuracy with the inclusions and exclusions really does make more of a difference as in the real world, when preparing/ serving food, people would most likely rather wait another minute or two rather than remake everything from scratch.

6. References:

"Binary Search Algorithm." *BINARY SEARCH ALGORITHM (Java, C++)*. N.p., n.d. Web. 12 Apr. 2015. <http://www.algolist.net/Algorithms/Binary_search>.

CS 150: Guidelines For Writing Lab and Project Reports: https://moodle.lafayette.edu/pluginfile.php/151579/mod_resource/content/1/write up-guidelines.pdf

CS 150: Project 2: https://moodle.lafayette.edu/pluginfile.php/159502/mod_resource/content/3/p2.pdf

"Java Platform SE 7." Java Platform SE 7. N.p., n.d. Web. 12 Mar. 2015. <http://docs.oracle.com/javase/7/docs/api/>.

Savard, Jean-François. "Iterate through a HashMap." *Java*. N.p., n.d. Web. 12 Apr. 2015. <http://stackoverflow.com/questions/1066589/iterate-through-a-hashmap>.

Weiss, Mark Allen. *Data Structures & Problem Solving Using Java*. Boston: Pearson/Addison Wesley, 2010. Print. Fourth Edition.