Craig Lombardo

CS150 Data Structures and Algorithms

Professor Liew

Project 2 Write up

5/9/15

1. Introduction:

One of the biggest problems that restaurant and business owners face is that it is rather difficult to keep track of all of the different ingredients they need, and where is the most cost effective to buy from. This project focuses specifically on analyzing the effect of distance from one location to another and the “going rate” at a given farm. Many of the better restaurants often advertise that use locally sourced ingredients in their cooking. This asserts that the total cost of a meal from said restaurant depends on a few things. As mentioned before, the first hurdle is the price that the farm charges for such an ingredient as well as the cost of transportation from the farm to the restaurant. For the purposes of this project we were given data on a list of restaurants, one pertaining to a list of farms in the area as well as the products the sell and finally, an interconnection graph, which shows the connecting roads between restaurants and farms. The program will be set to read in from text files, it will have a recipe.txt file which will have the recipes contained in a database. It will also have a restaurants.txt, farms.txt and a connectivity.txt file for the list of restaurants, farms and connections respectively. The program will also be able to prompt the user for a cost per distance (ie cost per unit distance along a road from the connectivity graph), prompt the user for the number of meals they want to prepare, and last but certainly not least read in a plan to generate a meal plan. Upon receiving the parameters for planning, a meal can be created and then the program will print out the minimum cost to acquire all of these ingredients and will also print out the paths taken to get everything. The goal of this project is to become more comfortable with graphs and effectively implement Dijkstra's algorithm (AlgoList.com, 2015) along with logic to make a program that can evaluate an optimal route. For this project I made the assumption that the information being pulled from my previous project worked correctly.

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 maintained the specs from the last project, making minor improvements to allow for new inputs required by the user, for example the need for a gas price prompt and likewise for the meal count prompt. 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. This project is broken up into two main parts, one of which handles the searching algorithm end, and one that handles the meal optimization part of things. For the purposes of this project I have decided to only use the “smart database” from the last project as that is the database used for planning, the other database was only convenient for quick and direct searches. I have also removed features such as adding and deleting as well as finding a recipe as for the purposes of this project it is superfluous and will not be used. 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 then it will do it’s best to figure it out given all possible forms of the unspecified category.

The Optimizer class is the overarching class that brings everything together. The DataHandler class is used to read the recipes, the read in is from a .txt file. 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 foods to cuisine types. This rule set may be called and referenced by the other classes. The InputReader class handles the actions specified by the user, whether it prompting the user to define constants, plan a meal or quit. It handles all of the prompts and prints out all of the results, if there are any, or prints error messages. 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.

That first part pertains entirely to the population of information and data handling, the second part deals entirely with the graph and it’s inner workings. Database is the first class and is essentially this half of the programs overarching class. In there the graph is actually created, connections and lists are stored, and that is where the shortest path logic is implemented. Next we have the Junction class, which acts as a road junction or intersection. At every Junction there is either a Farm, a Restaurant, or neither. Farms are where we will get all of our items. Each Item has a name and an associated price with it and a collection of these are stored in each Farm. Restaurants are only signified separately as even though they do not hold any information in them they make the logic a little easier to break down. From these classes my program is able to build up a graph with interconnections from one Junction to another and figure out what route is most cost effective.

3. Methods:

For the purposes of this project I decided to run things with varying recipe file sizes and various. I experimented with different meal requirements and exclusions, and asserted that the program provides me a meal plan that does not violate the parameters that were initially given. Given time was not an aspect of this project I could not evaluate the time parameter of the plan method, which is why I completely removed that function prompt, thus trimming my testing compared to last Projects testing regiment. I also tested the power of my shortest/best path logic by varying the number of meals and gas price as they affected the distance I would be willing to travel to get food/ the tipping point based on how many meals I needed to buy. I varied the number of meals by increasing one meal at a time to see how it affected the average price, as it means I most likely would not travel as far unless it guaranteed a cheaper price based on my total cost. I chose to increase each one one at a time, keeping the meal requirements the same but increasing the number of meals/the gas price incrementally. I also determined that there becomes a gas price by which it makes no difference as the gas cost already outweighs the cost of the ingredients.

4. Data and Analysis:

For the purpose of this project I kept my algorithm use to a minimum to avoid overcomplicating things. I tried keeping my methods small yet making them very powerful as I figured the longer the algorithm the higher the chance for something to break. The biggest factor in my success was by far the use of Dijkstra's algorithm (AlgoList.com, 2015). The reason I say Dijkstra's algorithm (AlgoList.com, 2015) was the biggest factor is because it is the backbone to my program and was used in modeling my logic. For the purposes of my design I decided to essentially build a very small scale version of Dijkstra's algorithm (AlgoList.com, 2015) in which there is a starting node and instead of looking for a certain node the program instead looks for a certain subset of ingredients. As it looks at each farm it checks to see what ingredients the farm has that have yet to be collected, and from there compares it to the current minimum average cost. I decided to use average cost because I wanted to save on gas, I figured if I can always purchase the most possible ingredients for the least average amount of money then I will, hopefully, always save on fuel as I should have to make less trips. I am aware that this is not always the case; however, the average price is also compared to the price in gas it took to get there, if the gas price is overpowering then I go with the closer option, but if for a little farther distance I save in the long run then I will definitely take the longer path that will get me more ingredients. I would like to digress for a second to discuss a flaw with this method. When a farm is visited, if it has duplicates of an ingredient (for different prices) then the program will count that ingredient 2x the amount it actually should have. This is a bug I happened to notice while testing; however, due to time constraints unable to fix. The program is “smart” enough to take the lower price yet unable to distinguish between the two as I have made a logical error somewhere. The algorithm performed its job by getting me a cheap price on my food costs; it was not perfect as it often made multiple “trips” to the same farm to get food in increments rather than collecting them all at once. I am not 100% sure why this was the case; however, it still knew enough to get all of the ingredients before actually leaving the farm it was currently on.

This graph displays the price per unit cost versus meal count. This data was taken consistently for finding the price of Salmon

For the purposes of this project I assumed that all of the prior requirements truly did hold up in this project as well. To this extent I mean everything from the meal plan acquisition to the list of potential ingredients that could be found at the farms remained the same. I fully understand that not all farms will have all ingredients; however, I assumed at least one farm will contain an ingredient in the list and all ingredients can be found somewhere.

As far as the quality of the solution goes it is a bit difficult to say as I have recognized there are flaws and I do not believe I could come up with a 100% perfect solution; however, I am happy with what I have put forth. I, as in the last project, cannot find a meal plan with generic inclusions such as meat, seafood, vegetarian etc. however, it works for specifics like beef, lettuce etc. (both work for exclusions). With that being said I would say my answer is pretty good, I may have lost a little bit of functionality and I am overpaying in a few instances (speculatively) but all in all the solutions follow the inclusions and restrictions and give me a generally low price which is great.

As far as the logic portion of my program goes I am fairly happy with it. I have tested it in various cases and have checked the different effect that meal and gas prices have on the result. When gas prices are extremely high the program tries to stay local, and as we approach free gas we get the overall lowest price, which is very nice. The size of the graph/ the number of restaurants and farms does not seem to have too much of an impact on the performance.

5. Conclusion

The only 2 people I really spoke to about this project were Matt Beck and Greg Flynn. Based on the results of my program I honestly believe my program does a relatively good job at finding a cheap solution. Things were not perfect but it got the job done!

6. References:

"Dijkstra's Algorithm in Java." - Algolist. AlgoList, n.d. Web. 09 May 2015. <http://www.algolist.com/code/java/Dijkstra%27s\_algorithm>.

CS 150: Guidelines For Writing Lab and Project Reports: https://moodle.lafayette.edu/pluginfile.php/151579/mod\_resource/content/1/writeup-guidelines.pdf

CS 150: Project 3:

https://moodle.lafayette.edu/pluginfile.php/163323/mod\_resource/content/2/p3.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/>.

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