CSC111 Project Report:
Look and Cook
Find Recipes Using Ingredients in Your Kitchen

Dana Al Shekerchi, Nehchal Kalsi, Kathy Lee, and Audrey Yoshino

Friday, April 16, 2021

Introduction: Problem Description and Research Question

With the global rise of COVID-19 cases last spring, the lockdown and increasing unemployment rates forced many to resort to exploring hobbies and interests to occupy their free time [1]. Where staying at home was mandatory, people tended to alternate rooms within their homes rather than touring the city. One's second favourite place after their bed was the kitchen. People started to try out new recipes to put their cooking skills to the test. Cooking allowed one to fulfill their boredom, improve their skills, and maybe get a good meal out of it–in fact, in a study conducted by Bloomberg in 2020, 43% of Gen Z respondents planned to cook at home more after the virus [2].

The endless number of complicated recipes found online drove our motivation to create an application that endangers the misery of not having the right ingredients–an issue exacerbated by pandemic-induced grocery shortages. Not all of us are willing to dedicate our time to a meal we don't know will be tasty. Our application will ameliorate these problems by allowing its user to input the ingredients they crave or have, and the time they have to create their ideal meal. In doing so, users may step out of their culinary comfort zones by creating something new out of their everyday ingredients. Once the user inputs the ingredients they already possess, they will get all the recipes that meet their criteria extracted from a multitude of renowned cooking sites.

Our goal is to create an intuitive, interactive program that outputs appropriate recipes given a users' inputs for ingredients and cooking time.

Dataset Description

There is one relevant dataset that we made use of for this project. This dataset is from Kaggle's "Recipe Ingredients and Reviews" [3]. There are four CSV files, viz., `recipes.csv`, `reviews.csv`, `clean_recipes.csv`, and `clean_reviews.csv`. The primary separator of these files is a semicolon (;). The recipes dataset has ten columns representing various recipe attributes to organize data and includes over 12200 rows constituting recipes. For computations, we mainly depend on the columns `ingredients` and `total_time`. However, to present our findings, all the columns are used. We also utilize data from both the reviews files to calculate average rating for recipes and display user reviews alongside the recipe directions.

Computational Overview

Data Cleaning and Reading

Our project mainly draws data for recipes from the file `clean_recipes` which contains ten columns representing various recipe attributes to organize data and includes over 12200 rows constituting recipes.

To convert this data into usable format, the file is passed into `read_recipes` function in the file `data_reading.py` which returns a dictionary of recipe ids mapping to the other attributes of the recipe. This dictionary is then used in various other functions and methods, including constructing a graph.

The ingredients in the file `clean_recipes.csv` are not clean either; they include various terms to specify quantities and include some other redundant words, such as, "English bread" or "2 cups of milk". In order to clean them, a function called `clean_ingredients`, also in the `data_reading.py` file is employed which is a mutating function.

Apart from this, to retrieve review scores and user reviews from the dataset `clean_reviews.csv` and `reviews.csv` respectively, two functions, viz., `get_review_scores` and `get_reviews` are called accordingly.

Data Classes and Graph

The graph used in our search algorithm is created using the file `data_type.py` and utilizes the data extracted by the `data_reading.py` file. The graph file uses the `_Vertex` and `Graph` classes from lecture with some adjustments to tailor the functions to our program.

The `_Vertex` class represents a vertex in the graph with three instance attributes: `item` and `neighbours` are found in the `_Vertex` class as discussed in lecture, while `kind` is an additional instance attribute (similar to that found in assignment 3) which assigns a vertex to be either an ingredient or a recipe.

The `Graph` class is again similar to that learned in lecture, however adjustments have been made to accommodate the new instance attribute `kind`. The graph methods consist of `__init__, add_vertex, add_edge, adjacent, get_neighbours` and `get_all_vertices`.

Finally, the `load_graph` function loads a recipe graph according to the provided `recipes_file`. This function calls the `read_recipes` function from the `data_reading` file and iterates through the provided dictionary to load and return an ingredient graph.

Computation

A GUI is presented to the user where the user picks the ingredients or, if they wish, stipulate a maximum time for all the recipes they intend to receive. The number of ingredients can be anywhere from one to ten. In order to implement this GUI, we used the library PyQt5 as exemplified in the files `main.py`, `ingredients_dialogue.py`, `recipes_dialogue.py`, and `display_recipe_dialogue.py`. As discussed in our project proposal, we planned to use the button and window manager modules provided by Tkinter, however, as further explained below, PyQt5 offered many modules and functions Tkinter did not. We specifically took advantage of the auto-complete search widgets used to input and search for ingredients on our second window.

The selected ingredients, along with the graph and data for recipes (complying to the format the function `read_recipes` returns data in), are passed to the function `ingrdnt_sort` in the file `sort_srch_rslts.py`. This function runs an algorithm to find recipes containing most ingredients out of those specified by the user. The recipes are then returned in a sorted decreasing fashion w.r.t. the number of ingredients being used.

Note that the recipes returned would contain at least one of the ingredient chosen by the user but also may contain some extra ingredients. This is to ensure that maximum possible recipes are presented. For instance, if seven ingredients are selected, then top few recipes might contain all seven ingredients but also a few others which might or might not be main ingredients of the recipe such as salt, pepper, lime juice. This is to guarantee that a recipe is not left out because the trivial ingredients were not selected.

Further, instead of indicating ingredients, if the user chooses an upper bound for the maximum time of recipes, the data for recipes (complying to the format the function `read_recipes` returns data in) is passed to another function called `time_sort` which, as the name suggests, sorts recipes in increasing or decreasing order of the total time of the recipes depending on how the user wants to view them. If no maximum time is stated, this function just returns the sorted recipes.

Algorithm used by `ingrdnt_sort`

The function `ingrdnt_sort` takes in three parameters, viz., the recipe data in the same format as that returned by `read_recipes`, the user-specified ingredients, and the graph.

The function picks one ingredient at a time, utilizes the graph to get a set of all adjacent vertices (neighbours) of that ingredient vertex, which are recipe ids, and add them to a loop accumulator: a dictionary mapping recipe id to the number of times each recipe occurs. As the loop for ingredients progresses, the occurrences of recipes also get updated to reflect that.

After the termination of the loop, the dictionary (loop accumulator) is sorted in a decreasing order of the occurrence times. Then, list of tuples, where the first element is the recipe id from the sorted dictionary and the second element is a list consisting of all the other recipe attributes referring to the corresponding, is returned.

Instructions to Obtain Data Sets and Run Program

a. Install Python Libraries

In order to be able to run our program, all of the Python libraries listed under `requirements.txt` must be first installed. No other preparation is deemed necessary.

b. Download Datasets

The datasets employed in our final submission can be obtained from Recipe Ingredients and Reviews. [3]

These files, in a compressed format, have also been shared with the CSC111 course email address via UTSend. Storing all data files under a folder named "data", which is then nested inside another folder containing all python files (that is, a folder containing Python files and another folder called "data" containing the dataset files) is vital for experiencing the program's expected functionality (Fig 1). The following is the claim ID and passcode to access compressed files on UTSend:

- Link to UTSend: https://send.utoronto.ca/pickup.php
- Claim ID: erNiiSgFy5bRmKer
- Claim Passcode: uG8VEH3ProU3qwMG

Please also ensure that `title_image.png`, `leaf.ico` and `creator_image.png` from MarkUs is stored in the same folder as the `.py` files.

c. Running `main.py`

Upon running the file `main.py`, the user is prompted with a yellow window (Fig 2). The window consists of the title of the project: "Look and Cook" and a `Start` button. On clicking the `Start` button, another window pops up which contains a list of all ingredients, 10 fields for ingredients, one input field for maximum time, if any, and a couple buttons to achieve varied functionality, (Fig 3).

The user has two choices: either to search recipes based on the ingredients they provide, or to filter recipes subject to total time. Both the options can be found on the current window.

If user wishes to perform an ingredient-specific search, they input the ingredients they have in their kitchen in the ingredients fields on the left-hand side of the window. To facilitate effective search, these fields provide assistance in filtering out the ingredient names containing letters typed into the field, similar to auto-complete. A maximum of 10 ingredients can be added on using the `Add ingredient` button at any particular instance. In case of wrong or mistaken ingredient being picked, the `Remove ingredient` button may be used to delete the ingredients in a "Last In First Out" fashion. One might also employ the `Clear` button to reset all progress and start afresh.

In case of a total time based filter on recipes, the user may choose to stipulate a maximum total time in minutes for the recipes they intend to receive. If a maximum total time is specified, they receive recipes satisfying the maximum time criteria. If no maximum time is stated, all recipes, sorted, are returned.

Upon satisfaction with their inputs, the user clicks `Submit`, which initiates computation. The user is then presented with a new window containing the names of recipes carefully picked as per their requirements. The further down the list, the fewer ingredients are used out of those provided. Note that the top 100 recipes satisfying the criterion are shown, else the list would be too long and also, the later recipes are not too significant. The sorting and filtering conditions for recipes too are visible on this window. At this point, either a recipe can be chosen to view directions for, or they can go back a step to edit their inputs (Fig 4).

If a recipe is selected, a new window with the name, ingredients, directions, and time for the recipe as well as the author's name pops up.

[4]

Changes to Project Proposal

In our former project proposal, we stated we would use trees to filter and search for recipes based on user-specified ingredients. After further discussion among our group based off feedback from the TA, we formulated a new search algorithm using graphs. Rather than assigning ingredients to nodes in a tree and creating leaves comprised of recipes using the provided combinations of ingredients, we chose an algorithm that required us to load a graph using `clean_recipes.csv` containing vertices whose items were either recipes or ingredients. In this graph, recipes are adjacent to the ingredients they utilize and, as a result, all recipe vertices are adjacent to only ingredient vertices and vice versa. Implementing a graph in this way allowed us to refine our search algorithm by searching for recipe vertices that contain the user-specified ingredients in their set of neighbours, essentially traversing the graph, as suggested. This implementation relies on the graph in our search function, as opposed to our former implementation, which did not necessarily require the tree to conduct our search.

Additionally, we chose to provide users with reviews in our visualization. When provided a list of possible recipes, users can now see a given recipe's rating out of five. The same Kaggle dataset we used for the recipes provided a review's csv file, named `clean_reviews.csv`, from which we extracted review data.

In addition to sorting recipes by the number of user-specified ingredients they use, we have provided users with the option of sorting by cooking-time. Users are given a drop down menu on the recipes page from which they can choose their desired sorting method.

Finally, we are no longer using Tkinter for our graphical user interface and instead are using PyQt5 because of the various limitations Tkinter has.

Discussion

Our computational exploration and final program achieved our project goal of creating an interactive interface allowing users to search recipes based on ingredients they input. We were able to use a graph and apply our learning from this semester to implement our search algorithms (by traversing said graph), while simultaneously implementing new python modules like PyQt5 for the interface. The interface is intuitive and the program itself is applicable to everyday life–especially to prevent unnecessary grocery store trips in the midst of the pandemic. In addition to achieving our project goal, we were also able to implement additional features we had not previously planned for including extra sorting options for the recipes as well as reviews (including a user interface for these reviews).

Limitations & Obstacles

The dataset we selected to extract recipe and ingredient data from provided two recipe files: `clean_recipes.csv` and `recipes.csv`. However, upon further inspection, we noticed the `clean_recipes.csv` file still contained ingredient quantities, numbers, and incorrectly placed strings that needed additional cleaning up. There were some challenges

in implementing this code as it was difficult to remove and account for all of the random inconsistencies. Due to the size of the dataset it was infeasible to manually clean up inconsistent ingredients, resulting in a relatively convoluted function. This, coupled with the difficulties that accompany mutating a set, made cleaning the dataset an additional obstacle we did not plan for and was ultimately a shortcoming in the one we selected.

In our project proposal we stated we would use the Tkinter library to create our GUI. The library itself was relatively straightforward and some of our group members had experience with it's functions and modules. However, this library failed to offer the functions and modules necessary to accomplish our project goal especially in terms of user interactivity. We discovered it's limitations relatively early on and as a result, we opted to use PyQt5 as our library to create the GUI: this library contained additional functions including auto-complete search that eased the overall user experience. While the new library vastly improved our program's interface, implementing it proved to be more challenging than previously thought. Our group members had little to no experience with PyQt5 which resulted in a learning curve. Indeed while switching to this library was an obstacle, the choice to do so was ultimately beneficial and produced a far more refined interface.

Further Exploration

For further exploration we discussed the possibility of adding on to our existing search algorithm in order to adjust for different diet restrictions (for example vegan, gluten free, vegetarian, etc.). To do so, we would need to find additional datasets containing restricted items and use these to filter out our recipes. Additionally, we would need to expand our search algorithm to carry out the filtering itself which would require further exploration. There are many possibilities for sorting in our program to improve user experience and cater to additional audiences that we could investigate further.

References

[1] Hamedy, S. (2021, April 4). *Wave of new hobbies during the pandemic mirrors trend during Great Depression.* CTVNews. Retrieved April 15, 2021, from https://www.ctvnews.ca/lifestyle/wave-of-new-hobbies-during-the-pandemic-mirrors-trend-during-great-depression-1.5373910.

[2] Querolo, N. (2020, July 7). *In Another Blow to Restaurants, Home Cooking During Covid is Here to Stay.* Bloomberg. Retrieved April 15, 2021, from https://www.bloomberg.com/news/articles/2020-07-07/newly-minted-home-chefs-mark-another-blow-to-u-s-restaurants.

[3] Burdurlu, Y. (2019, February 18). *Recipe ingredients and reviews.* Retrieved March 14, 2021, from https://www.kaggle.com/kanaryayi/recipe-ingredients-and-reviews/activity.

[4] *Inserting Images.* (n.d.). Retrieved April 12, 2020, from https://www.overleaf.com/learn/latex/Inserting_Images

[5] Jenn, J. (2019). *Always Position Your PyQt5 Window In The Center of Your Screen When Initializing.* YouTube. Retrieved April 10, 2021, from https://www.youtube.com/watch?v=735dNm-ar_0.

[6] Forogh, P. (2021). `PyQt5 Gui New Tutorial (2019)`. YouTube. Retrieved April 11, 2021 from https://www.youtube.com/playlist?list=PL1FgJUcJJ03uO70zDLDF3oaTu6s2QLOPa.

[7] Jenn, J. (2020). *Allowing user to enter number input only in PyQt5.* Youtube. Retrieved April 11, 2021 from https://www.youtube.com/watch?v=niIFdLxxc6o

[8] (2020). *PyQt5 Python 3 Tutorial.* Youtube. Retrieved April 15, 2021 from https://www.youtube.com/playlist?list=PLzMcBGfZo4-lB8MZfHPLTEHO9zJDDLpYj

Figures

Files and Folders Structure

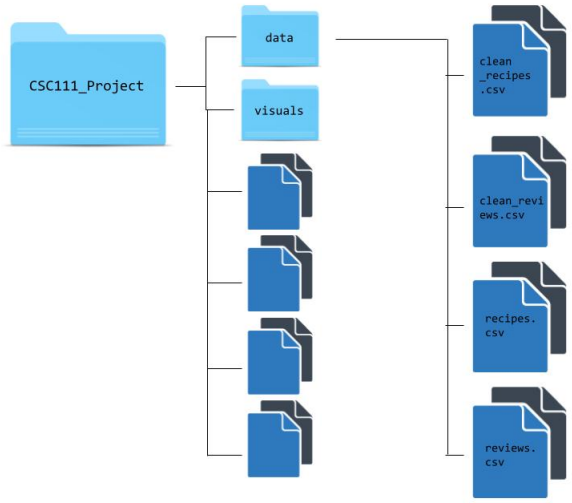

Figure 1: Proper structure for file organization

Different outputs of our program

Figure 2: Front window

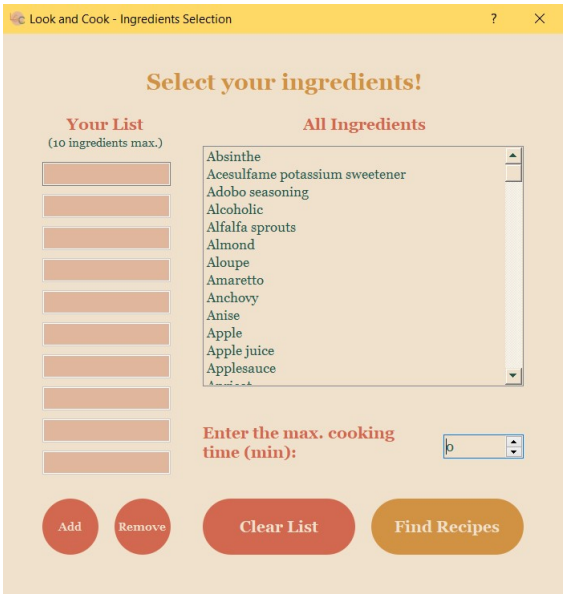

Figure 3: Main window

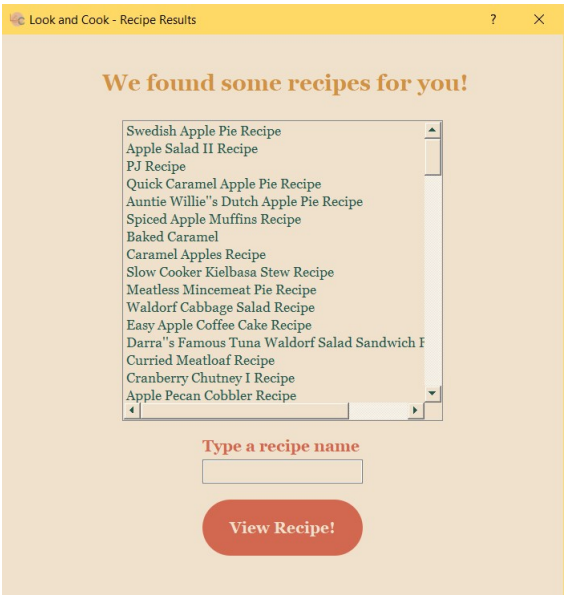

Figure 4: Window displaying various recipes to choose from

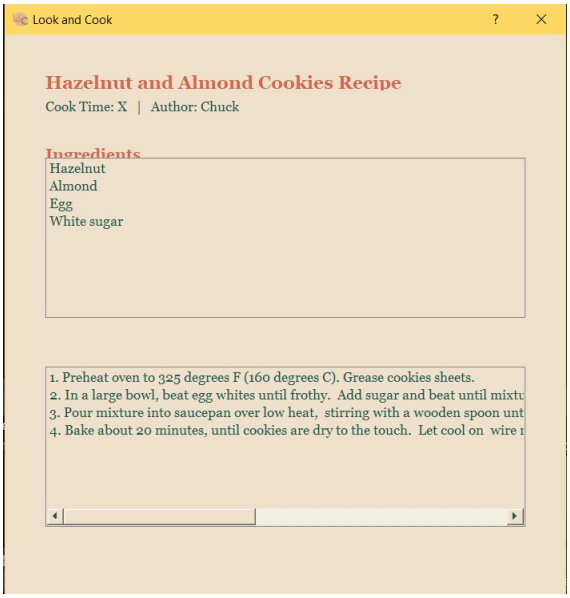

Figure 5: A typical window for recipe instructions