CSC A48 - Assignment 3 - Food Networks

This assignment is intended to help you practice and master the material we have covered in Unit 5 relating to ***Graphs and Recursion***.

The goal is to explore how graphs can be used for all kinds of applications you may not have considered to be typically associated with computer science. In this case, we will be working with food. Applications of CS to food-related issues have become more important, and cover a range of topics. For example, we can use CS to model the spread of plant disease, to study weather conditions and their effect on food production, to optimize transportation and food distribution networks, and even to analyze recipes and suggest novel ways to combine ingredients.

Chef Watson, an application developed using IBM's Watson computer (https://www.ibm.com/watson) was able to learn from a large database of recipes the ways in which ingredients in different types of dishes combine, and to produce new recipes with combinations that would be surprising (not used before), but would very likely taste delicious. At least that was the intended goal, the results, apparently, are... interesting (https://www.newyorker.com/magazine/2016/11/28/cooking-with-chef-watson-ibms-artificial-intelligence-app)

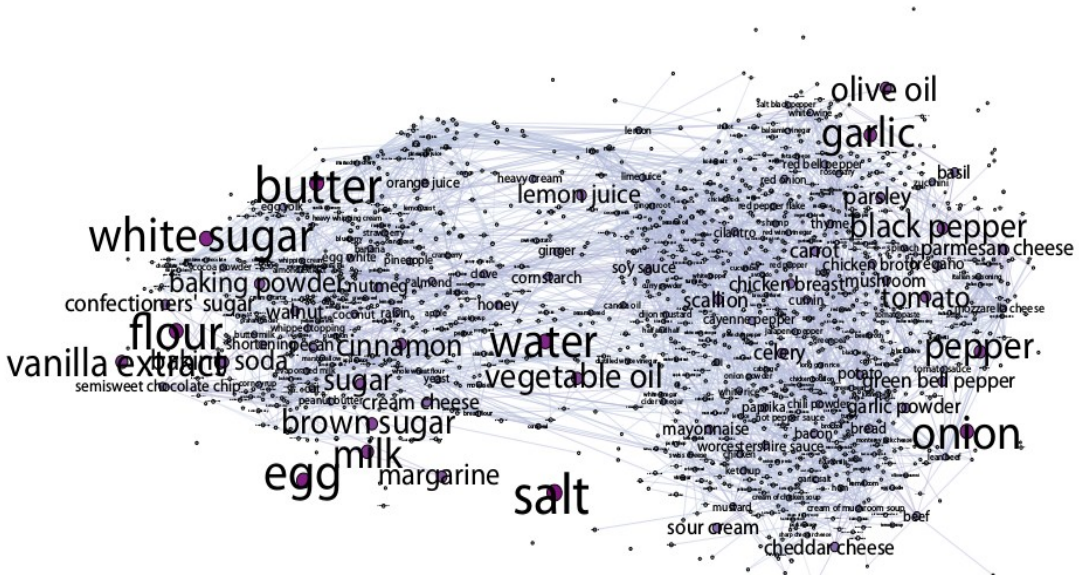

Figure 2: Ingredient complement network. Two ingredients share an edge if they occur together more than would be expected by chance and if their pointwise mutual information exceeds a threshold.

The image above, from *Chun-Yuen et al., "Recipe recoomendation using ingredient networks"*, shows a visualization of a graph where ingredients are grouped by how often they go together in recipes. The graph can be used to recommend flavour combinations. For this assignment, you will be working on a very small version of this graph, obtained from the *Recipe 1M dataset* (http://pic2recipe.csail.mit.edu/)

The data is organized as a list of ingredients, each ingredient will be a ***node*** in a graph that represents ***how often pairs of ingredients appear together*** in recipes. The graph itself is stored as an ***adjacency matrix where A[i][j] gives the number of times ingredient i appears with ingredient j in the same recipe.***

Your Task:

Implement all the functionality described in the starter code (the starter also describes how to select between 2 different sized graphs, how the graph and the ingredients are stored, and what each function needs to do).

The starter will provide basic functionality for you, including reading the ingredient information and adjacency matrix from file, and providing functions to manage linked lists of integers (please use these to implement anything requiring a linked list - *do not create your own*)

First step:

Download, unpack, and carefully study the starter code.

There is only one file where you have to implement code: *'ingredient_graph*.c'

The remaining .c files are:

- A3_driver.c - Your driver for A3, ***unlike A1 and A2 this driver does not try to provide testing***, it is there only to show you how to use each of the functions you have to implement, with an example. ***You have to design the tests for A3 - it is critical that you leave A48 knowing how to thoroughly test a piece of software you implemented.***

- AdjMat_full.dat - Large-size adjacency matrix (400x400) - for four hundred ingredients.

-AdjMat_small.dat - Small-size adjacency matrix (10x10) - for ten ingredients

- Ingredient_names_small.txt - Names of the ingredients for the small adj. matrix.
- Ingredient_names_full.txt - Names of the ingredients for the full-size adj. matrix.

I would recommend you develop and test your code first on the small graph and only move to using the large matrix once you're confident your code works.

Compiling your code:

To compile your BST code with the test driver, use:

```
gcc -Wall -Werror A3_driver.c
```

Do not include extra libraries or modify the code in ways that will cause it to not compile with the above command - our auto-marker will NOT customize the compilation process for your code.

Submitting your work:

As usual, submit your .c program file on Quercus, with the name:

ingredient_graphs_studentNo.c

Once your code is working, see what it does and think about whether it returns anything reasonable: Load the large graph, create a recipe with up to 10 ingredients from the available ones (make something tasty), then ask your program to suggest a replacement - see if the resulting recipe looks good, and if it does, ***try it!***

Cognitive Cooking in A48!

Problem solving (should now be completely familiar to you):

But we can't get too much practice. Therefore -

- Do not ask us to confirm if your idea is right or wrong: Try it out, improve it, or change it if it doesn't look promising. You're expected to come up with your solution without us telling you at each step that you're allowed to try things.

- Do come to talk to us in office hours if you're stuck with something, or if you want to discuss your plans (not your code) for implementing either of these functions. We'll be happy to help!

Debugging and checking your code:

It's recursive, so you will have to trace it by keeping track on paper of what the recursive calls are doing, the stack of un-resolved calls and what they are waiting for, and variables at each level of recursion. Use a debugging tool, or (like I do) use printf(). Either way, ***you have to trace it carefully and not skip anything***. Debugging recursive code requires discipline - making you practice that is part of the goal of this assignment.

Marking scheme:

[5 marks] - Completing a working *print_ingredients()* function.
[5 marks] - Completing a working *ingredient_index()* function.
[5 marks] - Related ingredients function
[10 marks] - Related ingredients, k-distance function
[10 marks] - Related ingredients with restrictions
[15 marks] - Substituting an ingredient

For a total of ***50 marks***.

That's it for A3 - And for A48 assignments - learn, cook, and make it count!