The University of Melbourne
School of Computing and Information Systems
COMP10002 Foundations of Algorithms
Semester 1, 2021
Assignment 2
Due: 4pm Friday 27th May 2022

1 Learning Outcomes

In this assignment you will demonstrate your understanding of dynamic memory allocation, linked data structures, and search algorithms. You will further extend your skills in program design and implementation.

2 The Story...

Mining is the largest industry in Australia. It delivered 10.4% (i.e., $202 billion) of the Australian economy in 2019-2020 and a total of 1.1 million jobs.[1]. The era of Big Data and Industry 4.0 call for a new kind of mining – data mining, or in more popular and generic terms, *data science*. Data science is an interdisciplinary field that focuses on mining patterns from data to support decision making and optimisation. In this assignment, you will gain an initial experience on how data science is impacting our daily lives.

The Victorian Dining and Entertainment Program offers 25% cash back on eligible dining and entertainment purchases across Victoria. [2] Through this program, the Victorian government can harvest minions of dining and entertainment transactions, which are a gold mine of opportunities. Now, the government is hiring you as a data scientist to build a program to mine this dataset[3], with the aim to identify Victorian's dining patterns so as to promote the restaurants and help with the recovery of the local business. Here, we focus just on dining to simplify the assignment setting.

3 Your Task

You are given a list of restaurant records and a list dining transactions in the following format. See the next page for a sample input.

The input starts with a list of **at least 1 and up to 99** restaurant records sorted by the restaurant IDs (e.g., ABNs) in ascending order. Each restaurant record takes one line with the following six fields:

- The restaurant ID, which is a unique 6-digit integer.
- The x-coordinate of the restaurant, which is a real number.
- The y-coordinate of the restaurant, which is a real number. For simplicity, all x- and y-coordinates have been normalised into the range of (0, 100).
- The average price per head, which is an integer.
- The cuisine type, which is a string of at least 1 and up to 20 lower-case letters.
- The restaurant name, which is a string of at least 1 and up to 100 (lower-case letters or `'_'`) characters.

[1] `https://www.australianmining.com.au/news/mining-industry-holds-largest-slice-of-australian-economy/`
[2] `https://www.vic.gov.au/victorian-dining-and-entertainment-program`
[3] Assume that we have user permission to analyse the data.

The list of restaurant records ends with a line of five `'#'`s which serves as a separator line. Then, the input continues with a non-preknown number (**at least 1**) of dining transactions sorted in ascending order of the (unique) transaction time. Each dining transaction takes one line with the following four fields:

- The transaction date and time, which uses the following format: `Year:Month:Day:Hour:Minute:Second`.
- The customer ID, which is a string of 6 lower-case letters (each unique ID represents a different customer).
- The restaurant ID, which is a 6-digit integer (and has appeared in the list of restaurants).
- The transaction amount, which is an integer.

You may assume that there are **at least 3** unique customer IDs in the list of dining transactions.

You may assume that all test input follows the format described above. No input validity checking is needed. Below is a sample set of input data following this format.

```
190947 34.2 37.6 50 aussie captain_cook
359210 31.7 07.4 30 british queen_of_the_fries
626944 97.4 84.8 40 chinese whale_fin_inn
680328 83.9 98.1 160 chinese tim_ho_wonton
699229 19.3 55.4 30 indian curry_palace
726136 94.8 67.2 10 italian tiramisu_pizza
789499 12.4 93.9 60 aussie seventeen_seeds
883358 37.9 72.4 110 italian lasagnes_kitchen
901093 33.3 92.6 140 british the_sherlock_home
987328 70.5 16.1 200 aussie mount_de_vue
#####
2022:04:05:17:05:55 jxrpfj 190947 42
2022:04:06:18:08:08 migbyt 359210 129
2022:04:07:13:50:45 syzbls 190947 170
2022:04:08:14:16:20 ftddia 680328 186
2022:04:08:19:39:00 ozhodc 883358 121
2022:04:15:09:06:49 ftddia 883358 195
2022:04:15:10:09:50 jxrpfj 190947 99
2022:04:15:13:45:29 migbyt 901093 173
2022:04:22:18:30:43 syzbls 680328 234
2022:04:23:17:26:20 syzbls 359210 45
2022:04:29:18:03:00 jxrpfj 789499 89
2022:04:29:20:07:00 jxrpfj 901093 192
```

3.1 Stage 1 - Read Restaurant Records (Up to 3 Marks)

Your first task is to read the restaurant records from the input data (using standard input functions with input redirection, just like in Assignment 1; no file operations needed such as `fopen`). You should start with designing a `struct` to represent a restaurant and then create an array to host the restaurant records.

When this stage is done, your program should output: the total number of restaurant records and the name of the restaurant with the smallest average price per head. If there is a tie, print the name of the restaurant with the smallest ID among the tied ones. The output of this stage based on the sample input is as follows.

```
=========================Stage 1=========================
Number of restaurants: 10
Restaurant with the smallest average price: tiramisu_pizza
```

3.2 Stage 2 - Read Dining Transactions (Up to 9 Marks)

Next, create a data structure to store the customer dining records. You need to create a `struct` to represent a customer and a customer linked list (using a modification of `listops.c` provided in the assignment package) to represent all unique customers in the dining transactions. The customer `struct` has two components:

1. A string to represent the customer ID, and

2. A *restaurant visiting array* to record the number of times that the customer has visited each restaurant in the restaurant list. This array needs to be of the same size of the array of restaurants created in Stage 1. The i-th element in this array stores the number of times that the customer has visited the i-th restaurant in the array of restaurants.

   For example, given the sample input above, the restaurant visiting array of customer `jxrpfj` should be `{2, 0, 0, 0, 0, 0, 1, 0, 1, 0}`. Here, customer `jxrpfj` has visited restaurant `#190947` (the 0-th restaurant) twice, restaurant `#789499` (the 6-th restaurant) once, and restaurant `#901093` (the 8-th restaurant) once. Thus, the 0-th element of the restaurant visiting array is 2, the 6-th and the 8-th elements of the array are both 1, and the rest of the array elements are all 0's.

Your program needs to populate data into the customer linked list by reading the dining transactions from the input. When a transaction is read:

- If the customer ID in this transaction is seen for the first time (for example, when reading the line `2022:04:05:17:05:55 jxrpfj 190947 42`), create a new node to represent the customer, fill the customer ID and initialise the restaurant visiting array with all 0's except for a 1 at the element corresponding to the restaurant in the transaction (that is, the first recorded visit to the restaurant), and append the node to the end of the customer linked list.
- If the customer ID in this transaction has been seen in the input before (for example, when reading the line `2022:04:15:10:09:50 jxrpfj 190947 99`), find the node corresponding to this customer ID in the customer linked list, and update its restaurant visiting array by increasing the element corresponding to the restaurant in the transaction by 1 (that is, one more visit to the restaurant).

When this stage is done, your program should output the customer linked list in the form of a matrix, where each row is the restaurant visiting record of a customer, and the i-th numeric column represents customers' visiting records to the i-th restaurant in the array of restaurants.

```
=========================Stage 2==========================
jxrpfj:  2  0  0  0  0  0  1  0  1  0
migbyt:  0  1  0  0  0  0  0  0  1  0
syzbls:  1  1  0  1  0  0  0  0  0  0
ftddia:  0  0  0  1  0  0  0  1  0  0
ozhodc:  0  0  0  0  0  0  0  1  0  0
```

Note: If you are not confident with using linked data structures, you may use an array of customer `structs` instead, assuming a maximum of 20 unique customers. However, if you do so, the full mark of this stage will be reduced by 3 marks.

3.3 Stage 3 - Recommend Based on Restaurant Similarity (Up to 12 Marks)

Your third task is to identify restaurants that a customer may want to visit next time, based on the similarity between the restaurants (a.k.a. *content-based recommendation*). In particular, for each customer C and each restaurant R that customer C has visited before (that is, its corresponding element in the restaurant visiting array of customer C is greater than zero), your program should identify all other unvisited restaurants that

- share the same cuisine with R, or
- are within a distance of `30` distance units (inclusive) from R, or
- have an average price per head which differs from that of R by no more than 20 (can be either higher or lower).

For each restaurant identified, your program should change the corresponding element in the restaurant visiting array of customer C to `-1`.

Here, we use the Euclidean distance to measure the distance between two restaurants. Given two restaurants R_1 and R_2 with coordinates $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$, respectively, their distance is calculated by:

$$\text{distance}(R_1, R_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

The output of this stage is the updated customer linked list, again in the form of a matrix. For example, given the sample input above, the output of this stage should be as shown in the next page.

```
=========================Stage 3=========================
jxrpfj:  2  -  -  -  -  0  1  -  1  -
migbyt:  -  1  -  -  -  -  -  -  1  0
syzbls:  1  1  -  1  -  -  -  0  -  -
ftddia:  0  0  -  1  -  -  0  1  -  0
ozhodc:  0  0  0  0  -  -  0  1  -  0
```

Here, a `'-'` has been printed for each array element with value -1. Consider customer `jxrpfj`, who has visited the 0-th (`aussie`, at ⟨`34.2, 37.6`⟩, average price per head: `50`), 6-th (`aussie`, at ⟨`12.4, 93.9`⟩, average price per head: `60`) and 8-th restaurants (`british`, at ⟨`33.3, 92.6`⟩, average price per head: `140`).

- The 1-st (`british`) and the 9-th (`aussie`) restaurants should both be recommended based on cuisine.
- The 4-th (at ⟨`19.3, 55.4`⟩ which is within distance 30 of the 0-th restaurant) and the 7-th (at ⟨`37.9, 72.4`⟩ which is within distance 30 of the 8-th restaurant) restaurants should both be recommended based on location.
- The 2-nd (average price per head: `40`) and the 3-rd (average price per head: `160`) restaurants should both be recommended based on average price per head.
- Note: Some restaurants may be recommended for more than one reason.

Overall, the restaurant visiting array of `jxrpfj` is updated to `{2, -1, -1, -1, -1, 0, 1, -1, 1, -1}`.

3.4 Stage 4 - Recommend Based on Customer Similarity (Up to 15 Marks)

Another way to make recommendations is based on the similarity between the customers (a.k.a. *user-based collaborative filtering*). Given two customers C_1 and C_2 with restaurant visiting arrays A_1 and A_2, respectively, their similarity is defined as follows, were r is the number of restaurants.

$$\text{similarity}(C_1, C_2) = \sum_{i \in [0, r-1], A_1[i] > 0, A_2[i] > 0} A_1[i] \cdot A_2[i] \tag{2}$$

In this stage, for each customer C, your program should calculate the top-2 most *similar customers* who have the largest similarity values with C. If there is a tie, your program to take the customers that appear earlier in the customer linked list among the tied ones.

Among the top-2 most similar customers found for C, we only keep the similar customers with similarity values greater than 0. For each similar customer kept, your program should go over each restaurant R visited by the similar customer. If R has been visited by customer C already, nothing further needs to be done. Otherwise, add `-1` to the element of the restaurant visiting array of C that corresponds to R.

After all customers are processed, your program should output the updated customer linked list, still in the form of a matrix. Given the sample input, the additional output of this stage is (note the final `'\n'`):

```
=========================Stage 4=========================
jxrpfj:  2  *  -  +  -  0  1  -  1  -
migbyt:  *  1  -  +  -  -  +  -  1  0
syzbls:  1  1  -  1  -  -  +  0  *  -
ftddia:  -  -  -  1  -  -  0  1  -  0
ozhodc:  0  0  0  -  -  -  0  1  -  0
```

Here, `'-'`, `'+'`, and `'*'` are printed to denote array element values -1, -2, and -3, respectively. Customer `jxrpfj`'s top-2 most similar customers are `syzbls` (similarity value: 2) and `migbyt` (similarity value: 1). Both similar customers have visited the 1-st restaurant, so the 1-st element of the restaurant visiting array of `jxrpfj` has been added with $-1 \times 2 = -2$, resulting in -3 (printed as `'*'`). Customer `syzbls` has visited the 3-rd restaurant, so the 3-rd element of the restaurant visiting array of `jxrpfj` has been added with -1, resulting in -2 (printed as `'+'`). Customer `migbyt` has visited the 8-the restaurant, which has also been visited by `jxrpfj`, so there was no further change to the restaurant visiting array of `jxrpfj`.

At the end of your submission file, you need to add a comment that states the time complexity of your algorithm for Stage 4, and explains why it has that time complexity, assuming that there are r restaurants, c customers, and use the top k most similar customers for updating the restaurant visiting array of each customer.

3.5 Stage 5 - Recommend by Matrix Factorisation

This stage is for a challenge and for your own exploration only. You should *not* submit this stage. Please do not start on this stage until your program through to Stage 4 is (in your opinion) perfect, and is submitted, and is verified.

A third way of recommendation (which is the foundation of modern recommender systems like those used by Amazon and Netflix) is to use matrix factorisation. The basic idea is to compute a $c \times d$ matrix $\mathbf{C}$ and a $d \times r$ matrix $\mathbf{R}$, where d is a system parameter that needs to be set by a data scientist. Each row of $\mathbf{C}$ represents the latent (that is, implicit) dining preference of a customer. Each column of $\mathbf{R}$ represents the latent features of a restaurant. To compute $\mathbf{C}$ and $\mathbf{R}$, we solve the following optimisation problem:

$$\mathbf{C}^*, \mathbf{R}^* = \arg\min_{\mathbf{C},\mathbf{R}} \sum_{i \in [0,c-1]} \sum_{j \in [0,r-1],\ \mathbf{M}_{i,j}>0} (\mathbf{M}_{i,j} - (\mathbf{C} \cdot \mathbf{R})_{i,j})^2 \tag{3}$$

Here, $\mathbf{M}$ denotes the matrix constructed in Stage 2, and $\mathbf{M}_{i,j}$ is its element at row-i and column-j; $\mathbf{C} \cdot \mathbf{R}$ denotes matrix product, and $(\mathbf{C} \cdot \mathbf{R})_{i,j}$ denotes the element at row-i and column-j of the product.

Intuitively, we aim to find $\mathbf{C}$ and $\mathbf{R}$ such that $(\mathbf{C} \cdot \mathbf{R})_{i,j}$ and $\mathbf{M}_{i,j}$ are as close as possible, for all elements where $\mathbf{M}_{i,j} > 0$. Note that $(\mathbf{C} \cdot \mathbf{R})_{i,j}$ is calculated by multiplying row-i of $\mathbf{C}$ and column-j of $\mathbf{R}$, which represent customer-i and restaurant-j, respectively. When $(\mathbf{C} \cdot \mathbf{R})_{i,j}$ and $\mathbf{M}_{i,j}$ are very close for all elements where $\mathbf{M}_{i,j} > 0$, we have uncovered vectors that precisely describe the customer preferences and restaurant features. We can then use the value of $(\mathbf{C} \cdot \mathbf{R})_{i,j}$ to indicate customer interest at elements where $\mathbf{M}_{i,j} = 0$, that is, to predict customer interest towards restaurants not visited before, and to make recommendations.

To solve this optimisation problem, we need matrix factorisation algorithms. See the Wiki page for details on such algorithms: `https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)`. As a challenge, implement a matrix factorisation algorithm for the matrix produced from Stage 2.

4 Submission and Assessment

This assignment is worth 15% of the final mark. A detailed marking scheme will be provided on Canvas.

To submit your code, you will need to:

1. Log in to Grok Learning Assignment 2 module via the **"Assignment 2" link in Canvas Assignments** page.
2. Write *all* your code in `program.c` and `listops.c` tab windows. Do *not* rename these files or add any new code files in your Grok workspace.
3. Compile your code by clicking on the `Compile` button.
4. Once the compilation is successful, click on the `Mark` button to submit your code. You can submit as many times as you want to. *Only the last submission made before the deadline will be marked.* Submissions made after the deadline will be marked with late penalties as detailed at the end of this document. Do *not* press the `Mark` button after the deadline unless a late submission is intended.
5. Two sample tests will be run automatically after you make a submission. Make sure that your submission passes these sample tests.
6. Two hidden tests will be run for marking purpose. Results of these tests will be released after the marking is done.

You can (and should) submit both **early and often** – to check that your program compiles correctly on our test system, which may have some different characteristics to your own machines.

You will be given a sample test file `test0.txt` and the sample output `test0-output.txt`. You can test your code on your own machine with the following command and compare the output with `test0-output.txt`:

```
mac: ./program < test0.txt    /* Here '<' feeds the data from test0.txt into program */
```

Note that we are using the following command to compile your code on the submission testing system (we name the source code file `program.c`).

```
gcc -Wall -std=c99 -o program program.c -lm
```

The flag "`-std=c99`" enables the compiler to use a modern standard of the `C` language – `C99`. To ensure that your submission works properly on the submission system, you should use this command to compile your code on your local machine as well.

You may discuss your work with others, but what gets typed into your program must be individual work, **not** from anyone else. Do **not** give (hard or soft) copy of your work to anyone else; do **not** "lend" your memory stick to others; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "no" when they ask for a copy of, or to see, your program, pointing out that your "no", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode.* See `https://academichonesty.unimelb.edu.au` for more information.

Deadline: Programs not submitted by **4pm Friday 27th May 2022** will lose penalty marks at the rate of 2 marks per day or part day late. Late submissions after 4pm Monday 30th May 2022 will **not** be accepted. Students seeking extensions for medical or other "outside my control" reasons should email the lecturer at jianzhong.qi@unimelb.edu.au. If you attend a GP or other health care professional as a result of illness, be sure to take a Health Professional Report (HRP) form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops into something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

Special consideration due to COVID-19: Please refer to "Advice for students affected by COVID-19" here: `https://students.unimelb.edu.au/your-course/manage-your-course/exams-assessments-and-results/special-consideration#advice-for-students-affected-by-covid-19`

And remember, *Algorithms are fun!*

Prepared by Jianzhong Qi