# Passenger Clearance Order System

# Yuhan Xu

# Assignment 6 2/15/2023

For this assignment, you will describe and implement the fifth and final release of your term project. You will incorporate *a greedy algorithm* as specified below. Submit this completed Word document as before, observing the 4-page limit excluding code etc..

## 1 SUMMARY DESCRIPTION—UPDATED (OR REPEATED IF UNCHANGED)

One-paragraph overall description of your proposed term project—half-page (12-point Times New Roman) limit.

In this part of the content, I used Greedy Algorithm to extend one of the homework part of the previous Dijkstra algorithm. Here, I let the program use the Greedy Algorithm to go to the airport where the passenger needs to go. The program will calculate the order of passenger release based on the airport information of the online database provided in the code.

The links of the database are as follows: https://raw.githubusercontent.com/gtarik/airports\_iata\_rices/Airports\_Iata\_Codes.csv

## 2 I/O EXAMPLE FROM PROJECTED COMPLETED PROJECT—UPDATED (OR REPEATED IF UNCHANGED)

Provide an example of projected *concrete* output for designated input as if the entire project were completed. You will not be held to fulfilling exactly this—it is just explanatory, to indicate where your project is going. We recognize that project direction and details will change as the term progress. This section refers to the project as a whole.

My input is a csv file, my code will be read it and run the program.

passenger\_infomation.csv:

passenger\_id,airport,city,state,country,airliner

A,LAX,Los Angeles,CA,US,AAL

B,ORD,Chicago,IL,US,DAL

C,KFK,New York,NY,US,AAL

D,HKG,Hong Kong,Hong Kong,China,CPA

E,LHR,London,London,UK,BAW

F,SYD,Sydney,Sydney,Australia,AAL

G,HNL,Honolulu,HI,US,UAL

H,ANC,Anchorage,AK,US,DAL

Expected output:

C KFK New York NY US AAL

E LHR London London UK BAW

B ORD Chicago IL US DAL

F SYD Sydney Sydney Australia AAL

A LAX Los Angeles CA US AAL

H ANC Anchorage AK US DAL

G HNL Honolulu HI US UAL

D HKG Hong Kong Hong Kong China CPA

## 3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE

Supply the [functional requirements](https://docs.google.com/document/d/1eU7eINLDxmrf793D4OF2yGT4ry_SW3GQGoVDYzecGHc/edit?usp=sharing) statement that you accomplished for this assignment, together with input where applicable, and output. Please state requirement in declarative form, as illustrated in the examples, because here we want to know the functionality intended (*what*, not *how*). Keep in mind that greedy programming is required in the implementation. Note whether or not true optimization is required.

### 3.1 Input Requirements

1. The path to a CSV file containing passenger information should be provided as input to the script.

2. The CSV file provided by airport should have a header row and the following fields for each passenger: passenger\_id, airport, city, state, country, and airliner.

### 3.2 Output Requirements

1. The script should output a list of passengers sorted based on the distance between their departure cities, in ascending order.

2. For each passenger, the script should output the following information: passenger\_id, airport, city, state, country, and airliner.

3. The output should be printed to the console.

## 4 ILLUSTRATIVE OUTPUT

### Provide illustrative output from your *actual* application showing that the requirements in part 3 have been met.

Please input the path of your csv file：/home/xuyuhan/Desktop/BU-learn-S2023/METCS566O1/HW6/pythonProject/passenger\_information.csv

C KFK New York NY US AAL

E LHR London London UK BAW

B ORD Chicago IL US DAL

F SYD Sydney Sydney Australia AAL

A LAX Los Angeles CA US AAL

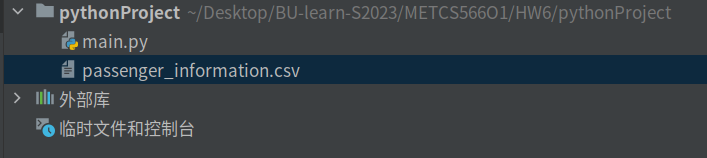
H ANC Anchorage AK US DAL

G HNL Honolulu HI US UAL

D HKG Hong Kong Hong Kong China CPA

## 5 YOUR DIRECTORY

### Show a screenshot of your directory, including methods.



## 6 GREEDY DESIGN IMPLEMENTED

Your implementation should include *greedy programming* in a manner that is useful to your application. This should be as specific to your application as possible. Avoid being generic. Explain where and how you applied this. Explain why is this optimal or suboptimal as the case may be.

The code in 6.2 applies a greedy algorithm in the process of sorting the list of passengers based on their cities. The algorithm selects the passenger with the shortest distance from the current city (starting with "Boston") and adds it to the sorted list. This process repeats until all the passengers have been sorted.

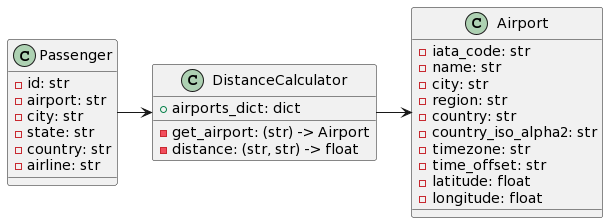
The greedy algorithm is optimal in this case because the goal is to minimize the total distance traveled, and at each step, the algorithm selects the passenger that will contribute the least to the total distance. The algorithm does not look ahead to consider the total distance of the entire route, but in this case, it is not necessary as the problem has the property of subproblem optimal solutions leading to global optimal solutions.

This algorithm is suboptimal in the sense that it doesn't guarantee the shortest possible route since there can be multiple ways to visit all the cities, and this algorithm does not explore all possibilities. Nonetheless, the greedy algorithm provides a good approximation for the traveling salesman problem when the number of cities is high.

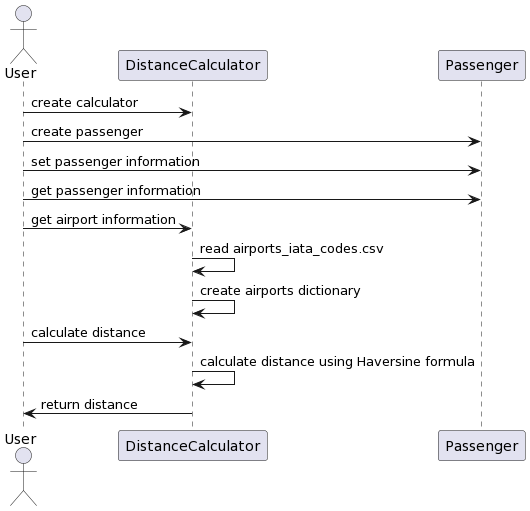
### 6.1 Class model and Sequence Diagram

Provide the class model (even if there is only one class). It should be as simple as possible but not simpler[[1]](#footnote-0). Identify where you included *graph processing.* To do this use tools (e.g., Visio), PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to cut and paste from). Insert indications in red (as in [this example](https://docs.google.com/spreadsheets/d/1ZvkerE9FkWHWwVGdzuy7YMBU6oBMFGZbA4sotFETs8Y/edit?usp=sharing)) to show where greedy programming is implemented.

Class Model

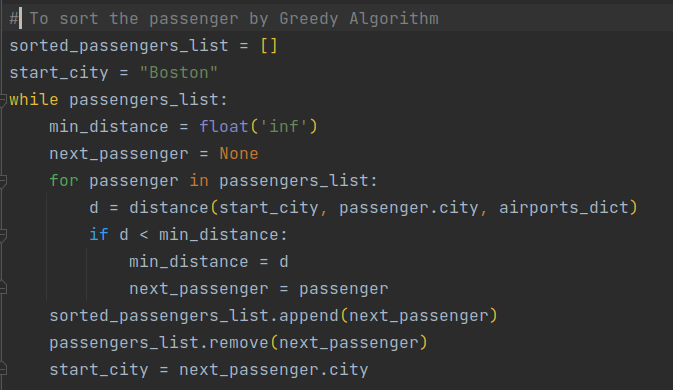


Sequence Diagram



### 6.2 Code showing greedy programming

Your graph processing code should clearly use the algorithm outline in the notes [here](https://docs.google.com/document/d/1XGfpvwFyQe9QSlbWBP5K50ksKMvG8Uq6R1aguIN67Jc/edit?usp=sharing). Show the relevant code (only) and explain why greedy programming is appropriate here. It should be clear where the code is located (class and method). This should be consistent with section 6.1.



### 6.3 Time complexity (efficiency)

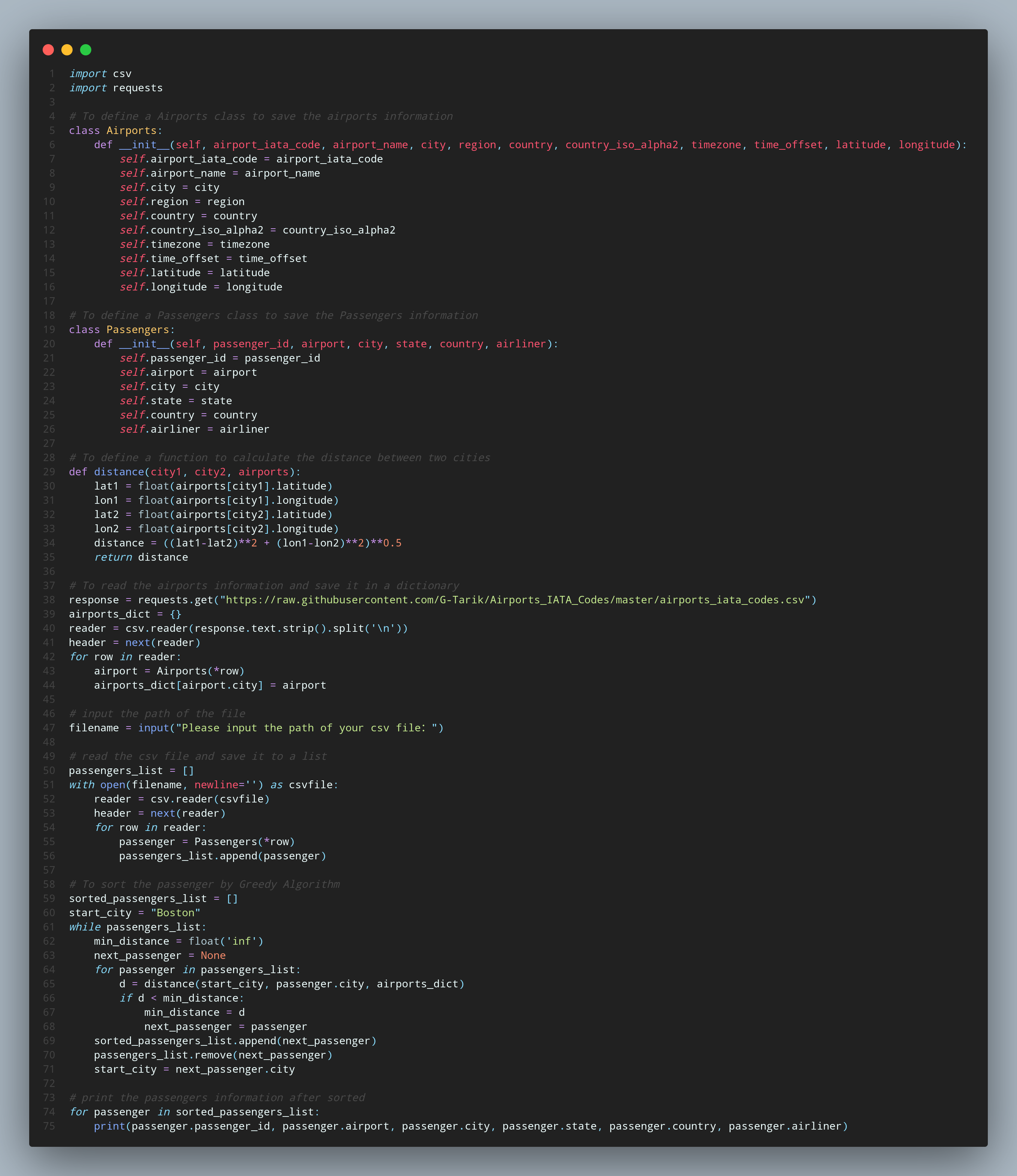
Explain the efficiency of the algorithm you implemented. How does this affect your application in a practical way? How might it if the input data count increases?

The algorithm reads a CSV file containing passenger information and then sorts the passengers by the distance between their starting city and the previous passenger's destination city, using the Greedy algorithm. The distances are calculated using the latitude and longitude of the airports in the cities, which are obtained from another CSV file. The algorithm has a time complexity of O(n^2), where n is the number of passengers. This is because for each passenger, the algorithm loops through the remaining passengers to find the one with the shortest distance to the previous passenger.

If the input data count increases, the algorithm's efficiency will decrease, as the number of iterations required to sort the passengers will increase. In practical terms, this means that the algorithm may take longer to run, which could be problematic if the application needs to process a large number of passengers in real-time. To improve the efficiency of the algorithm, one could consider using a different sorting algorithm that has a lower time complexity, such as merge sort or quick sort. Additionally, precomputing the distances between all pairs of cities and storing them in a lookup table could also improve the algorithm's efficiency.

## 7 YOUR CODE

Please provide your code in a mode that will make access straightforward for your facilitator. Please contact your facilitator in advance if you need to check on this.



## 8 Evaluation



## References (if used. Each of [1], [2], etc. should occur within the paper above.)

[1]

[2]

## Appendix 1 (if needed; should be referenced above, and will be read as-needed only)

## Appendix 2 (if needed; should be referenced above, and will be read as-needed only)

1. If a single class is appropriate, that’s fine. [↑](#footnote-ref-0)