Paint Factory Puzzle Demo

Author: Muhammad Ammad-ud-din
Email: mu.ammad.ud.din@gmail.com

March 9, 2019

1 Executive Summary

This demo explains the solution for the given paint factory code challenge. The solution works on the prinicple of least cost and seek to satisfy customers with minmium choices and would want the paint type: glossy. The robustness of the solutions are evaluated by manually designed numerous example case studies.

2 Configurations, Running and Testing

The solution is provided as a standalone python application and does not depend on external libraries or dependencies except python inbuilt-libaries: sys, os and unittest. The scripts are developed on python version 3.6.4 or above. The main file needed to run and execute the code is paint_factory.py. The file can be run from within python or terminal. To run from terminal, use "python paint_factory.py ORDER_NAME" so that the current working directory is set to contain paint_factory.py. The ORDER_NAME is expected to be a fully-qualified relative path ending with the order name (the input file). The output is shown on the terminal. In case no ORDER_NAME is provided, the script runs with a default example case study as given in the problem description.

To evaluate the robustness of the solution, several unit tests were performed using custom designed inputs and their corresponding outputs inferred manually. To run the unit test use the command: "python test.py". All the tests were successfully passed showing that the solution is robust and is capable of handling multitude of customer order.

3 Potential Solutions

There exist various approaches to solve this problem such as approximate search, permutation and algorithmic.

The constraints posed in the problem statement motivate to adopt an algorithm-based approach. Notably, the constraints such as one matte color per customer at maximum and preference of glossy over matte, restrict the search space of solutions considerably and an algorithm-based approach can become a natural choice. Hence, I chose to solve this problem with the this approach.

As an alternative approach, the approximate search uses evolutionary or genetic algorithms and intends to optimize a few chosen solutions based on the constraints. These algorithms can

find potential putative solutions; however, they may require a substantial longer convergence time thereby satisfying all the constraints, meanwhile comprising the theoretical guarantees. If the objective is to determine an approximate solution for the IMPOSSIBLE cases only; these approaches would become a potential approach to solve the problem.

On the other hand, a permutation-based approach is a powerful competitor to solve this problem. However, this approach can be computationally exhaustive, since the number of permutations to evaluate can grow exponentially with increasing number of input choices.

4 Solution

The following classes and functions produced the entire solution pipeline:

4.1 CustomerOrder

The class CustomerOrder implements the necessary functionality to read customer order (that is the input file) and store it in a list format. In Python, lists make a handy data structure, for example, Stack. Specifically, Stack works on the principle of last-in, first-out. Also, the inbuilt functions in Python allows performing the Stack operation in a short and simple way. To insert an element to the top of the list, i.e., to push an item, append() function and be used and to pop out an element, the pop() function is available, though direct indexing can also be utilized. These functions work quiet efficiently and fast in end operations.

4.2 Utilities

The solution include numerous utility functions to support the solving the task at hand. For instance, the compareCustomers() and compareChoices() are used as custom functions to sort the customers and their choices in ascending order.

4.3 DeliverOrder

The delivery_service contains the deliverOrder() function to present the solution in the desired output format.

4.4 PaintMaker

The class PaintMaker implements the actual solver using a Stack data structure. The solution has a very simple idea and work in three steps.

As a first step, the customer choices are sorted such that at first we yield customers with minimum choices and then sort each of the choices in a way the glossy paint type takes priority. This favors the principle of least cost. As a second step, a path of choices is maintained to support backtracking if all the customers cannot be satisfied (for a current path) As the final step, if all the choices have been considered (or tried), it results in the fail scenario (that is IMPOSSIBLE).

In this way, each of the customer case is solved and the paint type is inferred that satisfies a customer, if possible.

4.5 Tester:

The PaintMaker has been validated using multiple test cases. All of the tests were succesfully passed which demonstrate the usefulness and robustness of the solution. The notebook demostrate the applicability of the solution on several customer orders (input files), the results are presented below.

```
In [1]: import sys
        sys.path.append('paintfactory/')
        from order_services.customer_order import CustomerOrder
        from delivery_services.delivery_order import deliverOrder
        from paintmaker.paint_maker import PaintMaker
        from platform import python_version
        print(python_version())
```

Demo (as given in the problem statement)

```
In [2]: orderLocation = 'paintfactory/orders/demo_order.txt'
        with open(orderLocation, 'r') as order:
            order = order.readlines()
            for line in order:
                print(line.strip())

In [3]: customerOrder = CustomerOrder(orderLocation)
        orders = customerOrder.analyzeCustomerOrders()

        if orders is not None:
            paintMaker = PaintMaker(orders)
            deliveryReports = paintMaker.executeOrders()
            for report in deliveryReports:
                print(deliverOrder(report[0], report[1], report[2]))
```

Example 1

```
In [4]: orderLocation = 'paintfactory/orders/order1.txt'
        with open(orderLocation, 'r') as order:
            order = order.readlines()
            for line in order:
                print(line.strip())

In [5]: customerOrder = CustomerOrder(orderLocation)
        orders = customerOrder.analyzeCustomerOrders()

        if orders is not None:
            paintMaker = PaintMaker(orders)
            deliveryReports = paintMaker.executeOrders()

            for report in deliveryReports:
                print(deliverOrder(report[0], report[1], report[2]))
```

Example 2

```
In [6]: orderLocation = 'paintfactory/orders/order2.txt'
        with open(orderLocation, 'r') as order:
            order = order.readlines()
            for line in order:
                print(line.strip())
```

```
In [7]: customerOrder = CustomerOrder(orderLocation)
        orders = customerOrder.analyzeCustomerOrders()

        if orders is not None:
            paintMaker = PaintMaker(orders)
            deliveryReports = paintMaker.executeOrders()

            for report in deliveryReports:
                print(deliverOrder(report[0], report[1], report[2]))
```

Example 3

```
In [8]: orderLocation = 'paintfactory/orders/order3.txt'
        with open(orderLocation, 'r') as order:
            order = order.readlines()
            for line in order:
                print(line.strip())
```

```
In [9]: customerOrder = CustomerOrder(orderLocation)
        orders = customerOrder.analyzeCustomerOrders()
        if orders is not None:
            paintMaker = PaintMaker(orders)
            deliveryReports = paintMaker.executeOrders()

            for report in deliveryReports:
                print(deliverOrder(report[0], report[1], report[2]))
```

Example 4

```
In [10]: orderLocation = 'paintfactory/orders/order4.txt'
         with open(orderLocation, 'r') as order:
             order = order.readlines()
             for line in order:
                 print(line.strip())
```

```
In [11]: customerOrder = CustomerOrder(orderLocation)
         orders = customerOrder.analyzeCustomerOrders()
         if orders is not None:
             paintMaker = PaintMaker(orders)
             deliveryReports = paintMaker.executeOrders()

             for report in deliveryReports:
                 print(deliverOrder(report[0], report[1], report[2]))
```

Example 5

```
In [12]: orderLocation = 'paintfactory/orders/order5.txt'
         with open(orderLocation, 'r') as order:
             order = order.readlines()
             for line in order:
                 print(line.strip())
```

```
1
5
3
3 1 1 3 0 5 0
3 2 0 3 1 4 0
1 5 1
```

```
In [13]: customerOrder = CustomerOrder(orderLocation)
         orders = customerOrder.analyzeCustomerOrders()
         if orders is not None:
             paintMaker = PaintMaker(orders)
             deliveryReports = paintMaker.executeOrders()

             for report in deliveryReports:
                 print(deliverOrder(report[0], report[1], report[2]))
```

```
Case #1: 0 0 0 0 1
```

Example 6

```
In [14]: orderLocation = 'paintfactory/orders/order6.txt'
         with open(orderLocation, 'r') as order:
             order = order.readlines()
             for line in order:
                 print(line.strip())
```

```
1
3
3
2 1 1 2 0
2 2 0 3 0
1 3 1
```

```
In [15]: customerOrder = CustomerOrder(orderLocation)
         orders = customerOrder.analyzeCustomerOrders()

         if orders is not None:
             paintMaker = PaintMaker(orders)
```

```
            deliveryReports = paintMaker.executeOrders()

            for report in deliveryReports:
                print(deliverOrder(report[0], report[1], report[2]))
```

```
Case #1: 0 0 1
```

Example 7

```
In [16]: orderLocation = 'paintfactory/orders/order7.txt'
         with open(orderLocation, 'r') as order:
             order = order.readlines()
             for line in order:
                 print(line.strip())
```

```
2
5
1
2 1 0 2 0
1
2
1 1 0
1 1 1
```

```
In [17]: customerOrder = CustomerOrder(orderLocation)
         orders = customerOrder.analyzeCustomerOrders()

         if orders is not None:
             paintMaker = PaintMaker(orders)
             deliveryReports = paintMaker.executeOrders()

             for report in deliveryReports:
                 print(deliverOrder(report[0], report[1], report[2]))
```

```
Case #1: 0 0 0 0 0
Case #2: IMPOSSIBLE
```

Example 8

```
In [18]: orderLocation = 'paintfactory/orders/order8.txt'
         with open(orderLocation, 'r') as order:
             order = order.readlines()
             for line in order:
                 print(line.strip())
```

```
10
5
3
1 1 1
2 1 0 2 0
1 5 0
1
2
1 1 0
1 1 1
5
3
1 1 1
2 1 0 2 1
1 3 1
1
2
1 1 1
1 1 1
5
3
1 1 1
2 1 1 2 0
1 3 1
1
2
1 1 0
1 1 1
5
3
1 1 1
2 1 0 2 0
1 5 0
4
2
2 1 1 3 0
2 1 0 3 1
5
3
1 2 1
4 1 0 2 0 4 1 3 0
3 5 0 2 1 1 0
3
2
1 2 1
1 1 1
```

```
In [19]: customerOrder = CustomerOrder(orderLocation)
         orders = customerOrder.analyzeCustomerOrders()

         if orders is not None:
             paintMaker = PaintMaker(orders)
             deliveryReports = paintMaker.executeOrders()

             for report in deliveryReports:
                 print(deliverOrder(report[0], report[1], report[2]))
```

```
Case #1: 1 0 0 0 0
Case #2: IMPOSSIBLE
Case #3: 1 1 1 0 0
Case #4: 1
Case #5: 1 0 1 0 0
Case #6: IMPOSSIBLE
Case #7: 1 0 0 0 0
Case #8: 0 0 0 0
Case #9: 0 1 0 0 0
Case #10: 1 1 0
```