**Design/Coding Exercise**

* We have four design problems outlined in the document below. We require you to provide a solution, written in Java
* You may not use any external libraries to solve the problem itself, but you may use external libraries or tools for building or testing purposes.

**How we evaluate your code**

We will be looking at a number of things including the design aspect of your solution and your object oriented programming skills. Whilst these are small problems, we expect you to submit what you believe is “production-quality” code that you would be able to run, maintain and evolve. You don’t need to “gold plate” your solution, but we are looking for something more than a bare-bones algorithm. You should submit code that you would be happy to produce in a real project, or that you would be happy to receive from a colleague. We recommend you to provide adequate tests to indicate full code coverage. We also recommend you follow good code hygiene and clean code practices to reduce cyclomatic complexity of your classes.

**Problem 1: Airport Baggage**

Denver International Airport has decided to give an automated baggage system another shot. The hardware and tracking systems from the previous attempt are still in place, they just need a system to route the baggage.  The system will route baggage checked, connecting, and terminating in Denver.

You have been asked to implement a system that will route bags to their flights or the proper baggage claim.  The input describes the airport conveyor system, the departing flights, and the bags to be routed.  The output is the optimal routing to get bags to their destinations.  Bags with a flight id of “ARRIVAL” are terminating in Denver are routed to Baggage Claim.

**Input: The input consists of several sections.  The beginning of each section is marked by a line starting: “# Section:”**

Section 1: A weighted bi-directional graph describing the conveyor system.

Format: <Node 1> <Node 2> <travel\_time>

Section 2: Departure list Format:

<flight\_id> <flight\_gate> <destination> <flight\_time>

Section 3: Bag list Format:

<bag\_number> <entry\_point> <flight\_id>

**Output: The optimized route for each bag**

<Bag\_Number> <point\_1> <point\_2> [<point\_3>, …] : <total\_travel\_time>

The output should be in the same order as the Bag list section of the input.

**Example Input:**

# Section: Conveyor System

Concourse\_A\_Ticketing A5 5

A5 BaggageClaim 5

A5 A10 4

A5 A1 6

A1 A2 1

A2 A3 1

A3 A4 1

A10 A9 1

A9 A8 1

A8 A7 1

A7 A6 1

# Section: Departures

UA10 A1 MIA 08:00

UA11 A1 LAX 09:00

UA12 A1 JFK 09:45

UA13 A2 JFK 08:30

UA14 A2 JFK 09:45

UA15 A2 JFK 10:00

UA16 A3 JFK 09:00

UA17 A4 MHT 09:15

UA18 A5 LAX 10:15

# Section: Bags

0001 Concourse\_A\_Ticketing UA12

0002 A5 UA17

0003 A2 UA10

0004 A8 UA18

0005 A7 ARRIVAL

**Example Output:**

0001 Concourse\_A\_Ticketing A5 A1 : 11

0002 A5 A1 A2 A3 A4 : 9

0003 A2 A1 : 1

0004 A8 A9 A10 A5 : 6

0005 A7 A8 A9 A10 A5 BaggageClaim : 12

**Problem 2: Theatre Seating**

You run a small theater and each month, you have patrons mail in requests for pre-sale tickets. You need to process these ticket requests and either tell them where their party will sit or explain to the patron why you can't complete their order.

You have a few rules that you need to follow when you fill the orders

* Fill as many orders as possible
* Put parties as close to the front as possible.
* If there are not enough seats available in the theater to handle a party, tell them "Sorry, we can't handle your party."
* Each party must sit in a single row in a single section. If they won't fit, tell them "Call to split party".

Your program must parse a theater layout and a list of ticket requests and produce a list of tickets or explanations in the same order as the requests.

The theater layout is made up of 1 or more rows. Each row is made up of 1 or more sections separated by a space.

After the theater layout, there is one empty line, followed by 1 or more theater requests. The theater request is made up of a name followed by a space and the number of requested tickets.

**Sample input:**

6 6

3 5 5 3

4 6 6 4

2 8 8 2

6 6

Smith 2

Jones 5

Davis 6

Wilson 100

Johnson 3

Williams 4

Brown 8

Miller 12

Your program must produce results to standard output in the same order as the requests, with the name of the person who requested the ticket and either the row and section of the ticket or the explanations "Sorry, we can't handle your party" or "Call to split party."

**Sample output:**

Smith Row 1 Section 1

Jones Row 2 Section 2

Davis Row 1 Section 2

Wilson Sorry, we can't handle your party.

Johnson Row 2 Section 1

Williams Row 1 Section 1

Brown Row 4 Section 2

Miller Call to split party.

**Problem 3 Inventory Management 1**

Mr. X owns a store that sells almost everything you think about. Now he wants a inventory management system to manage his inventory. Mr. X feels that controlling his inventory through SMS from his mobile will be revolutionary. So as a prequel, he decides that he wants a system that accepts one line commands and performs the respective operation.

Below is the list of commands he needs in the system:

*a) create itemName costPrice sellingPrice*

Whenever Mr. X wants to add a new item to his store he issues a create command. This command creates a new item in the inventory with the given cost price and selling price. The prices are rounded off to two decimal places.

*b) delete itemName*

If Mr. X decides not to sell an item anymore, then he simply issues a delete command. This command will remove the item from the inventory.

*c) updateBuy itemName quantity*

Whenever Mr. X purchases additional quantity of the mentioned item, then he issues a updateBuy command. This command should increase the quantity of the mentioned item.

*d) updateSell itemName quantity*

Whenever Mr. X sells some item, then he issues a updateSell command. This command should deduct the quantity of the mentioned item.

*e) report*

Whenever Mr. X wants to view his inventory list he issues the report command. This command should print the current inventory details in the specified format sorted by alphabetical order. Apart from printing the inventory it has to report on the profit made by Mr. X since last report generation.

Where profit is calculated by: ∑ (sellingPrice-costPrice) of the sold items multiplied by no. of items sold- costPrice of the deleted items.

**Sample Input**

create Book01 10.50 13.79

create Food01 1.47 3.98

create Med01 30.63 34.29

create Tab01 57.00 84.98

updateBuy Tab01 100

updateSell Tab01 2

updateBuy Food01 500

updateBuy Book01 100

updateBuy Med01 100

updateSell Food01 1

updateSell Food01 1

updateSell Tab01 2

report

delete Book01

updateSell Tab01 5

create Mobile01 10.51 44.56

updateBuy Mobile01 250

updateSell Food01 5

updateSell Mobile01 4

updateSell Med01 10

report

#

**Expected Output**

INVENTORY REPORT

Item Name Bought At Sold At AvailableQty Value

--------- --------- ------- ----------- -------

Book01 10.50 13.79 100 1050.00

Food01 1.47 3.98 498 732.06

Med01 30.63 34.29 100 3063.00

Tab01 57.00 84.98 96 5472.00

---------------------------------------------------------------------------

Total value 10317.06

Profit since previous report 116.94

INVENTORY REPORT

Item Name Bought At Sold At AvailableQty Value

--------- --------- ------- ----------- -------

Food01 1.47 3.98 493 724.71

Med01 30.63 34.29 90 2756.70

Mobile01 10.51 44.56 246 2585.46

Tab01 57.00 84.98 91 5187.00

---------------------------------------------------------------------------

Total value 11253.87

Profit since previous report -724.75

**In-Office Extension:**

*f) updateSellPrice itemName newSellPrice*

Mr. X is now happy with the current system, however at times some goods are not selling out faster than expected. In this case he wants to change the initial selling price of that item. So he wants a new command. This command will update the sellingPrice of the specified item.

*report:*

Now while calculating the report profit, the system should take care of 2 different selling prices for a same item. Profits should be calculated with old selling price for the items sold before updating and with new selling price for the items bought after updating.

**Problem 4: Traffic Signals**

Write a program that controls the traffic signals for a four-way intersection. Initially, we consider traffic flowing in straight lines only, no turns. The four directions are S(outhbound) and N(orthbound) on Snell Rd; and W(estbound) and E(astbound) on Weaver Rd. The traffic lights should obey the following rules:

1. Cars arrive in each direction on both roads (Snell and Weaver) at the rate of 1 car per second. That is, 4 cars approach the intersection each second.

2. Only one road (Snell or Weaver) can have a "green" light at one time.

3. It is acceptable for both roads to have the "red" light at the same time. Of course, traffic backs up on both roads if this happens.

4. Start by turning on the traffic on Snell Rd "green" in both directions for 3 seconds; then turn it "red" for one second; then turn Weaver "green" for 3 seconds; and then red for one second.

5. When the light turns from red to green at any intersection, it takes the first car 2 seconds to start moving and cross the intersection. Subsequent cars take 1 second each.

6. At the instant the light turns from "green" to "red", a car may not start moving to cross the intersection; whether that car just arrived at the intersection or was waiting at that intersection.

7. The output should be the number of cars that are waiting at the intersection in each direction at each second, for the first 20 seconds. Do not make the program wait 20 seconds to produce the output: this is only a simulation, so print the output when it's ready.

8. Expected output

0: N = 0; S = 0; E = 0; W = 0

1: N = 0; S = 0; E = 1; W = 1

2: N = 0; S = 0; E = 2; W = 2

3: N = 0; S = 0; E = 3; W = 3

4: N = 1; S = 1; E = 4; W = 4

5: N = 2; S = 2; E = 5; W = 5

6: N = 3; S = 3; E = 5; W = 5

7: N = 4; S = 4; E = 5; W = 5

8: N = 5; S = 5; E = 6; W = 6

**In-Office Extension:**

Additional rules:

9. Cars drive on the right hand side of the road.

10. A car can make right turns during the one second when both signals are red. It takes a car 1 second to make a right turn (whether it is stopped at the intersection or just arrived at the intersection).

11. Cars may not make a right turn when traffic from the cross-road is moving. (For cars going straight through, rule #6 still applies.)

12. Each side of Weaver and Snell roads have two lanes: the right lane is for right turns only, the left lane is for cars going through the intersection or turning left.

13. One car can make a right turn from the right turn lane at the same time that another car is going through the intersection (or making a left turn) from the other lane.

14. Starting at time t=0, every third car arriving from all the four directions is making a right turn.

15. New expected output:

0: N = 0; S = 0; E = 0; W = 0

1: N = 0; S = 0; E = 1; W = 1

2: N = 0; S = 0; E = 2; W = 2

3: N = 0; S = 0; E = 3; W = 3

4: N = 1; S = 1; E = 3; W = 3

5: N = 2; S = 2; E = 4; W = 4

6: N = 3; S = 3; E = 4; W = 4

7: N = 4; S = 4; E = 4; W = 4

8: N = 4; S = 4; E = 4; W = 4

**Problem 5: Pricing Engine**

An online retail company conducts market research to competitively price their products.

Surveyed data contains Product code, Competitor and Price.

The retail company uses a Pricing engine which recommends most frequently occurring price. If multiple prices occur frequently, the least amongst them is chosen.

Products are classified based on parameters like Supply, Demand. Possible values are Low (L), High (H)

If Supply is High and Demand is High, Product is sold at same price as chosen price.

If Supply is Low and Demand is Low, Product is sold at 10 % more than chosen price.

If Supply is Low and Demand is High, Product is sold at 5 % more than chosen price.

If Supply is High and Demand is Low, Product is sold at 5 % less than chosen price.

Prices less than 50% of average price are treated as promotion and not considered.

Prices more than 50% of average price are treated as data errors and not considered.

Input consists of number of products, followed by each Product's supply and demand parameters.

followed by number of surveyed prices, followed by competitor prices.

Output must be recommended price for each product.

**Input 1:**

2

flashdrive H H

ssd L H

5

flashdrive X 1.0

ssd X 10.0

flashdrive Y 0.9

flashdrive Z 1.1

ssd Y 12.5

**Output 1:**

A 0.9

B 10.5

**Input 2:**

2

mp3player H H

ssd L L

8

ssd W 11.0

ssd X 12.0

mp3player X 60.0

mp3player Y 20.0

mp3player Z 50.0

ssd V 10.0

ssd Y 11.0

ssd Z 12.0

**Output 2:**

A 50.0

B 12.1