FINAL PRESENTATION

TEAM TA: TITHIRA WITHANAARACHCHI

26/07/2023

AGENDA

1. Development Environment and Outline of Work.
2. Task 6:Outline.
3. Task 7: Outline.
4. Task 7: Input Structure.
5. Task 7: Class Diagram.
6. Task 7: Concept of the Product Layout .
7. Example Scenario of Shelf and Product Placing.
8. Efficiency of the Proposed Store.
9. Testing and Code Demonstration.
10. Deliverables and Difficulties.
11. Contribution and the Learning Outcomes.

DEVELOPMENT ENVIRONMENT AND OUTLINE OF WORK

(2ND QUARTER)

Development Environment

Operating System: Windows.

Language: Java.

IDE: Vs Code.

Version Control: Git Hub.

Outline of Work

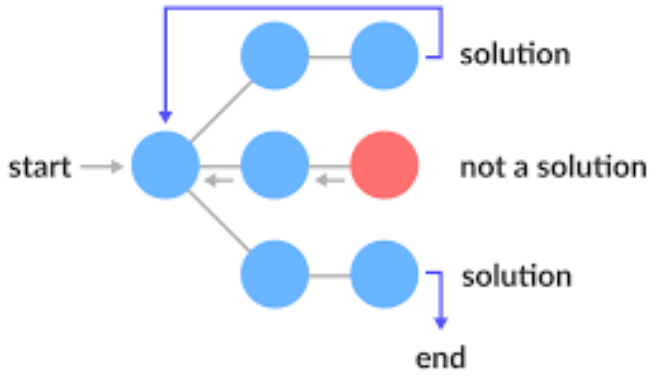
Task 6: Implemented using both Held Karp and Dijkstra's Algorithm. Held Karp is modified to return the path and distance both.

Task 7: Implemented considering the product pairs bought by customers and their frequency (own algorithm).

TASK 6:OUTLINE

- The task 6 was implemented using both Dijkstra's Algorithm and the Held Karp algorithm.

- In this task Held Karp algorithm is modified using **Back Tracking** in a way that it returns the shortest path it travelled within the graph in addition to the distance which was returned in Task 5.



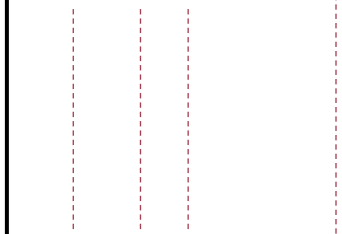
TASK 7:OUTLINE

- The task is implemented by own algorithm which considers the frequency of product pairs which was bought together in each order.

- History of the purchases are analyzed and used the output which contains product pairs and its frequencies to create the product Layout.

TASK7: INPUT STRUCTURE

N
T
i1 p1 p2…….pn
i2 p1 p2……..pn



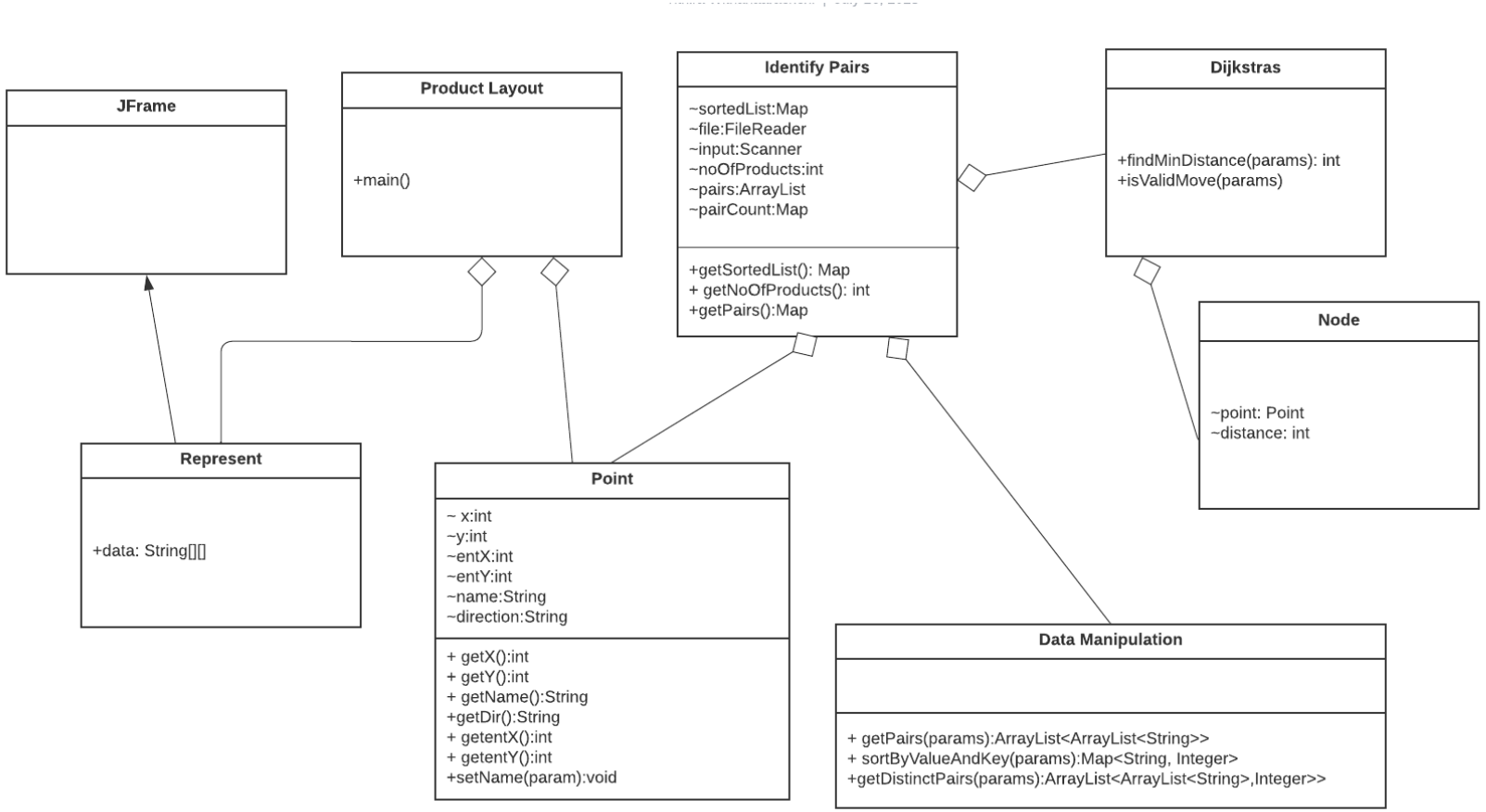
in p1 p2.....pn

Constraints:

- T->Number of Products is a constant(50).
- Size of the store is 12*12 which can accommodate 60 products

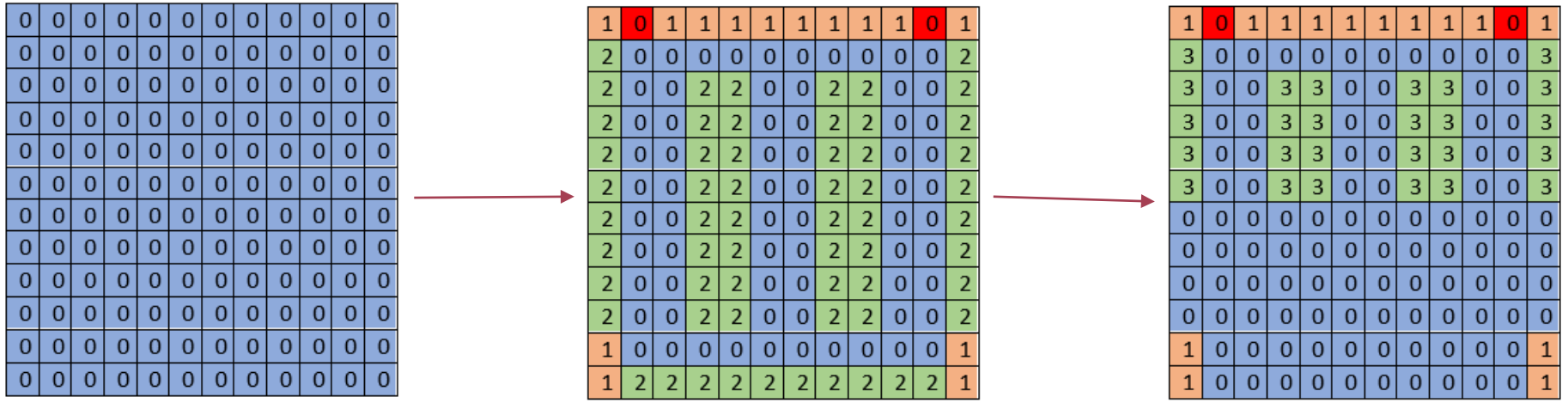
N: Total number of purchases.
T: Total number of products in the store.
i: number of products in the order(purchase).
p: list of products in the order(purchase).

TASK 7:CLASS DIAGRAM



TASK 7:CONCEPT OF THE PRODUCT LAYOUT

Shelf Placing:



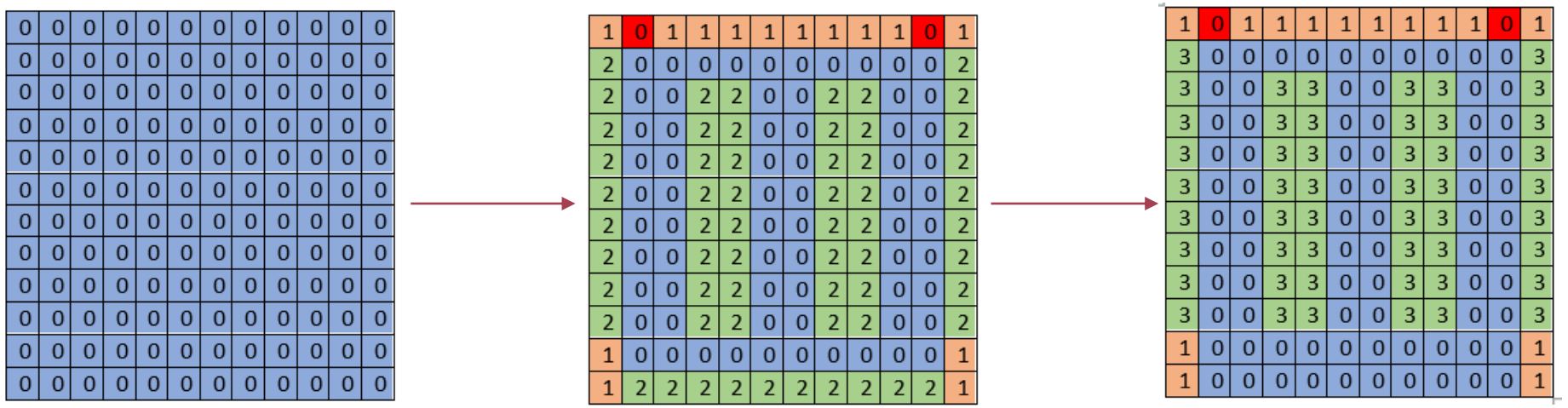
Empty Store

Maximum Possible Shelf Placement

Reduce the Shelves According to the Number of Products

0: Empty spaces of the store(Which customers can travel on)
1: Obstacles where customers cannot travel on.
2: Empty shelves.
3: Reduced set of shelves placed according to the number of products.

Example Scenario: Shelf Placing



Step 1 Step 2 Step 3

Example Scenario : Product Placing

1	0	1	1	1	1	1	1	1	1	0	1	
Potato 42	0	0	0	0	0	0	0	0	0	0	Apricot	9
Bread 43	0	0	Carrot 41	Tomato 26	0	0	Mango 25	Grapes 10	0	0	Pear	8
Cucumber 44	0	0	Onion 40	Raspberry 27	0	0	Greenbeans 24	Pasta 11	0	0	Papaya	7
Lettuce 45	0	0	Lentils 39	Apple 28	0	0	Blackeans 23	Celery 12	0	0	Pineapple	6
Lime 46	0	0	Peach 38	Artichoke 29	0	0	Broccoli 22	Cherry 13	0	0	Chickpeas	5
Strawberry 47	0	0	Corn 37	Bellpepper 30	0	0	Pomengranate 21	Watermelon 14	0	0	Pumpkin	4
Banana 48	0	0	Eggplant 36	Orange 31	0	0	Avacado 20	Fig 15	0	0	Asparagus	3
Zucchini 49	0	0	Cauliflower 35	Sweetpotato 32	0	0	Coconut 19	Radish 16	0	0	Lemon	2
Mushroom 50	0	0	Kiwi 34	Plum 33	0	0	Spinach 18	Rice 17	0	0	Blueberry	1
1	0	0	0	0	0	0	0	0	0	0	1	
1	0	0	0	0	0	0	0	0	0	0	1	

PAIR DISTANCE EFFICIENCY OF THE PROPOSED STORE

```
//Calculating Distance Product of Proposed Store
Dijkstra d=new Dijkstra();
for(ArrayList<String> h:sortedList.keySet()){
    ArrayList<Point> plist=new ArrayList<Point>();
    for(int j=0;j<h.size();j++ ){
            for(int i=0;i<shelves.size();i++){
                if(h.get(j).equals(shelves.get(i).getName())){
                    plist.add(shelves.get(i));

                }
            }
    }

    int distance=d.findMinDistance(graph, plist.get(index:0), plist.get(index:1));
    sum=sum+(distance*sortedList.get(h));
}
//Calculating Distance Product of Random Store
for(ArrayList<String> h:list.keySet()){
    ArrayList<Point> plist=new ArrayList<Point>();
    for(int j=0;j<h.size();j++ ){
            for(int i=0;i<shelves2.size();i++){
                if(h.get(j).equals(shelves2.get(i).getName())){
                    plist.add(shelves2.get(i));
                }
            }
    }
    int distance=d.findMinDistance(graph, plist.get(index:0), plist.get(index:1));
    sum2=sum2+(distance*list.get(h));
}

double percentage=((sum2-sum)/sum2)*100;


System.out.println("Distance Product of Proposed Approach : "+ sum);
System.out.println("Distance Product of Random Approach : "+ sum2);
System.out.println("Efficiency of the Proposed Store : "+String.format(format:"%.5f", percentage)+"%");
```

Pair Distance Product=(<distance between 2 products in the pairs list> * <the frequency of the pair>)/ (<distance between entrance and product1 of the product pair>*<distance between entrance and product2 of the product pair>)

Pair Distance Efficiency=((RandomSum-ProposedSum)/RandomSum)* 100

ProposedSum : Sum of “pair distance products” of proposed Store

RandomSum : Sum of “pair distance products” of Random Store

PAIR DISTANCE EFFICIENCY OF THE PROPOSED STORE

```
//Calculating Distance Product of Proposed Store
Dijkstra d=new Dijkstra();
for(ArrayList<String> h:sortedList.keySet()){
    ArrayList<Point> plist=new ArrayList<Point>();
    for(int j=0;j<h.size();j++ ){
            for(int i=0;i<shelves.size();i++){
                if(h.get(j).equals(shelves.get(i).getName())){
                    plist.add(shelves.get(i));

                }
            }
    }

    int distance=d.findMinDistance(graph, plist.get(index:0), plist.get(index:1));
    sum=sum+(distance*sortedList.get(h));
}
//Calculating Distance Product of Random Store
for(ArrayList<String> h:list.keySet()){
    ArrayList<Point> plist=new ArrayList<Point>();
    for(int j=0;j<h.size();j++ ){
            for(int i=0;i<shelves2.size();i++){
                if(h.get(j).equals(shelves2.get(i).getName())){
                    plist.add(shelves2.get(i));
                }
            }
    }
    int distance=d.findMinDistance(graph, plist.get(index:0), plist.get(index:1));
    sum2=sum2+(distance*list.get(h));
}

double percentage=((sum2-sum)/sum2)*100;


System.out.println("Distance Product of Proposed Approach : "+ sum);
System.out.println("Distance Product of Random Approach : "+ sum2);
System.out.println("Efficiency of the Proposed Store : "+String.format(format:"%.5f", percentage)+"%");
```

Approximate Pair Distance Efficiency range of Proposed Store:
0.4% ~ 1.02%

***As the distance between 2 related products reduce, the customer will be convinced to by more products.**

***And as the distance between entrance and the frequently bought products increase, the time that the customer will remain in the store will comparatively increase.**

***Therefore as the distance from entrance to frequently bought products increase, the efficiency of the proposed store increase.**

TESTING AND CODE DEMONSTRATION

Testing:

- Black Box Testing Approach.
- Inputs for testing were generated using a test data generator.

Lets Look at the Demo!

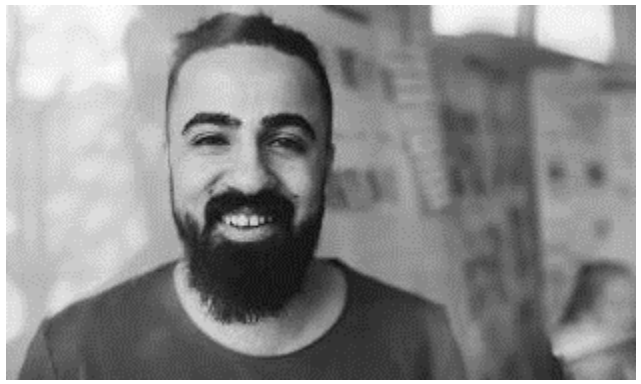
DELIVERABLES (QUARTER 2)

- Code which gives the shortest distance to travel within the store and the path that has to travel in order to achieve that(phase2).
- Optimized Product Layout of a store according to the purchase history of products by customers(Phase3).

DIFFICULTIES

- Working on the project alone.
- Manage Time for completing the task along with the other courses of the semester.



CONTRIBUTION AND THE LEARNING OUTCOMES

Contribution:

Tithira Withanaarachchi: 100%

Learnings:

- Learned the way of using pre-built algorithms in order to achieve the project goals.
- Improved the scope of thinking about a software problem and how to approach them after getting exposed to different algorithm.
- Test data generation.
- Improved the skill of achieving deadlines as possible.

THANK YOU

Team TA