CASE STUDY 1:
SOLVING REAL-WORLD PROBLEMS USING COMPUTATIONAL THINKING

GROUP 10:

MARQUEZ, CLARD JOZREIN

PADRIGANO, JOHN LLOYD

VILLAMOR, KURT RUSSEL

PROBLEM:

- You are an Appliances Warehouse manager, You want to maximize the profit, and ship the items from your warehouse to distributors as fast as possible. You have a Forklift that can **carry all items in one aisle at a time,** You also has a truck that has **10000kg capacity.**
- Your goal is to get the most preferable items in an aisle and putting it to the truck considering the how many minutes you consume in the process. (Assuming the minutes are constant and picking time is not considered)

TRUCK CAPACITY: 10000Kg

carry all items in one aisle at a time

ITERATION 1

In the Given Item List, How we Identify the Items should be delivered to the truck?

PROBLEM IDENTIFICATION

SUB-PROBLEMS:

- IDENTIFYING ITEMS THAT SATISFY HIGH PROFIT WHILE WITHIN THE TRUCK'S CAPACITY
- WHAT IS THE BEST ITEM TO PICK? SINCE NOT ALL BIG PROFIT CAN FIT TO THE TRUCK AND NOT ALL ITEMS THAT FIT IN THE TRUCK PRODUCE HIGH PROFIT (FILTERING THE ITEMS)

PATTERN RECOGNITION:

- WHEN WE PICKED AN ITEM, THE DIFFERENCE BETWEEN THE ITEM'S WEIGHT AND TRUCKS CAPACITY WILL BE THE CURRENT CAPACITY FOR THE NEXT ITEM
- THE ITEMS TO BE PICKED IS NOT HIGH IN CAPACITY WHILE HIGH IN PROFIT

ABSTRACTION:

- THE SIZE OF THE ITEM (LENGTH/WIDTH/HEIGHT) ARE NOT CONSIDERED

ITERATION 2

In the Item List you will pick,
How can we deliver it from the aisle to the truck fast using one forklift?

PROBLEM IDENTIFICATION

SUB-PROBLEMS:

- FIND POSSIBLE ROUTES IN THE WAREHOUSE THAT COULD SAVE TIME IN DELIVERING TO THE TRUCK (FASTER WAY)
- WHAT AISLE SHOULD BE THE FIRST AISLE TO PICK ITEMS FIRST?

PATTERN RECOGNITION:

- SELECTED ITEMS FROM AN AISLE CAN BE PICKED BY THE FORKLIFT
- THE PATH IS CONSIDERED ONE WAY SO THERE ARE 3 WAYS TO TRAVERSE THE PATH

ABSTRACTION:

- THE WEIGHT OF THE FORKLIFT AND THE ITEMS IS NOT CONSIDERED

ITEMS AND THE PATHWAYS

ITEMS PER AISLE

Name	Name	Weight	Profit
SELF-N-FORGET COOKER	1	1458	2949
WASHING MACHINE	3	1408	4590
THERMAL MASS REFRIGERATOR	5	1313	1686
SOUS-VIDE COOKER	3	751	2958
ELECTRIC WATER BOILER	1	1302	4590

Name	Name	Weight	Profit
ENERGY REGULATOR	4	646	4129
THERMAL IMMERSION CIRCULATOR	2	715	3656
AIR CONDITIONER	6	957	2389
ENERGY REGULATOR	1	598	564
INTERNET REFRIGERATOR	6	1193	2361

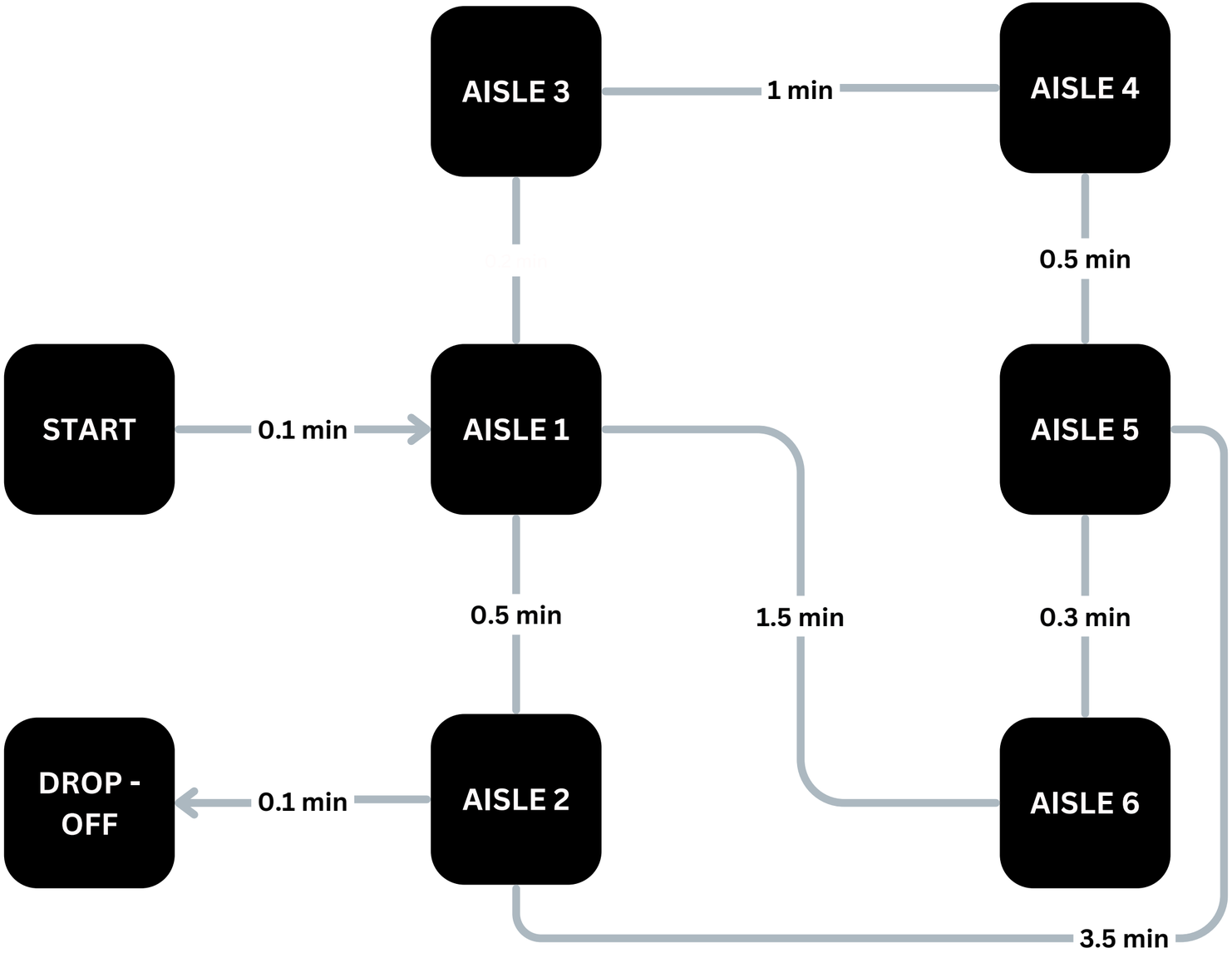
ITEMS PER AISLE

Name	Name	Weight	Profit
TURKEY FRYER	1	973	4791
BEVERAGE OPENER	6	1002	3812
HOME SERVER	4	934	4148
CLOTHES DRYER	2	1079	2671
VACUUM CLEANER	5	1346	864

Name	Name	Weight	Profit
HUMIDIFIER	4	792	2690
PANINI SANDWICH GRILL	2	1241	3564
FLATTOP GRILL	4	1352	1545
MICROWAVE OVEN	1	616	4390
ICEBOX	3	1275	2405

GRAPH OF PATHWAYS

10000 KG



CODE IMPLEMENTATION AND DISCUSSION

Uses:

- Knapsack (tabulation method)
- Greedy Algorithm in Shortest Path Graph (Weighted graphs)

01 ITEMS

```python
class Items:
  def __init__(self, name, aisle, price, weight): # Parameters for the Name(str), aisle(int),price(int),and weight(int)
    self.__name = name
    self.__aisle = aisle
    self.__price = price
    self.__weight = weight

  def get_name(self): # It gets the the encapsulated name
    return self.__name

  def get_aisle(self):  # It gets the the encapsulated aisle
    return self.__aisle

  def get_price(self):  # It gets the the encapsulated price
    return self.__price

  def get_weight(self):  # It gets the the encapsulated weight
    return self.__weight
```

First, Lets Create a class that will handle the items

02 ITEMS

```
def menu_of_items(arrOfNames, arrOfAisle,arrOfPrice,arrOfWeight):
  items = []
  for i in range(len(arrOfNames)):
    items.append(Items(arrOfNames[i], arrOfAisle[i], arrOfPrice[i], arrOfWeight[i]))
  return items
```

We also need to create a menu list so that all of the Items are organized well

03 ITEMS

```
# list of the items in the Warehouse
items = ["set-n-forget cooker", "washing machine", "thermal mass refrigerator", "sous-vide cooker", "electric water boiler",
         "energy regulator", "thermal immersion circulator","air conditioner", "energy regulator", "internet refrigerator",
         "clothes dryer", "turkey fryer", "beverage opener", "home server", "vacuum cleaner",
         "humidifier", "panini sandwich grill","flattop grill","microwave oven","icebox"]

aisle = [1,3,5,3,1,
         4,2,6,1,6,
         2,1,6,4,5,
         4,2,4,1,3]

price = [2949,2584,1686,2958,4590,
         4129,3656,2389,564,2361,
         2671,4791,3812,4148,864,
         2690,3564,1545,4390,2405]

weight = [1458,1408,1313,751,1302,
          646,715,957,598,1193,
          1079,973,1002,934,1346,
          792,1241,1352,616,1275]

capacity = 10000 # for the capacity of the truck
```

Now We made inndividual array per category but using the Class and the function, we can store it efficiently.

04 ITEMS

```
item_menu = menu_of_items(items, aisle, price, weight) # This variable will be sent to the knapsack function
print(item_menu) # Checking the Items in the List
```

Lets Feed in the data

05 KNAPSACK (TRUCK)

```python
def tab_knapsack(cap,items): # it accepts the Item class list we created
  n = len(items)
  table = [[0 for i in range(cap+1)] for i in range(n+1)]  # the use of tabulation Method
  for i in range(n+1):
    for j in range(cap+1):
      if i == 0 or j == 0:
        table[i][j] = 0
      elif items[i-1].get_weight() <= j:  # the the capacity can handle the weight of an item, then add to the table
        table[i][j] = max(items[i-1].get_price() + table[i-1][j - items[i-1].get_weight()], table[i-1][j])

  k = n
  l = cap
  while k > 0 and l > 0: # this loop for for analyzing the table populated with our data and print the items included or not
    if table[k][l] == table[k-1][l]:
      items.remove(items[k-1]) # It removes the item that is not Included to the knapsack
      k-=1
    else:
      k-=1
      l-=items[k].get_weight()

  items.sort(key=lambda x: x.get_aisle()) # sorting the items by Aisle
  aisles = []
  for i in items:
    print(f"{i.get_name()}, is in aisle {i.get_aisle()}, {i.get_weight()}") # Prints the items you include for the truck
    aisles.append(i.get_aisle())
  return list(set(aisles)) # Returns the list of the aisles that the items is in

aisle = tab_knapsack(capacity, item_menu) # Display the Included Items, Their Aisle Num and the Weight (For Checking).

print(aisle)
```

06 WAREHOUSE'S GRAPH

```
# STRUCTURE:
# { VERTICES: (EDGES, CORRESPONDING TIME FOR EDGE)}
g = { "START" : [(1,0.1)],
      1 : [(3,0.2),(2,0.5),(6,1.5)],
      2 : [(1, 0.5), (5, 3.5), ("DROP OFF", 0.1)],
      3 :[[1, 0.2], (4, 1)],
      4 : [(3,1),(5, 0.5)],
      5 : [(2, 3.5),(6, 0.3)],
      6 : [(1, 1.5),(5,0.3)],
      "DROP OFF":[]
    }
```

We decided to take the simple route in making graph, example is creating a dictionary to represent the graph and corresponding time in connecting edges

07 WAREHOUSE'S GRAPH

```python
class Graph:
    def __init__(self, graph_dict=None): # It accepts parameters of a graph dictionary, if theres None it creates One.
        """ initializes a graph object
            If no dictionary or None is given,
            an empty dictionary will be used
        """
        if graph_dict == None:
            graph_dict = {}
        self._graph_dict = graph_dict

    def edges(self, vertice): # Gets all the Edges of a certain vertice
        """ returns a list of all the edges of a vertice"""
        return [i for i in self._graph_dict[vertice]]

    def __generate_edges(self):
        """ A static method generating the edges of the
            graph "graph". Edges are represented as sets
            with one (a loop back to the vertex) or two
            vertices
        """
        edges = []
        for vertex in self._graph_dict:
            for neighbour in self._graph_dict[vertex]:
                if {neighbour, vertex} not in edges:
                    edges.append({vertex, neighbour})
        return edges
```

08 WAREHOUSE'S GRAPH

```
def find_all_paths(self, start_vertex, end_vertex, path=[], sums=0):
    """ find all paths from start_vertex to
        end_vertex in graph """
    graph = self._graph_dict
    path = path + [start_vertex]
    if start_vertex == end_vertex:  # THE BASE CASE, when this is satisfied, it returns all the posible paths,
                                    # additional the total time of the path to be finished (based on the time in the edges)
        return path, sums
    if start_vertex not in graph: # If vertex is not their, then give them nothing
        return []
    paths = []
    for vertex in graph[start_vertex]:
        if vertex[0] not in path:
            extended_paths = self.find_all_paths(vertex[0], end_vertex, path, sums + vertex[1]) # RECURSIVE
            for p in extended_paths:
                paths.append(p)
            self.sums = 0
    return paths
```

So The First goal is to Identify all the Possible routes from START to the DROP OFF point. so We are going to add a recursive function to find the routes.

09 WAREHOUSE'S GRAPH

```
def total_mins(self, paths):
    records = {}
    for i in range(len(paths)):
        if type(paths[i]) == list: # If the current Item is array, then it is recorded as
                                   # a value of the path index (path identified by the find_all_paths())
            records[len(records)] = [paths[i],paths[i+1]]
    return records
```

- With The Output of the finding all paths, we need to fixed it to make us understand more the process, so that this function fixes this paths to be a dictionary, noting that this paths are key for noting the paths.

10 WAREHOUSE'S GRAPH

```
def fastest_way(self, aisles, total_min):
    table = [[0 for i in range(len(total_min))] for i in range(len(aisles))]
    for i in range(len(aisles)):
      for j in range(len(total_min)):
        if aisles[i] in total_min[j][0]:
          table[i][j] += total_min[j][1]

    minimum_routes = []
    print(table)
    for i in table:
      smallest_time = min(num for num in i if num > 0)
      s = i.index(smallest_time)
      minimum_routes.append(s)

    return minimum_routes
```

- And Now we will find the fastest way to get all the items in the using the minimal value in the array. (Greedy Method- getting the Lowest possible time possible)

RESULTS

Items selected and their corresponding aisle

```
electric water boiler, is in aisle 1, 1302
turkey fryer, is in aisle 1, 973
microwave oven, is in aisle 1, 616
thermal immersion circulator, is in aisle 2, 715
panini sandwich grill, is in aisle 2, 1241
sous-vide cooker, is in aisle 3, 751
energy regulator, is in aisle 4, 646
home server, is in aisle 4, 934
humidifier, is in aisle 4, 792
air conditioner, is in aisle 6, 957
beverage opener, is in aisle 6, 1002
[1, 2, 3, 4, 6]
```

RESULTS

All the possible routes

```
{0: [['START', 1, 3, 4, 5, 2, 'DROP OFF'], 5.399999999999995],
 1: [['START', 1, 2, 'DROP OFF'], 0.7]
 2: [['START', 1, 6, 5, 2, 'DROP OFF'], 5.5]}
```

Order of pathways

```
[1, 1, 0, 0, 2]
```

THANK YOU FOR LISTENING

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis vulputate nulla at ante rhoncus, vel efficitur felis condimentum. Proin odio odio.

GROUP 10