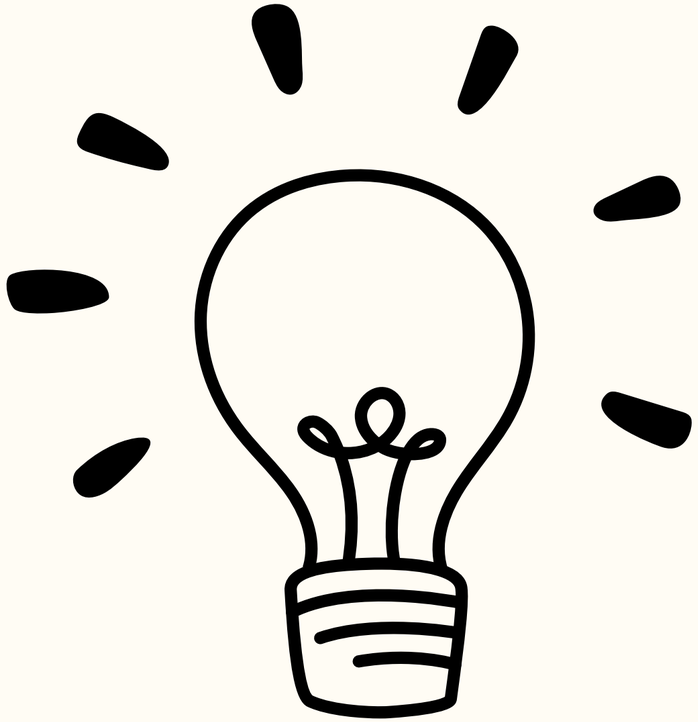
CASE STUDY 1

SOLVING REAL-WORLD PROBLEMS USING COMPUTATIONAL THINKING

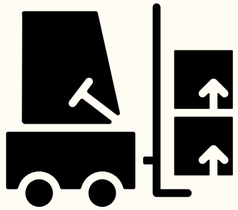
Learn how to optimize team tasks



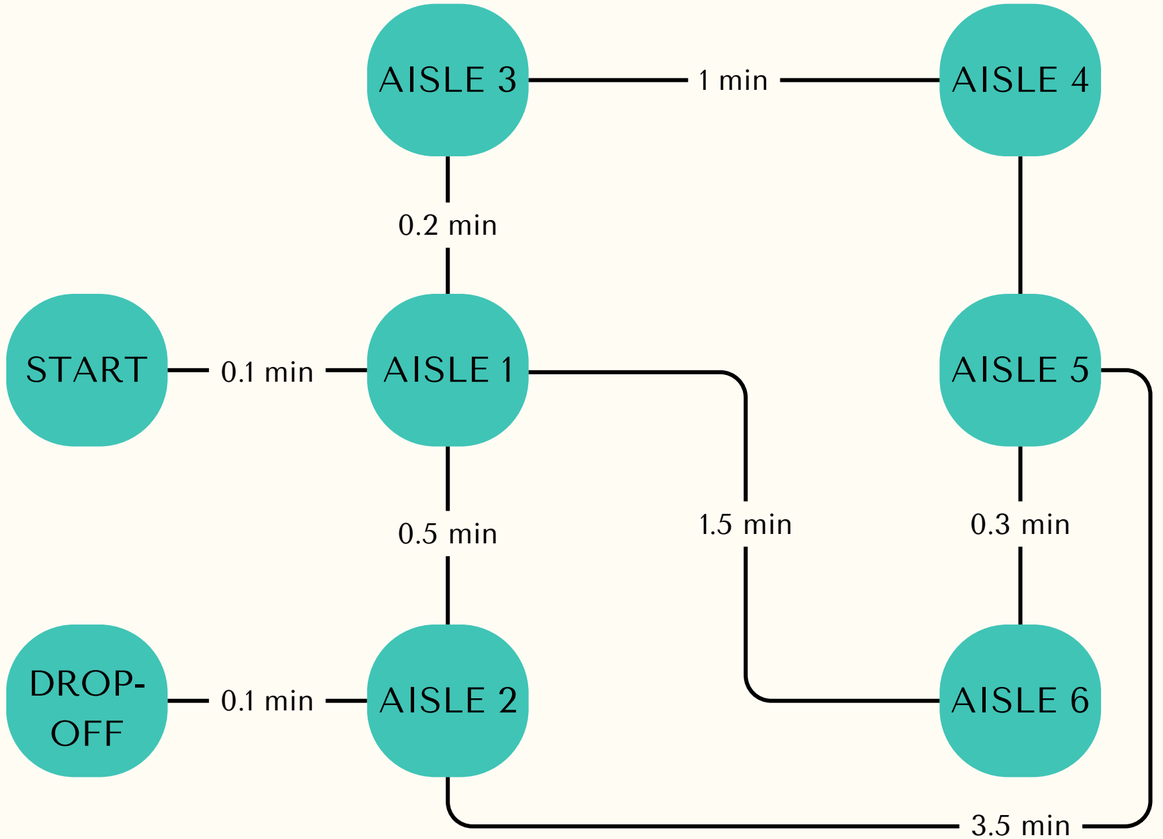
INTRODUCTION TO THE PROBLEM

You're an appliances supplier and you want to ship the items from your warehouse to give to your distributor and sell it as soon as possible. You have a forklift that can carry two items at a time and has a capacity weight of 2,000kg, you also have a truck that has 10,000kg capacity weight which is the storage of your items picked. You have a warehouse that has six aisles with different items. Your goal is to get the most preferable items in an aisle and putting it to the truck considering how many minutes you consume in the process. (Assuming the minutes are constant and picking time is not considered)

10,000 KG



2,000 KG



NAME	AISLE	WEIGHT	PRICE
SELF-N-FORGET COOKER	1	1458	2949
WASHING MACHINE	3	1408	2584
THERMAL MASS REFRIGERATOR	5	1313	1686
SOUS-VIDE COOKER	3	751	2958
ELECTRIC WATER BOILER	1	1302	4590
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
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
HUMIDIFIER	4	792	2690
PANINI SANDWICH GRILL	2	1241	3564
FLATTOP GRILL	4	1352	1545
MICROWAVE OVEN	1	616	4390
ICEBOX	3	1275	2405
		20951	58746

Iteration 1

Problem Identification

How to put items in the truck that satisfies the maximum weight in maximum price and consider the minutes that you consume in the process?

Decomposition

Sub-problems:

- Find the shortest parth/route of the forklift to aisle and back to truck
- Maximize the weight capacity of the forklift with preferable items
- Capacity of the truck must not exceed to 10,000 KG

Pattern Recognition

We can consider that the heavier items have a big price so we can consider on taking it first.

Abstraction

Relevant Information: Weight capacity and time

Irrelevant Information: Appliance supplier, distributors

Iteration 2

Problem Identification

I WANT TO FIND THE SHORTEST POSSIBLE ROUTE OF THE FORKLIFT TO THE AISLE AND BACK TO THE TRUCK WHICH ALGORITHM IS BEST FIT TO MAKE THIS WORK?

Decomposition

Pattern Recognition

Abstraction

IMPLEMENTATION OF CODE

First thing we do is make a class for the Items

```
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
```

Then, we created a function for menu list so that it is organized, and then proceed to create an array per category, by using the class and the function the values are stored efficiently.

```
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
```

```
# 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
```

Next, we will now create the knapsack function for the data and the truck, this also stores the aisle of the items to be picked

```
[ ]  item_menu = menu_of_items(items, aisle, price, weight) # This variable will be sent to the kanpsack function

     print(item_menu) # Checking the Items in the List
```

```
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
```

This part of the code will feed the items_menu in to the knapsack

```
[ ]  aisle = tab_knapsack(capacity, item_menu) # Display the Included Items, Their Aisle Num and the Weight (For Checking).
     print(aisle) #Checking the List of the aisle needed to be past through.
```

Now let's proceed to the making of the graph. We decided to take a simple route in making the graph, we create a dictionary to represent the graph and corresponding time in connecting edges.

```
# 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":[]
    }
```

```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
```

```python
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, a
        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

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 in
            records[len(records)] = [paths[i],paths[i+1]]
    return records
```

```
    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
```

RESULTS

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

```
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
```

```
[ ]  print(aisle)
```

```
[1, 2, 3, 4, 6]
```

```
[ ]  graph = Graph(g)
     path = graph.find_all_paths('START', 'DROP OFF') # All Possible Paths
     dicxts = graph.total_mins(path)
     print(dicxts) # The Dictionary Created
     print(graph.fastest_way(aisle, dicxts)) # The Order of the fastest way to get all the items in the warehouse
```

```
{0: [['START', 1, 3, 4, 5, 2, 'DROP OFF'], 5.3999999999999995], 1: [['START', 1, 2, 'DROP OFF'], 0.7], 2: [['START', 1, 6, 5, 2, 'DROP OFF'], 5.5]}
[[5.3999999999999995, 0.7, 5.5], [5.3999999999999995, 0.7, 5.5], [5.3999999999999995, 0, 0], [5.3999999999999995, 0, 0], [0, 0, 5.5]]
[1, 1, 0, 0, 2]
```