Hands-on Activity 1.1 | Optimization and Knapsack Problem

Objective(s):

This activity aims to demonstrate how to apply greedy and brute force algorithms to solve optimization problems

Intended Learning Outcomes (ILOs):

- Demonstrate how to solve knapsacks problems using greedy algorithm
- Demonstrate how to solve knapsacks problems using brute force algorithm

Resources:

- Jupyter Notebook

Procedures:

1. Create a Food class that defines the following:

- name of the food
- value of the food
- calories of the food

2. Create the following methods inside the Food class:

- A method that returns the value of the food
- A method that returns the cost of the food
- A method that calculates the density of the food (Value / Cost)
- A method that returns a string to display the name, value and calories of the food

```
1 class Food(object):
2     def __init__(self, n, v, w):
3         self.name = n
4         self.value = v
```

```
        self.calories = w
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + '>'
```

3. Create a buildMenu method that builds the name, value and calories of the food

```
def buildMenu(names, values, calories):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i]))
    return menu
```

4. Create a method greedy to return total value and cost of added food based on the desired maximum cost

```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

5. Create a testGreedy method to test the greedy method

```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)
```

```
def testGreedys(foods, maxUnits):
    print('Use greedy by value to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,          'calories')
```

```
4     print('\nUse greedy by cost to allocate', maxUnits,           'calories')
5     testGreedy(foods, maxUnits, lambda x: 1/Food.getCost(x))
6     print('\nUse greedy by density to allocate', maxUnits,          'calories')
7     testGreedy(foods, maxUnits, Food.density)
```

6. Create arrays of food name, values and calories
7. Call the buildMenu to create menu for food
8. Use testGreedys method to pick food according to the desired calories

```
1 names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
2 values = [89,90,95,100,90,79,50,10]
3 calories = [123,154,258,354,365,150,95,195]
4 foods = buildMenu(names, values, calories)
5 testGreedys(foods, 2000)
```

```
    Use greedy by value to allocate 2000 calories
    Total value of items taken = 603.0
        burger: <100, 354>
        pizza: <95, 258>
        beer: <90, 154>
        fries: <90, 365>
        wine: <89, 123>
        cola: <79, 150>
        apple: <50, 95>
        donut: <10, 195>

    Use greedy by cost to allocate 2000 calories
    Total value of items taken = 603.0
        apple: <50, 95>
        wine: <89, 123>
        cola: <79, 150>
        beer: <90, 154>
        donut: <10, 195>
        pizza: <95, 258>
        burger: <100, 354>
        fries: <90, 365>

    Use greedy by density to allocate 2000 calories
    Total value of items taken = 603.0
        wine: <89, 123>
        beer: <90, 154>
        cola: <79, 150>
        apple: <50, 95>
        pizza: <95, 258>
        burger: <100, 354>
        fries: <90, 365>
        donut: <10, 195>
```

Task 1: Change the maxUnits to 100

```
1 #type your code here
2 testGreedys(foods, 100)
```

```
    Use greedy by value to allocate 100 calories
    Total value of items taken = 50.0
        apple: <50, 95>

    Use greedy by cost to allocate 100 calories
    Total value of items taken = 50.0
        apple: <50, 95>

    Use greedy by density to allocate 100 calories
    Total value of items taken = 50.0
        apple: <50, 95>
```

Task 2: Modify codes to add additional weight (criterion) to select food items.

```
 1 # type your code here
 2 class FoodModified(object):
 3     def __init__(self, n, v, w, x):
 4         self.name = n
 5         self.value = v
 6         self.calories = w
 7         self.weight = x # ADDED
 8     def getValueMod(self):
 9         return self.value
10     def getCaloriesMod(self):
11         return self.calories
12     def densityMod(self):
13         return self.getValueMod()/self.getCaloriesMod()
14     def getWeightMod(self): # ADDED
15         return self.weight
16     def capacityMod(self):  # ADDED
17         return self.getValue()/self.getWeight()
18     def __str__(self):
19         return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', '+
20
21 def buildMenuMod(names, values, calories, weight): # Weight is ADDED
22     menu = []
23     for i in range(len(values)):
24       menu.append(FoodModified(names[i], values[i],calories[i], weight[i]))
25     return menu
26
27 def greedyMod(items, maxCalories, maxWeight, keyFunction): # Max weight is ADDED
28     """Assumes items a list, maxCalories >= 0,         keyFunction maps elements of it
29     itemsCopy = sorted(items, key = keyFunction, reverse = True)
30     result = []
31     totalValue, totalCalories, totalWeight = 0.0, 0.0, 0.0
32     for i in range(len(itemsCopy)):
33         if ((totalCalories+itemsCopy[i].getCaloriesMod()) <= maxCalories) and ((totall
```

```
34             result.append(itemsCopy[i])
35             totalCalories += itemsCopy[i].getCaloriesMod()
36             totalValue += itemsCopy[i].getValueMod()
37             totalWeight += itemsCopy[i].getWeightMod() # ADDED
38     return (result, totalValue)
39
40 def testGreedyMod(items, constraintCalories, contraintsWeight, keyFunction): # constra
41       taken, val = greedyMod(items, constraintCalories, contraintsWeight, keyFunction
42       print('Total value of items taken =', val)
43       for item in taken:
44           print('   ', item)
45
46 def testGreedysMod(foods, maxUnits, maxUnits2): # constraints for Weight is added
47       print('Use greedy by value to allocate', maxUnits,          'calories and ', max
48       testGreedyMod(foods, maxUnits, maxUnits2, FoodModified.getValueMod)
49       print('\nUse greedy by cost to allocate', maxUnits,          'calories and ', ma
50       testGreedyMod(foods, maxUnits, maxUnits2, lambda x: 1/FoodModified.getCaloriesMo
51       print('\nUse greedy by density to allocate', maxUnits, 'calories and ', maxUnits
52       testGreedyMod(foods, maxUnits, maxUnits2, FoodModified.densityMod)
```

Task 3: Test your modified code to test the greedy algorithm to select food items with your additional weight.

```
1 # type your code here
2 names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
3 values = [89,90,95,100,90,79,50,10]
4 calories = [123,154,258,354,365,150,95,195]
5 weight = [55, 75, 105, 125, 95, 29, 98, 49, 50] # Added Weight for Checking
6 foods = buildMenuMod(names, values, calories, weight)
7 testGreedysMod(foods, 2000, 200) # Checks if the item can satisfy both, Calories and
```

```
Use greedy by value to allocate 2000 calories and  200 weigths
Total value of items taken = 190.0
    burger: <100, 354, 125>
    beer: <90, 154, 75>

Use greedy by cost to allocate 2000 calories and  200 weigths
Total value of items taken = 218.0
    apple: <50, 95, 98>
    wine: <89, 123, 55>
    cola: <79, 150, 29>

Use greedy by density to allocate 2000 calories and  200 weigths
Total value of items taken = 258.0
    wine: <89, 123, 55>
    beer: <90, 154, 75>
    cola: <79, 150, 29>
```

9. Create method to use Bruteforce algorithm instead of greedy algorithm

```
def maxVal(toConsider, avail):
    """Assumes toConsider a list of items, avail a weight
       Returns a tuple of the total value of a solution to the
         0/1 knapsack problem and the items of that solution"""
    if toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getCost() > avail:
        #Explore right branch only
        result = maxVal(toConsider[1:], avail)
    else:
        nextItem = toConsider[0]
        #Explore left branch
        withVal, withToTake = maxVal(toConsider[1:],
                                     avail - nextItem.getCost())
        withVal += nextItem.getValue()
        #Explore right branch
        withoutVal, withoutToTake = maxVal(toConsider[1:], avail)
        #Choose better branch
        if withVal > withoutVal:
            result = (withVal, withToTake + (nextItem,))
        else:
            result = (withoutVal, withoutToTake)
    return result
```

```
def testMaxVal(foods, maxUnits, printItems = True):
    print('Use search tree to allocate', maxUnits,
          'calories')
    val, taken = maxVal(foods, maxUnits)
    print('Total costs of foods taken =', val)
    if printItems:
        for item in taken:
            print('   ', item)
```

```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
foods = buildMenu(names, values, calories)
testMaxVal(foods, 2400)
```

```
Use search tree to allocate 2400 calories
Total costs of foods taken = 603
    donut: <10, 195>
    apple: <50, 95>
    cola: <79, 150>
    fries: <90, 365>
    burger: <100, 354>
    pizza: <95, 258>
    beer: <90, 154>
    wine: <89, 123>
```

Supplementary Activity:

- Choose a real-world problem that solves knapsacks problem
- Use the greedy and brute force algorithm to solve knapsacks problem

PROBLEM

My mom sell Snacks so she wants to bring all the possible snacks she can sell, but since her bag has a certain capacity I need to help her identify the products that can give her more profit (given that she sell it all)

```
# Lets say my mom has 10 different packs of snacks
snacks = ['Toasted', 'Biscocho', 'Waffer', 'Tenga', 'Cookie Chips', 'Brownies', 'Eggnog
# with corresponding prices of
priceSnack = [55, 65, 70, 35, 100, 35, 85, 120, 45, 65]
# with these corresponding weights
weightSnack = [10, 20, 20, 10, 70, 50, 80, 50, 30, 50]
```

```
class Snack:
    def __init__(self, name, price, weight):
        self.name = name
        self.__price = price
        self.__weight = weight
    def getPrice(self):
        return self.__price
    def getWeight(self):
        return self.__weight
    def Ratio(self):
        return self.getPrice()/self.getWeight()
```

```
def buildLists(names, price, weight): # Weight is ADDED
    menu = []
    for i,j in enumerate(price):
      menu.append(Snack(names[i], j, weight[i]))
    return menu
```

```
def printG(result):
  for i in result:
    print(i, end=" ")
  print()
```

```
 7 def greedy(lists, maxWeight, keyDef):
 8   results = []
 9   sortedList = sorted(lists, key = keyDef ,reverse = True)
10   totalWeight, totalProfit = 0.0, 0.0
11   for i,j in enumerate(sortedList):
12     if j.getWeight() + totalWeight <= maxWeight:
13       results.append(j.name)
14       totalWeight += j.getWeight()
15       totalProfit += j.getPrice()
16   return printG([results, totalProfit, totalWeight])
17
18
19 def brute(consider, avail):
20   if consider == [] or avail == 0:
21         result = (0, ())
22   elif consider[0].getWeight() > avail:
23         result = brute(consider[1:], avail)
24   else:
25       nextItem = consider[0]
26       withVal, withToTake = brute(consider[1:], avail - nextItem.getWeight())
27       withVal += nextItem.getPrice()
28       withoutVal, withoutToTake = brute(consider[1:], avail)
29       if withVal > withoutVal:
30           result = (withVal, withToTake + (nextItem,))
31       else:
32           result = (withoutVal, withoutToTake)
33   return result
34
35 def printRes(result):
36   num =[]
37   for i in result:
38     if type(i) == int:
39       continue
40     else:
41       for j in i:
42         num.append(j.name)
43   print(num)
```

```
1 itemList = buildLists(snacks, priceSnack, weightSnack)
2 print("RESULTS IN GREEDY")
3 greedy(itemList, 100, Snack.getWeight)
4 print("RESULTS IN BRUTE FORCE")
5 printRes(brute(itemList, 100))
```

```
RESULTS IN GREEDY
['Eggnogs', 'Biscocho'] 150.0 100.0
RESULTS IN BRUTE FORCE
['Stick Os', 'Waffer', 'Biscocho', 'Toasted']
```

Conclusion:

Conclusion:

type your conclusion here

Greedy algorithm makes you blow your mind if you dont understand it but i will be good when you absorbed it well. brute force also, as the name suggest force all the positive patterns and results in his first or better answer while the greedy focused in more whose more profit in the end.

1