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
1. 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

```python
class Food(object):
    def __init__(self, n, v, w, d):
        # Make the variables private
        self.name = n
        self.value = v
        self.calories = w
        self.diseases = d
    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) + self.diseases + '>'
```

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

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

1. 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 to numbers"""
    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)
```

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

1. Create arrays of food name, values and calories
2. Call the buildMenu to create menu for food
3. Use testGreedys method to pick food according to the desired calories

```
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]
possible_disease = ['alcoholic', 'beercoholic', 'pizzcoholic',
'cardiac arrest', 'diabetie2', 'doctorphobia', 'diabetis3',
'googlemaps', 'covid 20205']
foods = buildMenu(names, values, calories, possible_disease)
testGreedys(foods, 2000)

Use greedy by value to allocate 2000 calories
Total value of items taken = 603.0
    burger: <100, 354cardiac arrest>
    pizza: <95, 258pizzcoholic>
    beer: <90, 154beercoholic>
    fries: <90, 365diabetie2>
    wine: <89, 123alcoholic>
    cola: <79, 150doctorphobia>
    apple: <50, 95diabetis3>
    donut: <10, 195googlemaps>

Use greedy by cost to allocate 2000 calories
Total value of items taken = 603.0
    apple: <50, 95diabetis3>
    wine: <89, 123alcoholic>
    cola: <79, 150doctorphobia>
    beer: <90, 154beercoholic>
    donut: <10, 195googlemaps>
    pizza: <95, 258pizzcoholic>
    burger: <100, 354cardiac arrest>
    fries: <90, 365diabetie2>

Use greedy by density to allocate 2000 calories
Total value of items taken = 603.0
    wine: <89, 123alcoholic>
    beer: <90, 154beercoholic>
    cola: <79, 150doctorphobia>
    apple: <50, 95diabetis3>
    pizza: <95, 258pizzcoholic>
    burger: <100, 354cardiac arrest>
    fries: <90, 365diabetie2>
    donut: <10, 195googlemaps>
```

Task 1: Change the maxUnits to 100

```
testGreedys(foods, 100) # Changed 2000 to 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.

```
possible_disease = ['alcoholic', 'beercoholic', 'pizzcoholic',
'cardiac arrest', 'diabetie2', 'doctorphobia', 'diabetis3',
'googlemaps', 'covid 20205']
foods = buildMenu(names, values, calories, possible_disease)
```

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

```
class Food(object):
    def __init__(self, n, v, w, d):
        # Make the variables private
        self.name = n
        self.value = v
        self.calories = w
        self.diseases = d
    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) + ', '+self.diseases + '>'

def buildMenu(names, values, calories, possible_disease):
    menu = []
    for i in range(len(values)):

menu.append(Food(names[i],values[i],calories[i],possible_disease[i]))
    return menu

testGreedys(foods, 2333)

Use greedy by value to allocate 2333 calories
Total value of items taken = 603.0
    burger: <100, 354, cardiac arrest>
    pizza: <95, 258, pizzcoholic>
    beer: <90, 154, beercoholic>
    fries: <90, 365, diabetie2>
    wine: <89, 123, alcoholic>
    cola: <79, 150, doctorphobia>
    apple: <50, 95, diabetis3>
    donut: <10, 195, googlemaps>

Use greedy by cost to allocate 2333 calories
Total value of items taken = 603.0
    apple: <50, 95, diabetis3>
    wine: <89, 123, alcoholic>
    cola: <79, 150, doctorphobia>
    beer: <90, 154, beercoholic>
    donut: <10, 195, googlemaps>
    pizza: <95, 258, pizzcoholic>
    burger: <100, 354, cardiac arrest>
    fries: <90, 365, diabetie2>

Use greedy by density to allocate 2333 calories
Total value of items taken = 603.0
    wine: <89, 123, alcoholic>
    beer: <90, 154, beercoholic>
    cola: <79, 150, doctorphobia>
    apple: <50, 95, diabetis3>
    pizza: <95, 258, pizzcoholic>
    burger: <100, 354, cardiac arrest>
    fries: <90, 365, diabetie2>
    donut: <10, 195, googlemaps>
```

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

```python
def bruteforcie(items, avail):
    bestVal = 0
    bestCombo = ()
    def generateSubsets(items):
        if not items:
            return [[]]
        first = items[0]
        restSubsets = generateSubsets(items[1:])
        return restSubsets + [subset + [first] for subset in
restSubsets]

    for subset in generateSubsets(items):
        totalCost = sum(item.getCost() for item in subset)
        totalValue = sum(item.getValue() for item in subset)

        if totalCost <= avail and totalValue > bestVal:
            bestVal = totalValue
            bestCombo = tuple(subset)

    return bestVal, bestCombo
```

```python
def testBruteForce(foods, maxUnits, printItems=True):
    print("Use brute force to allocate", maxUnits, "calories")
    val, taken = bruteforcie(foods, maxUnits)
    print("Total value 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]
possible_disease = ['alcoholic', 'beercoholic', 'pizzcoholic', 
'cardiac arrest', 'diabetie2', 'doctorphobia', 'diabetis3', 
'googlemaps', 'covid 20205']
foods = buildMenu(names, values, calories, possible_disease)
testBruteForce(foods, 2400)
```

```
Use brute force to allocate 2400 calories
Total value of foods taken = 603
    donut: <10, 195, googlemaps>
    apple: <50, 95, diabetis3>
    cola: <79, 150, doctorphobia>
    fries: <90, 365, diabetie2>
    burger: <100, 354, cardiac arrest>
    pizza: <95, 258, pizzcoholic>
    beer: <90, 154, beercoholic>
    wine: <89, 123, alcoholic>
```

Supplementary Activity:

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

```python
class List(object):
    def __init__(self, e, c):
        self.name = e
        self.value = c

    def getCost(self):
        return self.value

    def __str__(self):
        return self.name + ':<' + str(self.value) + ' Pesos>'

def listCreate(names, values):
    listing = []
    for i in range(len(values)):
        listing.append(List(names[i], values[i]))
    return listing

# Greedy Algo
```

```python
def greedy(items, maxCost):
    itemsCopy = sorted(items, key=lambda x: x.getCost(), reverse=True)
    result = []
    totalCost = 0.0
    for item in itemsCopy:
        if (totalCost + item.getCost()) <= maxCost:
            result.append(item)
            totalCost += item.getCost()
    return (result, totalCost)

def testgreedy(items, constraint):
    taken, val = greedy(items, constraint)
    print('Total cost of items taken =', val)
    for item in taken:
        print('    ', item)

# Bruteforce
def bruteforcie(items, avail):
    bestVal = 0
    bestCombo = ()

    def generateSubsets(items):
        if not items:
            return [[]]
        first = items[0]
        restSubsets = generateSubsets(items[1:])
        return restSubsets + [subset + [first] for subset in
restSubsets]

    for subset in generateSubsets(items):
        totalCost = sum(item.getCost() for item in subset)

        if totalCost <= avail and totalCost > bestVal:
            bestVal = totalCost
            bestCombo = tuple(subset)

    return bestVal, bestCombo

def testBruteForce(lists, maxUnits, printItems=True):
    print("Use brute force to allocate cost")
    val, taken = bruteforcie(lists, maxUnits)
    print("Total cost =", val)
    if printItems:
        for item in taken:
            print("    ", item)

expense = ['Snack', 'Jeep to LRT', 'LRT to Jeep', 'LRT to Anonas',
'LRT to Antipolo', 'Tricycle', 'E-Jeep', 'Meal']
cost = [30, 20, 19, 30, 40, 30, 50, 30]
```

```
lists = listCreate(expense, cost)
testBruteForce(lists, 248)
print("\nUse Greedy Algorithm to allocate cost")
testgreedy(lists, 250)
```

```
Use brute force to allocate cost
Total cost = 230
    Meal:<30 Pesos>
    E-Jeep:<50 Pesos>
    Tricycle:<30 Pesos>
    LRT to Antipolo:<40 Pesos>
    LRT to Anonas:<30 Pesos>
    Jeep to LRT:<20 Pesos>
    Snack:<30 Pesos>

Use Greedy Algorithm to allocate cost
Total cost of items taken = 249.0
    E-Jeep:<50 Pesos>
    LRT to Antipolo:<40 Pesos>
    Snack:<30 Pesos>
    LRT to Anonas:<30 Pesos>
    Tricycle:<30 Pesos>
    Meal:<30 Pesos>
    Jeep to LRT:<20 Pesos>
    LRT to Jeep:<19 Pesos>
```

Conclusion:

Doing this activity helped men understand the difference between greedy and brute force algorithms. The greedy method was faster and easier to apply, but I realized that it doesn't always give the most "Optimized"/"Accurate" results. I was also able to have a short review about Python classes, to use it to represent real-world problems like the activity here. From food choices or even expenses, which made the activity more practical and relatable. Overall, this exercise made me realized on how both brute force algorithm and greedy algorithm works, On how it can be used as a guide decision-making when resources are limited, and it gave me confidence in apply there concepts to everyday situations.