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

```
class Food(object):
    def __init__(self, n, v, w, x):
        self.name = n
        self.value = v
        self.calories = w
        self.weight = x
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def getWeight(self):
        return self.weight
    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, weight):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i], weight[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 to numbers"""
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost, totalWeight = 0.0, 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()
            totalWeight += itemsCopy[i].getWeight()
    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')
    testGreedy(foods, maxUnits, lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.density)
    print('\nUse greedy by weight to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getWeight)
```

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

```
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]
weight = [350, 300, 1000, 120, 150, 200, 100, 30]
foods = buildMenu(names, values, calories, weight)
testGreedys(foods, 2000) # an updated output
```

```
Allocate  2000 pesos by value using greedy
Total value of games taken = 150.0
  Palworld has a value of 80 to the buyer and costs 910 pesos
  GTA V has a value of 70 to the buyer and costs 635.91 pesos
Total amount of games bought = 1545.9099999999999  pesos
2454 is your change when rounded off.
Allocate  2000 pesos by cost using greedy
Total value of games taken = 110.0
  GTA V has a value of 70 to the buyer and costs 635.91 pesos
  The Outlast Trials has a value of 40 to the buyer and costs 682.5 pesos
Total amount of games bought = 1318.4099999999999  pesos
2682 is your change when rounded off.
```

Task 1: Change the maxUnits to 100

```
#type your code here
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.

```
# type your code here
# all comments direct the modifications
class Food(object):
    def __init__(self, n, v, w, x):
        # make variables private
        self.__name = n
        self.__value = v
        self.__calories = w
        self.__weight = x # added weight variable
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def getWeight(self): # added method for returning weight
        return self.weight
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + '>'

def buildMenu(names, values, calories, weight):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i], weight[i])) # including weight in the criteria of food menu
    return menu

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, totalWeight = 0.0, 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()
            totalWeight += itemsCopy[i].getWeight() # added getter of total for weight
    return (result, totalValue)

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)
    print('\nUse greedy by weight to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getWeight) # added function in which food weight is sorted in reverse order as well

weight = [350, 300, 1000, 120, 150, 200, 100, 30] # assigning weights to the foods
```

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

```
# type your code here\
testGreedys(foods, 2000) # the addition is in the 4th allocation of 2000 calories in which pizza is the heaviest and donut is the lightest

     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>

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

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']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
foods = buildMenu(names, values, calories, weight)
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

Budgeting money to buy online video games

A problem for gamers whenever there is a sale event on a gaming store.

```
class Game(object):
    def __init__(self, n, v, c):
        self.name = n
        self.value = v
        self.cost = c
    def getValue(self):
        return self.value
    def getCost(self):
        return self.cost
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ' has a value of ' + str(self.value)+ ' to the buyer and costs ' + str(self.cost) + ' pesos'

def buildOptions(names, values, cost):
        options = []
        for i in range(len(values)):
            options.append(Game(names[i], values[i], cost[i]))
        return options

def greedy(items, maxCost, keyFunction):
        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, totalCost)

def testGreedy(items, constraint, keyFunction):
        taken, val, amount = greedy(items, constraint, keyFunction)
        print('Total value of games taken =', val)
        for item in taken:
            print(' ', item)
        print('Total amount of games bought =', amount, 'pesos')
        change = 4000 - amount
        print(round(change), 'is your change when rounded off.')

def testGreedyz(options, maxUnits):
        print('Allocate', maxUnits,          'pesos by value using greedy:')
        testGreedy(games, maxUnits, Game.getValue)
        print()
        print('Allocate', maxUnits,          'pesos by cost using greedy:')
        testGreedy(games, maxUnits, lambda x: 1/Game.getCost(x))


names = ['Tekken 8', 'GTA V', 'Rainbow Six Siege', 'Palworld', 'The Outlast Trials']
values = [90, 70, 60, 80, 40]
costs = [2799.95, 635.91, 750, 910, 682.5]
games = buildOptions(names, values, costs)
testGreedyz(games, 4000)
```

```
Allocate 4000 pesos by value using greedy:
Total value of games taken = 170.0
  Tekken 8 has a value of 90 to the buyer and costs 2799.95 pesos
  Palworld has a value of 80 to the buyer and costs 910 pesos
Total amount of games bought = 3709.95  pesos
290 is your change when rounded off.

Allocate 4000 pesos by cost using greedy:
Total value of games taken = 250.0
  GTA V has a value of 70 to the buyer and costs 635.91 pesos
  The Outlast Trials has a value of 40 to the buyer and costs 682.5 pesos
  Rainbow Six Siege has a value of 60 to the buyer and costs 750 pesos
  Palworld has a value of 80 to the buyer and costs 910 pesos
Total amount of games bought = 2978.41  pesos
1022 is your change when rounded off.
```

```
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('Allocate', maxUnits, 'pesos by using search tree')
    val, taken = maxVal(foods, maxUnits)
    print('Total value of games taken =', val)
    if printItems:
        for item in taken:
            print('  ', item)

names = ['Tekken 8', 'GTA V', 'Rainbow Six Siege', 'Palworld', 'The Outlast Trials']
values = [90, 70, 60, 80, 40]
costs = [2799.95, 635.91, 750, 910, 682.5]
games = buildOptions(names, values, costs)
testMaxVal(games, 4000)
```

```
Allocate 4000 pesos by using search tree
Total value of games taken = 250
   The Outlast Trials has a value of 40 to the buyer and costs 682.5 pesos
   Palworld has a value of 80 to the buyer and costs 910 pesos
   Rainbow Six Siege has a value of 60 to the buyer and costs 750 pesos
   GTA V has a value of 70 to the buyer and costs 635.91 pesos
```

Conclusion:

> In order to grasp optimization, one must acknowledge that Knapsack problems are a common issue in our daily lives. Consequently, algorithms such as greedy and brute force rise to the occasion. These algorithms exist for problems to be solved using the most optimal solution or to detect the most unideal one. In conclusion, I managed to solve optimization problems by applying greedy and brute force algorithms by studying the provided code despite its complexity.