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):
        # Make the variables private
        self.name = n
        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 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)
```

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)
```

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]
foods = buildMenu(names, values, calories)
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

```
#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

class Food(object):
    def __init__(self, n, v, w, cap):
        self.name = n
        self.value = v
        self.calories = w
        self.capacity = cap
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def getCapacity(self):
        return self.capacity
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', ' + str(self.capacity) + '>'

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

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 capacity to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getCapacity)
```

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

```
# type your code here

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]
capacity = [2,3,4,5,6,7,8,9]

foods = buildMenu(names, values, calories, capacity)
testGreedys(foods, 2000)

     Use greedy by value to allocate 2000 calories
     Total salary of Tutor Hours = 603.0
         burger: <100, 354, 5>
         pizza: <95, 258, 4>
         beer: <90, 154, 3>
         fries: <90, 365, 6>
         wine: <89, 123, 2>
         cola: <79, 150, 7>
         apple: <50, 95, 8>
         donut: <10, 195, 9>

     Use greedy by cost to allocate 2000 calories
     Total salary of Tutor Hours = 603.0
         apple: <50, 95, 8>
         wine: <89, 123, 2>
         cola: <79, 150, 7>
         beer: <90, 154, 3>
         donut: <10, 195, 9>
         pizza: <95, 258, 4>
         burger: <100, 354, 5>
         fries: <90, 365, 6>

     Use greedy by density to allocate 2000 calories
     Total salary of Tutor Hours = 603.0
         wine: <89, 123, 2>
         beer: <90, 154, 3>
         cola: <79, 150, 7>
         apple: <50, 95, 8>
         pizza: <95, 258, 4>
         burger: <100, 354, 5>
         fries: <90, 365, 6>
         donut: <10, 195, 9>

     Use greedy by capacity to allocate 2000 calories
     Total salary of Tutor Hours = 603.0
         donut: <10, 195, 9>
         apple: <50, 95, 8>
         cola: <79, 150, 7>
         fries: <90, 365, 6>
         burger: <100, 354, 5>
         pizza: <95, 258, 4>
         beer: <90, 154, 3>
         wine: <89, 123, 2>
```

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 real-world problem that I chose that solves knapsacks problem is list of subjects and days of tutoring with their corresponding salary and hours. The concept is that to find the job that has the least consume hours and high salary.

```
class Tutor(object):
    def __init__(self, name, salary, hours, subjects):
        self.name = name
        self.value = salary
        self.hour = hours
        self.subject = subjects

    def getName(self):
        return self.name

    def getValue(self):
        return self.value

    def getCost(self):
        return self.hour

    def getSubject(self):
        return self.subject

    def density(self):
        return self.value/self.hour

    def __str__(self):
        return self.name + ' <' + str(self.value) + ', ' + str(self.hour) + ', ' + str(self.subject) + '>'

def buildMenu(names, values, hour, subject):
    menu = []
    for i in range(len(values)):
        menu.append(Tutor(names[i], values[i],hour[i],subject[i]))
    return menu

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


# Data
TutorDay = ['Day1','Day2','Day3','Day4','Day5','Day6','Day7','Day8','Day9','Day10']
hours = [2,4,6,2,3,4,6,3,5,2]
salaries = [500,700,1000,500,600,700,1000,900,800,500]
subs = ['Filipino','English','Differential Equation','General Education','Algebra','Science','Calculus','Discrete Math', 'Statis

tutorVal = buildMenu(TutorDay, salaries, hours, subs)


def testMaxVal(tutorVal, maxUnits, printItems = True):
    print('Use search tree to allocate', maxUnits,
          'hours')
    val, taken = maxVal(tutorVal, maxUnits)
    print('Total salary of Tutor Hours =', val)
    if printItems:
        for item in taken:
            print('   ', item)


testMaxVal(tutorVal, 6)
```

```
 Use search tree to allocate 6 hours
 Total salary of Tutor Hours = 1500
     Day8 <900, 3, Discrete Math>
     Day5 <600, 3, Algebra>
```

```
def testGreedys(foods, maxUnits):
    print('Use greedy by hours to allocate', maxUnits,          'hours')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by salary to allocate', maxUnits,          'hours')
    testGreedy(foods, maxUnits, lambda x: 1/Tutor.getCost(x))
    print('\nUse greedy by hourly rate to allocate', maxUnits,          'hours')
    testGreedy(foods, maxUnits, Food.density)
```

```
testGreedys(tutorVal, 6)

     Use greedy by hours to allocate 6 hours
     Total salary of Tutor Hours = 1000.0
         Day3 <1000, 6, Differential Equation>

     Use greedy by salary to allocate 6 hours
     Total salary of Tutor Hours = 1500.0
         Day1 <500, 2, Filipino>
         Day4 <500, 2, General Education>
         Day10 <500, 2, Economics>

     Use greedy by hourly rate to allocate 6 hours
     Total salary of Tutor Hours = 1400.0
         Day8 <900, 3, Discrete Math>
         Day1 <500, 2, Filipino>
```

- Conclusion: