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, l):
        self.name = n
        self.value = v
```

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

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 nu
    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]
weight = [10, 20, 30, 11, 25, 12, 34, 18, 22]
foods = buildMenu(names, values, calories, weight)
testGreedys(foods, 2000)
```

```
Use greedy by value to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>

Use greedy by cost to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>
```

Task 1: Change the maxUnits to 100

```
#type your code here
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 density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getWeight)

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 = [10, 20, 30, 11, 25, 12, 34, 18, 22]
```

```
foods = buildMenu(names, values, calories, weight)
testGreedys(foods, 100)
```

```
     Use greedy by value to allocate 100 calories
     Total value of items taken = 50.0
         apple: <50, 95, 34>

     Use greedy by cost to allocate 100 calories
     Total value of items taken = 50.0
         apple: <50, 95, 34>

     Use greedy by density to allocate 100 calories
     Total value of items taken = 50.0
         apple: <50, 95, 34>

     Use greedy by density to allocate 100 calories
     Total value of items taken = 50.0
         apple: <50, 95, 34>
```

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

```
class Food(object):
    def __init__(self, n, v, w, l):
        self.name = n
        self.value = v
        self.calories = w
        self.weight = l
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def getWeight(self):
        return self.weight
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', ' + st

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

def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items to
    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)

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

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, l):
        self.name = n
        self.value = v
        self.calories = w
        self.weight = l
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def getWeight(self):
        return self.weight
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', ' + st

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

def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items to
    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)

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 density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getWeight)

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 = [10, 20, 30, 11, 25, 12, 34, 18, 22]
foods = buildMenu(names, values, calories, weight)
testGreedys(foods, 100)
```

```
Use greedy by value to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>

Use greedy by cost to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 34>
```

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

```
class Food(object):
    def __init__(self, n, v, w, l):
        self.name = n
        self.value = v
        self.calories = w
```

```
        self.weight = l
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def getWeight(self):
        return self.weight
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', ' + st

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

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]
weight = [10, 20, 30, 11, 25, 12, 34, 18, 22]
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, 18>
    apple: <50, 95, 34>
    cola: <79, 150, 12>
    fries: <90, 365, 25>
    burger: <100, 354, 11>
    pizza: <95, 258, 30>
    beer: <90, 154, 20>
    wine: <89, 123, 10>
```

Supplementary Activity:

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

```
class Supplements(object):
    def __init__(self, n, v, w, l):
        self.name = n
        self.value = v
        self.calories = w
        self.protein = l
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def getProtein(self):
        return self.protein
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', ' + st

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

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

```
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items to
    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)

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(supplements, maxUnits):
    print('Use greedy by value to allocate', maxUnits,          'calories')
    testGreedy(supplements, maxUnits, Supplements.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,          'calories')
    testGreedy(supplements, maxUnits, lambda x: 1/Supplements.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(supplements, maxUnits, Supplements.density)
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(supplements, maxUnits, Supplements.getProtein)

names = ['Oats', 'Egg', 'Milk', 'Coffee', 'Shake','Amino', 'Fish oil', 'Whey', 'Greek yog
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
protein = [10, 20, 30, 11, 25, 12, 34, 18, 22]
supplements = buildMenu(names, values, calories, protein)
testGreedys(supplements, 100)
```

```
Use greedy by value to allocate 100 calories
Total value of items taken = 50.0
    Fish oil: <50, 95, 34>

Use greedy by cost to allocate 100 calories
Total value of items taken = 50.0
    Fish oil: <50, 95, 34>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    Fish oil: <50, 95, 34>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    Fish oil: <50, 95, 34>
```

```
class Supplements(object):
```

```
    def __init__(self, n, v, w, l):
        self.name = n
        self.value = v
        self.calories = w
        self.protein = l
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def getProtein(self):
        return self.protein
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', ' + str(

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

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(supplements, maxUnits, printItems = True):
    print('Use search tree to allocate', maxUnits,
          'calories')
    val, taken = maxVal(supplements, maxUnits)
```

```
    print('Total costs of foods taken =', val)
    if printItems:
        for item in taken:
            print('   ', item)

names = ['Oats', 'Egg', 'Milk', 'Coffee', 'Shake','Amino', 'Fish oil', 'Whey', 'Greek yogu
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
protein = [10, 20, 30, 11, 25, 12, 34, 18, 22]
supplements = buildMenu(names, values, calories, protein)
testMaxVal(supplements, 500)
```

```
Use search tree to allocate 500 calories
Total costs of foods taken = 258
    Amino: <79, 150, 12>
    Egg: <90, 154, 20>
    Oats: <89, 123, 10>
```

Conclusion:

In conclusion, the things i learned todays le
method and also the brute force but now inter
series of tupple which is required for the cc
already given to us which made this activity
a lot.

In conclusion, the things i learned todays lesson is how to use the greedy method and also the brute force but now intentionally using it. We used a series of tupple which is required for the code to work, the functions are already given to us which made this activity quite easy but also made us learn a lot.