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

In [461]:
```
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

In [462]:
```
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

In [463]:
```
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

In [464]:
```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)
```

In [465]:
```
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

In [466]:
```
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

In [467]:
```
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, 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.

In [468]:
```
class Food(object):
    def __init__(self, n, v, w, d):
        self.name = n
        self.value = v
        self.calories = w
        self.price = d
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def getPrice(self):
        return self.price
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' +
str(self.calories) + ', ' + str(self.price) + '>'

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

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]
price = [900, 200, 650, 75, 50, 25, 10, 80, 500]
foods = buildMenu(names, values, calories, price)
```

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

In [469]:
```
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 price to allocate', maxUnits,
'Pesos')
    testGreedy(foods, maxUnits, lambda x: 1/Food.getPrice(x))


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]
price = [900, 200, 650, 75, 50, 25, 10, 80, 500]
foods = buildMenu(names, values, calories, price)
testGreedys(foods, 1500)
```

```
Use greedy by value to allocate 1500 calories
Total value of items taken = 593.0
    burger: <100, 354, 75>
    pizza: <95, 258, 650>
    beer: <90, 154, 200>
    fries: <90, 365, 50>
    wine: <89, 123, 900>
    cola: <79, 150, 25>
    apple: <50, 95, 10>

Use greedy by cost to allocate 1500 calories
Total value of items taken = 513.0
    apple: <50, 95, 10>
    wine: <89, 123, 900>
    cola: <79, 150, 25>
    beer: <90, 154, 200>
    donut: <10, 195, 80>
    pizza: <95, 258, 650>
    burger: <100, 354, 75>

Use greedy by density to allocate 1500 calories
Total value of items taken = 593.0
    wine: <89, 123, 900>
```

```
    beer: <90, 154, 200>
    cola: <79, 150, 25>
    apple: <50, 95, 10>
    pizza: <95, 258, 650>
    burger: <100, 354, 75>
    fries: <90, 365, 50>

Use greedy by price to allocate 1500 Pesos
Total value of items taken = 508.0
    apple: <50, 95, 10>
    cola: <79, 150, 25>
    fries: <90, 365, 50>
    burger: <100, 354, 75>
    donut: <10, 195, 80>
    beer: <90, 154, 200>
    wine: <89, 123, 900>
```

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

In [470]:

```
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
```

In [471]:
```
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)
```

In [472]:
```
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]
price = [900, 200, 650, 75, 50, 25, 10, 80, 500]
foods = buildMenu(names, values, calories, price)
testMaxVal(foods, 2400)
```

```
Use search tree to allocate 2400 calories
Total costs of foods taken = 603
    donut: <10, 195, 80>
    apple: <50, 95, 10>
    cola: <79, 150, 25>
    fries: <90, 365, 50>
    burger: <100, 354, 75>
    pizza: <95, 258, 650>
    beer: <90, 154, 200>
    wine: <89, 123, 900>
```

Supplementary Activity:

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

Using Greedy Algorithm

In [473]:
```
class Commute(object):
    def __init__(self, n, d, w):
        self.place = n
        self.distance = d
        self.travel_fare = w
    def getPlace(self):
        return self.place
    def getDistance(self):
        return self.distance
    def getFare(self):
        return self.travel_fare
    def __str__(self):
        return self.place + ' : <' + str(self.distance)+ ', ' +
str(self.travel_fare) + '>'
```

In [474]:
```
def travel(place, distance, travel_fare):
    A = []
    for i in range(len(values)):
        A.append(Commute(place[i], distance[i], travel_fare[i]))
    return A
```

In [475]:
```
def greedy(places, maxCost, keyFunction):
    placesCopy = sorted(places, key = keyFunction,
                        reverse = True)
    result = []
    totalCost, totalValue = 0.0, 0.0
    for i in range(len(placesCopy)):
        if (totalCost+placesCopy[i].getFare()) <= maxCost:
            result.append(placesCopy[i])
            totalCost += placesCopy[i].getFare()
            totalValue += placesCopy[i].getDistance()

    return (result, totalValue)
```

In [476]:
```
def testGreedy(places, constraint, keyFunction):
    taken, val = greedy(places, constraint, keyFunction)
    print('Total places that can be visited =', val)
    for item in taken:
        print(" ", item)
```

In [477]:
```
def testGreedys(visits, maxUnits):
    print('Use greedy by distance to allocate', maxUnits,
'Pesos')
    testGreedy(visits, maxUnits, Commute.getDistance)
    print('\nUse greedy by travel fare to allocate', maxUnits,
'Pesos')
    testGreedy(visits, maxUnits, lambda x: 1/Commute.getFare(x))
```

In [478]:
```
place = ['Pasig', 'Taytay', 'Quezon', 'Cainta', 'Davao','Bicol',
'Dumaguete', 'Cebu', 'Boracay']
distance = [10,13,4,10,1470,444,936,448] #measured in km
travel_fare = [19,22,13,19,300,150,250,155]
visits = A(place, distance, travel_fare)
testGreedys(visits, 3000)
```

```
Use greedy by distance to allocate 3000 Pesos
Total places that can be visited = 3335.0
  Davao : <1470, 300>
  Dumaguete : <936, 250>
  Cebu : <448, 155>
  Bicol : <444, 150>
  Taytay : <13, 22>
  Pasig : <10, 19>
  Cainta : <10, 19>
  Quezon : <4, 13>

Use greedy by travel fare to allocate 3000 Pesos
Total places that can be visited = 3335.0
  Quezon : <4, 13>
  Pasig : <10, 19>
  Cainta : <10, 19>
  Taytay : <13, 22>
  Bicol : <444, 150>
  Cebu : <448, 155>
  Dumaguete : <936, 250>
  Davao : <1470, 300>
```

Using Brute Force Algorithm

In [479]:
```
def maxVal(toConsider, avail):
    if toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getFare() > avail:
        #Explore right branch only
        result = maxVal(toConsider[1:], avail)
    else:
        nextItem = toConsider[0]
        #Explore left branch
        withVal, withToTake = maxVal(toConsider[1:],
                                     avail - nextItem.getFare())
        withVal += nextItem.getDistance()
        #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
```

In [480]:
```
def testMaxVal(visits, maxUnits, printItems = True):
    print('Use search tree to allocate', maxUnits,
          'Pesos')
    val, taken = maxVal(visits, maxUnits)
    print('Total costs of foods taken =', val)
    if printItems:
        for item in taken:
            print('   ', item)
```

In [481]:
```
place = ['Pasig', 'Taytay', 'Quezon', 'Cainta', 'Davao','Bicol',
'Dumaguete', 'Cebu', 'Boracay']
distance = [10,13,4,10,1470,444,936,448] #measured in km
travel_fare = [19,22,13,19,300,150,250,155]
visits = A(place, distance, travel_fare)
testGreedys(visits, 3000)
```

```
Use greedy by distance to allocate 3000 Pesos
Total places that can be visited = 3335.0
  Davao : <1470, 300>
  Dumaguete : <936, 250>
  Cebu : <448, 155>
  Bicol : <444, 150>
  Taytay : <13, 22>
  Pasig : <10, 19>
  Cainta : <10, 19>
  Quezon : <4, 13>

Use greedy by travel fare to allocate 3000 Pesos
Total places that can be visited = 3335.0
  Quezon : <4, 13>
  Pasig : <10, 19>
  Cainta : <10, 19>
  Taytay : <13, 22>
  Bicol : <444, 150>
  Cebu : <448, 155>
  Dumaguete : <936, 250>
  Davao : <1470, 300>
```

Conclusion:

This hands-on activity has aided me in learning even more about greedy and brute force algorithm. It was fun being able to learn how to code a program that chooses the optimal solution based on the limitations that I have set. Usually it takes a much longer time to think about these solutions manually, but seeing how a program does it so quickly truly fascinates me. In all honesty, although it is not my first time encountering the brute force algorithm, I have not really understood it well in the past. Being able to accomplish this activity makes me truly proud of myself. I am looking forward to learning even more in the following weeks to come. I am excited to try new things in the future laboratories as well.