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

In [56]:
```
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) + '>'
```

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

In [57]:
```
def buildMenu(names, values, calories):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i]))
    return menu
```

1. Create a method greedy to return total value and cost of added food based on the desired maximum cost

In [58]:
```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items
    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

In [59]:
```
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 [60]:
```
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

In [61]:
```
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 [62]:
```
#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]
foods = buildMenu(names, values, calories)
testGreedys(foods, 100) #maxUnits changed from 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.

In [67]:
```
# type your code here
def greedy(items, maxCost, keyFunction, secondKeyFunction=None):
    """Assumes items a list, maxCost >= 0,
```

```
        keyFunction maps elements of items to numbers
        secondKeyFunction maps elements, items to numbers"""
    itemsCopy = sorted(items, key=lambda x: (keyFunction(x), secondKeyFunction(x) if s
                       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)
```

In [70]:
```
def testGreedy(items, constraint, keyFunction, secondKeyFunction=None):
    taken, val = greedy(items, constraint, keyFunction, secondKeyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)
```

In [68]:
```
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 value with additional weight to allocate', maxUnits, 'calor
    testGreedy(foods, maxUnits, Food.getValue, lambda x: x.calories)  # addtional weig
```

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

In [73]:
```
# type your code here
testGreedys(foods, 1000)
```

```
Use greedy by value to allocate 1000 calories
Total value of items taken = 424.0
    burger: <100, 354>
    pizza: <95, 258>
    beer: <90, 154>
    wine: <89, 123>
    apple: <50, 95>

Use greedy by cost to allocate 1000 calories
Total value of items taken = 413.0
    apple: <50, 95>
    wine: <89, 123>
    cola: <79, 150>
    beer: <90, 154>
    donut: <10, 195>
    pizza: <95, 258>

Use greedy by density to allocate 1000 calories
Total value of items taken = 413.0
    wine: <89, 123>
    beer: <90, 154>
    cola: <79, 150>
    apple: <50, 95>
    pizza: <95, 258>
    donut: <10, 195>

Use greedy by value with additional weight to allocate 1000 calories
Total value of items taken = 285.0
    burger: <100, 354>
    pizza: <95, 258>
    fries: <90, 365>
```

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

In [ ]:

```python
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 [ ]:
```
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 [ ]:
```
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:

- You are in a bookstore having a specific amount of money. The goal is to pick however number of books according to the given constraint (budget) that contains the best ratings/review (1-5).

In [150…
```
# greedy method

class Books(object):
  def __init__(self, n, v, w):
    self.title = n
    self.rating = v
    self.price = w
  def getRating(self):
    return self.rating
  def getCost(self):
    return self.price
  def __str__(self):
    return self.title + ': <' + str(self.rating) + ', ' + str(self.price) + '>'
```

In [151…
```
def buildBasket(title, rating, price):
    basket = []
    for i in range(len(rating)):
```

```
        basket.append(Books(title[i], rating[i], price[i]))
    return basket
```

In [156…
```
def greedy(books, maxCost, keyFunction):
  booksCopy = sorted(books, key = keyFunction, reverse = True)
  result = []
  totalRating, totalCost = 0.0, 0.0
  for i in range(len(booksCopy)):
    if(totalCost+booksCopy[i].getCost()) <= maxCost:
      result.append(booksCopy[i])
      totalCost += booksCopy[i].getCost()
      totalRating += booksCopy[i].getRating()
  return (result, totalRating)
```

In [157…
```
def testGreedy(books, constraint, keyFunction):
    taken, val = greedy(books, constraint, keyFunction)
    print('Rating of books taken =', val)
    for book in taken:
        print('    ', book)
```

In [158…
```
def testGreedys(books, maxCost):
    print('Use greedy by ratings to allocate', maxCost, 'pesos')
    testGreedy(books, maxCost, Books.getRating)
    print('\nUse greedy by cost to allocate', maxCost, 'pesos')
    testGreedy(books, maxCost, lambda x: 1/Books.getCost(x))
```

In [181…
```
title = ['book1', 'book2', 'book3', 'book4', 'book5','book6', 'book7', 'book8']
ratings = [2,5,3,4.5,5,4,2,5]
price = [599,650,600,435,550,499,399,499]
books = buildBasket(title, ratings, price)
testGreedys(books, 2000)
```

```
Use greedy by ratings to allocate 2000 pesos
Rating of books taken = 15.0
    book2: <5, 650>
    book5: <5, 550>
    book8: <5, 499>

Use greedy by cost to allocate 2000 pesos
Rating of books taken = 15.5
    book7: <2, 399>
    book4: <4.5, 435>
    book6: <4, 499>
    book8: <5, 499>
```

In [182…
```
# brute force method

def bruteForce(books, maxCost):
    n = len(books)
    maxBooks = []
    maxRating = 0.0

    for i in range(2**n):
        currentBooks = []
        currentCost, currentRating = 0.0, 0.0

        for j in range(n):
            if (i & (1 << j)) > 0:
                currentBooks.append(books[j])
```

```
                currentCost += books[j].getCost()

                currentRating += books[j].getRating()

        if currentCost <= maxCost and currentRating > maxRating:
            maxBooks = currentBooks
            maxRating = currentRating

    return (maxBooks, maxRating)
```

In [183…

```
def bruteForce(books, maxCost):
    n = len(books)
    maxBooks = []
    maxRating = []

    for i in range(2**n):
        currentBooks = []
        currentCost = 0.0
        currentRating = []

        for j in range(n):
            if (i & (1 << j)) > 0:
                currentBooks.append(books[j])
                currentCost += books[j].getCost()
                currentRating.append(books[j].getRating())

        if currentCost <= maxCost and currentRating > maxRating:
            maxBooks = currentBooks
            maxRating = currentRating

    return (maxBooks, maxRating)
```

In [184…

```
def testBruteForce(books, maxCost):
    taken, val = bruteForce(books, maxCost)
    print('Rating of books taken =', val)

    for book in taken:
        print('    ', book)
```

In [192…

```
title = ['book1', 'book2', 'book3', 'book4', 'book5','book6', 'book7', 'book8']
ratings = [2,3,3,4.5,4.5,4,2,3]
price = [599,650,600,435,550,499,399,499]
books = buildBasket(title, ratings, price)
testBruteForce(books, 1700)
```

```
Rating of books taken = [4.5, 4.5, 4]
    book4: <4.5, 435>
    book5: <4.5, 550>
    book6: <4, 499>
```

Conclusion:

In conclusion, this activity gave me an introduction to knapsack problems, how to implement greedy algorithm as well as brute force algorithm to it. I am not familiar with these concepts so it was quite confusing when understanding the code and how it works. But with the examples given, I already gained some basic knowledge with it. I still need to study this concept further because I still don't understand how some of it works, I just followed the example above and

made it my own but with some different but still similar parameters and constraints. Still, I'm pleased with how it turned out and I got to manipulate the codes and made it do what I initially intended.