Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report 4

COMP 314

(For partial fulfillment of 3rd Year/ 2nd Semester in Computer Engineering)

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submitted by:

Neha Malla

C.E.

Roll No. 27

1. Pseudocodes:

a. Brute-force method (0/1 Knapsack)

```
BruteForceZO(p, w, m):
        if len(p) != len(w) then print "Error msg!"

        n = len(p)
        bit_strings = getStrings(n)
        max_profit = 0
        solution = ''

        for s in bit_strings:
                profit = sum([int(s[i]) * p[i] for i in range(n)])
                weight = sum([int(s[i]) * w[i] for i in range(n)])

                if weight <= m and profit > max_profit:
                        max_profit = profit
                        solution = s

        return solution, max_profit
```

b. Brute-force method (fractional Knapsack)

```
BruteForceF(p, w, m):
        if len(p) != len(w) then print "Error msg!"

        n = len(p)
        bit_strings = getStrings(n)
        max_profit = 0
        solution = ''

        for s in bit_strings:
                profit, weight = 0, 0
                s1 = ''

                for i = 0 to n-1:
                        if weight < m and int(s[i]) == 1:
                                if w[i] <= (m - weight):
                                        profit += p[i]
                                        weight += w[i]
```

```
                    else: // fractional part
                        s1 = s[0:i] + 'f'
                        // f is the ratio of weight taken
                        f = 'f = (' + str(m-weight) + '/' + str(w[i]) + ')'
                        unit = round(p[i]/w[i], 3)
                        profit += (m - weight)*unit
                        weight += m - weight

                if profit > max_profit:
                    max_profit = profit

                    if s1 != '': // else block is entered above & fractional value is taken
                        solution = s1
                    else:
                        solution = s

        return(solution, f, max_profit)
```

c. Greedy method (fractional Knapsack)

```
Greedy(array, maxWeight):
        // array tuple = profit, weight, key
        for item in array:
            item.append(item[0]/item[1], 3) // unit profit is added as tuple

        array.sort() // sorting array on the basis of unit profit in descending order
        profit = 0
        selected = []

        for item in array:
            if item[1] <= maxWeight:
                selected.append(item[0:3])
                maxWeight -= item[1]
                profit += item[0]
                if maxWeight == 0:
                    break
            else: // fractional part
                selected.append([maxWeight * item[3], maxWeight, item[2]])
                profit += maxWeight * item[3]
```

```
                maxWeight = 0
                break

    return selected, profit
```

d. Dynamic programming (0/1 Knapsack)

```
Dynamic(p, w, m):
    if len(p) != len(w) then print "Error msg!"

    n = len(p)
    V = [[0] * (m + 1)] * (n + 1) // 2D array of m+1 columns & n+1 rows

    // bottom up approach
    for i = 0 to n:
        for j = 0 to m:
            if i == 0 or j == 0:
                V[i][j] = 0
            else if w[i - 1] <= j:
                V[i][j] = max(V[i - 1][j], V[i - 1][j - w[i - 1]] + p[i - 1])
            else:
                V[i][j] = V[i - 1][j]

    // finding the solution
    rm, rp = m, V[n][m]
    solution = ''
    for i = n to 1; step = -1:
        if rp <= 0:
            break
        if rp != V[i - 1][rm]: // included item
            solution = '1' + solution
            rp -= p[i - 1]
            rm -= w[i - 1]
        else:
            solution = '0' + solution

    return solution, V[n][m]
```

1. Source Code

➔ Filename: knapsack.py

```
def getStrings(n):
   return [bin(x)[2:].rjust(n, '0') for x in range(2**n)]


# 0/1 Knapsack problem with Brute force
def BruteForceZO(p, w, m):
      assert len(p) == len(w), "Profit and weight do not have the same
number of elements!"

    n = len(p)
    bit_strings = getStrings(n)

    max_profit = 0
    solution = ''

    for s in bit_strings:
        profit = sum([int(s[i]) * p[i] for i in range(n)])
        weight = sum([int(s[i]) * w[i] for i in range(n)])

        if weight <= m and profit > max_profit:
            max_profit = profit
            solution = s

    return solution, max_profit


# Fractional Knapsack problem with Brute force
def BruteForceF(p, w, m):
      assert len(p) == len(w), "Profit and weight do not have the same
number of elements!"

    n = len(p)
    bit_strings = getStrings(n)

    max_profit = 0
    solution = ''
```

```
    for s in bit_strings:
        profit, weight = 0, 0
        s1 = ''
        for i in range(n):
            if weight < m and int(s[i]) == 1:
                if w[i] <= (m - weight):
                    profit += p[i]
                    weight += w[i]
                else:# fractional part
                    s1 = s[0:i] + 'f'
                    # f is the ratio of weight taken
                    f = 'f = (' + str(m - weight) + '/' + str(w[i]) + ')'
                    unit = round(p[i]/w[i], 3)
                    profit += (m - weight)*unit
                    weight += m - weight

        if profit > max_profit:
            max_profit = profit
            if s1 != '':
                # fractional item completes the max weight, so rest of the
item are not considered
                solution = s1.ljust(4, '0')
            else:
                solution = s

    return(solution, f, max_profit)


# Fractional Knapsack problem with Greedy
def Greedy(array, maxWeight):
    # array tuple = profit, weight, key
    for item in array:
          item.append(round(item[0]/item[1], 3))# tuple = profit, weight,
key, unit profit

     array.sort(key = lambda x:x[3], reverse = True)# sorting array on the
basis of unit profit
    profit = 0
    selected = []
```

```
    for item in array:
        if item[1] <= maxWeight:
            selected.append(item[0:3])
            maxWeight -= item[1]
            profit += item[0]
            if maxWeight == 0:
                break
        else:# fractional part
            selected.append([maxWeight * item[3], maxWeight, item[2]])
            profit += maxWeight * item[3]
            maxWeight = 0
            break

    return selected, profit


# 0/1 Knapsack problem with Dynamic
def Dynamic(p, w, m):
      assert len(p) == len(w), "Profit and weight do not have the same
number of elements!"

    n = len(p)
    V = [[0 for j in range(m + 1)] for i in range(n + 1)]

    # bottom up approach
    for i in range(n + 1):
        for j in range(m + 1):
            if i == 0 or j == 0:
                V[i][j] = 0
            elif w[i - 1] <= j:
                 V[i][j] = max(V[i - 1][j], V[i - 1][j - w[i - 1]] + p[i -
1])
            else:
                V[i][j] = V[i - 1][j]

    # finding the solution
    rm, rp = m, V[n][m]
    solution = ''
    for i in range(n, 0, -1):
        if rp <= 0:
```

```
            break
        if rp != V[i - 1][rm]:
            # included item
            solution = '1' + solution

            rp -= p[i - 1]
            rm -= w[i - 1]
        else:
            solution = '0' + solution

    solution = solution.rjust(4, '0')
    return solution, V[n][m]
```

2. Test Cases

➔ Filename: test_knapsack.py

```
import unittest
from knapsack import Greedy, BruteForceZO, BruteForceF, Dynamic


class TestKnapsack(unittest.TestCase):
    def setUp(self):
        self.p = [5, 6, 7, 2]# profit
        self.w = [4, 2, 3, 1]# weight
        self.m = 8# max weight

    def test_Greedy(self):
        # array tuple = profit, weight, key
        array = [[5, 4, '1'], [6, 2, '2'],[7, 3, '3'], [2, 1, '4']]
          #  solution tuple have the fractional profit and weight that are
chosen
        solution = [[6, 2, '2'], [7, 3, '3'], [2, 1, '4'], [2.5, 2, '1']]
        profit = 17.5
        self.assertEqual(Greedy(array, 8), (solution, profit))

    def test_BruteForceZO(self):
           self.assertEqual(BruteForceZO(self.p, self.w, self.m), ('0111',
15))
```

```
    def test_BruteForceF(self):
         self.assertEqual(BruteForceF(self.p, self.w, self.m), ('11f0', 'f
= (2/3)', 15.666))

    def test_Dynamic(self):
        self.assertEqual(Dynamic(self.p, self.w, self.m), ('0111', 15))


if __name__ == "__main__":
    unittest.main()
```

3. Output of Test Cases

```
C:\Users\neha\Algorhythm\lab4>python test_knapsack.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK

C:\Users\neha\Algorhythm\lab4>
```

4. Conclusion

Here, Knapsack problem is solved using different strategies such as Brute-force method (for fractional and 0/1 Knapsack), Greedy method (for fractional Knapsack) and Dynamic programming (for 0/1 Knapsack).

The input for the Greedy method is different from others as in this method, we sort the array for highest unit profit, and the order of the input array is hard to keep track of if there is no key for each item. Thus a key is introduced to each item in the array, so as to know the solution in the output without confusion.

All test cases ran successfully on the four of the methods, as seen on the screenshot provided above.