Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report III

[Code No: COMP 314]

Submitted by:

Prashant Manandhar

Roll No:30

Group: Computer Engineering

Level: III year/II sem

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date:20/06/2024

Task I - Pseudocode

Knapsack 01 by brute force

function knapsack_01_brute_force(weights, values, capacity):

1. n = len(weights)
2. max_value = 0
3. all_accepted_weight_sequence = []
4. all_accepted_binary_sequence = []
5. sequences_values = []
6. for x from 0 to (2^n) - 1:
    a. value = 0
    b. weight = 0
    c. current_binary_sequence = []
    d. current_weight_sequence = []
    e. temp = x
    f. for j from 0 to n - 1:
        i. if temp % 2 != 0 then
            1. value = value + values[j]
            2. weight = weight + weights[j]
        ii. end if
        iii. current_binary_sequence.append(temp % 2)
        iv. temp = floor(temp / 2)
    g. end for
    h. if weight <= capacity then
        i. temp = x
        ii. for j from 0 to n - 1:
            1. if temp % 2 != 0 then
                a. current_weight_sequence.append(weights[j])
            2. end if
            3. temp = floor(temp / 2)
        iii. end for

iv. sequences_values.append(value)
v. all_accepted_weight_sequence.append(current_weight_sequence)
vi. all_accepted_binary_sequence.append(current_binary_sequence)

i. end if

7. end for
8. max_value = max(sequences_values)
9. max_value_index = index_of(max(sequences_values))
10. return {"max_value": max_value,"weight_sequence": all_accepted_weight_sequence[max_value_index], "binary_sequence": all_accepted_binary_sequence[max_value_index] }

end function

Knapsack 01 by dyanmic

function knapsack_01_dynamic(weights, values, capacity):

1. n = length(weights)
2. dp = create_table(n + 1, capacity + 1, 0)
3. for i from 1 to n:
    a. for w from 0 to capacity:
        i. if weights[i - 1] <= w then
            1. dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
        ii. else
            1. dp[i][w] = dp[i - 1][w]
        iii. end if
    b. end for
4. end for
5. max_value = dp[n][capacity]
6. weight_sequence = []
7. inary_sequence = []

8. w = capacity
9. for i from n to 1 by -1:
    a. if dp[i][w] != dp[i - 1][w] then
        i. weight_sequence.append(weights[i - 1])
        ii. binary_sequence.append(1)
        iii. w -= weights[i - 1]
    b. else
        i. binary_sequence.append(0)
    c. end if
10. end for
11. binary_sequence.reverse()
12. weight_sequence.reverse()
13. return {"max_value": max_value, "weight_sequence": weight_sequence,"binary_sequence": binary_sequence}

end function

Knapsack Fractional brute force

function knapsack_fractional_brute_force(weights, values, capacity):

1. n = length(weights)
2. max_value = 0
3. all_accepted_weight_sequence = []
4. all_accepted_binary_sequence = []
5. sequences_values = []
6. for x from 0 to (2^n) - 1:
    a. value = 0
    b. weight = 0
    c. current_binary_sequence = []
    d. current_weight_sequence = []
    e. temp = x
    f. for j from 0 to n - 1:
        i. if temp % 2 != 0 then

1. remaining_capacity = capacity - weight
2. if weights[j] <= remaining_capacity then
    a. value = value + values[j]
    b. weight = weight + weights[j]
    c. current_binary_sequence[j] = 1
3. else
    a. ratio = remaining_capacity / weights[j]
    b. value = value + values[j] * ratio
    c. weight = weight + weights[j] * ratio
    d. current_binary_sequence[j] = ratio
4. end if

ii. else
1. current_binary_sequence[j] = 0

iii. end if

iv. temp = floor(temp / 2)

g. end for

h. if weight <= capacity then

i. temp = x

ii. for j from 0 to n - 1:
1. if temp % 2 != 0 then
    a. current_weight_sequence.append(weights[j])
2. end if

iii. temp = floor(temp / 2)

i. end if

j. if weight < capacity then

i. for j from 0 to n - 1:
1. if current_binary_sequence[j] == 0 and weight < capacity then
    a. remaining_capacity = capacity - weight
    b. if weights[j] <= remaining_capacity then
        i. ratio = remaining_capacity / weights[j]

ii. current_binary_sequence[j] = ratio
iii. current_weight_sequence.append(weights[j])
iv. value = value + values[j] * ratio
v. weight = weight + weights[j] * ratio

c. else
i. can_add_weight = remaining_capacity
ii. ratio = can_add_weight / weights[j]
iii. current_binary_sequence[j] = ratio
iv. value = value + values[j] * ratio
v. weight = weight + weights[j] * ratio

d. end if

e. if weight >= capacity then
i. break

f. end if

2. end if

ii. end for

k. end if

l. If weight <= capacity
i. sequences_values.append(value)
ii. all_accepted_weight_sequence.append(current_weight_sequence)
iii. all_accepted_binary_sequence.append(current_binary_sequence)

m. end if

7. end for

8. if sequences_values is not empty then
a. max_value = max(sequences_values)
b. max_value_index = index_of(max_value)

c. return {"max_value": max_value,"weight_sequence": all_accepted_weight_sequence[max_value_index],"binary_sequence":all_accepted_binary_sequence[max_value_index] }

9. else
    a. return {"max_value": 0, "weight_sequence": [], "binary_sequence": []}
10. end if

end function

KnapSnack fraction by greedy

function knapsack_fractional_greedy(weights, values, capacity):

1. n = length(weights)
2. items = []
3. for i from 0 to n - 1:
    a. items.append((weights[i], values[i], values[i] / weights[i], i))
4. end for
5. items.sort(key=lambda x: x[2], reverse=True)
6. total_value = 0
7. total_weight = 0
8. chosen_weights = []
9. binary_sequence = array of size n filled with 0s
10. for weight, value, ratio, index in items:
    a. if total_weight + weight <= capacity then
        i. total_weight = total_weight + weight
        ii. total_value = total_value + value
        iii. chosen_weights.append(weight)
        iv. binary_sequence[index] = 1
    b. else
        i. remaining_capacity = capacity - total_weight

ii. if remaining_capacity > 0 then

1. fraction = remaining_capacity / weight
2. total_value = total_value + value * fraction
3. total_weight = total_weight + remaining_capacity
4. binary_sequence[index] = fraction

iii. end if

iv. break

c. end if

11. end for
12. return { "max_value": total_value,"weight_sequence": chosen_weights,"binary_sequence": binary_sequence }

end function

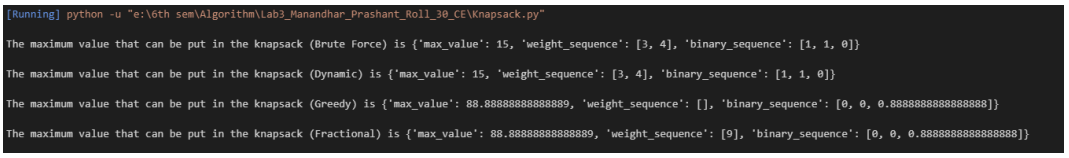

Fig: Output from all knapsack implemented

Task II - Some test cases to test the Knapsack Algorithm

For the testing of the four algorithms, there is two functions that test for a random number, one for zero capacity, both empty weight and value, single items, single items with a capacity exceeding the capacity, and all items' total weight equal to the weight of capacity.

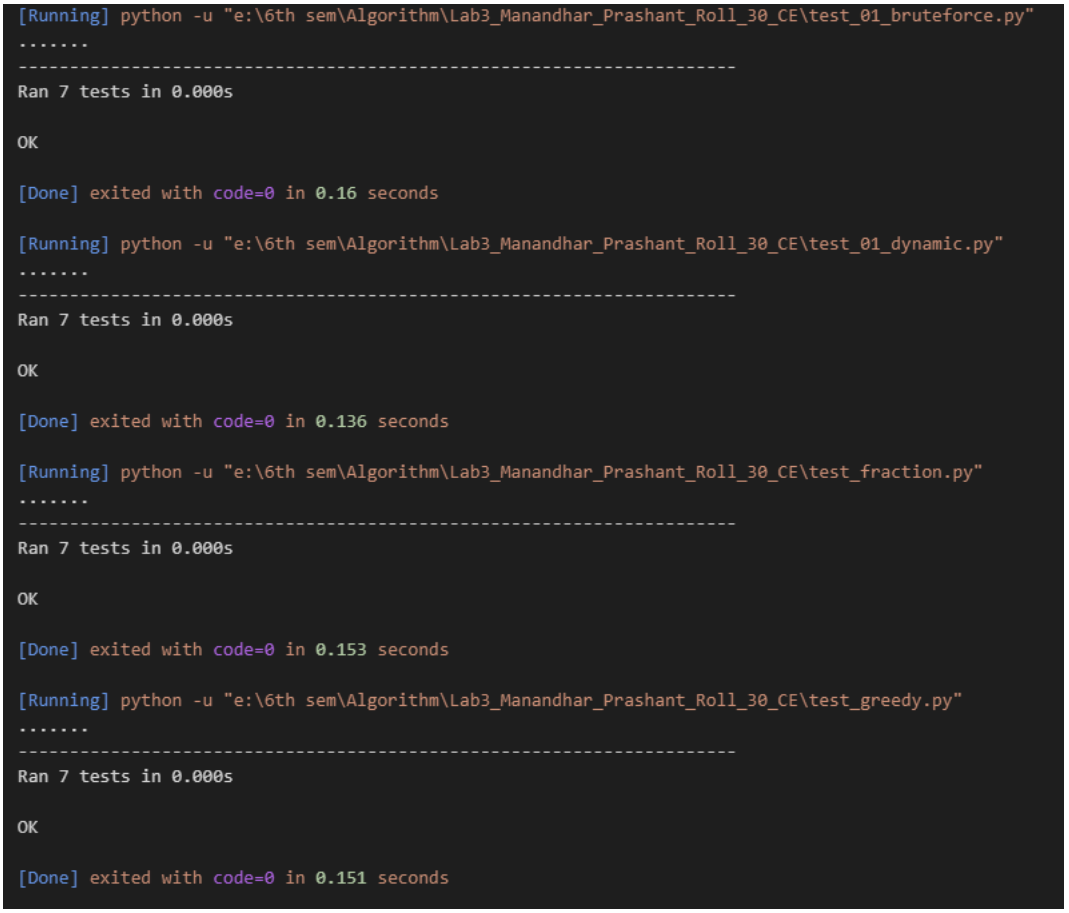

Fig: Test result from the four algorithm

GitHub Link:

The code of the Lab 3 can be found in the following GitHub Link:

https://github.com/Eemayas/Algorithm-and-Complexity-CE2020-Roll-30-COMP-314-Labs/tree/main/Lab3_Manandhar_Prashant_Roll_30_CE