Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre



Lab 4

Algorithms and Complexity (COMP 314)

Submitted By:

Mani Dumaru

Roll no.: 15

CE 3rd Year/ 2nd Semester

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 3rd June 2023

Solving Knapsack problem using different algorithm design strategies.

1. Brute Force Method:
2. *0/1 Knapsack using Brute Force strategy*

Pseudocode:

01knapsackBrute (length, data, size, init):

if size <= 0: return;

if (weight of current data) <= size:

profit\_including \_data = data.profit + 01knapsackBrute(length, data, size -data.weight, init+1)

profit\_excluding\_data = 01knapsackBrute(length, data, size, init+1)

return max of (profit\_including\_data, profit\_excluding\_data)

else:

profit\_excluding\_data = 01knapsackBrute(length, data, size, init+1)

return profit\_excluding\_data

1. *Fractional Knapsack using Brute Force*

Pseudocode:

FractionalBrute (length, data, size, init):

if size <= 0: return;

if (weight of current data) <= size:

profit\_including \_data = data.profit + FractinalBrute(length, data, size -data.weight, init+1)

profit\_excluding\_data = FractionalBrute(length, data, size, init+1)

else:

profit\_including = data.profit \* (size/data.weight)

excluding\_profit = FractionalBrute(length, data, size, init+1)

return max(profit\_including, profit, excluding)

1. Greedy Approach:

Pseudocode:

greedyKnapsack (data, size):

profit = 0

for every i in data:

i[“profit/weight”] = i.profit / i.weight

sort data in descending order of profit/weight

for every i in data:

if size <= 0:

break

if i.weight <= size:

profit = profit + i.profit

size = size – i.weight

else:

profit = profit + i.profit / (size/i.weight)

size = 0

return profit