# GROUP A: ASSIGNMENT No. 4

Title: O-1 Knapsack	Problem
---------------------	---------

Objective: To solve 0.1 Knapsack Problem using dynamic programming or branch & bound strategy

#### Problem Statement:

Write a program to solve 0.1 Knaprack problem using dynamic programming or branch & bound stockedy

# Software & Hardwerre Requirement:

- 1. Desktop/Laptop
- 2. Any Operating System
- 3. Python
- 4. IDE or Code Editor

# Theory:

#### Knapsack Problem:

You are given -

- 1) A knapsack with limited weight capacity 2) Few items each having weight & value.

Hems should be placed into the knapsack such that -

- 17 The value of profit obtained by putting the items into the knapsack is maximum.
- 2) And the weight limit af knaprack does not exceed

### 0/1 knapsack Problem:

In 011 Knapsack Problem

) As the name suggests items one indivisible here.

- R> We can not take fraction of any item.

  3> We have to either take an item completely or
- leave it completely.

  4> It is solved using dynamic programming approach

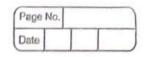
Dynamic Programming: Dynamic Programming is mainly an optimization over plain recursion. Inherever we see a recursive solution that har repeated calls for some inputs, we Can optimize it using Dynamic Programming. The idea is to simply store the result of subproblems, so that we do not have to se-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

A thief is probbing a store & can carry a maximal weight of winto his knaprack. There are n items & weight of it item is will the provist of selecting this item is pi. What item thief should take?

Dynamic Programmia Approach:

Let ; be the highest numbered item in an optimal solution 5 For W dollars. Then 5'= 5- {: 3 is an optimal solution For W-W: dollars & the value at the solution Sir V: plus the value of the subproblem.

Me can express this fact in the following formula. define (i,w) to be the solution for items 1,2,..., if the maximum weight w.



The algorithm takes the following inputs

The maximum weight W

The number of items h

The two sequences  $V = \langle V_1, V_2, ..., V_n \rangle = \langle w_1, w_2, ..., w_n \rangle$ 

Algorithm:

Dynamic-0-1- knapsack (v,w,n,W)

For w = 0 to W do

([0,w] = 0

for i=1 to n do

((i,0] =0

For w=1 to W do

if wis w then

if Vi+ ((i-1, w-wi) then

((i,w) = Vi+((i-1, w-wi)

else

c Ci, w] = ( (i-1, w]

else

((i,w) = ( (i-1,w)

The set at items to take can be deduced from the table, starting at ( [n,w] & tracing backwards where the optime! values came from.

If C(i,w) = C(i+,w), then item i is not part at the solution, & we continue tracing with C(i-1,w). Otherwise, item

i is part af the solution, & we continue tracing with C(i-1,w-w)

Page No.	
Date	

Analysis:

This algorithm takes  $\Theta(n_1w)$  times as table c has (n+1) (w+1) entries, where each entry requires  $\Theta(1)$  time to compute.

Test Cases:

Input:

Hem A B C D
Proprit 24 18 18 10
Weight 24 10 10 7

Expected Output = 36

Actual Output = 18+18 = 36

Result: Pass

#### Conclusion:

Successfully implemented 0-1 knapsack problem using Dynamic Programming.

Joen Con

#### Code:

#### Output:

11