**Experiment 2.3**

**Aim:** *Develop a program and analyze complexity to implement 0-1 Knapsack using Dynamic*

**Objectives:** *To implement 0-1 Knapsack using Dynamic Programming*

**Input/Apparatus Used:** *VS CODE*

# Procedure/Algorithm:

# *Dynamic Programming Approach: The solution is based on dynamic programming to solve the 0-1 Knapsack problem.*

# *DP Table: Create a DP table with columns representing all possible weights from 1 to the maximum capacity 'W' and rows representing the weights that can be kept.*

# *DP State: The state DP[i][j] represents the maximum value for a weight of 'j' considering all values from '1' to 'ith'.*

# *Two Possibilities: When considering weight 'wi' (weight in the 'ith' row), two possibilities exist:*

# *Fill 'wi' in the Column: If the weight 'wi' can be accommodated in the current column (if the weight is less than or equal to 'j'), the value in DP[i][j] can be updated as wi + DP[i-1][j-wi].*

# *Do Not Fill 'wi' in the Column: If 'wi' cannot be added to the current column, then the value remains the same as DP[i-1][j].*

# *Maximum Value: Take the maximum of these two possibilities to update the current state DP[i][j].*

# *Visualization Example: A visualization example is provided with weight elements, weight values, and capacity, where the DP table is filled step by step, considering these principles.*

# Code:

# *#include <iostream>*

# *using namespace std;*

# *int max(int a, int b) {*

# *return (a > b) ? a : b;*

# *}*

# *int knapSack(int W, int wt[], int val[], int n) {*

# *int i, w;*

# *int K[n + 1][W + 1];*

# *for (i = 0; i <= n; i++) {*

# *for (w = 0; w <= W; w++) {*

# *if (i == 0 || w == 0)*

# *K[i][w] = 0;*

# *else if (wt[i - 1] <= w)*

# *K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);*

# *else*

# *K[i][w] = K[i - 1][w];*

# *}*

# *}*

# *return K[n][W];*

# *}*

# *int main() {*

# *int val[] = {60, 100, 120};*

# *int wt[] = {10, 20, 30};*

# *int W = 50;*

# *int n = sizeof(val) / sizeof(val[0]);*

# *cout << knapSack(W, wt, val, n) << endl;*

# *return 0;*

# *}*

# Observations/Outcome :

# 

# Time Complexity:

* *Time Complexity: O(nW) where n is the number of items and W is the capacity of knapsack.,*