AOA EXPERIMENT 8

Name: Mohammed Arhaan Ali
Div: S11
Roll No: 02

Aim: To study and implement 0/1 Knapsack problem using Dynamic Programming Approach

Theory:
The Knapsack problem is an example of the combinational optimization problem. This problem is also commonly known as the “Rucksack Problem“. The name of the problem is defined from the maximization problem as mentioned below:

Given a bag with maximum weight capacity of W and a set of items, each having a weight and a value associated with it. Decide the number of each item to take in a collection such that the total weight is less than the capacity and the total value is maximized.

0/1 Knapsack Problem:

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.
Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

Examples:
Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
Output: 3
Explanation: There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible profit is 3. Note that we cannot put both the items with weight 4 and 1 together as the capacity of the bag is 4.
Input: N = 3, W = 3, profit[] = {1, 2, 3}, weight[] = {4, 5, 6}
Output: 0

Recursion Approach for 0/1 Knapsack Problem:

To solve the problem follow the below idea:
A simple solution is to consider all subsets of items and calculate the total weight and profit of all subsets. Consider the only subsets whose total weight is smaller than W. From all such subsets, pick the subset with maximum profit.
Optimal Substructure: To consider all subsets of items, there can be two cases for every item.

- Case 1: The item is included in the optimal subset.
- Case 2: The item is not included in the optimal set.

Follow the below steps to solve the problem:
The maximum value obtained from ‘N’ items is the max of the following two values.

- Case 1 (include the Nth item): Value of the Nth item plus maximum value obtained by remaining N-1 items and remaining weight i.e. (W-weight of the Nth item).
- Case 2 (exclude the Nth item): Maximum value obtained by N-1 items and W weight.
- If the weight of the ‘Nth‘ item is greater than ‘W’, then the Nth item cannot be included and Case 2 is the only possibility.

Code:

```
#include <stdio.h>

#define MAX_ITEMS 100
#define MAX_WEIGHT 100

// Function to calculate maximum of two integers
int max(int a, int b)
{
   return (a > b) ? a : b;
}

// Function to solve 0/1 knapsack problem
int knapsack(int weights[], int values[], int n, int capacity)
{
   int i, w;
   int dp[MAX_ITEMS + 1][MAX_WEIGHT + 1];

   // Build dp[][] in bottom-up manner
   for (i = 0; i <= n; i++)
   {
      for (w = 0; w <= capacity; w++)
      {
```

```
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i - 1] <= w)
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    return dp[n][capacity];
}

int main()
{
    printf("---------KNAPSACK--------\n");

    int n;
    printf("Enter the number of items: ");
    scanf("%d", &n);

    int weights[MAX_ITEMS], values[MAX_ITEMS];
    printf("Enter the weights:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &weights[i]);
    }

    printf("Enter the Profits:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &values[i]);
    }

    int capacity = 50; // Assuming a fixed capacity for demonstration purposes
    int max_value = knapsack(weights, values, n, capacity);
    printf("Maximum value that can be obtained: %d\n", max_value);

    return 0;
}
```

Output:

```
/tmp/jh1qcCnSqU.o
---------KNAPSACK--------
Enter the number of items: 5
Enter the weights:
1 2 3 4 5
Enter the Profits:
2 3 4 6 7
Maximum value that can be obtained: 22
```