**SE-Comps\_A Batch-C**

**Mark Lopes**

**9913**

**Knapsack\_DP:-**

#include <stdio.h>

int max(int a, int b) //calculate max of two integers

{

    return (a > b) ? a : b;

}

int knapsack(int W, int wt[], int val[], int n)

{

    int mat[n + 1][W + 1];

    // Initialize matrix with 0

    for (int i = 0; i <= n; i++)

    {

        for (int j = 0; j <= W; j++)

        {

            if (i == 0 || j == 0)

                mat[i][j] = 0;

            else

            {

                int maxValWithoutCurr = mat[i - 1][j];

                int maxValWithCurr = 0;

                if (j >= wt[i - 1])

                {

                    maxValWithCurr = val[i - 1];

                    int remainingCapacity = j - wt[i - 1];

                    maxValWithCurr += mat[i - 1][remainingCapacity];

                }

                mat[i][j] = max(maxValWithoutCurr, maxValWithCurr);

            }

        }

    }

    return mat[n][W];

}

int main()

{

    int W = 10;                   // Max weight

    int n = 4;                    // Number of items

    int val[] = {10, 40, 30, 50}; // Values of items

    int wt[] = {5, 4, 6, 3};      // Weights of items

    int maxValue = knapsack(W, wt, val, n);

    printf("Maximum value that can be obtained: %d\n", maxValue);

    return 0;

}

****

**Coin exchange DP:-**

#include <stdio.h>

// Define the available coin denominations and the target sum

int coins[] = {1, 2, 3};

int targetSum = 4;

int numCoins = 3;

// Function to initialize the dynamic programming table

void initializeTable(int table[][5]) {

    // Initialize the first row to 0 (base case)

    for (int i = 0; i <= targetSum; i++) {

        table[0][i] = 0;

    }

    // Initialize the first column to 1 (base case)

    for (int i = 0; i <= numCoins; i++) {

        table[i][0] = 1;

    }

}

// Function to calculate the number of ways to reach the target sum

int countWays(int table[][5]) {

    for (int coinIndex = 1; coinIndex <= numCoins; coinIndex++) {

        for (int currentSum = 1; currentSum <= targetSum; currentSum++) {

            if (coins[coinIndex - 1] > currentSum) {

                // If the coin value is greater than the current sum, exclude it

                table[coinIndex][currentSum] = table[coinIndex - 1][currentSum];

            } else {

                // otherwise include the current coin in the count

                table[coinIndex][currentSum] = table[coinIndex - 1][currentSum] +

                                                table[coinIndex][currentSum - coins[coinIndex - 1]];

            }

        }

    }

    // Return the final count of ways to reach the target sum

    return table[numCoins][targetSum];

}

int main() {

    // Create a 2D array for the dynamic programming table

    int dpTable[numCoins + 1][5];

    // Initialize the dynamic programming table

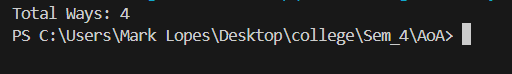
    initializeTable(dpTable);

    // Calculate and print the total number of ways to reach the target sum

    printf("Total Ways: %d\n", countWays(dpTable));

    return 0;

}

****