**CHAROTAR UNIVERSITY OF SCIENCE &**

**TECHNOLOGY**

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

**Computer Science & Engineering**

**NAME: PARTH NITESHKUMAR PATEL**

**ID: 19DCS098**

**SUBJECT: DESIGN AND ANALYSIS OF**

**ALGORITHM**

**CODE: CS 351**

**DYNAMIC PROGRAMMING**

**PRACTICAL-5.1**

**AIM:**

Implement a program which has BNMCOEF() function that takes two parameters n and k and returns the value of Binomial Coefficient C(n, k). Compare the dynamic programming implementation with recursive implementation of BNMCOEF(). (In output, entire table should be displayed.)

|  |  |  |
| --- | --- | --- |
| **Test Case** | **n** | **k** |
| 1 | 5 | 2 |
| 2 | 11 | 6 |
| 3 | 12 | 5 |

**PROGRAM CODE:**

#include<iostream>

using namespace std;

int BNFCOEF(int n, int k)

{

    if (k == 0 || k == n){

        return 1;

    }

    return BNFCOEF(n - 1, k - 1) +

                BNFCOEF(n - 1, k);

}

int main()

{

    int n,k;

    cout<<"ENTER THE VALUE OF n AND k : ";

    cin>>n>>k;

    cout<<"BINOMIAL COEFFIECIENT : " <<BNFCOEF(n,k)<<endl;

    cout<<endl;

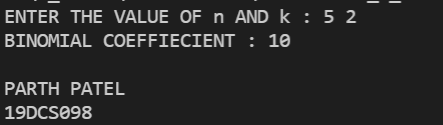
    cout<<"PARTH PATEL\n19DCS098"<<endl;

    return 0;

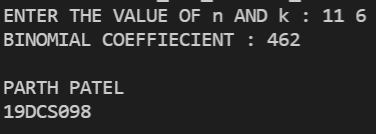
}

**OUTPUT:**

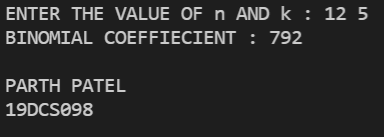
**TEST CASE-1:**



**TEST CASE-2:**



**TEST CASE-3:**



**PRACTICAL-5.2**

**AIM:**

Implement the program 4.2 using Dynamic Programing. Compare Greedy and Dynamic approach

**PROGRAM CODE:**

#include <iostream>

using namespace std;

int max(int x, int y)

{

    if (x > y)

        return x;

    else

        return y;

}

int knapsackSolution(int bagCapacity, int weight[], int profit[], int number)

{

    int matrix[number + 1][bagCapacity + 1];

    for (int i = 0; i < number + 1; i++)

        for (int j = 0; j < bagCapacity + 1; j++)

        {

            if (i == 0 || j == 0)

                matrix[i][j] = 0;

            else if (j >= weight[i - 1])

                matrix[i][j] = max(matrix[i - 1][j], profit[i - 1] + matrix[i - 1][j - weight[i - 1]]);

            else

                matrix[i][j] = matrix[i - 1][j];

        }

    return matrix[number][bagCapacity];

}

int main()

{

    int number, bagCapacity;

    cout << "\nENTER THE SIZE OF ARRAY : ";

    cin >> number;

    int weight[number], profit[number];

    cout << "\nENTER THE WEIGHTS :";

    for (int i = 0; i < number; i++)

        cin >> weight[i];

    cout << "ENTER THE PROFITS :";

    for (int i = 0; i < number; i++)

        cin >> profit[i];

    cout << "ENTER THE CAPACITY OF BAG : ";

    cin >> bagCapacity;

    cout << "\nMAXIMUM POOSIBLE PROFIT: " << knapsackSolution(bagCapacity, weight, profit, number) << endl;

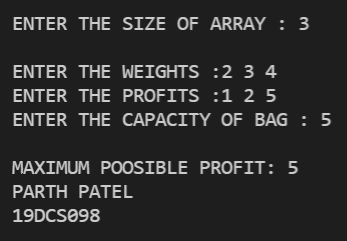
    cout << "PARTH PATEL\n19DCS098" << endl;

    return 0;

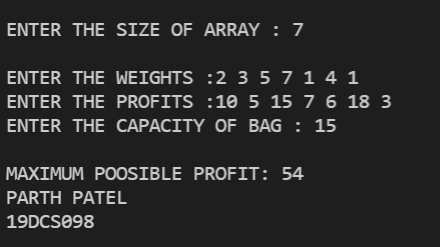
}

**OUTPUT:**

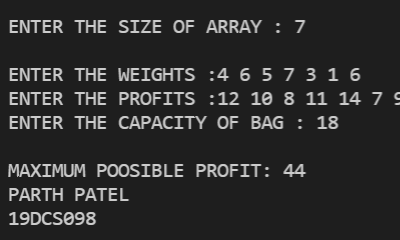
**Test Case-1:**



**Test Case-2:**



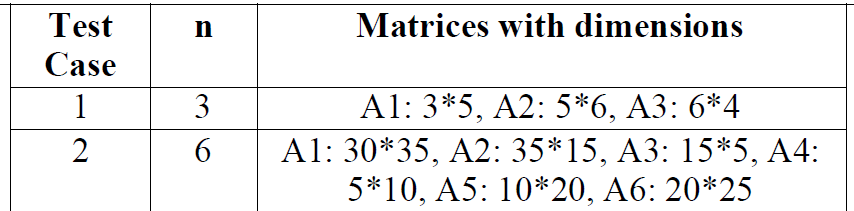
**Test Case-3:**



**PRACTICAL-5.3**

**AIM:**

Given a chain < A1, A2,…,An> of n matrices, where for i=1,2,…,n matrix Ai with dimensions. Implement the program to fully parenthesize the product A1,A2,…,An in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar multiplications for all possible combinations of matrices



**PROGRAM CODE:**

#include <bits/stdc++.h>

using namespace std;

int MatrixMultiplication(int p[], int n)

{

    int m[n][n];

    int i, j, k, L, q;

    for (i = 1; i < n; i++)

        m[i][i] = 0;

    for (L = 2; L < n; L++)

    {

        for (i = 1; i < n - L + 1; i++)

        {

            j = i + L - 1;

            m[i][j] = INT\_MAX;

            for (k = i; k <= j - 1; k++)

            {

                q = m[i][k] + m[k + 1][j] +

                    p[i - 1] \* p[k] \* p[j];

                if (q < m[i][j])

                    m[i][j] = q;

            }

        }

    }

    return m[1][n - 1];

}

int main()

{

    int n;

    cout << "ENTER THE TOTAL DIMENSIONAL VALUE : ";

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cout << "ENTER THE VALUE OF P : " << i << " : ";

        cin >> arr[i];

    }

    int length = sizeof(arr) / sizeof(arr[0]);

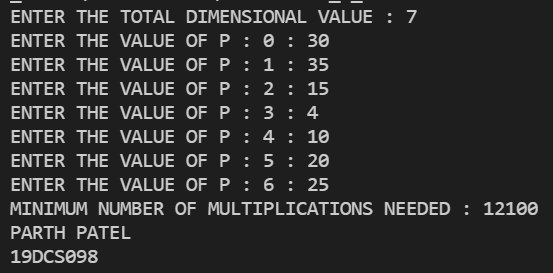
    cout << "MINIMUM NUMBER OF MULTIPLICATIONS NEEDED : " << MatrixMultiplication(arr, length) << endl;

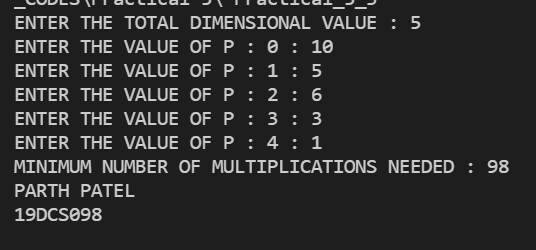
    cout << "PARTH PATEL\n19DCS098" << endl;

    return 0;

}

**OUTPUT:**

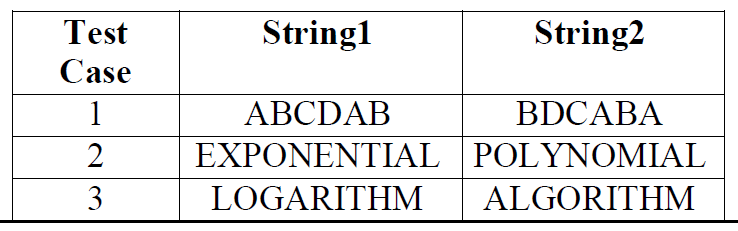




**PRACTICAL-5.4**

**AIM:**

Implement a program to print the longest common subsequence for the following strings:



**PROGRAM CODE:**

#include <iostream>

#include <string.h>

using namespace std;

int maximum(int a, int b);

int longestCommonSubsequence(char \*X, char \*Y, int m, int n)

{

    if (m == 0 || n == 0)

        return 0;

    if (X[m - 1] == Y[n - 1])

        return 1 + longestCommonSubsequence(X, Y, m - 1, n - 1);

    else

        return maximum(longestCommonSubsequence(X, Y, m, n - 1), longestCommonSubsequence(X, Y, m - 1, n));

}

int maximum(int a, int b)

{

    return (a > b) ? a : b;

}

int main()

{

    char X[100], Y[100];

    cout << "ENTER THE SEQUENCE OF STRING-1 : ";

    cin >> X;

    cout << "ENTER THE SEQUENCE OF STEING-2 : ";

    cin >> Y;

    int m = strlen(X);

    int n = strlen(Y);

    cout << "Length of Longest Common Subsequence is : " << longestCommonSubsequence(X, Y, m, n);

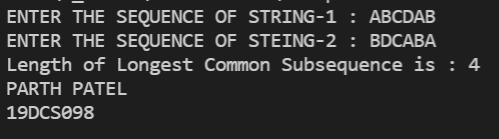
    cout<<"\nPARTH PATEL\n19DCS098"<<endl;

    return 0;

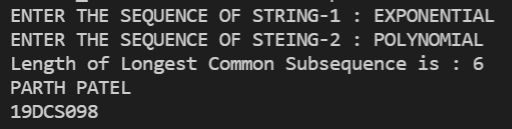
}

**OUTPUT:**

**Test Case-1:**



**Test Case-2:**



**Test Case-3:**

