**PRACTICAL-5**

**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 <bits/stdc++.h>

using namespace std;

int binomialCoeff(int n, int k)

{

if (k == 0 || k == n)

return 1;

return binomialCoeff(n - 1, k - 1) +

binomialCoeff(n - 1, k);

}

int main()

{

int n, k;

cout<<”Enter value of n\n”;

cin>>n;

cout<<”Enter value of k\n”;

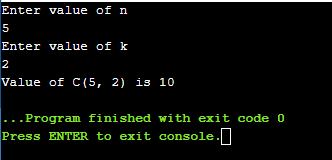
cin>>k;

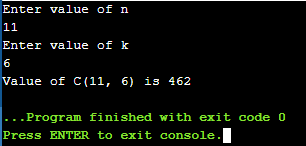
cout << "Value of C("<<n<<", "<<k<<") is " << binomialCoeff(n, k);

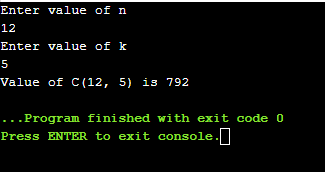
return 0;

}

**OUTPUT:**







**CONCLUSION:** I have learnt about Binomial Coefficient using recursive method.

**5.2**

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

**PROGRAM:**

#include<iostream>

using namespace std;

int maximum(int x,int y)

{

if(x>y)

return x;

else

return y;

}

int knapsack(int bag\_capacity,int weight[],int profit[],int number)

{

int matrix[number+1][bag\_capacity+1];

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

for(int j=0;j<bag\_capacity+1;j++)

{

if(i==0 || j==0)

matrix[i][j]=0;

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

matrix[i][j]=maximum(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][bag\_capacity];

}

int main()

{

int number,bag\_capacity;

cout<<".....BINARY KNAPSACK PROBLEM.....";

cout<<"\nEnter the size of arrays : ";

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 bag capacity : ";

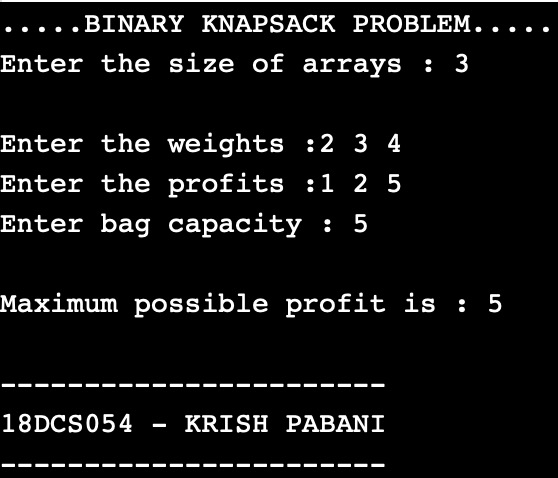
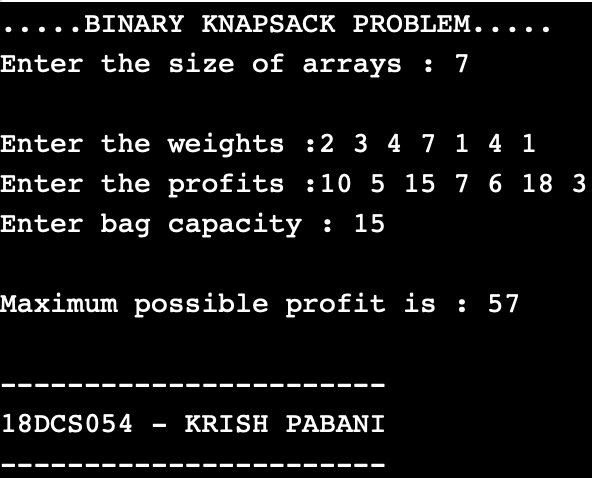
cin>>bag\_capacity;

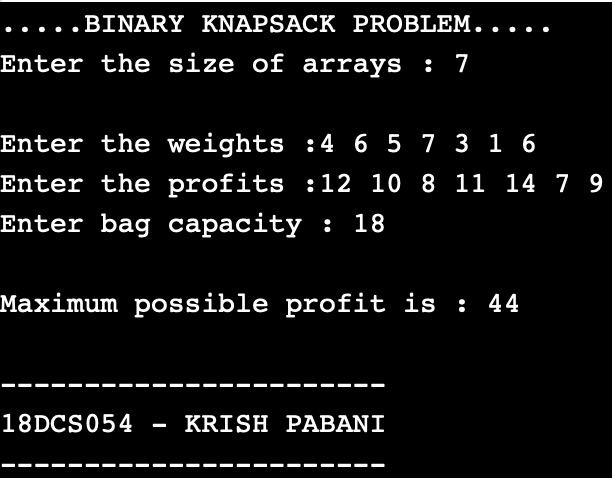
cout<<"\nMaximum possible profit is: " << knapsack(bag\_capacity,weight,profit,number) <<endl;

return 0;

}

**OUTPUT:**

** **

****

**CONCLUSION:**

**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.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **n** | **Matrices with dimensions** |
| 1 | 3 | A1:3\*5,A2:5\*6,A3:6\*4 |
| 2 | 6 | A1: 30\*35, A2: 35\*15, A3: 15\*5, A4: 5\*10, A5: 10\*20, A6: 20\*25 |

**PROGRAM:**

#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;

// L is chain length.

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 = cost/scalar multiplications

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<<"...MATRIX CHAIN MULTIPLICATION...\n\n";

cout<<"Enter total number of dimension values : ";

cin>>n;

int arr[n];

for(int i=0;i<n;i++)

{

cout<<"Enter P"<<i<<" : ";

cin>>arr[i];

}

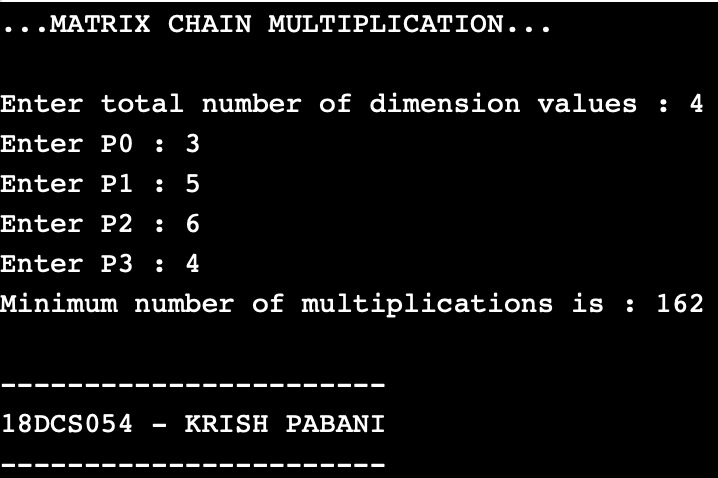
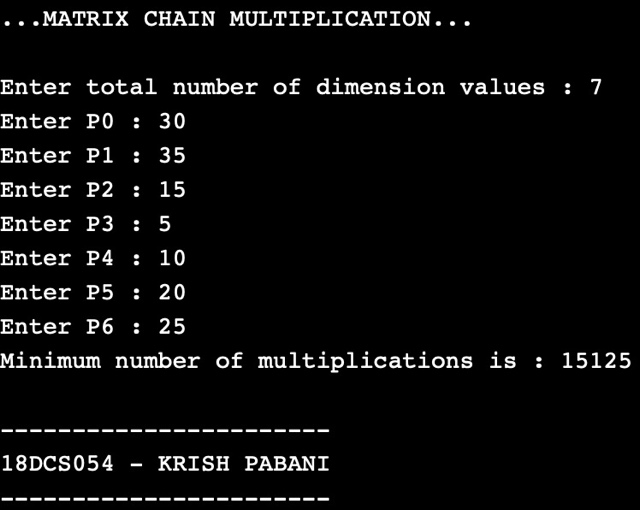
int length=sizeof(arr)/sizeof(arr[0]);

cout << "Minimum number of multiplications is : "<<MatrixMultiplication(arr,length)<<endl;

return 0;

}

**OUTPUT:**

** **

**CONCLUSION:**

**5.4**

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

Check the program for following test cases:

|  |  |  |
| --- | --- | --- |
| **Test Case** | **String1** | **String2** |
| 1 | ABCDAB | BDCABA |
| 2 | EXPONENTIAL | POLYNOMIAL |
| 3 | LOGARITHM | ALGORITHM |

**PROGRAM:**

#include<iostream>

#include<string.h>

using namespace std;

int max(int a, int b);

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

{ if (m == 0 || n == 0)

return 0;

if (X[m-1] == Y[n-1])

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

else

return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));} int max(int a, int b)

{

return (a > b)? a : b;

}

int main()

{ cout<<"...LONGEST COMMON SUBSEQUENCE..."<<endl;

char X[100],Y[100];

cout<<"Enter 1st string sequence : ";cin>>X;

cout<<"Enter 2nd string sequence : ";cin>>Y;

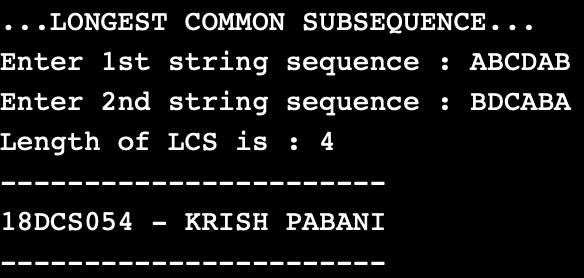
int m = strlen(X);int n = strlen(Y);

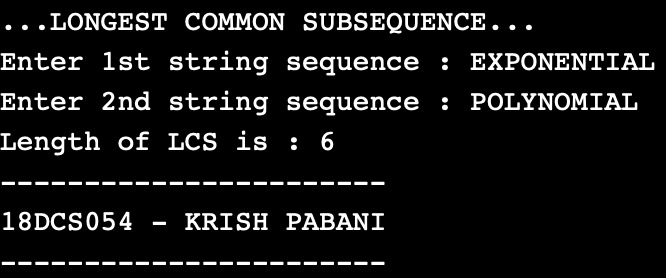
cout<<"Length of LCS is : "<<lcs(X,Y,m,n);

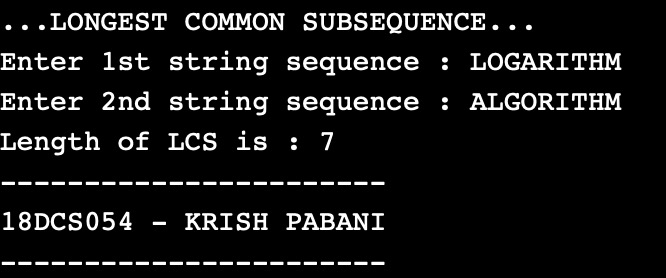
return 0;

}

**OUTPUT:**

****

****

****

**CONCLUSION:**