**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem V

**Course: Design and Analysis of Algorithms**

**List of Experiments**

w.e.f. 1st Jul 2020

**Faculty:** Prof. Abhay Kolhe

LAB Manual

PART B

|  |  |
| --- | --- |
| Roll No. B032 | Name: Naman Garg |
| Class : B Tech CS B | Batch : B2 |
| Date of Experiment: 15-09-2020 | Date of Submission: 15-09-2020 |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

# Name: NAMAN GARG

# Roll No: B032

# AIM:Implementation of Dynamic Programming Technique – Matrix Chain Multiplication.

import numpy as np

from math import \*

#difining the divide function

def divide(start, end):

    global parans

    if(start == end):

        return

    else:

        divide(start, s[start-1][end-1])

        divide(s[start-1][end-1]+1, end)

        parans.append((start, end))

if \_\_name\_\_ == "\_\_main\_\_":

    # taking the input for matrix size and number of matrix

    n = int(input('Enter number of matrices : '))

    d = []

    for i in range(n):

        a, b = list(map(int, input(f'Enter order of matrix {i+1} : ').split()))

        d.append(a)

        if(i == n-1):

            d.append(b)

    # initializing to 0

    c = np.zeros((n, n), dtype=int).tolist()

    s = np.zeros((n, n), dtype=int).tolist()

    # getting factorial

    matVals = factorial(n-1)

    a = 1

    i = 0

    j = 0

    differencebw = 1

    # running the while loop/ algorithim for the factorial amount

    while(a <= matVals):

        if(i+differencebw != n):

            j = i+differencebw

        else:

            i = 0

            differencebw += 1

            j = i+differencebw

        k = i

        minimum = []

        while(k < j):

            minimum.append(c[i][k]+c[k+1][j]+d[i]\*d[k+1]\*d[j+1])

            k += 1

        # getting minimum value from the array

        c[i][j] = min(minimum)

        s[i][j] = minimum.index(min(minimum))+i+1

        i += 1

        a += 1

    # printing the results

    print("\n")

    print('Matrix Multiplication')

    for row in c:

        print(row)

    print("\n")

    print('K value matrix')

    for row in s:

        print(row)

    print("\n")

    print(f'Minimum multiplication is {c[0][n-1]} and the optimal way is ', end='')

    parans = []

    mat\_str = ''.join(chr(i) for i in range(65, 65+n))

    mat\_str = '('+mat\_str+')'

    divide(1, n)

    for start, end in parans[:-1]:

        mat\_list = list(mat\_str)

        mat\_list.insert(mat\_str.index(chr(64+start)), '(')

        mat\_str = ''.join(mat\_list)

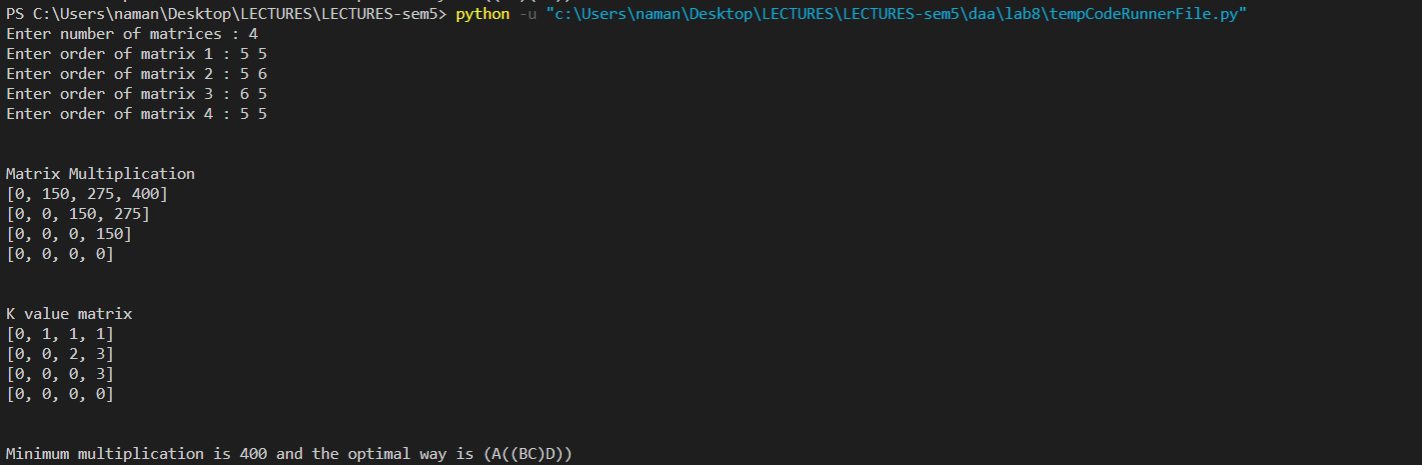
        mat\_list = list(mat\_str)

        mat\_list.insert(mat\_str.index(chr(64+end))+1, ')')

        mat\_str = ''.join(mat\_list)

    print(mat\_str)

**B.2 Input and Output:**

** B.3 Observations and learning:**

We used dynamic programming, Dynamic programming is a method for solving optimization problems.

It is algorithm technique to solve a complex and overlapping sub-problems. Compute the solutions to the sub-problems once and store the solutions in a table, so that they can be reused (repeatedly) later.

First observation I have is that we need a valid input, we must have matrices that are multipliable as given in the ss taken above

**B.4 Conclusion:**

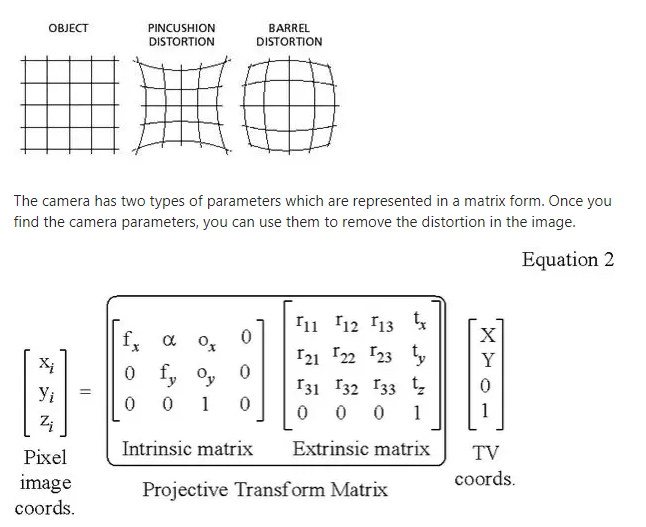
Thus using this technique, we can figure out which is the optimal way of matrix multiplication and the minimum multiplication of these matrices.

I also conclude the dynamic programming approach is better than the recursive approach

**B.5 Question of Curiosity**

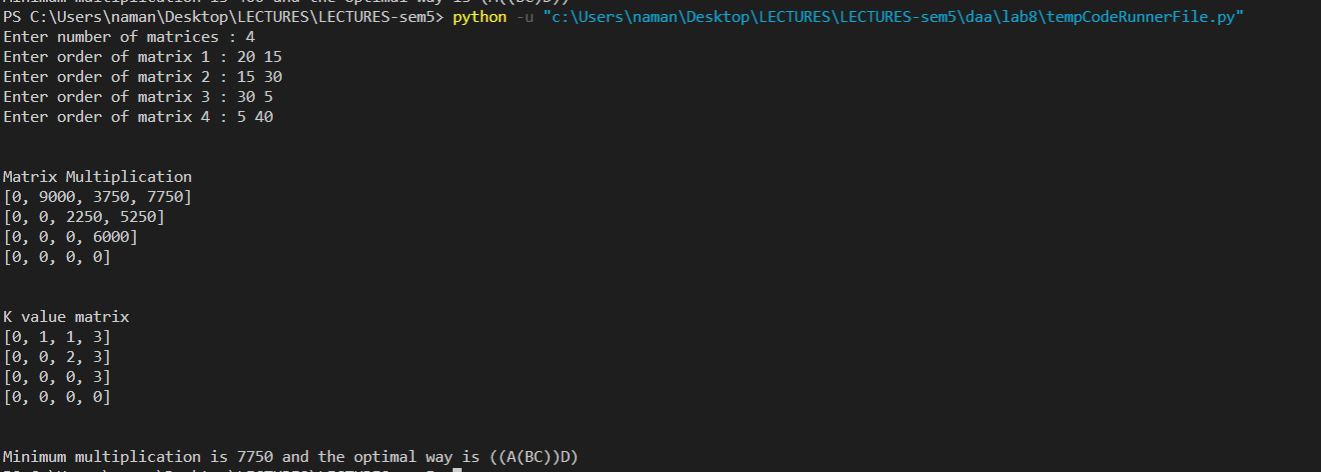
Q.1 Identify & discuss the real life applications Matrix Chain Multiplication.

* Solving linear equations
* Image transforms



* perspective projections, which is the foundation for 3D animation

Q.2 Consider the matrices P, Q, R and S which are 20 x 15, 15 x 30, 30 x 5 and 5 x 40 matrices respectively. What is the minimum number of multiplications required to multiply the four matrices?



This will take 7750 elementary multiplications.

Q.3 Compare various technique of matrix chain multiplication.

A general formula can be devised for the number of operations between 2 matrices but that too can depend on the number of operations between some other pair of matrices. This can lead to 2 approaches of solving problems : recursive where we don’t have to worry about the sequence of steps to be followed but that will take time and will consume stack memory ; and the other approach is dynamic programming where we follow the steps whose input is not some other step’s output. In this case, the diagonal of the matrix storing the elementary operations will always be 0. The cells just right to the diagonal will depend only on the outputs of the diagonal which we have figured out ie 0. So we can get the matrix whose last value in the 1st row will give the optimal number of elementary operations. The 2nd matrix will store the values of k which will help in the parenthesization process. The k values are the positions in the sequence about which to divide the sequence of operations. In the latter case of dynamic programming, it is noteworthy that we are filling the matrices in a diagonal fashion and not row wise or column wise. Using recursion, memory is consumed in the form of stack memory while in dynamic programming, memory is consumed in the form of the 2 matrices. The time complexity in recursion is exponential as we will calculate the same subproblems again and again while in dynamic programming, time complexity is of the order n^3.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*