**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 A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.08**

**A.1 Aim:**

Implementation of Dynamic Programming Technique – Matrix Chain Multiplication.

**A.2 Prerequisite:**

1. Concepts of Dynamic Programming Technique of algorithm design.

2. Knowledge of Matrix Handling.

3. Knowledge of Implementing Recursion.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Design & implement a solution using Dynamic Programming Technique.
2. Identify different problems that can be solved by using Dynamic Programming Technique.
3. Identify applications of Matrix Chain Multipliacation Problem.

**A.4 Theory:**

**A.4.1.**

The chain matrix multiplication problem is perhaps the most popular example of dynamic programming used in the upper undergraduate course (or review basic issues of dynamic programming in advanced algorithm's class).

The chain matrix multiplication problem involves the question of determining the optimal sequence for performing a series of operations. This general class of problem is important in complier design for code optimization and in databases for query optimization. We will study the problem in a very restricted instance, where the dynamic programming issues are clear. Suppose that our problem is to multiply a chain of n matrices A1 A2 ... An. Recall (from your discrete structures course), matrix multiplication is an associative but not a commutative operation. This means that you are free to parenthesize the above multiplication however we like, but we are not free to rearrange the order of the matrices. Also, recall that when two (non-square) matrices are being multiplied, there are restrictions on the dimensions.

Suppose, matrix A has p rows and q columns i.e., the dimension of matrix A is p × q. You can multiply a matrix A of p × q dimensions times a matrix B of dimensions q × r, and the result will be a matrix C with dimensions p × r. That is, you can multiply two matrices if they are **compatible:**the number of columns of A must equal the number of rows of B.

In particular, for 1 ≤ i ≤  p and 1 ≤ j ≤ r, we have

C[i, j] = ∑1 ≤ k ≤ q A[i, k] B[k, j].

There are p . r total entries in C and each takes O(q) time to compute, thus the total time to multiply these two matrices is dominated by the number of scalar multiplication, which is p . q . r.

***Dynamic Programming Approach***

The first step of the dynamic programming paradigm is to characterize the structure of an optimal solution. For the chain matrix problem, like other dynamic programming problems, involves determining the optimal structure (in this case, a parenthesization). We would like to break the problem into subproblems, whose solutions can be combined to obtain a solution to the global problem.

For convenience, let us adopt the notation Ai .. j, where i ≤ j, for the result from evaluating the product  Ai Ai + 1 ... Aj. That is,

Ai .. j ≡ Ai Ai + 1 ... Aj,    where i ≤ j,

It is easy to see that is a matrix Ai .. j  is of dimensions pi × pi + 1.

In parenthesizing the expression, we can consider the highest level of parenthesization. At this level we are simply multiplying two matrices together. That is, for any k, 1 ≤  k ≤  n − 1,

A1..n = A1..k  Ak+1..n .

Therefore, the problem of determining the optimal sequence of multiplications is broken up into two questions:

Question 1: How do we decide where to split the chain? (What is k?)

Question 2: How do we parenthesize the subchains A1**..**k  Ak+1..n?

The subchain problems can be solved by recursively applying the same scheme. On the other hand, to determine the best value of k, we will consider all possible values of k, and pick the best of them. Notice that this problem satisfies the principle of optimality, because once we decide to break the sequence into the product , we should compute each subsequence optimally. That is, for the global problem to be solved optimally, the subproblems must be solved optimally as well.

The key observation is that the parenthesization of the "prefix" subchain A1..kwithin this optimal parenthesization of A1..n. must be an optimal parenthesization of A1..k.

**Dynamic Programming Formulation**

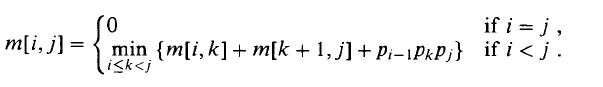
The second step of the dynamic programming paradigm is to define the value of an optimal solution recursively in terms of the optimal solutions to subproblems. To help us keep track of solutions to subproblems, we will use a table, and build the table in a bottom­up manner. For 1 ≤ i ≤ j ≤ n, let m[i, j] be the minimum number of scalar multiplications needed to compute the Ai..j. The optimum cost can be described by the following recursive formulation.

Basis: Observe that if i = j then the problem is trivial; the sequence contains only one matrix, and so the cost is 0. (In other words, there is nothing to multiply.) Thus,

m[i, i] = 0 for i = 1, 2, ..., n.

Step: If i ≠ j, then we are asking about the product of the subchain Ai..jand we take advantage of the structure of an optimal solution. We assume that the optimal parenthesization splits the product, Ai..j into for each value of k, 1 ≤  k ≤  n − 1 as Ai..k . Ak+1..j.

The optimum time to compute is m[i, k], and the optimum time to compute is m[k + 1, j]. We may assume that these values have been computed previously and stored in our array. Since Ai..kis a matrix, and Ak+1..jis a matrix, the time to multiply them is pi − 1. pk . pj. This suggests the following recursive rule for computing m[i, j].



To keep track of optimal subsolutions, we store the value of k in a table s[i, j]. Recall, k is the place at which we split the product Ai..j to get an optimal parenthesization. That is,

s[i, j] = k such that m[i, j] = m[i, k] + m[k + 1, j] + pi − 1. pk . pj.

**A.5 Procedure/Algorithm:**

**A.5.1:**

Matrix-Chain(array p[1 .. n], int n) {

Array s[1 .. n − 1, 2 .. n];

FOR i = 1 TO n DO m[i, i] = 0; // initialize

FOR L = 2 TO n DO { // L=length of subchain

FOR i = 1 TO n − L + 1 do {

j = i + L − 1;

m[i, j] = infinity;

FOR k = i TO j − 1 DO { // check all splits

q = m[i, k] + m[k + 1, j] + p[i − 1] p[k] p[j];

IF (q < m[i, j]) {

m[i, j] = q;

s[i, j] = k;

}

}

}

}

return m[1, n](final cost) and s (splitting markers);

}

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

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. | Name: |
| Class : | Batch : |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

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

***(Paste your c/c++ code completed during the 2 hours of practical in the lab here)***

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Input Data:**

**Output Data:**

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

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

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?

Q.3 Compare various technique of matrix chain multiplication.

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