Name	Kinnari Shah
UID no.	2021700058
Experiment No.	5

AIM:	To implement dynamic algorithms
Program 1	
PROBLEM STATEMENT :	**Details** – Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping sub-problems and optimal substructure property. If any problem can be divided into sub-problems, which in turn are divided into smaller sub-problems, and if there are overlapping among these sub-problems, then the solutions to these sub-problems can be saved for future reference. The approach of solving problems using dynamic programming algorithm has following steps: 1. Characterize the structure of an optimal solution. 2. Recursively define the value of an optimal solution. 3. Compute the value of an optimal solution, typically in a bottom-up fashion. 4. Construct an optimal solution from computed information. --- **Problem Definition & Assumptions** – The aim of this experiment is two-fold. First, it finds the efficient way of multiplying a sequence of k matrices (called ***Matrix Chain Multiplication***) using Dynamic Programming. The chain of multiplication M_1 x M_2 x M_3 x M_4 x...x M_k may be computed in $(2N)!/((N+1)!\,N!) = \binom{2N}{N}/(N+1)$ ways due to associative property where $N = k - 1$ of matrix multiplication. Second, it compares regular matrix multiplication which has complexity of $O(n^3)$ and Strassen's Matrix Multiplication which has complexity of $O(n^{2.81})$ using Divide and Conquer. Consider the optimization problem of efficiently multiplying a randomly generated sequence of 10 matrices (M_1, M_2, M_3, M_4 ,..., M_{10}) using Dynamic programming approach. The dimension of these matrices are stored in an array p[i] for i = 0 to 9, where the dimension of the matrix M_i is (p[i-1] x p[i]). All p[i] are randomly generated and they are between 15 and 46. For example, $p[0..10] = (23, 20, 25, 45, 30, 35, 40, 22, 15, 29, 21)$. All ten matrices are generated randomly and each matrix value can be between 0 and 1. Determine following values of ***Matrix Chain Multiplication (MCM)*** using Dynamic Programming: 1) m[1..10][1..10] = Two dimension matrix of optimal solutions (No. of multiplications) of all possible matrices M_1... M_{10} 2) c[1..9][2..10] = Two dimension matrix of optimal solutions (parenthesizations) of all combinations of matrices M_1...M_{10} 3) the optimal solution (i.e.parenthesization) for the multiplication of all ten matrices M_1x M_2x M_3xM_4 x...x M_{10} Find the running time of 10 matrices using regular matrix multiplication and Strassen's Matrix Multiplication as a trivial sequence i.e. (((((((((M_1x M_2)x M_3) xM_4 x...x M_{10}) and the sequence of matrix multiplication suggested by ***Matrix Chain Multiplication*** in Step No. 3 Links for Understanding basic concepts of MCM: 1. Youtube MCM video link - https://www.youtube.com/watch?v=prx1psByp7U 2. Reading resource link - https://www.javatpoint.com/matrix-chain-multiplication-example --- **Input –** **1)** All p[i] for i=0 to 9 are randomly generated and they are between 15 and 46. 2) All ten matrices are generated randomly and each matrix value can be between 0 and 1.

	Input – **1)** All p[i] for i=0 to 9 are randomly generated and they are between 15 and 46. 2) All ten matrices are generated randomly and each matrix value can be between 0 and 1. **Output –** 1) m[1..10][1..10] = Two dimension matrix of optimal solutions (No. of multiplications) of all possible matrices M_1... M_{10} 2) c[1..9][2..10] = Two dimension matrix of optimal solutions (parenthesizations) of all combinations of matrices M_1...M_{10} 3) The optimal solution (i.e.parenthesization) for the multiplication of all ten matrices M_1x M_2x M_3xM_4 x...x M_{10} 4) Print the time required to multiply ten matrices using four combinations as discussed above. **Submission –** 1) C/C++ source file of implementation 2) Verified output for the written source code with multiple inputs (four combinations) 3) One page report of Exp. 4 (PDF file)
ALGORITHM:	If a chain of matrices is given, we have to find the minimum number of the correct sequence of matrices to multiply. We know that the matrix multiplication is associative, so four matrices ABCD, we can multiply A(BCD), (AB)(CD), (ABC)D, A(BC)D, in these sequences. Like these sequences, our task is to find which ordering is efficient to multiply. In the given input there is an array say arr, which contains arr[] = {1, 2, 3, 4}. It means the matrices are of the order (1 x 2), (2 x 3), (3 x 4). So here we need to create n*n matrix where n is the number of matrices and in this matrix all the diagonal elements are zero and all the non diagonal elements are evaluated by the following formula $$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j \\ i \le k < j & \end{cases}$$ This n*n matrix is upper triangular.

PROGRAM:	#include<stdio.h>
	#include<stdlib.h>
	#include<stdbool.h>
	#include<limits.h>
	
	//utils
	void print_matrix(int n,int mat[n][n]){
	for(int i=1;i<n;i++){
	for(int j=1;j<n;j++){
	if(i>j)printf(" ");
	else printf(" %7d ",*(*(mat+i)+j));
	}

```
        printf(" \n");
    }

}

//constructing the solution
void print_optimal_parentheses(int n,int s[n][n],int i,int j){
    if(i==j){
        printf("A%d",i);
    }else{
        printf("(");
        print_optimal_parentheses(n,s,i,s[i][j]);
        print_optimal_parentheses(n,s,s[i][j]+1,j);
        printf(")");
    }

}

//computing the solution
void matrix_chain_multiplication(int dimensions[],int n){

    //initialization
    int m[n+1][n+1];
    int s[n+1][n+1];
    for(int i=0;i<n+1;i++){
        m[i][i]=0;
        s[i][i]=0;
    }

    //diagonal traversal
    int len=2;
    while(len<n+1){
        int second=len;
        int first=1;
        while(second<n+1){
            int min=INT_MAX;
            int mink=0;
            int k=first;
            while(k<second){
                int temp_cost=  m[first][k]
```

```
                    + m[k+1][second]
                    + dimensions[first-1]*dimensions[k]*dimensions[second];
            if(temp_cost<min){
               min=temp_cost;
               mink=k;
            }
            k++;
         }
         m[first][second]=min;
         s[first][second]=mink;
         first++;
         second++;
      }
      len++;
   }

   //printing results
   printf("\nthe cost matrix is: \n");
   print_matrix(n+1,m);
   printf("\nthe k matrix is: \n");
   print_matrix(n+1,s);

   printf("\noptimal cost is: %d\n",m[1][n]);

   printf("\noptimal parentheses is:\n\n");
   print_optimal_parentheses(n+1,s,1,n);
   printf(" \n");
}


int main(){
   int n;
   printf("enter number of matrices:\n");
   scanf("%d",&n);
   int dimensions[n+1];
   printf("enter dimension array:\n");
   for(int i=0;i<n+1;i++)scanf("%d",&dimensions[i]);
   matrix_chain_multiplication(dimensions,n);
}
```

RESULT:

```
enter number of matrices:
4
enter dimension array:
5
4
6
2
7

the cost matrix is:
       0      120       88      158
                0       48      104
                         0       84
                                  0

the k matrix is:
       0        1        1        3
                 0        2        3
                          0        3
                                   0

optimal cost is: 158

optimal parentheses is:

((A1(A2A3))A4)


...Program finished with exit code 0
Press ENTER to exit console.
```

OBSERVATION:	There are three nested loops. Each loop executes a maximum n times. 1. l, length, O (n) iterations. 2. i, start, O (n) iterations. 3. k, split point, O (n) iterations Body of loop constant complexity **Total Complexity is: O (n^3)**
CONCLUSION:	In this experiment, we computed optimal cost as well as optimal parenthesis for matrix chain multiplication using dynamic programming.