|  |  |
| --- | --- |
| **Name** | Ronak Thakar |
| **UID no.** | 2021700066 |
| **Experiment No.** | 5 |

|  |  |
| --- | --- |
| **AIM:** | Dynamic Programming - Matrix Chain Multiplication. |
| **Program** | |
| **PROBLEM STATEMENT:** | Use Dynamic Programming method to find the optimal way to multiply(parenthesize) the matrices to find the minimum number of multiplications required to solve the matrix. |
| **ALGORITHM/**  **THEORY:** | 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 subproblems, 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.   Given the dimension of a sequence of matrices in an array **arr[]**, where the dimension of the **ith** matrix is **(arr[i-1] \* arr[i]),** the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimum.  Note: Here we just find the way to multiply them but we don’t multiply the content of matrices as such.  1] Optimal Substructure: Here we break the number of matrices into smaller groups and solve them to find the minimum number of multiplications.  2] Recursive method: We use recursive call to find the possible ways to multiply them and solve them. The recursive formula is:  c[i, j] = 0 if i=j  = min(c[i, j],c[i, k] + c[k+1, j] + p[i-1]\*p[k]\*p[j]) if i<=k<=j  3] Computing the optimal cost  4] Constructing an optimal solution.  if(i==j)        printf("A%d",i);  else{        printf("(");        POP(i,s[i][j]);        POP(s[i][j]+1,j);        printf(")");  } |
| **PROGRAM:** | #include<stdio.h>  int mat[100][100],s[100][100],count=0;  int MCM(int p[], int i, int j){      if(i==j){        mat[i][j] = 0;        return 0;      }      mat[i][j] = 30000;      for(int k=i; k<j; k++){         count = MCM(p,i,k) + MCM(p,k+1,j) + p[i-1]\*p[k]\*p[j];         if(count<mat[i][j]){           mat[i][j] = count;           s[i][j] = k;         }      }      return mat[i][j];  }  void POP(int i,int j){      if(i==j)        printf("A%d",i);      else{        printf("(");        POP(i,s[i][j]);        POP(s[i][j]+1,j);        printf(")");      }  }  void main(){      int num;      printf("\nEnter the number of inputs you want to give: ");      scanf("%d",&num);      int p[num];      printf("\nEnter the order of matrices: ");      for(int i=0;i<num;i++){         printf("\nEnter value for place %d: ",i+1);         scanf("%d",&p[i]);      }      printf("\nThe minimum number of multiplications required are: %d\n\n",MCM(p,1,num-1));      for(int i=1;i<num;i++){         for(int j=1;j<num;j++){            printf("%d\t",mat[i][j]);         }         printf("\n");      }      printf("\nHence the optimal solution is: \n");      POP(1,num-1);  } |
| **RESULT:** | |
| **CONCLUSION:** | We used Dynamic Programming steps to solve Matrix Chain Multiplication problem. |