|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **21 / 02 / 2024** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Dr. Aarti Phadke** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 6**

**Title: Matrix Chain Multiplication**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation Matrix Chain Multiplication of Dynamic Programming. |

|  |
| --- |
| **COs to be achieved:** |
| **CO2:** Describe various algorithm design strategies to solve different problems. |

|  |
| --- |
| **Apparatus / Software Tools Used:** |
| 1. VS Code 2. Microsoft Excel |

|  |
| --- |
| **Theory:** |
| Problem Definition:  Given a sequence of N matrices, the matrix chain multiplication problem is to find the most efficient way to multiply these matrices by minimizing the number of computations involved during multiplications.  Optimal Substructure:  Parameterization/select the subgroup of matrices that will result in least number of computations.  For multiplication of matrix series Ai to Aj, choose Ak such that multiplication of matrices through Ai..k and Ak+1…j will incur least number of computations for any k such that i<=k<j.  Recursive Formula: |

|  |
| --- |
| **Upload the code/output:** |
| Code:  #include <stdio.h>  #include <limits.h>  #define MAX\_SIZE 100  void printMatrix(int mat[MAX\_SIZE][MAX\_SIZE], int rows, int cols) {      for (int i = 1; i <= rows; i++) {          for (int j = 1; j <= cols; j++) {              printf("%d\t", mat[i][j]);          }          printf("\n");      }  }  void printOptimalParenthesization(int s[MAX\_SIZE][MAX\_SIZE], int i, int j) {      if (i == j)          printf("A%d", i);      else {          printf("(");          printOptimalParenthesization(s, i, s[i][j]);          printOptimalParenthesization(s, s[i][j] + 1, j);          printf(")");      }  }  int matrixChainMultiplication(int dims[], int n, int m[MAX\_SIZE][MAX\_SIZE], int s[MAX\_SIZE][MAX\_SIZE]) {      for (int i = 1; i <= n; i++)          m[i][i] = 0;      for (int L = 2; L <= n; L++) {          for (int i = 1; i <= n - L + 1; i++) {              int j = i + L - 1;              m[i][j] = INT\_MAX;              for (int k = i; k <= j - 1; k++) {                  int cost = m[i][k] + m[k + 1][j] + dims[i - 1] \* dims[k] \* dims[j];                  if (cost < m[i][j]) {                      m[i][j] = cost;                      s[i][j] = k;                  }              }          }      }        return m[1][n];  }  int main()  {      int n, dims[MAX\_SIZE], m[MAX\_SIZE][MAX\_SIZE], s[MAX\_SIZE][MAX\_SIZE];      printf("\nenter the number of matrices: ");      scanf("%d", &n);      printf("\nenter the dimensions of each matrix:\n");      for (int i = 0; i <= n; i++) {          printf("dimension %d: ", i + 1);          scanf("%d", &dims[i]);      }      int minMultiplications = matrixChainMultiplication(dims, n, m, s);      printf("\nminimum number of multiplications is %d\n", minMultiplications);      printf("\npptimal parenthesization: ");      printOptimalParenthesization(s, 1, n);      printf("\n");      printf("\npartition matrix:\n");      printMatrix(s, n, n);      printf("\n");      return 0;  }  Output:    Handwritten Solution: |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Solve the Matrix Chain Multiplication for the order using dynamic method only. Using dimensions <4, 10, 3, 12, 20>.**  Output:    Handwritten Solution: |

|  |
| --- |
| **Conclusion:** |
| In conclusion, through experimentation with matrix chain multiplication, we've gained insight into the application of a new greedy technique, providing a methodical approach for optimizing matrix multiplication operations, crucial in various computational tasks, such as in algorithm design and optimization strategies. |

**Signature of faculty in-charge with Date:**