|  |  |
| --- | --- |
| **Name** | Manish Shashikant Jadhav |
| **UID** | 2023301005 |
| **Subject** | Design and Analysis of Algorithms (DAA) |
| **Experiment No.** | 4 |
| **Aim** | To implement Dynamic Algorithms.   1. Assembly Line Scheduling. 2. Longest Common Subsequence. |
| **Code:** | #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <limits.h>  #define NUM\_MATRICES 10  *// Function to allocate memory for a 2D matrix*  int\*\* allocateMatrix(int *rows*, int *cols*) {      int \*\*matrix = (int \*\*)malloc(*rows* \* sizeof(int \*));      for (int i = 0; i < *rows*; i++) {          matrix[i] = (int \*)malloc(*cols* \* sizeof(int));      }      return matrix;  }  *// Function to free memory for a 2D matrix*  void freeMatrix(int \*\**matrix*, int *rows*) {      for (int i = 0; i < *rows*; i++) {          free(*matrix*[i]);      }      free(*matrix*);  }  *// Function to generate random matrices*  void generateRandomMatrices(int \*\*\**matrices*, int \**dims*) {      for (int i = 0; i < NUM\_MATRICES; i++) {          int rows = *dims*[i];          int cols = *dims*[i + 1];  *matrices*[i] = allocateMatrix(rows, cols);          for (int row = 0; row < rows; row++) {              for (int col = 0; col < cols; col++) {  *matrices*[i][row][col] = rand() % 2; *// Random values between 0 and 1*              }          }      }  }  *// Function to perform Matrix Chain Multiplication using dynamic programming*  void matrixChainMultiplication(int *dims*[], int *n*, int \*\**m*, int \*\**c*) {      for (int i = 1; i <= *n*; i++) {  *m*[i][i] = 0;      }      for (int len = 2; len <= *n*; len++) {          for (int i = 1; i <= *n* - len + 1; i++) {              int j = i + len - 1;  *m*[i][j] = INT\_MAX;              for (int k = i; k < j; 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;  *c*[i][j] = k;                  }              }          }      }  }  int main() {      srand(time(NULL));  *// Generate random dimensions for matrices*      int dims[NUM\_MATRICES + 1]; *// +1 to include the dimensions of result matrix*      printf("Random Dimensions for Matrices:\n");      for (int i = 0; i <= NUM\_MATRICES; i++) {          dims[i] = rand() % 32 + 15; *// Random dimensions between 15 and 46*          printf("M%d: %2d x %2d\n", i, dims[i-1], dims[i]);      }  *// Generate random matrices*      int \*\*\*matrices = (int \*\*\*)malloc(NUM\_MATRICES \* sizeof(int \*\*));      generateRandomMatrices(matrices, dims);  *// Allocate memory for storing optimal solutions and parenthesizations*      int \*\*m = allocateMatrix(NUM\_MATRICES + 1, NUM\_MATRICES + 1);      int \*\*c = allocateMatrix(NUM\_MATRICES + 1, NUM\_MATRICES + 1);  *// Perform Matrix Chain Multiplication and measure time*      clock\_t start = clock();      matrixChainMultiplication(dims, NUM\_MATRICES, m, c);      clock\_t end = clock();      double duration\_mcm = ((double)(end - start)) / CLOCKS\_PER\_SEC;  *// Print optimal solutions*      printf("\nOptimal Solutions (No. of Multiplications):\n");      for (int i = 1; i <= NUM\_MATRICES; i++) {          for (int j = 1; j <= NUM\_MATRICES; j++) {              printf("%6d", m[i][j]);          }          printf("\n");      }  *// Print time for Matrix Chain Multiplication*      printf("\nTime for Matrix Chain Multiplication: %.6f seconds\n", duration\_mcm);  *// Print the cost matrix and the k matrix*      printf("\nCost Matrix:\n");      for (int i = 1; i <= NUM\_MATRICES; i++) {          for (int j = 1; j <= NUM\_MATRICES; j++) {              printf("%6d", m[i][j]);          }          printf("\n");      }        printf("\nK Matrix:\n");      for (int i = 1; i <= NUM\_MATRICES; i++) {          for (int j = 1; j <= NUM\_MATRICES; j++) {              printf("%6d", c[i][j]);          }          printf("\n");      }  *// Free allocated memory for matrices*      for (int i = 0; i < NUM\_MATRICES; i++) {          freeMatrix(matrices[i], dims[i]);      }      free(matrices);  *// Free allocated memory for optimal solutions and parenthesizations*      freeMatrix(m, NUM\_MATRICES + 1);      freeMatrix(c, NUM\_MATRICES + 1);      return 0;  } |
| **Output** |  |
| **Pseudo Code** |  |
| **Conclusion** | Hence, by completing this experiment I came to know about implementation of Matrix Chain Multiplication. |