Shubhan Singh

SE-Comps B/Batch C

2022300118

DAA Experiment 5

Aim – To implement Dynamic programming algorithms to find the optimal order for Matrix chain multiplication.

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 subproblems, then the solutions to these sub-problems can be saved for future reference.

Problem statement:

Consider the optimization problem of efficiently multiplying a sequence of n matrices (M1, M2, M3, M4 ,..., Mn) using Dynamic programming approach. The dimension of these matrices are stored in an array p[i] for i = 0 to n, where the dimension of the matrix Mi is (p[i-1] x p[i]).

Determine following values of Matrix Chain Multiplication (MCM) using Dynamic Programming:
1) m[1..n][1..n] = Two dimension matrix of optimal solutions (No. of multiplications) of all possible matrices M1... Mn

2) the optimal solution (i.e.parenthesization) for the multiplication of all n matrices M1x M2x M3xM4 x...x Mn

Pseudocode :

Algorithm Matrix-Chain-Multiplication(p, n)

Input: Sequence p[0..n] of matrix dimensions, number of matrices n

Output: The minimum number of scalar multiplications needed to compute the product of the matrices

1. Let m[1..n, 1..n] and s[1..n, 1..n] be new tables

```
2. for i = 1 to n
3.     m[i, i] = 0
4. for l = 2 to n    // l is the chain length
5.     for i = 1 to n-l+1
6.         j = i+l-1
7.         m[i, j] = infinity
8.         for k = i to j-1
9.             q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
10.            if q < m[i, j]
11.                m[i, j] = q
12.                s[i, j] = k
13. Print the matrix of costs m
14. Print the matrix s
15. Call the function Parenthesize(1, n, s) to print the optimal parenthesization
16. return m[1, n]
```

M[1,n] contains the minimum number of scalar multiplications required

```
Function Parenthesize(i, j, s)
  Input: Matrix s of splitting points, indices i and j
  Output: A string representing the optimal parenthesization

  1. if i == j
  2.     return "A" + i
  3. else
  4.     return "(" + Parenthesize(i, s[i, j], s) + " x " + Parenthesize(s[i, j] + 1, j, s) + ")"
```

Source code(C language):

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

char* parenthesize(int i, int j, int** arr){
    if(i==j){
        char* s=malloc(2*sizeof(char));
        s[0]='A'+i;
        s[1]='\0';
        return s;
    }
    char *l,*r;
    l=parenthesize(i,arr[i][j],arr);
    r=parenthesize(arr[i][j]+1,j,arr);
    char* s=malloc(256*sizeof(char));
    s[0]='(';
    s[1]='\0';
    strcat(s,l);
    strcat(s," x ");
    strcat(s,r);
    strcat(s,")");
    free(l);
    free(r);
    return s;
}


signed main(){
    srand(time(NULL));
    int n,temp;
    printf("Enter number of Martices: ");
    scanf("%d",&n);
    char a='A';
    int p[n+1];
    printf("Enter dimensions of matrix %c: ",a++);
    scanf("%dx%d",&p[0],&p[1]);
    // for(int i=0;i<=n;i++){
    //     p[i]=rand()%30;
    //     if(p[i]<2){
```

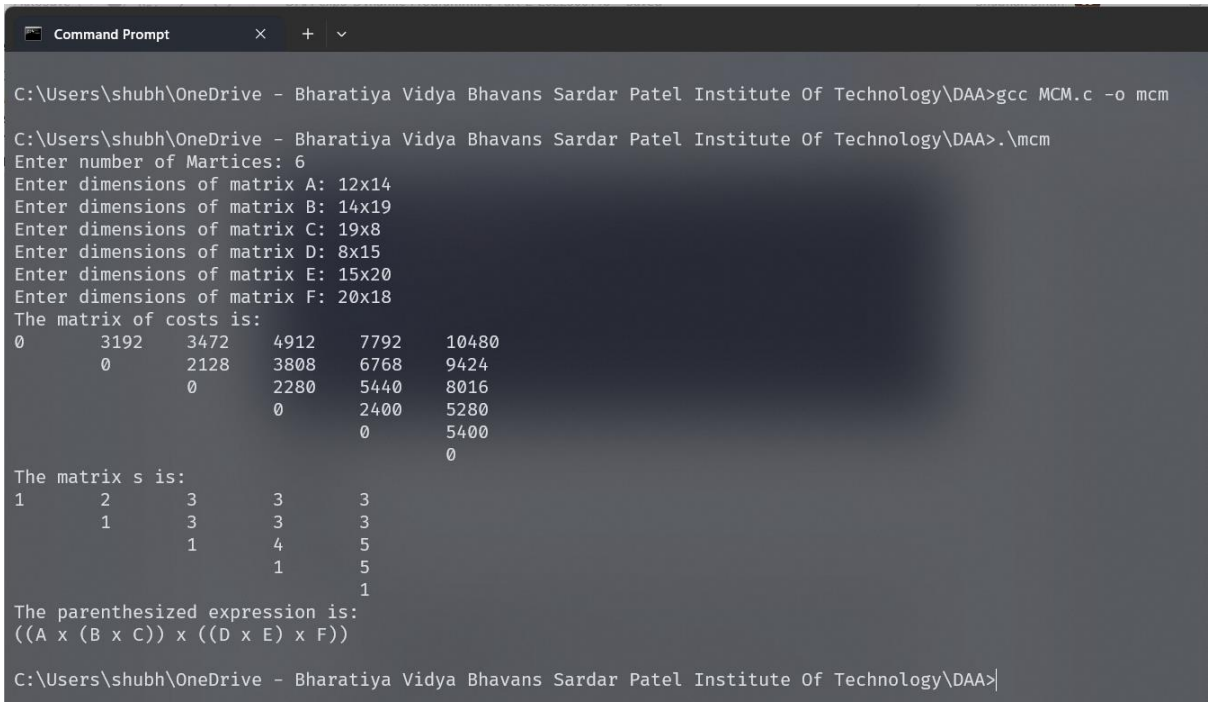
```
//          p[i]=2;
//      }
//      printf("%d ",p[i]);
// }
// printf("\n");
temp=p[1];
for(int i=1;i<n;i++){
    printf("Enter dimensions of matrix %c: ",a++);
    scanf("%dx%d",&temp,&p[i+1]);
    if(temp!=p[i]){
        printf("Invalid dimensions\n");
        return 0;
    }
}
int m[n][n];
int** s=malloc(n*sizeof(int*));
for(int i=0;i<n;i++){
    s[i]=malloc(n*sizeof(int));
    m[i][i]=0;
    s[i][i]=0;
}
int j;
for(int l=2;l<=n;l++){
    for(int i=0;i<=n-l;i++){
        j=l+i-1;
        m[i][j]=INT_MAX;
        for(int k=i;k<j;k++){
            temp=m[i][k]+m[k+1][j]+p[i]*p[k+1]*p[j+1];
            if(temp<m[i][j]){
                m[i][j]=temp;
                s[i][j]=k;
            }
        }
    }
}
printf("The matrix of costs is: \n");
for(int i=0;i<n;i++){
    for(int x=0;x<i;x++){printf("\t");}
    for(int j=i;j<n;j++){
        printf("%d\t",m[i][j]);
    }
    printf("\n");
```

```
    }
    printf("The matrix s is: \n");
    for(int i=1;i<n;i++){
        for(int x=1;x<i;x++){printf("\t");}
        for(int j=i;j<n;j++){
            printf("%d\t",s[i][j]+1);
        }
        printf("\n");
    }

    printf("The parenthesized expression is:\n");
    char* exp=parenthesize(0,n-1,s);
    printf("%s\n",exp);
    for(int i=0;i<n;i++){
        free(s[i]);
    }
    free(s);
    free(exp);

return 0;
}
```

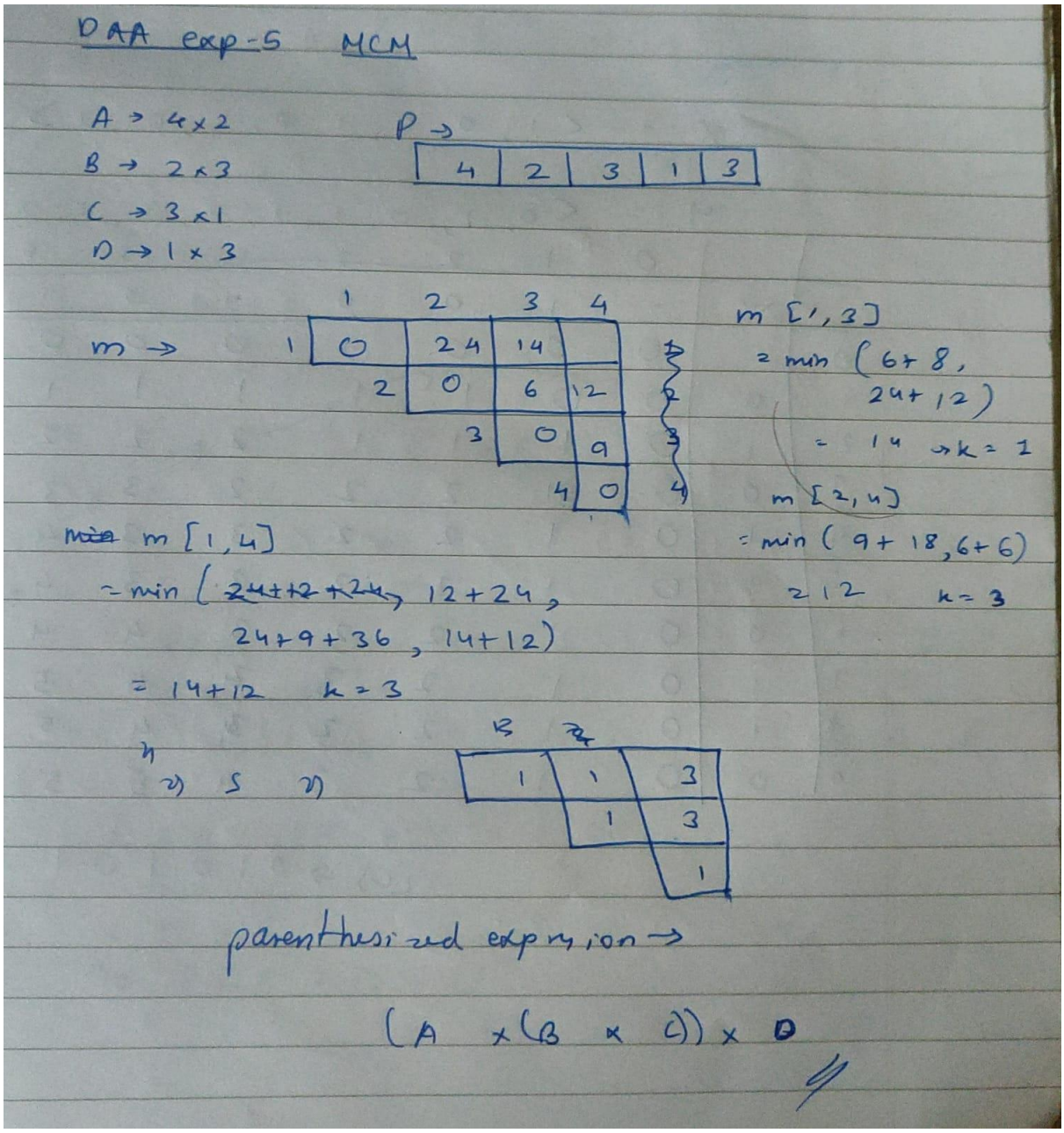
Output:



Conclusion:

- This dynamic programming solution executes in O(n^3) time complexity, where n is the number of matrices we are trying to multiply.
- This solution satisfies the optimal substructure property as in the process of arriving to the final answer, we also found the optimal solutions to all of the all of the subproblems (the minimum number of scalar multiplications required to multiply Matrices i…j).

Rough Working :



Theory:

1. **Problem Statement**: The Matrix Chain Multiplication problem is an optimization problem that deals with the most efficient way to multiply a chain of matrices. The problem is not to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved.
2. **Order Matters**: The order of matrix multiplication matters because the cost of multiplication can vary dramatically depending on the order. For example, if you have three matrices A, B, and C with dimensions 10x100, 100x5, and 5x50 respectively, then (A(BC)) would require 7500 scalar multiplications, while ((AB)C) would require 25000.
3. **Dynamic Programming Solution**: The problem can be solved using dynamic programming by breaking it down into smaller subproblems, solving each subproblem only once, and storing their results in case they are needed later (this is known as memoization).
4. **Subproblems**: The subproblems are defined by a starting and ending position for the chain of matrices to be multiplied (i.e., for each pair (i, j) where $1 \leq i \leq j \leq n$, find the most efficient way to multiply matrices i through j in the chain).
5. **Recurrence Relation**: The dynamic programming solution uses a recurrence relation to express the solution of the problem in terms of smaller subproblems. The minimum number of multiplications needed to multiply matrices i through j is found by trying all possibilities for the final multiplication, and choosing the one that costs the least.
6. **Parenthesization**: After the table is filled, the solution to the problem can be found by tracing back through the decisions that led to the optimal cost. This gives the optimal parenthesization of the matrix chain.
7. **Time Complexity**: The time complexity of the dynamic programming solution to the Matrix Chain Multiplication problem is $O(n^3)$, where n is the number of matrices in the chain.