Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Jhanvi Parekh

60009210033

CSE(Data Science)

Experiment 5

(Dynamic Programming)

Aim: Implementation of Matrix Chain Multiplication using dynamic programming.

Theory:

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

For example: A is a 10 x 30 matrix, B is a 30 x 5 matrix, and C is a 5 x 60 matrix.

(AB)C = (10x30x5) + (10x5x60) = 1500 + 3000 = 4500 operations

A(BC) = (30x5x60) + (10x30x60) = 9000 + 18000 = 27000 operations.

APPROACH: -

1) Optimal Substructure:

- A simple solution is to place parenthesis at all possible places, calculate the cost for each placement and return the minimum value. In a chain of matrices of size n, we can place the first set of parenthesis in n-1 ways. For example, if the given chain is of 3 matrices. let the chain be ABC, then there are 2 ways to place first set of parenthesis outer side: (A)(BC) and (AB)(C).
- So when we place a set of parenthesis, we divide the problem into sub-problems of smaller size. Therefore, the problem has optimal substructure property and can be easily solved using recursion.
- Minimum number of multiplication needed to multiply a chain of size n = Minimum of all n-1 placements.

RECURSIVE ALGORITHM to find the minimum cost:-

- Take the sequence of matrices and separate it into two subsequences.
- Find the minimum cost of multiplying out of each subsequence.
- Add these costs together, and add in the cost of multiplying the two matrices.
- Do this for each possible position at which the sequence of matrices can be split and take the minimum over all of them.
- The time complexity of above solution is exponential.

Since same suproblems are called again, this problem has overlapping subprolems property. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array dp[][] in bottom up manner.

Pseudocode:

MATRIX-CHAIN-ORDER (p)
n ← length[p]-1
for i ← 1 to n do
m [i, i] ← 0
4. for l ← 2 to n   // l is the chain length
do for i ← 1 to n-l + 1
do j ← i+ l -1 m[i,j] ←
∞ for k ← i to j-1
do q ← m [i, k] + m [k + 1, j] + pi-1 pk pj
if q < m [i,j] then m [i,j] ← q s [i,j] ← k
return m and s

Complexity:

Best Case Time Complexity: $O(n^2)$

Lab Assignment to Complete:

Find a minimum number of multiplications required to multiply: A [1 × 5], B [5 × 4], C [4 × 3], D [3 × 2], and E [2 × 1]. Also, give optimal parenthesization.

CODE:

```
#include <stdio.h>

#include <limits.h>

#define N 5

int A[N+1] = {1, 5, 4, 3, 2, 1};

int m[N+1][N+1];

int s[N+1][N+1];
```

```
void print_optimal_parens(int i, int j) {
 if (i == j) {
 printf("A%d", i);
 } else {
 printf("(");
 print_optimal_parens(i, s[i][j]);
 print_optimal_parens(s[i][j] + 1, j);
 printf(")");
 }
}
int main() {
 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 q = m[i][k] + m[k+1][j] + A[i-1]*A[k]*A[j];
 if (q < m[i][j]) {
 m[i][j] = q;
 s[i][j] = k;
 }
```

```
 }
 }
}
printf("60009210033 Jhanvi Parekh \n");
printf("Minimum number of multiplications: %d\n", m[1][N]);
printf("Optimal parenthesization: ");
print_optimal_parens(1, N);
printf("\n");
return 0;}
```

main.c

Run

```c
#include <stdio.h>

#include <limits.h>

#define N 5

int A[N+1] = {1, 5, 4, 3, 2, 1};

int m[N+1][N+1];

int s[N+1][N+1];

void print_optimal_parens(int i, int j) {

 if (i == j) {

 printf("A%d", i);

 } else {

 printf("(");

 print_optimal_parens(i, s[i][j]);

 print_optimal_parens(s[i][j] + 1, j);
```

main.c

Run

```
print_optimal_parens(s[i][j] + 1, j);

printf(")");

}

}

int main() {

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++) {
```

main.c

Run

```
for (int k = i; k <= j - 1; k++) {

int q = m[i][k] + m[k+1][j] + A[i-1]*A[k]*A[j];

if (q < m[i][j]) {

m[i][j] = q;

s[i][j] = k;

}

}

}

}

printf("60009210033 Jhanvi Parekh \n");

printf("Minimum number of multiplications: %d\n", m[1][N]);

printf("Optimal parenthesization: ");

print_optimal_parens(1, N);
```

main.c

```c
 m[i][j] = q;

 s[i][j] = k;

 }

 }

 }

}

printf("60009210033 Jhanvi Parekh \n");

printf("Minimum number of multiplications: %d\n", m[1][N]);

printf("Optimal parenthesization: ");

print_optimal_parens(1, N);

printf("\n");

return 0;}
```

OUTPUT:

Output

```
/tmp/H7zi2Clv5h.o
60009210033 Jhanvi Parekh
Minimum number of multiplications: 40
Optimal parenthesization: ((((A1A2)A3)A4)A5)
```

WRITTEN:

Q.

P_0	P_1	P_2	P_3	P_4	P_5
1	5	4	3	2	1

M_1 M_2 M_3 M_4 M_5

cost

	1	2	3	4	5	
	0	20	32	38	40	1
		0	60	64	88	2
			0	24	18	3
				0	6	4
					0	5

k matrix

	1	2	3	4	5	
	0	1	2	3	4	1
		0	2	2	2	2
			0	3	3	3
				0	4	4
					0	5

$M[1,2] = M[1,1] + M[2,2] + P_0 * P_1 * P_2$

$k = 1 \quad = 1 \times 5 \times 4 = 20$

$M[2,3] = M[2,2] + M[3,3] + P_1 \times P_2 \times P_3$

$k = 2 \quad = 5 \times 4 \times 3 = 60$

$M[3,4] = M[3,3] + M[4,4] + P_2 \times P_3 \times P_4$

$k = 3 \quad = 4 \times 3 \times 2 = 24$

$M[1,3] = \min \begin{cases} k=1: & M[1,1] + M[2,3] + P_0 \times P_2 \times P_3 \\ & = 60 + 1 \times 5 \times 3 = 75 \\ k=2: & M[1,2] + M[3,3] + P_0 \times P_2 \times P_3 \\ & = 20 + 1 \times 4 \times 3 = 32 \end{cases}$

$k = 1, 2$

$$M[2,4] = \min_{k=2,3} \begin{cases} k=2: M[2,2] + M[3,4] + P_1 \times P_2 \times P_3 \\ \quad = 24 + 5 \times 4 \times 2 = 24 \\ k=3: M[2,3] + M[4,4] + P_1 \times P_3 \times P_4 \\ \quad = 60 + 5 \times 3 \times 2 = 90 \end{cases}$$

$$M[3,5] = \min_{k=3,4} \begin{cases} k=3: M[3,3] + M[4,5] + P_2 \times P_3 \times P_5 \\ \quad = 6 + 4 \times 3 \times 1 = 18 \\ k=4: M[3,4] + M[5,5] + P_2 \times P_4 \times P_5 \\ \quad = 24 + 4 \times 2 \times 1 = 32 \end{cases}$$

$$M[1,4] = \min_{k=1,2,3} \begin{cases} k=1: M[1,1] + M[2,4] + P_0 \times P_1 \times P_4 \\ \quad = 64 + 1 \times 5 \times 2 = 74 \\ k=2: M[1,2] + M[3,4] + P_0 \times P_2 \times P_4 \\ \quad = 20 + 24 + 1 \times 4 \times 2 = 52 \\ k=3: M[1,3] + M[4,4] + P_0 \times P_3 \times P_4 \\ \quad = 32 + 1 \times 3 \times 2 = 38 \end{cases}$$

$$M[2,5] = \min_{k=2,3,4} \begin{cases} k=2: M[2,2] + M[3,5] + P_1 \times P_2 \times P_5 \\ \quad = 18 + 5 \times 4 \times 1 = 38 \\ k=3: M[2,3] + M[4,5] + P_1 \times P_3 \times P_5 \\ \quad = 60 + 6 + 5 \times 3 \times 1 = 81 \\ k=4: M[2,4] + M[5,5] + P_1 \times P_4 \times P_5 \\ \quad = 64 + 5 \times 2 \times 1 = 74 \end{cases}$$

$$M[1,5] = \min \begin{cases} K=1: M[1,1] + M[2,5] + P_0 \times P_1 \times P_5 \\ \quad = 38 + 5 \times 1 \times 1 = 43 \\ K=2: M[1,2] + M[3,5] + P_0 \times P_2 \times P_5 \\ \quad = 20 + 18 + 1 \times 4 \times 1 = 42 \\ K=3: M[1,3] + M[4,5] + P_0 \times P_3 \times P_5 \\ \quad = 32 + 6 + 1 \times 3 \times 1 = 41 \\ K=4: M[1,4] + M[2,5] + P_0 \times P_4 \times P_5 \\ \quad = 38 + 1 \times 2 \times 1 = 40 \end{cases}$$

$K = 1, 2, 3, 4$

Optimal Parenthesis

$((((M_1 \; M_2) \; M_3) \; M_4) \; M_5)$