get a detailed insight of dynamic programming approach by the implementation of Matrix Chain Multiplication problem and see the impact of parenthesis positioning on

time requirements for matrix multiplication.

#include <stdio.h>

#include <limits.h>

// Function to perform Matrix Chain Multiplication using Dynamic Programming

void matrixChainOrder(int p[], int n) {

// m[i][j] stores the minimum number of scalar multiplications

int m[n][n];

int s[n][n]; // s[i][j] stores the index of the split point that achieved the optimal cost

// Cost is zero when multiplying one matrix

for (int i = 1; i < n; i++)

m[i][i] = 0;

// L is chain length

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;

// Try placing the parenthesis between each possible split point `k`

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

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

// Update m[i][j] if we find a smaller cost

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

m[i][j] = q;

s[i][j] = k; // Store the split point

}

}

}

}

// Output the minimum cost

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

// Print optimal parenthesis placement

printf("Optimal Parenthesization is: ");

printOptimalParenthesis(s, 1, n - 1);

printf("\n");

}

// Recursive function to print the optimal parenthesis placement

void printOptimalParenthesis(int s[][10], int i, int j) {

if (i == j) {

printf("A%d", i);

} else {

printf("(");

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

printOptimalParenthesis(s, s[i][j] + 1, j);

printf(")");

}

}

int main() {

// Array p represents the dimensions of matrices

// For matrices A1: 30x35, A2: 35x15, A3: 15x5, A4: 5x10, A5: 10x20, A6: 20x25

// Array p will be {30, 35, 15, 5, 10, 20, 25}

int p[] = {30, 35, 15, 5, 10, 20, 25};

int n = sizeof(p) / sizeof(p[0]);

// Find and display the optimal order

matrixChainOrder(p, n);

return 0;

}