**DAA [Day - 2]**

UID: 24MCI10204

Name: Rahul Saxena

Branch: 24MCA – AI & ML

**Question 1:** Implement Strassen’s Matrix Multiplication algorithm to compute the product of two square matrices.

**Answer:**

Strassen’s Matrix Multiplication is an efficient divide-and-conquer algorithm used to multiply two square matrices faster than the conventional method.

* **Traditional method time complexity:** O(n³)
* **Strassen's algorithm time complexity:** O(n^log₂7) ≈ O(n^2.81)

Strassen’s algorithm reduces the number of **multiplications** by computing **7 products** instead of 8 when multiplying two 2×2 matrices.

**Source Code:**

import java.util.\*;

public class StrassenMultiplication {

public static int[][] strassen(int[][] A, int[][] B) {

int n = A.length;

if (n == 1) {

int[][] C = {{A[0][0] \* B[0][0]}};

return C;

}

int newSize = n / 2;

int[][] A11 = new int[newSize][newSize];

int[][] A12 = new int[newSize][newSize];

int[][] A21 = new int[newSize][newSize];

int[][] A22 = new int[newSize][newSize];

int[][] B11 = new int[newSize][newSize];

int[][] B12 = new int[newSize][newSize];

int[][] B21 = new int[newSize][newSize];

int[][] B22 = new int[newSize][newSize];

split(A, A11, 0, 0);

split(A, A12, 0, newSize);

split(A, A21, newSize, 0);

split(A, A22, newSize, newSize);

split(B, B11, 0, 0);

split(B, B12, 0, newSize);

split(B, B21, newSize, 0);

split(B, B22, newSize, newSize);

int[][] M1 = strassen(add(A11, A22), add(B11, B22));

int[][] M2 = strassen(add(A21, A22), B11);

int[][] M3 = strassen(A11, subtract(B12, B22));

int[][] M4 = strassen(A22, subtract(B21, B11));

int[][] M5 = strassen(add(A11, A12), B22);

int[][] M6 = strassen(subtract(A21, A11), add(B11, B12));

int[][] M7 = strassen(subtract(A12, A22), add(B21, B22));

int[][] C11 = add(subtract(add(M1, M4), M5), M7);

int[][] C12 = add(M3, M5);

int[][] C21 = add(M2, M4);

int[][] C22 = add(subtract(add(M1, M3), M2), M6);

int[][] C = new int[n][n];

join(C11, C, 0, 0);

join(C12, C, 0, newSize);

join(C21, C, newSize, 0);

join(C22, C, newSize, newSize);

return C;

}

public static int[][] add(int[][] A, int[][] B) {

int n = A.length;

int[][] C = new int[n][n];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

C[i][j] = A[i][j] + B[i][j];

return C;

}

public static int[][] subtract(int[][] A, int[][] B) {

int n = A.length;

int[][] C = new int[n][n];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

C[i][j] = A[i][j] - B[i][j];

return C;

}

public static void split(int[][] P, int[][] C, int row, int col) {

for (int i = 0; i < C.length; i++)

for (int j = 0; j < C.length; j++)

C[i][j] = P[i + row][j + col];

}

public static void join(int[][] C, int[][] P, int row, int col) {

for (int i = 0; i < C.length; i++)

for (int j = 0; j < C.length; j++)

P[i + row][j + col] = C[i][j];

}

public static void main(String[] args) {

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

int[][] B = {{5, 6}, {7, 8}};

int[][] C = strassen(A, B);

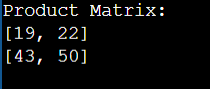
System.out.println("Product Matrix:");

for (int[] row : C)

System.out.println(Arrays.toString(row));

}

}

**Output:**

**Question 2:** A company has only one conference room that can be used for meetings. Each meeting request has a start time and end time, and no two meetings can overlap. Given a list of n meeting requests, maximize the number of non-overlapping meetings that can be scheduled in the room.

You must choose the maximum number of meetings that can be accommodated without overlaps.

**Answer:**

import java.util.\*;

class Meeting implements Comparable<Meeting> {

int start, end;

public Meeting(int start, int end) {

this.start = start;

this.end = end;

}

public int compareTo(Meeting m) {

return this.end - m.end;

}

}

public class MeetingScheduler {

public static int maxMeetings(List<Meeting> meetings) {

Collections.sort(meetings);

int count = 1;

int lastEndTime = meetings.get(0).end;

for (int i = 1; i < meetings.size(); i++) {

if (meetings.get(i).start >= lastEndTime) {

count++;

lastEndTime = meetings.get(i).end;

}

}

return count;

}

public static void main(String[] args) {

List<Meeting> meetings = new ArrayList<>();

meetings.add(new Meeting(1, 3));

meetings.add(new Meeting(2, 4));

meetings.add(new Meeting(3, 5));

meetings.add(new Meeting(0, 6));

meetings.add(new Meeting(5, 7));

meetings.add(new Meeting(8, 9));

meetings.add(new Meeting(5, 9));

int max = maxMeetings(meetings);

System.out.println("Maximum number of non-overlapping meetings: " + max);

}

}

**Output**