import java.util.\*;

public class BankersAlgorithm {

    static int P, R;

    static int[][] allocation; // Allocation matrix

    static int[][] max; // Max demand matrix

    static int[][] need; // Need matrix

    static int[] available; // Available resources

    // Function to check whether the system is in a safe state

    public static boolean isSafe() {

        int[] work = new int[R];

        System.arraycopy(available, 0, work, 0, R);

        boolean[] finish = new boolean[P];

        Arrays.fill(finish, false);

        int count = 0;

        List<Integer> safeSequence = new ArrayList<>();

        while (count < P) {

            boolean found = false;

            for (int i = 0; i < P; i++) {

                if (!finish[i] && isLessThanOrEqual(need[i], work)) {

                    // If process i can finish, do it

                    for (int j = 0; j < R; j++) {

                        work[j] += allocation[i][j];

                    }

                    finish[i] = true;

                    count++;

                    safeSequence.add(i);  // Add process to the safe sequence

                    found = true;

                    break;

                }

            }

            if (!found) {

                // If no process can finish, the system is not in a safe state

                return false;

            }

        }

        // If all processes can finish, the system is in a safe state

        // Print the safe sequence

        System.out.print("Safe Sequence: ");

        for (int i = 0; i < safeSequence.size(); i++) {

            System.out.print("P" + safeSequence.get(i));

            if (i != safeSequence.size() - 1) {

                System.out.print(" -> ");

            }

        }

        System.out.println();

        return true;

    }

    // Helper function to check if a process's need is less than or equal to available resources

    private static boolean isLessThanOrEqual(int[] need, int[] available) {

        for (int i = 0; i < R; i++) {

            if (need[i] > available[i]) {

                return false;

            }

        }

        return true;

    }

    // Function to initialize the matrices and available resources

    public static void initialize(int processes, int resources) {

        P = processes;

        R = resources;

        allocation = new int[P][R];

        max = new int[P][R];

        need = new int[P][R];

        available = new int[R];

    }

    // Function to input the data

    public static void inputData() {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the Allocation Matrix:");

        for (int i = 0; i < P; i++) {

            for (int j = 0; j < R; j++) {

                allocation[i][j] = sc.nextInt();

            }

        }

        System.out.println("Enter the Max Matrix:");

        for (int i = 0; i < P; i++) {

            for (int j = 0; j < R; j++) {

                max[i][j] = sc.nextInt();

            }

        }

        System.out.println("Enter the Available Resources:");

        for (int i = 0; i < R; i++) {

            available[i] = sc.nextInt();

        }

        // Calculating the Need matrix

        for (int i = 0; i < P; i++) {

            for (int j = 0; j < R; j++) {

                need[i][j] = max[i][j] - allocation[i][j];

            }

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of processes: ");

        int processes = sc.nextInt();

        System.out.print("Enter number of resources: ");

        int resources = sc.nextInt();

        initialize(processes, resources);

        inputData();

        if (isSafe()) {

            System.out.println("The system is in a safe state.");

        } else {

            System.out.println("The system is not in a safe state.");

        }

    }

}