Assignment 1: Banker's Algorithm Implementation (Deadlock Avoidance)

Write a program to check if a system is in a safe state and simulate resource requests. Implement the Banker's Algorithm to determine the safety state of a system and handle resource requests.

1. Data Structures: Define the necessary data structures for a system with N processes and M resource types:
    - Available: A vector of length M indicating the number of available resources of each type.
    - Max: An N X M matrix defining the maximum demand of each process.
    - Allocation: An N X M matrix defining the number of resources of each type currently allocated to each process.
    - Need: An N X M matrix defining the remaining resource need of each process, calculated as Need = Max - Allocation
2. Safety Algorithm: Implement the Safety Algorithm to check if the current state of the system is safe. If safe, output a safe sequence of processes.
3. Banker's Algorithm (Resource Request Allocation Algorithm): Implement the Banker's Algorithm. When a process P_i requests a vector of resources $Request_i$:

Assignment 2: Deadlock Detection Algorithm Simulation

Write the deadlock detection algorithm, which is used in systems that allow processes to request resources as needed and periodically check for deadlocks. Implement the Deadlock detection algorithm to detect if a set of processes is currently deadlocked.

Assignment 3: Comparing Avoidance vs. Detection in a System Simulator

Write an interactive system simulator to compare avoidance and detection. Simulate resource management under both deadlock avoidance (Banker's) and deadlock detection mechanisms, and analyze their performance and impact on process completion.

1. Simulation Mode Selector: Allow the user to choose between two modes:
    - Mode A: Deadlock Avoidance (Banker's Algorithm): Use the Banker's Algorithm to determine if a request should be granted or if requests must wait.
    - Mode B: Deadlock Detection: Always grant a resource request if $Request_i$ <= Available. Periodically (after every 3 requests) run the Deadlock Detection Algorithm. If a deadlock is found, one of the deadlocked processes (e.g., the one with the lowest ID) must be preempted (all its resources added to Available).
2. For both modes, simulate a continuous cycle where processes randomly generate small resource requests, and upon completion of their Need, release all their resources.
3. Output:

- Find the total time taken for all processes to complete in both modes.
- In Mode B, track the number of times a deadlock was detected and the number of processes preempted.