Name	Anurag Nair
UID no.	2022600035
Experiment No.	7

AIM:	Deadlock Avoidance
Program	
PROGRAM:	

```
import java.util.*;

class Main {

   static int totalProcesses;
   static int totalResources;
   static int[][] needMatrix;
   static boolean[] finishProcesses;
   static int[] safeSequence;
   static int[] workArray;

   static void calculateNeed(int[][] maxMatrix, int[][]
allocationMatrix) {
       needMatrix = new int[totalProcesses][totalResources];
       for (int i = 0; i < totalProcesses; i++) {
           for (int j = 0; j < totalResources; j++) {
               needMatrix[i][j] = maxMatrix[i][j] -
allocationMatrix[i][j];
           }
       }
   }

   static void checkSafeState(int[] available, int[][]
maxMatrix, int[][] allocationMatrix) {

       calculateNeed(maxMatrix, allocationMatrix);

       finishProcesses = new boolean[totalProcesses];
       safeSequence = new int[totalProcesses];
       workArray = Arrays.copyOf(available, available.length);

       int counter = 0;
       while (counter < totalProcesses) {
           boolean foundSafeState = false;
           for (int m = 0; m < totalProcesses; m++) {
               if (!finishProcesses[m]) {
```

```
                    int j;
                    for (j = 0; j < totalResources; j++) {
                        if (needMatrix[m][j] > workArray[j])
                            break;
                    }
                    if (j == totalResources) {
                        for (int k = 0; k < totalResources;
k++) {
                            workArray[k] +=
allocationMatrix[m][k];
                        }

                        safeSequence[counter++] = m;
                        finishProcesses[m] = true;
                        foundSafeState = true;
                    }
                }
            }
            if (!foundSafeState) {
                System.out.println("The system is not in the
safe state due to lack of resources.");
                return;
            }
        }
        System.out.print("The system is in a safe state and the
safe sequence is: ");
        for (int i = 0; i < totalProcesses; i++) {
            System.out.print("P" + safeSequence[i] + " ");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of processes and
resources:");
        totalProcesses = sc.nextInt();
        totalResources = sc.nextInt();

        System.out.println("Enter the availability of each
resource:");
        int[] availableResources = new int[totalResources];
        for (int i = 0; i < totalResources; i++) {
            availableResources[i] = sc.nextInt();
        }

        System.out.println("Enter the maximum resources that
can be allocated to each process:");
```

```
        int[][] maxMatrix = new
int[totalProcesses][totalResources];
        for (int i = 0; i < totalProcesses; i++) {
            System.out.print("For process P" + i + ":");
            for (int j = 0; j < totalResources; j++) {
                maxMatrix[i][j] = sc.nextInt();
            }
        }

        System.out.println("Enter the allocated resources for
each process:");
        int[][] allocationMatrix = new
int[totalProcesses][totalResources];
        for (int i = 0; i < totalProcesses; i++) {
            System.out.print("For process P" + i + ":");
            for (int j = 0; j < totalResources; j++) {
                allocationMatrix[i][j] = sc.nextInt();
            }
        }

        checkSafeState(availableResources, maxMatrix,
allocationMatrix);
    }
}
```

OUTPUT:

```
Enter the number of processes and resources:5 3
Enter the availability of each resource:
3 3 2
Enter the maximum resources that can be allocated to each process:
For process P0:7 5 3
For process P1:3 2 2
For process P2:9 0 2
For process P3:2 2 2
For process P4:4 3 3
Enter the allocated resources for each process:
For process P0:0 1 0
For process P1:2 0 0
For process P2:3 0 2
For process P3:2 1 1
For process P4:0 0 2
The system is in a safe state and the safe sequence is: P1 P3 P4 P0 P2
Process finished with exit code 0
```

CONCLUSION:	From this experiment, I learned about the Banker's algorithm and how to implement it to manage resource allocation and detect potential deadlocks in computing systems. This algorithm proved essential in ensuring system stability by balancing available resources with process requirements, highlighting the importance of resource management in maintaining optimal system performance and reliability.