Name: Akhil Katlagunta
Roll Number: **Division:**
Enrollment Number:

Operating Systems Assignment
(Implementation of **Banker's Algorithm** for Deadlock Avoidance)

Theory

The Banker's Algorithm is a deadlock avoidance algorithm used in operating systems. It is named so because it compares the resource allocation problem to a bank loaning money such that it never allocates funds in a way that could lead to unsafe states (where all funds may not be returned).

The algorithm uses three main matrices:

- **Allocation Matrix:** The current allocation of each resource type to each process.
- **Maximum Demand Matrix:** The maximum number of each resource that each process may request.
- **Available Vector:** The number of available instances for each resource type.

From these, a Need Matrix is calculated:

```
Need[i][j] = Max[i][j] - Allocation[i][j]
```

Procedure

The algorithm works as follows:

1. It checks if there is a safe sequence in which all processes can complete.
2. A process is said to be safely executable if its needs can be satisfied with the currently available resources.
3. If such a process is found, it is assumed to execute and release its resources, which are added back to the available pool.
4. This continues until all processes can complete, proving the system is in a safe state.

If no such sequence exists, the system is in an unsafe state, which may lead to deadlock.

Screenshots

Os assign.py > is_safe_state

```python
def is_safe_state(processes, available, max_demand, allocation):
    n = len(processes)
    m = len(available)

    # Calculate Need matrix
    need = [[max_demand[i][j] - allocation[i][j] for j in range(m)] for i in range(n)]
    finish = [False] * n
    safe_sequence = []
    work = available.copy()
    while len(safe_sequence) < n:
        allocated = False
        for i in range(n):
            if not finish[i] and all(need[i][j] <= work[j] for j in range(m)):
                for j in range(m):
                    work[j] += allocation[i][j]
                safe_sequence.append(f'P{i}')
                finish[i] = True
                allocated = True
                break
        if not allocated:
            break
    if len(safe_sequence) == n:
        print("\n System is in a safe state.")
        print("Safe sequence:", " -> ".join(safe_sequence))
    else:
        print("\n System is NOT in a safe state.")

# Input from User
n = int(input("Enter number of processes: "))
m = int(input("Enter number of resource types: "))

print("\nEnter Allocation Matrix:")
allocation = []
for i in range(n):
    row = list(map(int, input(f"Process P{i}: ").split()))
    allocation.append(row)

print("\nEnter Maximum Demand Matrix:")
max_demand = []
for i in range(n):
    row = list(map(int, input(f"Process P{i}: ").split()))
    max_demand.append(row)
available = list(map(int, input("\nEnter Available Resources: ").split()))
processes = [f'P{i}' for i in range(n)]

# Run Banker's Algorithm
is_safe_state(processes, available, max_demand, allocation)
```

Figure 1: Input of Banker's Algorithm Simulation

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console

```
PS C:\Users\akhil\Desktop\Study\code\rough> & 'c:\Users\akhil\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\akhil\.vscode\extensions\ms-python.debugpy
-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '57454' '--' 'C:\Users\akhil\Desktop\Study\code\rough\Os assign.py'
Enter number of processes: 5
Enter number of resource types: 3

Enter Allocation Matrix:
Process P0: 0 1 0
Process P1: 2 0 0
Process P2: 3 0 2
Process P3: 2 1 1
Process P4: 0 0 2

Enter Maximum Demand Matrix:
Process P0: 7 5 3
Process P1: 3 2 2
Process P2: 9 0 2
Process P3: 2 2 2
Process P4: 4 3 3

Enter Available Resources: 3 3 2

 System is in a safe state.
Safe sequence: P1 -> P3 -> P0 -> P2 -> P4
PS C:\Users\akhil\Desktop\Study\code\rough>
```

Figure 2: Output of Banker's Algorithm Simulation