Experiment No: 05
Date of Experiment: 05/ 05 / 2025
Experiment Name: Banker's Algorithm

Theory:

Banker’s Algorithm is a deadlock avoidance algorithm used in operating systems to allocate resources to processes in a way that avoids deadlock. It is named so because it simulates the way a banker might allocate funds to clients to ensure that the bank never runs out of money.

The algorithm works by checking the system's state whenever a new resource is requested. It determines whether the system will remain in a safe state after the allocation. A safe state is one where there exists a safe sequence of all processes such that each process can finish execution even if all of them demand their maximum resources.

Key Terms:

- **Available**: The number of available instances of each resource.
- **Max**: The maximum demand of each process.
- **Allocation**: The number of resources currently allocated to each process.
- **Need**: Remaining resources needed by a process = Max - Allocation.

The algorithm works in two parts:

1. **Safety Algorithm**: Checks whether the system is in a safe state by trying to find a sequence of all processes where each can finish with the currently available resources.
2. **Resource-Request Algorithm**: When a process *P* requests resources, the algorithm tentatively allocates the requested resources and then runs the safety algorithm to check if the resulting state is safe. If safe, the allocation proceeds; otherwise, the request is denied, and the process must wait.

Explanation:

• **Input Handling**: User inputs number of processes and resource types, followed by matrices for Allocation, Maximum, and Available resources.

• **Need Matrix Calculation**: Automatically calculated as Max - Allocation for each process.

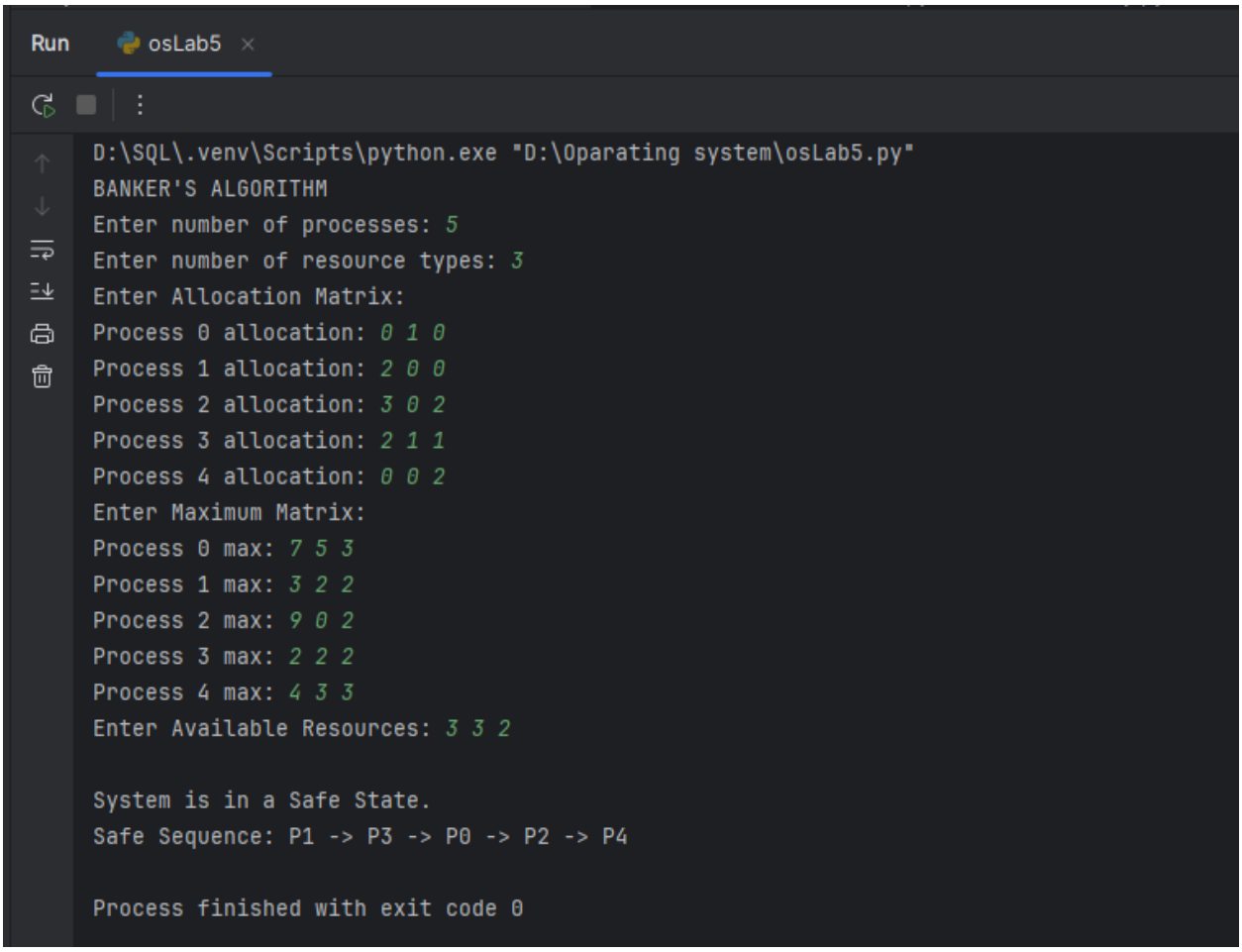
• **Safety Check Loop**:

- The algorithm checks if any process can complete with the current available resources.
- If yes, the process is marked as finished, and its allocated resources are returned to the pool.
- This continues until all processes are completed (safe state) or no further process can proceed (unsafe state).

Code:

```
print("BANKER'S ALGORITHM")
n = int(input("Enter number of processes: "))
m = int(input("Enter number of resource types: "))
print("Enter Allocation Matrix:")
allocation = []
for i in range(n):
    allocation.append(list(map(int, input(f"Process {i} allocation: ").split())))
print("Enter Maximum Matrix:")
maximum = []
for i in range(n):
    maximum.append(list(map(int, input(f"Process {i} max: ").split())))
available = list(map(int, input("Enter Available Resources: ").split()))
need = []
for i in range(n):
    row = []
    for j in range(m):
        row.append(maximum[i][j] - allocation[i][j])
    need.append(row)
finish = [False] * n
safe_sequence = []
while len(safe_sequence) < n:
    allocated_in_this_round = False
    for i in range(n):
        if not finish[i]:
            if all(need[i][j] <= available[j] for j in range(m)):
                for j in range(m):
                    available[j] += allocation[i][j]
                safe_sequence.append(f"P{i}")
                finish[i] = True
                allocated_in_this_round = True
                break
    if not allocated_in_this_round:
        break
if len(safe_sequence) == n:
    print("\nSystem is in a Safe State.")
    print("Safe Sequence:", ' -> '.join(safe_sequence))
else:
    print("\nSystem is NOT in a Safe State.")
```

Input and Output:



Discussion:

Banker's Algorithm is a classic method of deadlock avoidance. It ensures that resources are only allocated if the system will remain in a safe state. This prevents circular wait conditions and potential deadlocks.

It has **some limitations**:

1. **Complexity**: It is computationally intensive, especially in systems with many resources and processes.
2. **Requirement of Prior Knowledge**: It requires knowing each process's maximum resource need in advance, which may not always be practical.
3. **Not Scalable**: Less suitable for large systems or real-time environments due to overhead.

Explanation:

- Need matrix is computed by subtracting Allocation from Max Demand for each process.
- The safety check starts with the Available vector. It looks for any process whose Need is less than or equal to Available, “allocates” its resources (i.e., adds its allocation back to Available), and repeats until all processes finish or no further allocations are possible.
- If all processes can finish, the system is in a safe state, and the order found is the safe sequence.
- In the sample, processes complete in the order **P1, P3, P0, P2, P4**, ensuring no deadlock.

Conclusion:

The Banker's Algorithm provides a robust mechanism for deadlock avoidance by ensuring resource allocation only occurs when the system remains in a safe state. Its practical application is limited by its overhead and the need for precise maximum demand information, but it offers clear insights into resource management strategies and system safety guarantees in operating system design.