DEADLOCK AVOIDANCE ALGORITHM

CODE:

```
def input_matrix():
    rows = int(input("Enter number of rows: "))
    cols = int(input("Enter number of columns: "))
    matrix = []
    print("Enter the elements row by row:")
    for i in range(rows):
        row = list(map(int, input().split()))
        while len(row) != cols:
            print(f"Please enter exactly {cols} elements.")
            row = list(map(int, input().split()))
        matrix.append(row)
    return matrix

def check_row(allocation, claim, available, row_index):
    # Ensure the row index is within the bounds
    if row_index >= len(claim) or row_index >= len(allocation):
        print("Row index out of bounds.")
        return False
    # Calculate claim - allocation for the specified row
    claim_minus_allocation = [claim[row_index][j] - allocation[row_index][j] for j in
range(len(claim[row_index]))]

    # Check if the claim minus allocation is less than or equal to available resources
    for j in range(len(claim_minus_allocation)):
        if claim_minus_allocation[j] > available[j]:
            return False
    return True
# Example usage
print("Enter the Claim Matrix")
Claim_Matrix = input_matrix()
print("Enter the Allocation Matrix")
Allocation_Matrix = input_matrix()
# Ensure the Available_Matrix is a single row vector
print("Enter Available Matrix:")
Available_Matrix = list(map(int, input().split()))  # Single line input for available resources
print("Enter the Resource Matrix")
Resource_Matrix = input_matrix()
# Track the execution status of processes
process_executed = [False] * len(Claim_Matrix)
# Keep looping until all processes are executed
i = 0  # Start with the first process
while not all (process_executed):
# Check each row of Claim minus Allocation
    if not process_executed[i]:
        is_valid = check_row(Allocation_Matrix, Claim_Matrix, Available_Matrix, i)
        if is_valid:
            print(f"Process P{i + 1} is running.")
            print(f"Previous Available Matrix (V): {Available_Matrix}")  # Display updated available
resources
            # Update the Available_Matrix by adding the allocated resources back
            for j in range(len(Available_Matrix)):
                Available_Matrix[j] += Allocation_Matrix[i][j]  # Add the current allocation to available

            print(f"Updated Available Matrix (Vnew): {Available_Matrix}")  # Display updated available
resources
            # Mark the process as executed
            process_executed[i] = True
            # After executing a process, reset index to check from the start
            i = -1  # Set i to -1 because it will be incremented to 0 by the loop
        else:
            print(f"Process P{i + 1} cannot be executed due to insufficient resources.")
    i += 1  # Move to the next process
    # If we've checked all processes, reset to start again
    if i >= len(Claim_Matrix):
        # If no more processes can be executed, detect deadlock
        if all(process_executed):
            print("All processes have been executed successfully.")
        else:
            print("Deadlock detected. No more processes can be executed.")
        break  # Exit the loop if deadlock occurs
```

OUTPUT:

```
Enter the Claim Matrix
Enter number of rows: 4
Enter number of columns: 3
Enter the elements row by row:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the Allocation Matrix
Enter number of rows: 4
Enter number of columns: 3
Enter the elements row by row:
1 0 0
6 1 2
2 1 1
0 0 2
Enter Available Matrix:
0 1 1
Enter the Resource Matrix
Enter number of rows: 1
Enter number of columns: 3
Enter the elements row by row:
9 3 6
Process P1 cannot be executed due to insufficient resources.
Process P2 is running.
Previous Available Matrix (V): [0, 1, 1]
Updated Available Matrix (Vnew): [6, 2, 3]
Process P1 is running.
Previous Available Matrix (V): [6, 2, 3]
Updated Available Matrix (Vnew): [7, 2, 3]
Process P3 is running.
Previous Available Matrix (V): [7, 2, 3]
Updated Available Matrix (Vnew): [9, 3, 4]
Process P4 is running.
Previous Available Matrix (V): [9, 3, 4]
Updated Available Matrix (Vnew): [9, 3, 6]
```