Assignment - 5

Aim : Implement Bankers algorithm for Deadlock avoidance

Theory :

Banker's Algortihm:
The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for the predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Available :

- It is a 1-d array of size 'm' indicating the number of available resources of each type.
- Available[j] = k means there are 'k' instances of resource type Rj

Max :

- It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system.
- Max[i, j] = k means process Pi may request at most 'k' instances of resource type Rj.

Allocation :

- It is a 2-d array of size 'n*m' that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process Pi is currently allocated 'k' instances of resource type Rj

Need :

- It is a 2-d array of size 'n*m' that indicates the remaining resource need of each process.
- Need [i, j] = k means process Pi currently needs 'k' instances of resource type Rj
- Need [i, j] = Max [i, j] – Allocation [i, j]

Program & Output :

```
def display_table():
   print("Process |\t Allocation \t|\t Max Need \t|\t Remaining Need|")

print("--------+-----------------------+-----------------------+-------
----------------+")
   for i in range(no_of_processes):
       print(f"P{i} \t|\t {allocation[i]} \t|\t {max_need[i]} \t|\t
{rem_need[i]} \t|")

if __name__ == '__main__':
   no_of_processes = 5
   no_of_resources = 3
   total = [10,5,7]
   allocation = [
       [0,1,0],
       [2,0,0],
       [3,0,2],
       [2,1,1],
       [0,0,2]
   ]
   max_need = [
       [7,5,3],
       [3,2,2],
       [9,0,2],
       [2,2,2],
       [4,3,3]
   ]
   available = [3,3,2]
   rem_need = [[0] * no_of_resources for _ in range(no_of_processes)]
   safe_seq = []

   for i in range(no_of_processes):
       for j in range(no_of_resources):
           rem_need[i][j] = max_need[i][j] - allocation[i][j]

   finished = [False] * no_of_processes
   for _ in range(no_of_processes):
       for i in range(no_of_processes):
          if not finished[i]:
              count = 0
              for j in range(no_of_resources):
                  if rem_need[i][j] <= available[j]:
```

```
                    count+=1
            if count == no_of_resources:
                safe_seq.append(f"P{i+1}")
                for j in range(no_of_resources):
                    available[j] += allocation[i][j]
                finished[i] = True

if len(safe_seq) == no_of_processes:
    print("System is in safe state\n")
    display_table()
    print("\nSafe Sequence : ", safe_seq)
else:
    print("System is in an unsafe state")
```

```
➜  OS python3 bankers_algo.py
System is in safe state

Process |        Allocation     |        Max Need       |        Remaining Need |
--------+-----------------------+-----------------------+-----------------------+
P0      |        [0, 1, 0]      |        [7, 5, 3]      |        [7, 4, 3]      |
P1      |        [2, 0, 0]      |        [3, 2, 2]      |        [1, 2, 2]      |
P2      |        [3, 0, 2]      |        [9, 0, 2]      |        [6, 0, 0]      |
P3      |        [2, 1, 1]      |        [2, 2, 2]      |        [0, 1, 1]      |
P4      |        [0, 0, 2]      |        [4, 3, 3]      |        [4, 3, 1]      |

Safe Sequence :  ['P2', 'P4', 'P5', 'P1', 'P3']
➜  OS
```

```
16          [0,0,2]
17      ]
18      max_need = [
19          [7,5,3],
20          [3,2,2],
21          [11,0,2], #max need changes to
22          [2,2,2],
23          [4,3,3]
24      ]
25      available = [3,3,2]

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

➜  OS python3 bankers_algo.py
System is in an unsafe state
➜  OS
```

Conclusion : Here in this assignment, we studied banker's algorithm, why banker's algorithm is used and implemented in python using 2d arrays.