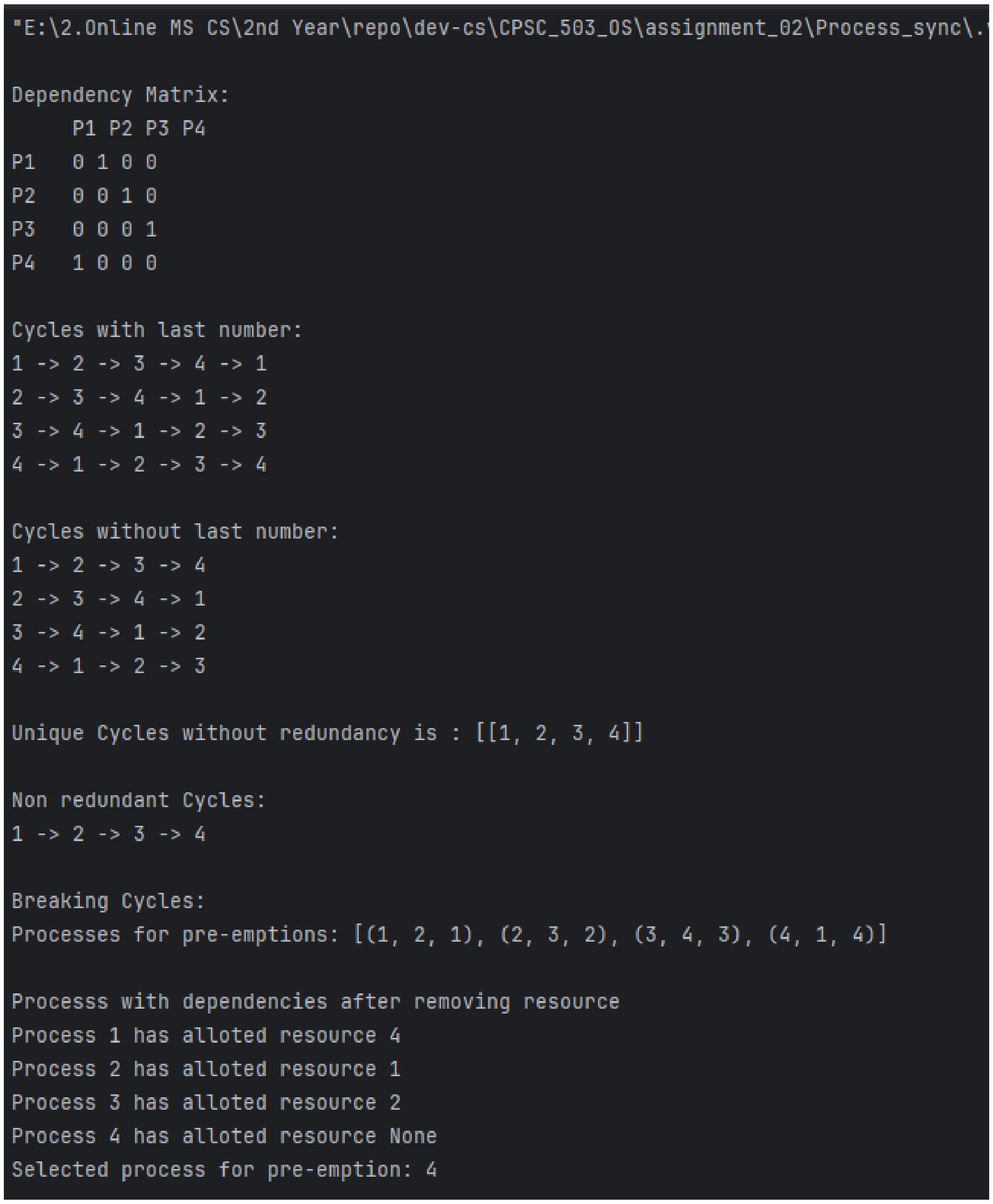
STUDENT ID: 1174066

Deadlock Management Modules Implementation (graph reduction algorithm ,deadlock detection algorithm and deadlock avoidance module)

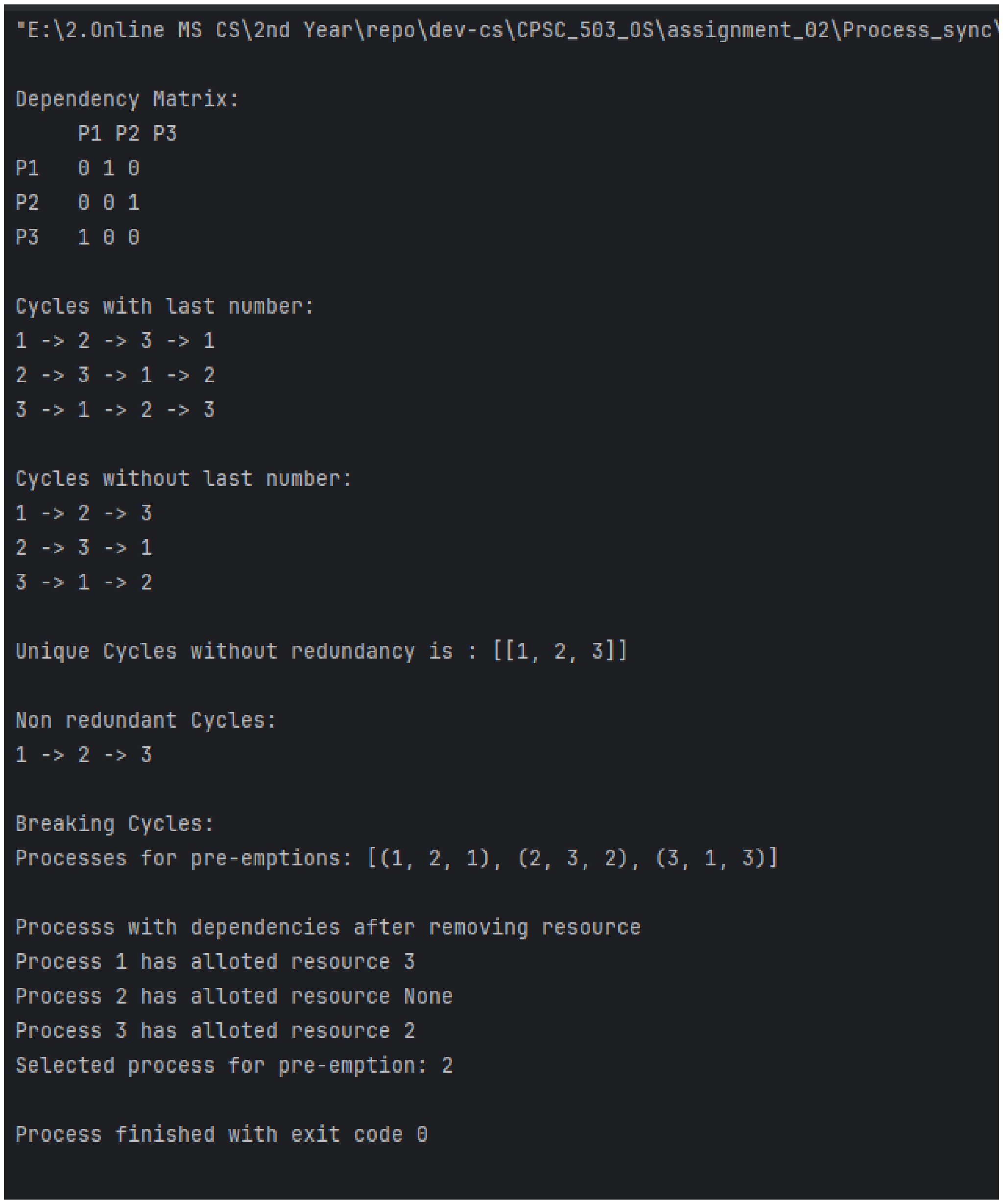
Implementation final output

Process Resource Allocation 1



```
"E:\2.Online MS CS\2nd Year\repo\dev-cs\CPSC_503_OS\assignment_02\Process_sync\.

Dependency Matrix:
     P1 P2 P3 P4
P1   0 1 0 0
P2   0 0 1 0
P3   0 0 0 1
P4   1 0 0 0

Cycles with last number:
1 -> 2 -> 3 -> 4 -> 1
2 -> 3 -> 4 -> 1 -> 2
3 -> 4 -> 1 -> 2 -> 3
4 -> 1 -> 2 -> 3 -> 4

Cycles without last number:
1 -> 2 -> 3 -> 4
2 -> 3 -> 4 -> 1
3 -> 4 -> 1 -> 2
4 -> 1 -> 2 -> 3

Unique Cycles without redundancy is : [[1, 2, 3, 4]]

Non redundant Cycles:
1 -> 2 -> 3 -> 4

Breaking Cycles:
Processes for pre-emptions: [(1, 2, 1), (2, 3, 2), (3, 4, 3), (4, 1, 4)]

Processs with dependencies after removing resource
Process 1 has alloted resource 4
Process 2 has alloted resource 1
Process 3 has alloted resource 2
Process 4 has alloted resource None
Selected process for pre-emption: 4
```

Process Resource Allocation 2



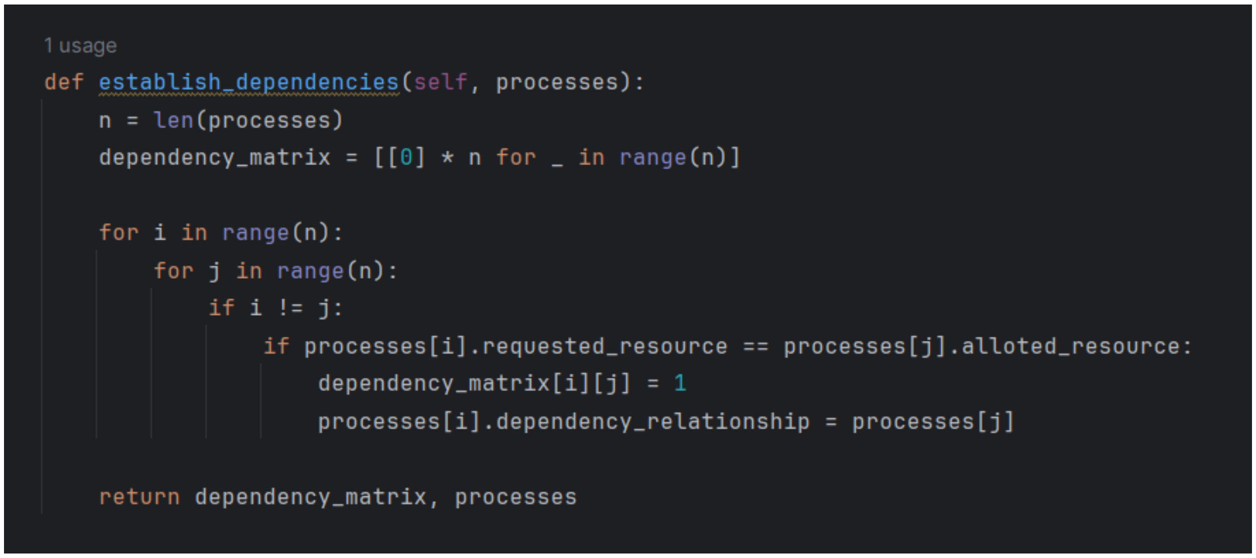
```
"E:\2.Online MS CS\2nd Year\repo\dev-cs\CPSC_503_OS\assignment_02\Process_sync\

Dependency Matrix:
     P1 P2 P3
P1   0 1 0
P2   0 0 1
P3   1 0 0

Cycles with last number:
1 -> 2 -> 3 -> 1
2 -> 3 -> 1 -> 2
3 -> 1 -> 2 -> 3

Cycles without last number:
1 -> 2 -> 3
2 -> 3 -> 1
3 -> 1 -> 2

Unique Cycles without redundancy is : [[1, 2, 3]]

Non redundant Cycles:
1 -> 2 -> 3

Breaking Cycles:
Processes for pre-emptions: [(1, 2, 1), (2, 3, 2), (3, 1, 3)]

Processs with dependencies after removing resource
Process 1 has alloted resource 3
Process 2 has alloted resource None
Process 3 has alloted resource 2
Selected process for pre-emption: 2

Process finished with exit code 0
```

Complexity Analysis

Assumption – number of processes and number of resources are equal

In the algorithm implemented for Deadlock Management Module a large cost arises when we need to determine dependency graph. To be able to do that in the implementation, each process's requested resource(m) is checked against another process's allotted resource(n) to see if they depend on each other

In the code below the nested loops iterate over all pairs of processes. For each pair, it performs a comparison and a potential assignment for dependency matrix.



```python
1 usage
def establish_dependencies(self, processes):
    n = len(processes)
    dependency_matrix = [[0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            if i != j:
                if processes[i].requested_resource == processes[j].alloted_resource:
                    dependency_matrix[i][j] = 1
                    processes[i].dependency_relationship = processes[j]

    return dependency_matrix, processes
```

The code consists of outer loop for processes with requested resources and inner loop for processes with allotted resource

Outer loop complexity is $O(n)$

Inner loop complexity is $O(n)$

Because above loops are nested loops iterate over all pairs of processes

Combined loop complexity is $O(n^2)$

Looking at other complexities

Initialization of n is $O(1)$

Initialization of dependency_matrix is $O(n^2)$

Summing up all complexities, the overall time complexity is: $O(n^2)$, This means that the method scales quadratically with the number of processes.

Conclusion

The ***establish_dependencies*** method has a time complexity of $O(n^2)$. This is because it involves creating an $n\,X\,n$ matrix and iterating over all pairs of processes to check for dependencies and establish relationships.

Note: because we assumed that number of processes is equal to number of resources the complexity is $O(n^2)$, however when number of resources is m and number of processes is n , $m \approx n$ for complexity to be $O(n^2)$