Assignment 4

Assumptions:

1. It is assumed that within the given network there is exactly one node which is already in the Critical Section.
2. The nodes are numbered from A to Z up to n terms which is given by the user.
3. The number of nodes wanting to enter into the Critical Section is given by the user. The nodes are however chosen randomly.

Source Code:

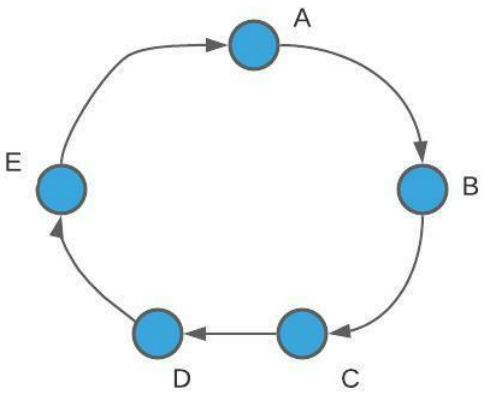
```
import random

# Represents a particular node in the network
class Node:
    def __init__(self, name: str):
        self.name: str = name

    def enterCriticalSection(self):
        print(f"In Critical Section of Node {self.name}")

def nameConvert(node_index):
        return node_list[node_index].name

if(__name__ == "__main__"):
    # Get number of nodes from user
    try:
        nos_nodes = int(input("Enter number of nodes: "))
        nos_nodes_critical = int(input("Enter Number of Nodes wanting to enter
Critical Section: "))
    except:
        print("Invalid Input")
        exit()

    # Check for general correctness
    if(nos_nodes < 2):
        print("Invalid Input")
        exit()
    elif(nos_nodes_critical >= nos_nodes or nos_nodes_critical < 0):
        print("Invalid Input")
        exit()

    # Generate the nodes in the network
    node_list: "list[Node]" = []
    for i in range(nos_nodes):
        new_node = Node(chr(ord("A") + i))
        node_list.append(new_node)
```

```
    # Randomly generate a starting node
    critical_node = random.randint(0, (nos_nodes - 1))
    print("\nNode already in Critical Section:",
node_list[critical_node].name)

    # Randomly generate the required nodes wanting to enter critical section
    print("\nRequest Phase:")
    nodes_critical_section: "list[int]" = []
    counter: int = 0
    while(True):
        node = random.randint(0, (nos_nodes - 1))
        # Accept only those nodes which is not the initial critical section
node and has not been chosen before
        if(node != critical_node and node not in nodes_critical_section):
            nodes_critical_section.append(node)
            counter += 1

            # Display the flow of message in the network
            first_iter = True
            while(node != critical_node):
                if(first_iter == True):
                    print(f"Requesting Node {node_list[node].name}")
                    first_iter = False
                else:
                    print(f"Request with Node {node_list[node].name}")
                node = (node + 1) % nos_nodes
            print(f"Request with Node {node_list[node].name} and queued")

        # If the required number of nodes has been generated, break
        if(counter == nos_nodes_critical):
            break

    print(f"\nNodes willing to enter Critical Section: {list(map(nameConvert,
nodes_critical_section))}\n")

    # Enter the node which is already sitting in critical section into the
queue
    nodes_critical_section.insert(0, critical_node)

    print("Execution Phase:")
    token = critical_node
    while True:
        print(f"Token with Node {node_list[token].name}")

        # If the node is eligible to enter the critical section, then do so
        # Once the node has been found out, remove its entry from the queue
```

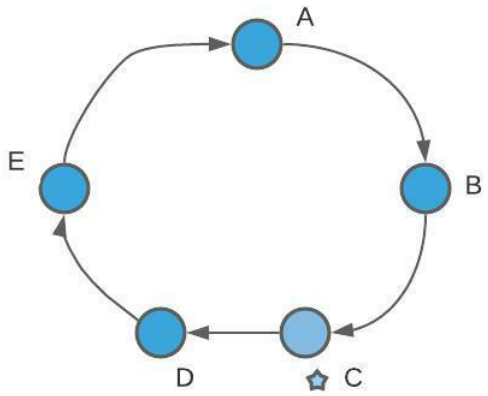
```
        if(node_list[token].name ==
node_list[nodes_critical_section[0]].name):
            nodes_critical_section.pop(0)
            node_list[token].enterCriticalSection()

        # Move token to next node
        token = (token + 1) % nos_nodes

        print(f"Status of queue at node {node_list[token].name}:
{list(map(nameConvert, nodes_critical_section))}\n")

        # If the queue is empty, it denotes that no other nodes are willing to
enter critical section.
        # In that case, stop the program
        if(len(nodes_critical_section) == 0):
            break
```

Dataset Used

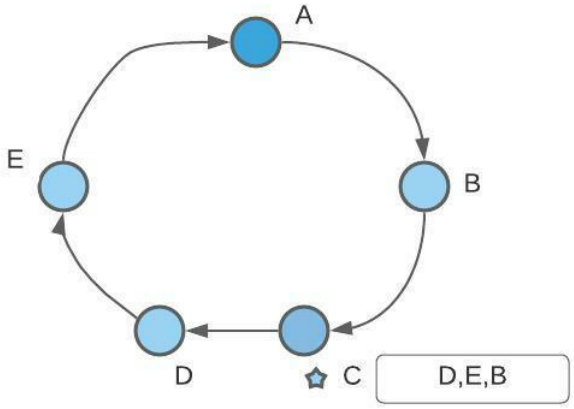
Initial Network as given by user



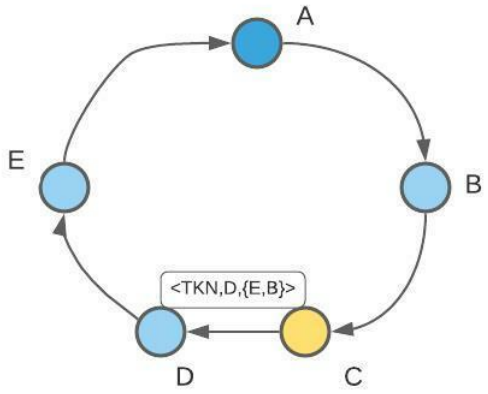
Node C is in critical section



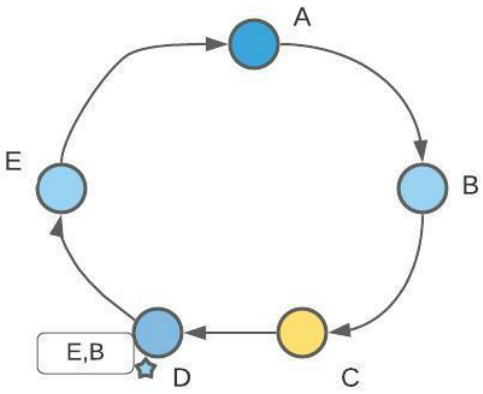
Nodes D, E, B want to enter the critical section. So, they pass their node number with the token until it gets queued.



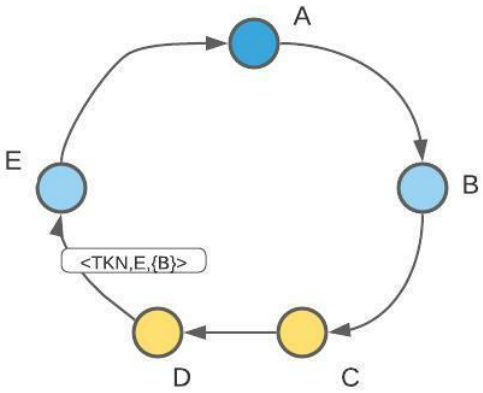
Node C comes out of the critical section. It passes the token to node D along with the list of Queued nodes.



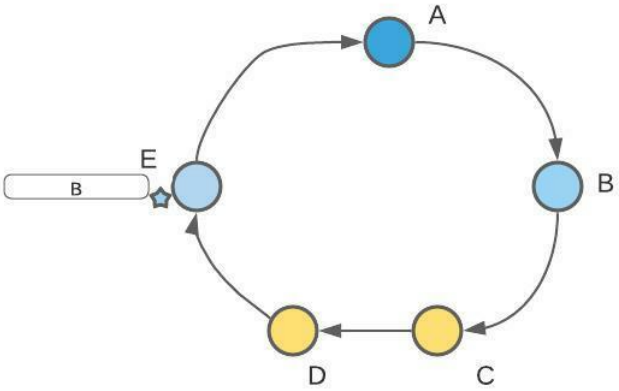
Now, node D enters the critical section



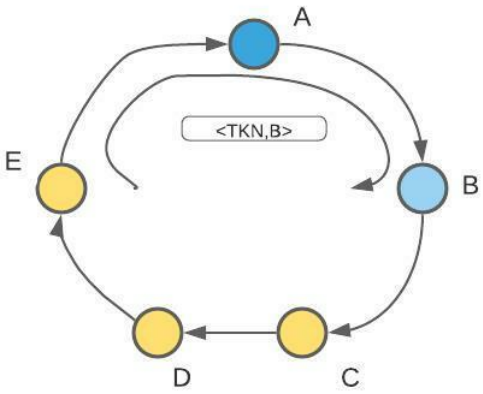
Node D comes out of the critical section. It passes the token to node E along with the list of Queued nodes.



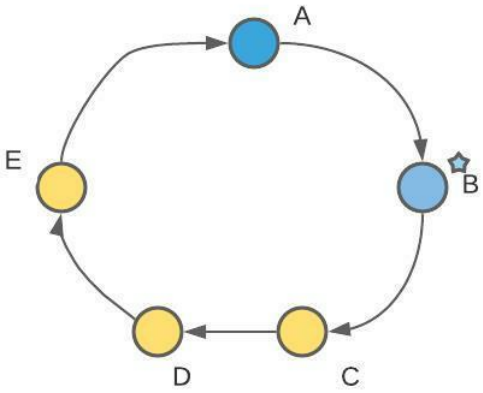
Now, node E enters the critical section



Node E comes out of the critical section. It passes the token to node A which in turn passes the same to node B as it does not wish to enter into Critical Section.



Finally, node B enters the critical section.



After the execution of the Critical Section is completed by the node B, it holds the token as no further nodes request for the token.

Input-Output:

```
PS C:\Users\debal\Documents\Assignments\msc-sem3-AOS\Assignment4> python .\TokenRingModified.py
Enter number of nodes: 5
Enter Number of Nodes wanting to enter Critical Section: 3

Node already in Critical Section: C

Request Phase:
Requesting Node D
Request with Node E
Request with Node A
Request with Node B
Request with Node C and queued
Requesting Node E
Request with Node A
Request with Node B
Request with Node C and queued
Requesting Node B
Request with Node C and queued

Nodes willing to enter Critical Section: ['D', 'E', 'B']

Execution Phase:
Token with Node C
In Critical Section of Node C
Status of queue at node D: ['D', 'E', 'B']

Token with Node D
In Critical Section of Node D
Status of queue at node E: ['E', 'B']

Token with Node E
In Critical Section of Node E
Status of queue at node A: ['B']

Token with Node A
Status of queue at node B: ['B']

Token with Node B
In Critical Section of Node B
Status of queue at node C: []

PS C:\Users\debal\Documents\Assignments\msc-sem3-AOS\Assignment4>
```