

Data Structures 2023-2

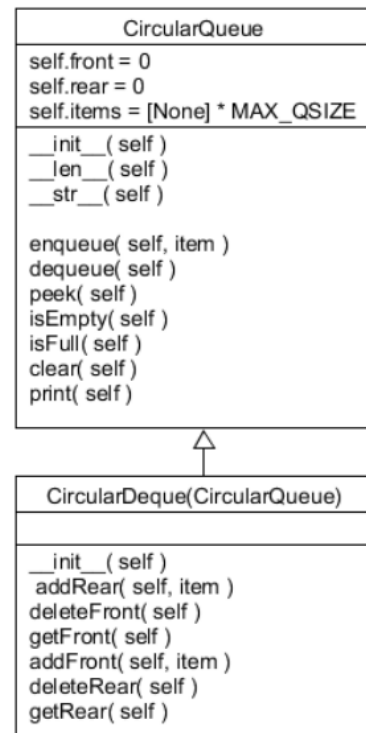
Lab 04: Queue Abstract Data Type

1. Task-1: Implement Deque Data Structure

Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends. Write code for Deque abstract data type and test it.

Deque Operations: The following basic operations are performed on the deque:

- ✓ addFront(): Adds an item at the front of Deque.
- ✓ addRear(): Adds an item at the rear of Deque.
- ✓ deleteFront(): Deletes an item from front of Deque.
- ✓ deleteRear(): Deletes an item from rear of Deque.
- ✓ getFront(): Gets the front item from queue.
- ✓ getRear(): Gets the last item from queue.
- ✓ isEmpty(): Checks whether Deque is empty or not.
- ✓ isFull(): Checks whether Deque is full or not.



CircularQueue Code

```
1 MAX_QSIZE = 10
2
3 class CircularQueue:
4
5     def __init__(self):
6         self.front = 0
7         self.rear = 0
8         self.items = [None] * MAX_QSIZE
9
10    def isEmpty(self):
11        return self.front == self.rear
12
13    def isFull(self):
14        return self.front == (self.rear + 1) % MAX_QSIZE
15
16    def clear(self):
17        self.front = self.rear
18
19    def __len__(self):
20        return (self.rear - self.front + MAX_QSIZE) % MAX_QSIZE
21
22    def enqueue(self, item):
23        if not self.isFull():
24            self.rear = (self.rear + 1) % MAX_QSIZE
25            self.items[self.rear] = item
26
27    def dequeue(self):
28        if not self.isEmpty():
29            self.front = (self.front + 1) % MAX_QSIZE
30            return self.items[self.front]
31
32    def peek(self):
33        if not self.isEmpty():
34            return self.items[(self.front + 1) % MAX_QSIZE]
35
36    def print(self):
37        out = []
38        if self.front < self.rear:
39            out = self.items[self.front + 1 : self.rear + 1]
40        else:
41            out = self.items[self.front + 1 : MAX_QSIZE] \
42                + self.items[0 : self.rear + 1]
43
44        print(f"[f={self.front}, r={self.rear} ==> {out}]")
```

16

CircularDeque Code

```
46 class CircularDeque(CircularQueue):
47     def __init__(self):
48         super().__init__()
49
50     def addRear(self, item):
51         self.enqueue(item)
52
53     def deleteFront(self):
54         return self.dequeue()
55
56     def getFront(self):
57         return self.peek()
58
59     def addFront(self, item):
60         if not self.isFull():
61             self.items[self.front] = item
62             self.front = (self.front - 1 + MAX_QSIZE) % MAX_QSIZE
63
64     def deleteRear(self):
65         if not self.isEmpty():
66             item = self.items[self.rear]
67             self.rear = (self.rear - 1 + MAX_QSIZE) % MAX_QSIZE
68             return item
69
70     def getRear(self):
71         return self.items[self.rear]
```

Results/Output

[Test code]

```
TestLab04.py x Stack.py x TCS.py x Maze.py x Media...
1 from queueADT import *
2 from TCS import *
3 from MediaPlayer import *
4 from Maze import *
5
6 def testCircularDeque():
7     print("Deque Test")
8     q = CircularDeque()
9     for i in range(10):
10         q.enqueue(i)
11     print("\tenqueue()x9 : ", end="")
12     q.print()
13     print("\t\tdequeue()-->", q.deleteFront())
14     print("\t\t\tdequeue()-->", q.deleteFront())
15     print("\t\t\tdequeue()-->", q.deleteFront())
16     print("\t\t\tdequeue()-->", q.deleteRear())
17     print("\t\t\tdequeue()-->", q.deleteRear())
18     print("\t\t\tdequeue()x5", end='')
19     q.print()
20
21     q.clear()
22     q.enqueue('aaa')
23     q.enqueue('bbb')
24     q.enqueue('ccc')
25     q.enqueue('ddd')
26     print("\t\t\tenqueue()x4: ", end="")
27     q.print()
28     print("\t\t\tdequeue()-->", q.deleteRear())
29     print("\t\t\tdequeue()x9 ", end="")
30     q.print()
31     print("\t\t\tpeek()-->", q.peek())
32     print("\n")
33
34 def testCircularQueue():
35     print('Test Queue')
36     q = CircularQueue()
37     for i in range(10):
38         q.enqueue(i)
39
40     print('\tenqueue()x9: ', end='')
41     q.print()
42     print('\t\t\tdequeue()-->', q.dequeue())
43     print('\t\t\tdequeue()-->', q.dequeue())
44     print('\t\t\tdequeue()-->', q.dequeue())
45     print('\t\t\tdequeue()x3', end='')
46
47     q.clear()
48     print()
49     q.enqueue('aaa')
50     q.enqueue('bbb')
51     q.enqueue('ccc')
52     q.enqueue('ddd')
53     print('\tenqueue()x4: ', end='')
54     q.print()
55     print('\t\t\tdequeue()-->', q.dequeue())
56     print('\t\t\tdequeue()x9', end='')
57     q.print()
58     print("\t\t\tpeek()-->", q.peek())
59     print("\n")
60
61 def main():
62     testCircularQueue()
63     testCircularDeque()
64     #runSimulation()
65     #testMPQ()
66     #m = Maze()
67     #m.DFS1()
68     #m.BFS2()
69
70 if __name__ == '__main__':
71     main()
```

[Test Queue and Test Deque]

```
TestLab04 (1) x
C:\Users\iqeq1\anaconda3\envs\data mining\python.exe "C:\4-2\Data Structure\Lab04\TestLab04.py"
Test Queue
enqueue()x9: [f=0, r=9 ==> [0, 1, 2, 3, 4, 5, 6, 7, 8]
dequeue()--> 0
dequeue()--> 1
dequeue()--> 2
dequeue()x3
enqueue()x4: [f=9, r=3 ==> ['aaa', 'bbb', 'ccc', 'ddd']
dequeue()--> aaa
dequeue()x9[f=0, r=3 ==> ['bbb', 'ccc', 'ddd']
peek()--> bbb

Deque Test
enqueue()x9 : [f=0, r=9 ==> [0, 1, 2, 3, 4, 5, 6, 7, 8]
dequeue()--> 0
dequeue()--> 1
dequeue()--> 2
dequeue()--> 8
dequeue()--> 7
dequeue()x5[f=3, r=7 ==> [3, 4, 5, 6]
enqueue()x4: [f=7, r=1 ==> ['aaa', 'bbb', 'ccc', 'ddd']
dequeue()--> ddd
dequeue()x9 [f=7, r=0 ==> ['aaa', 'bbb', 'ccc']
peek()--> aaa
```

2. Task2: Ticketing Counter system (Simulation)

A computer simulation can be developed to model this Ticketing Counter system using the Queue data structure.

An object-oriented solution with multiple classes.

- ✓ CircularQueue : Data structure to hold the passengers.
- ✓ Passenger : store info related to a passenger.
- ✓ TicketAgent : store info related to an agent.
- ✓ TicketCounterSimulation : manages the actual simulation.



CircularQueue Code : Data structure to hold the passengers

```
py x Stack.py x TCS.py x Maze.py x MediaPlayer.py x queueADT.py x
1 MAX_QSIZE = 10
2 class CircularQueue:
3     def __init__(self):
4         self.front = 0
5         self.rear = 0
6         self.items = [None] * MAX_QSIZE
7
8     def isEmpty(self):
9         return self.front == self.rear
10
11    def isFull(self):
12        return self.front == (self.rear + 1) % MAX_QSIZE
13
14    def clear(self):
15        self.front = self.rear
16
17    def __len__(self):
18        return (self.rear - self.front + MAX_QSIZE) % MAX_QSIZE
19
20    def enqueue(self, item):
21        if not self.isFull():
22            self.rear = (self.rear + 1) % MAX_QSIZE
23            self.items[self.rear] = item
24
25    def dequeue(self):
26        if not self.isEmpty():
27            self.front = (self.front + 1) % MAX_QSIZE
28            return self.items[self.front]
29
30    def peek(self):
31        if not self.isEmpty():
32            return self.items[(self.front + 1) % MAX_QSIZE]
33
34    def print(self):
35        out = []
36        if self.front < self.rear:
37            out = self.items[self.front + 1 : self.rear + 1]
38        else:
39            out = self.items[self.front + 1 : MAX_QSIZE] \
40                + self.items[0 : self.rear + 1]
41        print(f"[f={self.front}, r={self.rear}] ==> {out}")
```

passenger Code : store info related to a passenger.

```
TestLab04.py x Stack.py x TCS.py x Maze.py x MediaPlayer.py x
1  from queueADT import *
2  from random import randint
3
4  class passenger:
5      def __init__(self, pID, ArrivalTime):
6          self._pID = pID
7          self._arrivalTime = ArrivalTime
8
9          # Return id Number
10     def getPID(self):
11         return self._pID
12
13     # Return Arrival Time
14     def timeArrived(self):
15         return self._arrivalTime
```

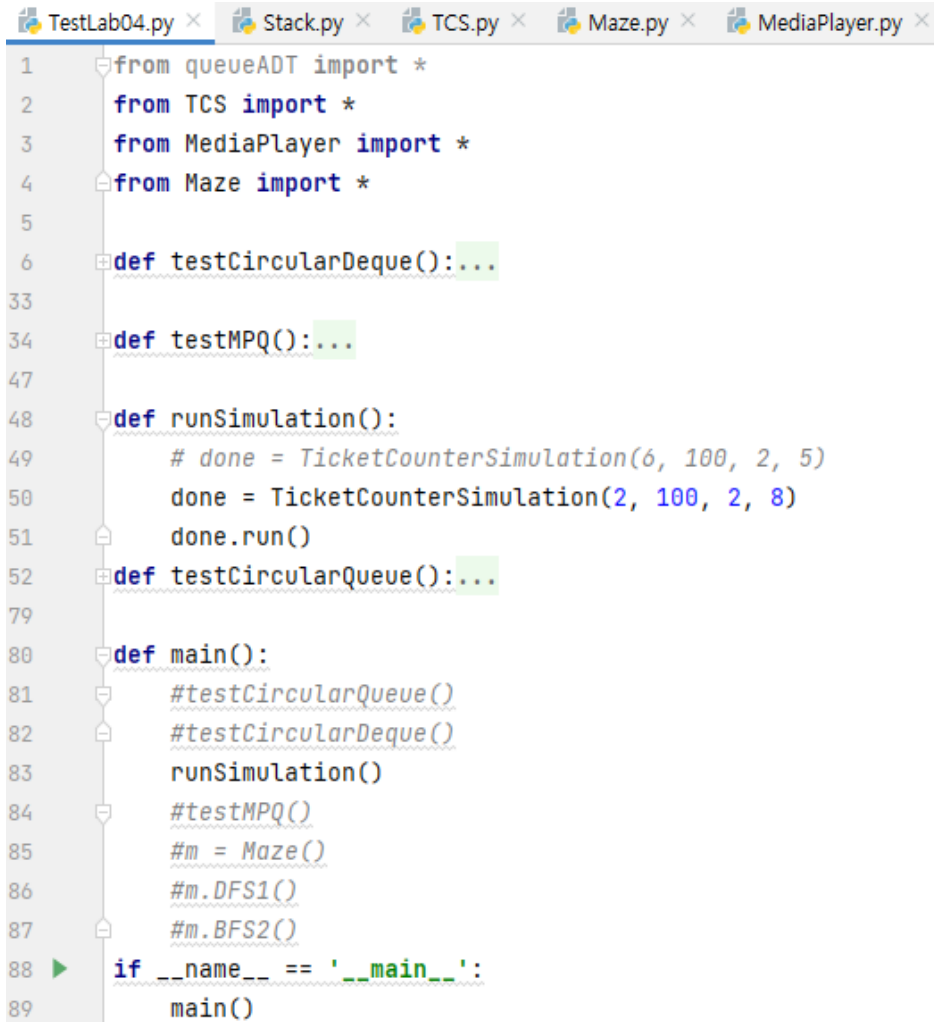
TicketAgent Code : store info related to an agent.

```
17 class TicketAgent:
18     def __init__(self, aID):
19         self._aID = aID
20         self._passenger = None
21         self._stopTime = -1
22         # Return Id Number
23     def getAID(self):
24         return self._aID
25
26     # Determine if Agent is Free
27     def isFree(self):
28         return self._passenger is None
29
30     # Determine if Agent has finished a Service
31     def isFinished(self, curTime):
32         return self._passenger is not None and curTime == self._stopTime
33
34
35     # Start Attending to a Passenger
36     def startService(self, passenger, stopTime):
37         self._passenger = passenger
38         self._stopTime = stopTime
39
40     # Stop Service to a Passenger
41     def stopService(self):
42         thepassenger = self._passenger
43         self._passenger = None
44         return thepassenger
```

✓ **TicketCounterSimulation Code** : manages the actual simulation.

```
47 class TicketCounterSimulation:
48     # Create a simulation object.
49     def __init__(self, numAgents, numMinutes, betweenTime, serviceTime):
50         #Parameters supplied by the user
51         self._arriveprob = 1.0 / betweenTime
52         self._serviceTime = serviceTime
53         self._numMinutes = numMinutes
54         self.served = 0
55
56         # Simulation components.
57         self._passengers = CircularQueue()
58         self._Agents = [None] * numAgents
59         for i in range(numAgents):
60             self._Agents[i] = TicketAgent(i+1)
61
62         # Computed during the simulation.
63         self._totalWaitTime = 0
64         self._numPassengers = 0
65
66     # Run the simulation using the parameters supplied
67     def run(self):
68         for curTime in range(self._numMinutes + 1):
69             self._handleArrival ( curTime)
70             self._handleBeginService( curTime )
71             self._handleEndService( curTime )
72         self.printResult()
73
74
75     def printResult(self):
76         numServed = self._numPassengers - len(self._passengers)
77         avgwait = float(self._totalWaitTime) / numServed
78         print("")
79         print(f"Number or passengers served = {numServed}")
80         print(f"Number of passengers remaining in line = {len(self._passengers)}")
81         print(f"The average wait time was {avgwait :.2f} minutes.")
82
83
84     # Handle Customer Arrival
85     def _handleArrival(self, curTime):
86         prob = randint(0,0, 1.0)
87         if 0.0 <= prob <= self._arriveprob:
88             person = passenger(self._numPassengers + 1, curTime)
89             self._passengers.enqueue( person )
90             self._numPassengers += 1
91             print( f"Time {curTime} : Passenger {person.getPID()} arrived.")
92
93
94     # Begin Customer Service
95     def _handleBeginService(self, curTime):
96
97         i = 0
98         while i < len(self._Agents):
99             if self._Agents[i].isFree() and not self._passengers.isEmpty() and curTime != self._numMinutes:
100                 passenger = self._passengers.dequeue()
101                 self.served += 1
102                 stoptime = curTime + self._serviceTime
103                 self._Agents[i].startService(passenger, stoptime)
104                 self._totalWaitTime += (curTime - passenger.timeArrived())
105                 print(f"Time {curTime} : Agent {self._Agents[i].getAID()} started serving passenger {passenger.getPID()}")
106                 i += 1
107
108
109     # Stop Customer Service
110     def _handleEndService(self, curTime):
111         i = 0
112         while i < len(self._Agents):
113             if self._Agents[i].isFinished(curTime):
114                 passenger = self._Agents[i].stopService()
115                 print(f"Time {curTime} : Agent {self._Agents[i].getAID()} stopped serving passenger {passenger.getPID()}")
116                 i += 1
117
118
```

Results/Output Code



```
1 from queueADT import *
2 from TCS import *
3 from MediaPlayer import *
4 from Maze import *
5
6 def testCircularDeque():...
33
34 def testMPQ():...
47
48 def runSimulation():
49     # done = TicketCounterSimulation(6, 100, 2, 5)
50     done = TicketCounterSimulation(2, 100, 2, 8)
51     done.run()
52 def testCircularQueue():...
79
80 def main():
81     #testCircularQueue()
82     #testCircularDeque()
83     runSimulation()
84     #testMPQ()
85     #m = Maze()
86     #m.DFS1()
87     #m.BFS2()
88 if __name__ == '__main__':
89     main()
```

Results/Output

```
TestLab04 (1) × C:\Users\iqeq1\anaconda3\envs\datamining\python.exe
Time 3 : Passenger 1 arrived.
Time 3 : Agent 1 started serving passenger 1
Time 6 : Passenger 2 arrived.
Time 6 : Agent 2 started serving passenger 2
Time 7 : Passenger 3 arrived.
Time 8 : Passenger 4 arrived.
Time 10 : Passenger 5 arrived.
Time 11 : Passenger 6 arrived.
Time 11 : Agent 1 stopped serving passenger 1
Time 12 : Agent 1 started serving passenger 3
Time 13 : Passenger 7 arrived.
Time 14 : Agent 2 stopped serving passenger 2
Time 15 : Passenger 8 arrived.
Time 15 : Agent 2 started serving passenger 4
Time 16 : Passenger 9 arrived.
Time 17 : Passenger 10 arrived.
Time 19 : Passenger 11 arrived.
Time 20 : Agent 1 stopped serving passenger 3
Time 21 : Passenger 12 arrived.
Time 21 : Agent 1 started serving passenger 5
Time 22 : Passenger 13 arrived.
Time 23 : Passenger 14 arrived.
Time 23 : Agent 2 stopped serving passenger 4
Time 24 : Passenger 15 arrived.
Time 24 : Agent 2 started serving passenger 6
Time 27 : Passenger 16 arrived.
Time 28 : Passenger 17 arrived.
Time 29 : Agent 1 stopped serving passenger 5
Time 30 : Passenger 18 arrived.
Time 30 : Agent 1 started serving passenger 7
Time 31 : Passenger 19 arrived.
Time 32 : Agent 2 stopped serving passenger 6
Time 33 : Agent 2 started serving passenger 8
Time 35 : Passenger 20 arrived.
Time 36 : Passenger 21 arrived.
Time 37 : Passenger 22 arrived.
Time 38 : Agent 1 stopped serving passenger 7
Time 39 : Passenger 23 arrived.
Time 39 : Agent 1 started serving passenger 9
Time 40 : Passenger 24 arrived.
Time 41 : Passenger 25 arrived.
Time 41 : Agent 2 started serving passenger 8
Time 42 : Agent 2 started serving passenger 10
Time 43 : Passenger 26 arrived.
Time 44 : Passenger 27 arrived.
Time 46 : Passenger 28 arrived.
Time 47 : Passenger 29 arrived.
Time 47 : Agent 1 stopped serving passenger 9
Time 48 : Passenger 30 arrived.
Time 48 : Agent 1 started serving passenger 11
Time 49 : Passenger 31 arrived.
Time 50 : Passenger 32 arrived.
Time 50 : Agent 2 stopped serving passenger 10
Time 51 : Passenger 33 arrived.
Time 51 : Agent 2 started serving passenger 12
Time 55 : Passenger 34 arrived.
Time 56 : Agent 1 stopped serving passenger 11
Time 57 : Agent 1 started serving passenger 13
Time 58 : Passenger 35 arrived.
Time 59 : Agent 2 stopped serving passenger 12
Time 60 : Passenger 36 arrived.
Time 60 : Agent 2 started serving passenger 14
Time 61 : Passenger 37 arrived.
Time 63 : Passenger 38 arrived.
Time 65 : Passenger 39 arrived.
Time 65 : Agent 1 stopped serving passenger 13
Time 66 : Agent 1 started serving passenger 16
Time 68 : Agent 2 stopped serving passenger 14
Time 69 : Passenger 40 arrived.
Time 69 : Agent 2 started serving passenger 19
Time 70 : Passenger 41 arrived.
Time 71 : Passenger 42 arrived.
Time 72 : Passenger 43 arrived.
Time 74 : Agent 1 stopped serving passenger 16
Time 75 : Agent 1 started serving passenger 20
Time 77 : Passenger 44 arrived.
Time 77 : Agent 2 stopped serving passenger 19
Time 78 : Passenger 45 arrived.
Time 78 : Agent 2 started serving passenger 24
Time 79 : Passenger 46 arrived.
Time 80 : Passenger 47 arrived.
Time 81 : Passenger 48 arrived.
Time 83 : Agent 1 stopped serving passenger 20
Time 84 : Agent 1 started serving passenger 26
Time 86 : Agent 2 stopped serving passenger 24
Time 87 : Agent 2 started serving passenger 31
Time 89 : Passenger 49 arrived.
Time 90 : Passenger 50 arrived.
Time 92 : Passenger 51 arrived.
Time 92 : Agent 1 stopped serving passenger 26
Time 93 : Passenger 52 arrived.
Time 93 : Agent 1 started serving passenger 34
Time 94 : Passenger 53 arrived.
Time 95 : Passenger 54 arrived.
Time 95 : Agent 2 stopped serving passenger 31
Time 96 : Passenger 55 arrived.
Time 96 : Agent 2 started serving passenger 35

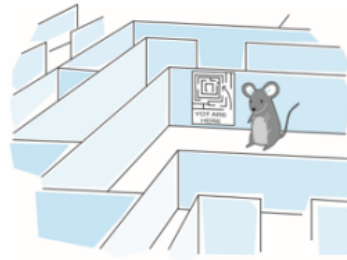
Number of passengers served = 47
Number of passengers remaining in line = 8
The average wait time was 11.91 minutes.
```


3. Task-3: Solve the Maze problem through DFS and BFS using different data structures

A Maze is given as $N \times N$ binary matrix of blocks where source block is `maze[0][0]` and destination block is `maze[N-1][N-1]`.

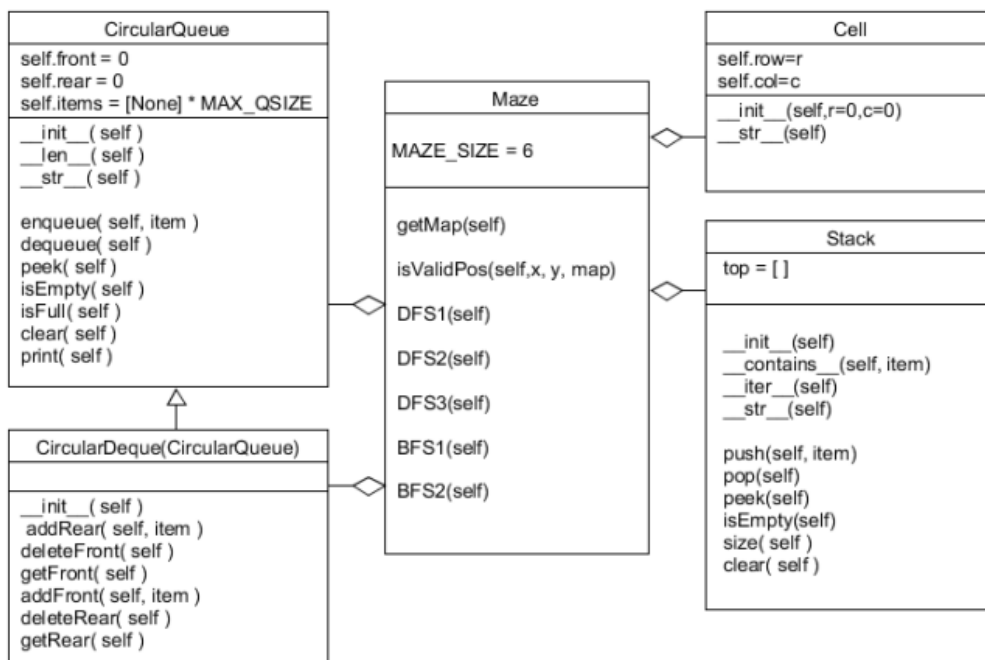
- ✓ A rat starts from the source and has to reach the destination. The rat can move only in four directions: upward, down, forward, and back word.
- ✓ In the maze matrix, 1 means the block is a dead end and 0 means the block can be used in the path from source to destination
- ✓ The task is to check if there exists any path so that the rat can reach at the destination or not.

```
char maze[N][N] = {
{ 'e', '1', '1', '1', '1', '1' },
{ '0', '0', '1', '0', '0', '1' },
{ '1', '0', '0', '0', '1', '1' },
{ '1', '0', '1', '0', '1', '1' },
{ '1', '0', '1', '0', '0', '1' },
{ '1', '1', '1', '1', '0', '1' },
};
```



Write the following functions in Maze class

- ✓ DFS1() : It searches maze using stack from CircularDeque (front end operations)
- ✓ DFS2() : It searches maze using Stack
- ✓ DFS1() : It searches maze using stack from CircularDeque (rear end operation)
- ✓ BFS1() : It searches maze using Queue from CircularDeque
- ✓ BFS2() : It searches maze using Queue from CircularQueue



CircularQueue Code

```
1 MAX_QSIZE = 10
2
3 class CircularQueue:
4
5     def __init__(self):
6         self.front = 0
7         self.rear = 0
8         self.items = [None] * MAX_QSIZE
9
10    def isEmpty(self):
11        return self.front == self.rear
12
13    def isFull(self):
14        return self.front == (self.rear + 1) % MAX_QSIZE
15
16    def clear(self):
17        self.front = self.rear
18
19    def __len__(self):
20        return (self.rear - self.front + MAX_QSIZE) % MAX_QSIZE
21
22    def enqueue(self, item):
23        if not self.isFull():
24            self.rear = (self.rear + 1) % MAX_QSIZE
25            self.items[self.rear] = item
26
27    def dequeue(self):
28        if not self.isEmpty():
29            self.front = (self.front + 1) % MAX_QSIZE
30            return self.items[self.front]
31
32    def peek(self):
33        if not self.isEmpty():
34            return self.items[(self.front + 1) % MAX_QSIZE]
35
36    def print(self):
37        out = []
38        if self.front < self.rear:
39            out = self.items[self.front + 1 : self.rear + 1]
40        else:
41            out = self.items[self.front + 1 : MAX_QSIZE] \
42                  + self.items[0 : self.rear + 1]
43
44        print(f"[f={self.front}, r={self.rear} ==> {out}]")
```

16

CircularDeque Code

```
46 class CircularDeque(CircularQueue):
47     def __init__(self):
48         super().__init__()
49
50     def addRear(self, item):
51         self.enqueue(item)
52
53     def deleteFront(self):
54         return self.dequeue()
55
56     def getFront(self):
57         return self.peek()
58
59     def addFront(self, item):
60         if not self.isFull():
61             self.items[self.front] = item
62             self.front = (self.front - 1 + MAX_QSIZE) % MAX_QSIZE
63
64     def deleteRear(self):
65         if not self.isEmpty():
66             item = self.items[self.rear]
67             self.rear = (self.rear - 1 + MAX_QSIZE) % MAX_QSIZE
68             return item
69
70     def getRear(self):
71         return self.items[self.rear]
```

Cell Code

```
TestLab04.py x Stack.py x TCS.py x Maze.py x MediaPlayer.py x
1 from queueADT import *
2 from Stack import *
3 class Cell:
4     def __init__(self, r, c):
5         self.row = r
6         self.col = c
7     def __str__(self):
8         return '(' + str(self.row) + ", " + str(self.col) + ")
```

Stack Code

```
TestLab04.py x Stack.py x TCS.py x Maze.py x MediaPlayer.py x
1 class Stack:
2     def __init__(self):
3         self.top = []
4     def __str__(self):
5         #return str(self.top[::-1])
6         return str(self.top)
7     def __iter__(self):
8         return self
9     def __len__(self):
10        return len(self.top)
11    def __contains__(self, item):
12        return item in self.top
13    def push(self, item):
14        self.top.append(item)
15    def pop(self):
16        if not self.isEmpty():
17            return self.top.pop()
18        else:
19            print("Stack is Empty...")
20            exit()
21    def peek(self):
22        if not self.isEmpty():
23            return self.top[-1]
24        else:
25            print("Stack is Empty...")
26            exit()
27
28    def size(self):
29        return len(self.top)
30    def display(self):
31        str(self.top[:])
32    def isEmpty(self):
33        return len(self.top) == 0
34    def clear(self):
35        self.top=[]
```

Maze Code

```
9 class Maze:
10     MAZE_SIZE = 6
11     def getMap(self):
12         _map = [ ['1','1','1','1','1','1'],
13                 ['e','0','1','0','0','1'],
14                 ['1','0','0','0','1','1'],
15                 ['1','0','1','0','1','1'],
16                 ['1','0','1','0','0','x'],
17                 ['1','1','1','1','1','1']]
18         return _map
19
20     def isValidPos(self, x, y, _map):
21         if (x < 0 or y < 0 or x >= self.MAZE_SIZE or y >= self.MAZE_SIZE):
22             return False
23         else:
24             return _map[y][x] == '0' or _map[y][x] == 'x'
25
26     def DFS1(self):
27         _map = self.getMap()
28         deq = CircularDeque()
29         entry = Cell(0,1)
30         deq.addFront(entry)
31         print("\nDFS1: Using Deque Data Structure : ")
32
33         while not deq.isEmpty():
34             here = deq.deleteFront()
35             print(here, end=">")
36             x = here.row
37             y = here.col
38             if (_map[y][x] == 'x') : return True
39             else:
40                 _map[y][x] = '.'
41                 if self.isValidPos(x-1, y, _map) : deq.addFront(Cell(x-1, y))
42                 if self.isValidPos(x+1, y, _map) : deq.addFront(Cell(x+1, y))
43                 if self.isValidPos(x, y-1, _map) : deq.addFront(Cell(x, y-1))
44                 if self.isValidPos(x, y+1, _map) : deq.addFront(Cell(x, y+1))
45         return False
46
47     def DFS2(self):
48         _map = self.getMap()
49         s = Stack()
50         entry = Cell(0,1)
51         s.push(entry)
52         print("\nDFS2 : using Stack Data Structure : ")
53
54         while (s.isEmpty() == False):
55             here = s.pop()
56             print(here, end=">")
57             x = here.row
58             y = here.col
59             if (_map[y][x] == 'x') : return True
60             else:
61                 _map[y][x] = '.'
62                 if self.isValidPos(x - 1, y, _map) : s.push(Cell(x - 1, y))
63                 if self.isValidPos(x + 1, y, _map) : s.push(Cell(x + 1, y))
64                 if self.isValidPos(x, y - 1, _map) : s.push(Cell(x, y - 1))
65                 if self.isValidPos(x, y + 1, _map) : s.push(Cell(x, y + 1))
66         return False
```

```

68 def DFS3(self):
69     _map = self.getMap()
70     deq = CircularDeque()
71     entry = Cell(0, 1)
72     deq.addRear(entry)
73     print("\nDFS3: Using Deque Data Structure : ")
74
75     while not deq.isEmpty():
76         here = deq.deleteRear()
77         print(here, end=">")
78         x = here.row
79         y = here.col
80         if _map[y][x] == 'x': return True
81         else:
82             _map[y][x] = '.'
83             if self.isValidPos(x - 1, y, _map): deq.addRear(Cell(x - 1, y))
84             if self.isValidPos(x + 1, y, _map): deq.addRear(Cell(x + 1, y))
85             if self.isValidPos(x, y - 1, _map): deq.addRear(Cell(x, y - 1))
86             if self.isValidPos(x, y + 1, _map): deq.addRear(Cell(x, y + 1))
87     return False
88
89 def BFS1(self):
90     _map = self.getMap()
91     deq = CircularDeque()
92     entry = Cell(0, 1)
93     deq.addRear(entry)
94     print("\nBFS1 : using Deque Data Structure: ")
95     while not deq.isEmpty():
96         here = deq.deleteFront()
97         print(here, end=">")
98         x = here.row
99         y = here.col
100        if _map[y][x] == 'x':
101            return True
102        else:
103            _map[y][x] = '.'
104            if self.isValidPos(x - 1, y, _map): deq.addRear(Cell(x - 1, y))
105            if self.isValidPos(x + 1, y, _map): deq.addRear(Cell(x + 1, y))
106            if self.isValidPos(x, y - 1, _map): deq.addRear(Cell(x, y - 1))
107            if self.isValidPos(x, y + 1, _map): deq.addRear(Cell(x, y + 1))
108    return False
109
110 def BFS2(self):
111     _map = self.getMap()
112     que = CircularQueue()
113     entry = Cell(1, 0)
114     que.enqueue(entry)
115     print("\nBFS2 : using Queue Data Structure: ")
116
117     while not que.isEmpty():
118         here = que.dequeue()
119         print(here, end=">")
120         x = here.row
121         y = here.col
122         if _map[y][x] == 'x':
123             return True
124         else:
125             _map[y][x] = '.'
126             if self.isValidPos(x, y - 1, _map): que.enqueue(Cell(x, y - 1))
127             if self.isValidPos(x, y + 1, _map): que.enqueue(Cell(x, y + 1))
128             if self.isValidPos(x - 1, y, _map): que.enqueue(Cell(x - 1, y))
129             if self.isValidPos(x + 1, y, _map): que.enqueue(Cell(x + 1, y))
130    return False

```

Results/Output Code

```
TestLab04.py x Stack.py x TCS.py x Maze.py x MediaPlayer.py x
1 from queueADT import *
2 from TCS import *
3 from MediaPlayer import *
4 from Maze import *

80 def main():
81     #testCircularQueue()
82     #testCircularDeque()
83     #runSimulation()
84     # testMPQ()
85     m = Maze()
86     m.DFS1()
87     m.DFS2()
88     m.DFS3()
89     m.BFS1()
90     m.BFS2()
91 if __name__ == '__main__':
92     main()
```

Results/Output

```
TestLab04 (1) x
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab04\TestLab04.py"

DFS1: Using Deque Data Structure :
(0, 1)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(2, 2)->(3, 2)->(3, 3)->(3, 4)->(4, 4)->(5, 4)->
DFS2 : using Stack Data Structure :
(0, 1)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(2, 2)->(3, 2)->(3, 3)->(3, 4)->(4, 4)->(5, 4)->
DFS3: Using Deque Data Structure :
(0, 1)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(2, 2)->(3, 2)->(3, 3)->(3, 4)->(4, 4)->(5, 4)->
BFS1 : using Deque Data Structure:
(0, 1)->(1, 1)->(1, 2)->(2, 2)->(1, 3)->(3, 2)->(1, 4)->(3, 1)->(3, 3)->(4, 1)->(3, 4)->(4, 4)->(5, 4)->
BFS2 : using Queue Data Structure:
(1, 0)->(1, 1)->(1, 2)->(1, 3)->(2, 2)->(1, 4)->(3, 2)->(3, 1)->(3, 3)->(4, 1)->(3, 4)->(4, 4)->(5, 4)->
```

4. Conclusion

Conclude the Lab. Write your views about it, i.e. what have you learned from this lab? It was helpful or difficult etc

Through this task, I was able to learn the Queue and Deque data structure. He was also able to learn how to create a ticketing center system using him. And I was able to learn how to implement DFS and BFS by comprehensively using the data structures I have learned so far.

It was an opportunity to realize once again that Python is a really good tool in implementing the data structure.