K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Data Structure

ENCS205

School of Engineering & Technology (SOET)
K.R. MANGALAM University

Session 28: **Queues: A Revision**

Session 28 Outlook

- Recap of Queue basics.
- Numerical problems for Linear Queue operations.
- Numerical problems for Circular Queue operations.
- Q&A and Summarization.

Recap of Queues

- **Definition:** A linear data structure with FIFO order.
- **Analogy:** Like a ticket counter queue.
- **Order:** Front = deletion end, Rear = insertion end.

Key Characteristics:

- Abstract Data Type (ADT): Defined by its behavior.
- Restricted Access: Elements are added at one end (Rear/Tail) and removed from the other end (Front/Head).

Linear Queue Operations

- **Enqueue (Insertion):** Adds an element to the rear of the queue.
  - If the queue is full, it's an Overflow condition.
- **Dequeue (Deletion):** Removes an element from the front of the queue.
  - If the queue is empty, it's an Underflow condition.
- **Front**: Points to the first element in the queue.
- **Rear:** Points to the last element in the queue.
- **Limitation:** After several Dequeue operations, even if space becomes available at the front, new elements can only be added at the rear, leading to "queue full" even if the array isn't entirely used.

Numerical Problem: Linear Queue Trace

Perform the following sequence of operations on an initially empty linear queue with a maximum size of 5. Show the queue's contents and the Front and Rear pointers after each operation.

1. ENQUEUE(10)
2. ENQUEUE(20)
3. DEQUEUE()
4. ENQUEUE(30)
5. ENQUEUE(40)
6. DEQUEUE()
7. ENQUEUE(50)
8. ENQUEUE(60)
9. DEQUEUE()

Numerical Problem: Linear Queue Trace

Solution

Operation	Queue Contents (Front to Rear)	Front	Rear	Notes
Initially Empty	[ ]	-1	-1	
ENQUEUE(10)	[10]	0	0	
ENQUEUE(20)	[10, 20]	0	1	
DEQUEUE()	[20]	1	1	10 removed
ENQUEUE(30)	[20, 30]	1	2	
ENQUEUE(40)	[20, 30, 40]	1	3	
DEQUEUE()	[30, 40]	2	3	20 removed
ENQUEUE(50)	[30, 40, 50]	2	4	
ENQUEUE(60)	**Queue Full**	2	4	Cannot add 60. Array size is 5 (indices 0-4).
DEQUEUE()	[40, 50]	3	4	30 removed

Circular Queue Operations

- **Enqueue:** Adds to the rear.

  Rear = (Rear + 1) % MaxSize.

- **Dequeue**: Removes from the front.

  Front = (Front + 1) % MaxSize.

- **Conditions:**
  - **Empty:** Front = -1 (or Front = Rear depending on implementation).
  - **Full**: (Rear + 1) % MaxSize == Front.

Circular Queue Numerical problem

- Perform the following sequence of operations on an initially empty circular queue with a maximum size of 5. Show the queue's contents and the Front and Rear pointers after each operation. Assume Front = -1 and Rear = -1 for an empty queue.
  1. ENQUEUE(A)
  2. ENQUEUE(B)
  3. DEQUEUE()
  4. ENQUEUE(C)
  5. ENQUEUE(D)
  6. ENQUEUE(E)
  7. DEQUEUE()
  8. ENQUEUE(F)
  9. ENQUEUE(G)

Circular Queue Numerical problem

Solution:

Operation	Queue Contents	Front	Rear	Notes
Initially Empty	[, , , ,]	-1	-1	
ENQUEUE(A)	[**A**, , , ,]	0	0	
ENQUEUE(B)	[**A**, **B**, , ,]	0	1	
DEQUEUE()	[, **B**, , ,]	1	1	'A' removed.

Circular Queue Numerical problem

Operation	Queue Contents	Front	Rear	Notes
ENQUEUE(D)	[, **B**, **C**, **D**,]	1	3	
ENQUEUE(E)	[, **B**, **C**, **D**, **E**]	1	4	
DEQUEUE()	[, , **C**, **D**, **E**]	2	4	'B' removed.
ENQUEUE(F)	[**F**, , **C**, **D**, **E**]	2	0	'F' added at index 0 (circular wrap-around, Rear = (4+1)%5 = 0).
ENQUEUE(G)	**Queue Full** (B,C,D,E,F -> 5 elements)	2	0	Cannot add 'G'. (Rear + 1) % MaxSize == Front ((0+1)%5 == 1 which is not 2). Let's recheck full condition. If (Rear+1)%MaxSize == Front, it is full. Here (0+1)%5 = 1, and Front is 2. So not full.

Queue Practice Question

Question 1. Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. What are conditions to detect queue full and queue empty .

Solution:

Explanation: Suppose we start filling the queue.

Let the max QueueSize (Capacity of the Queue) is 4.So the size of the array which is used to implement this circular queue is 5, which is n. In the beginning when the queue is empty, FRONT and REAR point to 0 index in the array. REAR represents insertion at the REAR index. FRONT represents deletion from the FRONT index.

enqueue(“a”); REAR = (REAR+1)%5; (FRONT = 0, REAR = 1)

enqueue(“b”); REAR = (REAR+1)%5; (FRONT = 0, REAR = 2)

enqueue(“c”); REAR = (REAR+1)%5; (FRONT = 0, REAR = 3)

enqueue(“d”); REAR = (REAR+1)%5; (FRONT = 0, REAR = 4)

Now the queue size is 4 which is equal to the maxQueueSize. Hence overflow condition is reached.

Now, we can check for the conditions.

When Queue Full :

(REAR+1)%n = (4+1)%5 = 0

FRONT is also 0. Hence (REAR + 1) %n is equal to FRONT.

When Queue Empty :

REAR was equal to FRONT when empty (because in the starting before filling the queue FRONT = REAR = 0)

Queue

Question 2:

Implement a circular queue using an array with a capacity of 5 elements. Perform enqueue operations with values 2, 4, 6, and 8. What will be the front and rear indices of the queue after these operations?

Solutions :-

To implement a circular queue using an array with a capacity of 5 elements and perform the specified operations, let's follow these steps:

1. **Initialize the circular queue:**
   - **Create an array of size 5.**
   - **Initialize front and rear indices to -1 to indicate that the queue is initially empty.**
2. **Enqueue operations:**
   - **Enqueue 2**
   - **Enqueue 4**
   - **Enqueue 6**
   - **Enqueue 8**

Solutions :-

Circular Queue Initialization

Array: [, , , ,] // Initial empty array of size 5Front: -1Rear: -1

Enqueue Operation 1: Enqueue 2

- **Since the queue is initially empty (front is -1), set front to 0.**
- **Increment rear to 0 and place the value at index 0.makefile**

- Array: [2, , , ,] Front: 0 Rear: 0

Solutions :-

Enqueue Operation 2: Enqueue 4Increment rear to 1 and place the value at index 1.
Array: [2, 4, , ,] Front: 0 Rear: 1

Enqueue Operation 3: Enqueue 6Increment rear to 2 and place the value at index 2.
Array: [2, 4, 6, ,]Front: 0Rear: 2

Enqueue Operation 4: Enqueue 8Increment rear to 3 and place the value at index 3.
Array: [2, 4, 6, 8,] Front: 0 Rear: 3

Final State of the Circular Queue
After performing all the enqueue operations, the circular queue looks like this:
Array: [2, 4, 6, 8,] Front: 0 Rear: 3

Queue Applications

- Operating Systems:
  - CPU Scheduling
  - Printer Spooling
  - Disk Scheduling
- Networking:
  - Packet Buffering
  - Traffic Management:
- Data Processing:
  - Asynchronous Data Transfer
  - Breadth-First Search (BFS): Graph traversal algorithm.
- Simulation


THANK
YOU