Circular Queue Implementation

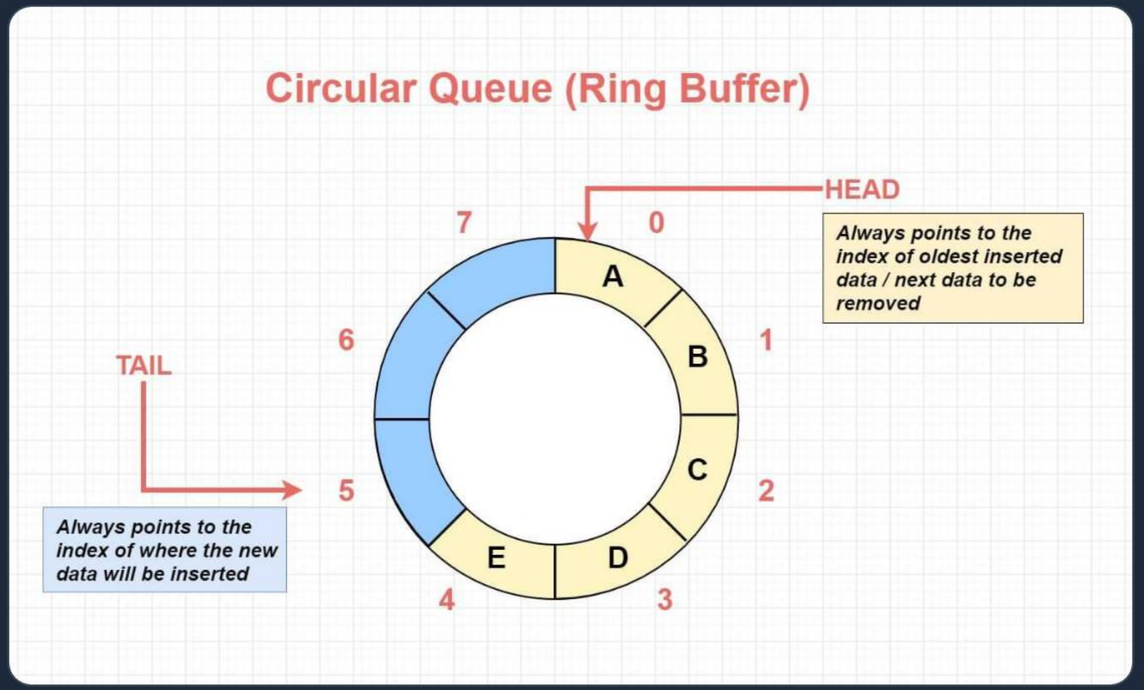
Logic, Algorithms, and Java Code

What is a Circular Queue?

A **Circular Queue** is a linear data structure that follows the FIFO (First In, First Out) principle.

Unlike a standard queue, the last position is connected back to the first position to make a circle.

> 💡 **Key Benefit:** It solves the problem of unutilized space in a linear queue by reusing empty blocks.



1. Initialization (Constructor)

Capacity

Store the maximum size `k` and initialize an integer array of size `k`.

Pointers

Initialize `front = 0` and `rear = -1` to track the start and end of the queue.

Size

Initialize `size = 0` to keep track of the current number of elements.

2. Enqueue Operation (Insert)

- **Step 1:** Check if the queue is full. If `size == capacity`, return false.
- **Step 2:** Move the `rear` pointer circularly using the modulo operator.
- **Step 3:** Insert the new value at `queue[rear]`.
- **Step 4:** Increment the `size` counter.

Logic Formula

Circular Indexing:

```
rear = (rear + 1) % capacity
```

This ensures the index wraps around to 0 after reaching the last index.

3. Dequeue Operation (Remove)

Logic Formula

Circular Indexing:

```
front = (front + 1) % capacity
```

Moves the front pointer forward, wrapping to 0 if needed.

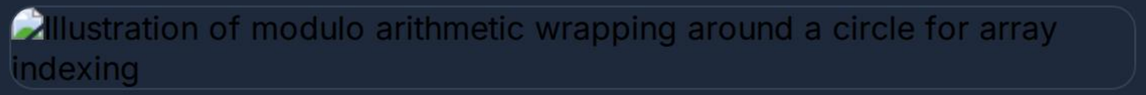
- **Step 1:** Check if the queue is empty. If `size == 0`, return false.
- **Step 2:** Move the `front` pointer circularly.
- **Step 3:** Decrement the `size` counter.
- **Step 4:** Return true to indicate successful removal.

Visualizing Modulo Arithmetic

Why use Modulo (%)?

In a circular queue, we need the index to go back to `0` after it reaches `capacity - 1`.

The operation `index % capacity` achieves this mathematically.



- If capacity is 5 and current index is 4:
- Next index = (4 + 1) % 5 = **0**
- This creates the "infinite loop" effect.

Helper Operations

Front()

Returns the element at `queue[front]`.

Returns -1 if empty.

Rear()

Returns the element at `queue[rear]`.

Returns -1 if empty.

isEmpty()

Checks if the queue is empty.

```
return size == 0;
```

isFull()

Checks if the queue is full.

```
return size == capacity;
```

Java Implementation

We define a class `CircularQueue` with an array and pointers.

```
class CircularQueue {
    int[] queue;
    int front, rear, size, capacity;

    // Constructor
    CircularQueue(int k) {
        capacity = k;
        queue = new int[k];
        front = 0;
        rear = -1;
        size = 0;
    }
}
```

The constructor sets up the initial state with a fixed capacity.

Code: Enqueue Method

```
boolean enqueue(int value) {
    if (isFull())
        return false;

    rear = (rear + 1) % capacity;
    queue[rear] = value;
    size++;
    return true;
}
```

Code Explanation

- Check Full: Prevents overflow.
- Update Rear: Uses modulo to wrap around.
- Insert: Places value at the new rear index.
- Size++: Updates the count.

Code: Dequeue Method

Code Explanation

- Check Empty: Prevents underflow.
- Update Front: Moves head pointer forward, wrapping if needed.
- Size--: Updates the count.
- Note: We don't need to "delete" the data; moving the pointer is enough.

```
boolean dequeue() {
    if (isEmpty())
        return false;

    front = (front + 1) % capacity;
    size--;
    return true;
}
```

Testing the Implementation

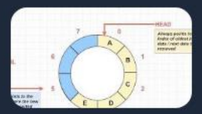

```
> Output:
Front Element: 10
Rear Element: 40
```

Q & A

Any questions about the Circular Queue logic or the Java implementation?

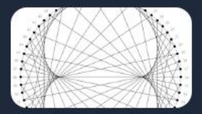
Next Topic: Linked List Implementation of Queues

Image Sources



https://www.sahinarslan.tech/static/63103e976b608cdcdfcc43424724cddb/f704e/circular-queue-anatomy.jpg

Source: www.sahinarslan.tech



https://i.sstatic.net/b9l86.png

Source: math.stackexchange.com

https://img.freepik.com/premium-photo/java-programming-code-abstract-technology-background_272306-149.jpg

Source: www.freepik.com