To implement the concatenate(LinkedQueue Q2) method for the LinkedQueue class, the goal is to append all the elements of queue Q2 to the end of the original queue (this), and do so in constant time, O(1).

To achieve this efficiently, we can:

1. **Directly link the last element of Q2 to the rear of the original queue (this)**.
2. **Update the rear pointer of the original queue** to point to the last element of Q2.
3. **Empty Q2** by setting its front and rear to null and its size to 0.

This approach works because a linked queue uses linked nodes, where each node points to the next node, and we can directly manipulate the pointers without having to traverse the queues.

**Code Implementation**

public class LinkedQueue<T> {

private static class Node<T> {

T data;

Node<T> next;

Node(T data) {

this.data = data;

this.next = null;

}

}

private Node<T> front; // Points to the front node of the queue

private Node<T> rear; // Points to the rear node of the queue

private int size; // Size of the queue

public LinkedQueue() {

this.front = null;

this.rear = null;

this.size = 0;

}

public boolean isEmpty() {

return size == 0;

}

public void enqueue(T item) {

Node<T> newNode = new Node<>(item);

if (isEmpty()) {

front = newNode;

rear = newNode;

} else {

rear.next = newNode;

rear = newNode;

}

size++;

}

public T dequeue() {

if (isEmpty()) throw new IllegalStateException("Queue is empty");

T item = front.data;

front = front.next;

size--;

if (isEmpty()) {

rear = null; // If the queue becomes empty, rear should also be null

}

return item;

}

public int size() {

return size;

}

// Concatenate method: append all elements of Q2 to this queue, and empty Q2

public void concatenate(LinkedQueue<T> Q2) {

if (Q2.isEmpty()) return; // If Q2 is empty, no need to do anything

if (this.isEmpty()) {

// If the original queue is empty, just set this queue to Q2

this.front = Q2.front;

this.rear = Q2.rear;

} else {

// Otherwise, append the elements of Q2 to the rear of this queue

this.rear.next = Q2.front;

this.rear = Q2.rear;

}

// Clear Q2 to make it empty

Q2.front = null;

Q2.rear = null;

Q2.size = 0;

}

}

**Explanation of concatenate(LinkedQueue<T> Q2):**

1. **Check if Q2 is Empty**: If Q2 is empty, there’s nothing to concatenate, so we simply return from the method.
2. **If the Original Queue (this) is Empty**: If the original queue is empty, we directly set its front and rear to the front and rear of Q2. This effectively moves all elements of Q2 into the original queue.
3. **Otherwise (both queues have elements)**: We append the elements of Q2 to the rear of the original queue by setting the next pointer of the original queue's rear node to point to Q2's front node. We then update the original queue's rear to point to Q2's rear.
4. **Empty Q2**: After the elements are concatenated, we clear the front, rear, and size of Q2, effectively making it an empty queue.

**Time and Space Complexity:**

* **Time Complexity**: The concatenate() operation runs in constant time, O(1), because we are only modifying pointers (no traversal or copying of elements). Updating the rear pointers and clearing Q2 takes constant time.
* **Space Complexity**: The space complexity is O(1) because we are not allocating any additional space or making copies of the elements in Q2. We are only changing the pointers.

**Usage Example:**

LinkedQueue<Integer> queue1 = new LinkedQueue<>();

queue1.enqueue(1);

queue1.enqueue(2);

LinkedQueue<Integer> queue2 = new LinkedQueue<>();

queue2.enqueue(3);

queue2.enqueue(4);

queue1.concatenate(queue2); // queue1 should now contain [1, 2, 3, 4], queue2 is empty

System.out.println("queue1 size: " + queue1.size()); // 4

System.out.println("queue2 size: " + queue2.size()); // 0