Documentation

The problem requires implementing a stack (Last-In-First-Out, LIFO) using only queue operations, specifically using two queues. A stack allows operations such as push (adding an element to the top of the stack), pop (removing and returning the aspect from the top), top (retrieving the top element without removing it), and empty (checking if the stack is empty). Since a queue operates in a First-In-First-Out (FIFO) manner, the challenge is to manipulate the queue operations to simulate the LIFO behavior of a stack.

The stack uses two queues, queue1 and queue2. The main idea is to ensure that the most recently pushed element always appears at the front of queue1, so it can be easily accessed for pop and top operations. When making an element, we first add it to queue2. Then, all existing elements from queue1 are moved to queue2, ensuring that the most recently pushed element is at the front. After that, the roles of queue1 and queue2 are swapped, making queue1 the main queue containing elements in the correct order for stack operations. This ensures that subsequent pop or top operations can access the last pushed element first, mimicking a stack's behavior.

The pop operation removes the front element of queue1, which represents the stack's top element. Similarly, the top operation retrieves this element without removing it. The empty operation checks whether queue1 is empty, indicating whether the stack contains any elements.

This approach uses two queues to maintain the stack order by rearranging the elements during every push operation. The key insight is to manipulate the two queues in such a way that the last element pushed to the stack is always at the front of the queue, which allows efficient access to the top element for both pop and top operations.

This implementation leverages basic queue operations: appending to the back, removing from the front, and checking if the queue is empty. By swapping between the two queues during each push, we ensure that the stack maintains its LIFO nature.