Documentation

The MyQueue class implements a queue using only two stacks, achieving a first-in, first-out (FIFO) order with stack operations. This is accomplished by using two main stacks: stack_in and stack_out. The stack_in stack is used for all push operations, where elements are added to the end of the queue. The stack_out stack is utilized for pop and peek operations, storing elements in reverse order to simulate the behavior of removing elements from the front of the queue. This dual-stack approach provides a reliable way to achieve FIFO order using the LIFO (last-in, first-out) nature of stacks.

In the push operation, elements are simply added to the stack_in stack, which represents the end of the queue. Since no other actions are required, this operation is straightforward and efficient. When the pop or peek method is called, the function checks if stack_out is empty. If stack_out contains elements, they are processed directly; otherwise, elements from stack_in are transferred to stack_out in reverse order. This transfer reverses the order of elements, placing the oldest element on top of stack_out, which enables pop and peek operations to retrieve elements from the front of the queue, preserving the queue's FIFO behavior.

The pop operation first ensures that all elements in stack_in have been moved to stack_out if stack_out is empty. Then, it removes the top element of stack_out, effectively dequeuing the oldest element in the queue. Similarly, the peek operation checks the top component of stack_out, looking at the front of the queue without removing it. This process ensures that stack_out always has the oldest element on top, maintaining FIFO order in the queue. The use of two stacks is crucial here because transferring elements only when stack_out is empty minimizes the need for repeated transfers, optimizing the time complexity.

The empty method is a straightforward check to determine if the queue has any elements. By checking both stack_in and stack_out, this method confirms whether there are no elements in either stack, returning True if both are empty and False otherwise. This method provides an efficient way to determine the queue's emptiness without having to iterate through the elements. The helper function _transfer_in_to_out facilitates the movement of elements from stack_in to stack_out, handling this transition only when necessary, further enhancing the efficiency of the pop and peek operations.

Overall, this implementation provides an efficient way to implement a queue using only stack operations, adhering to the problem's constraints. The approach of using two stacks achieves an amortized O(1) time complexity for each operation, meaning that even though some operations, such as transferring elements, may take longer, the overall time complexity across multiple operations remains constant. This is particularly advantageous in scenarios where large numbers of operations need to be performed while preserving the FIFO order. The MyQueue class is an excellent example of how stack operations can be creatively applied to mimic queue behavior, providing a practical solution to implement a queue with stack constraints.