SQP4 Queue Implementation Using Stack

Wednesday, November 16, 2022 8:00 PM

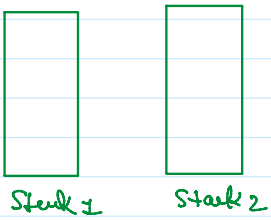
In this problem we learned How to Implement Queue using Stack data Structures.

→ Now we previously learn How to work Stack and Queue as well as its Implementation

→ How do we? let's see

Thought Process

→ Now Just previously we learn Same steps we? all steps also follow here only change one (position of the steps)



Steps :-

push(x)

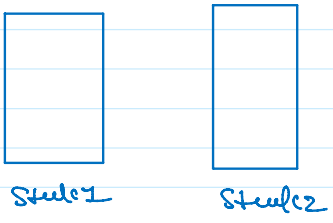
→ Stack1 → Stack2
→ x → Stack 1
→ Stack2 → Stack 1

pop(x)

Stack 1.pop()

Let's Understand

We required (2) Stack for Implementation

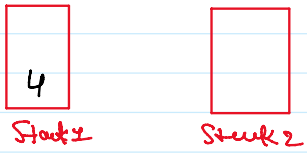


push(4)
push(3)
push(2)
push(5)
top()
pop()
top()

push(4)

→ Now follow the all Steps Initially both Stack are Empty So 1st Empty()
→ put (4) into the Stack 1
→ after put (x) into the Stack 1 put all the element Stack 2 to Stack 1

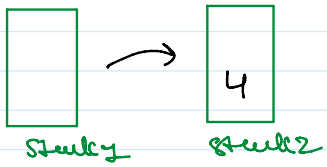
→ after push (x) into the stack 1 push all the element stack 2 to stack 1

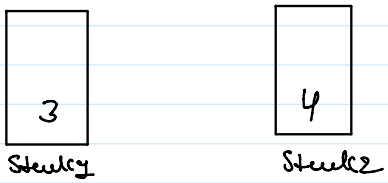


push(3)

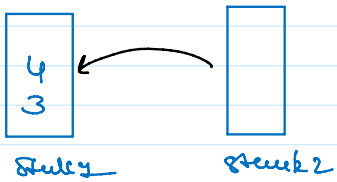
Same steps and processes follow again

↳ after put all the element into stack (2) push (x) into stack (1)





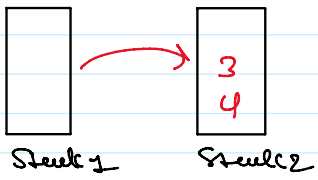
→ again put all the element of stack 2 into the stack 1



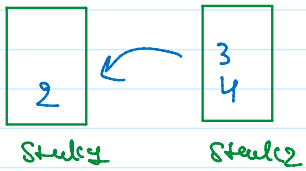
push(2)

Again follow same Approach and steps

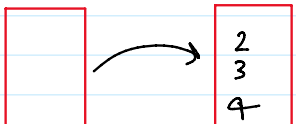
→ element by element push from the top of the stack 2

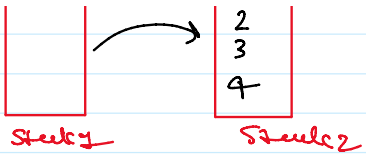


→ put x into the stack 1



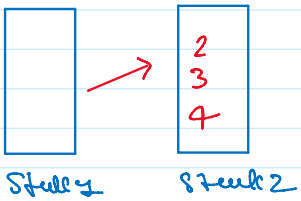
→ again put all the element from the stack 2 to stack 1



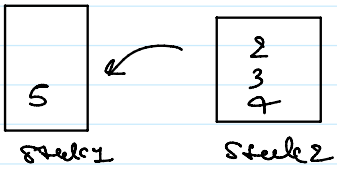


push(5)

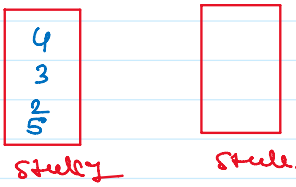
same process and steps follow again



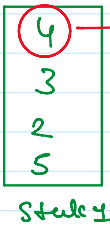
put the (5) into the stack 1



again put All the element of the stack 2 to stack 1



that is the way to how to push() Operation work



→ top element

Now Apply the FIFO concept on stack 1

Time Complexity = O(N)

→ in worst case all the Operation takes O(N) times

Space Complexity = O(N) + O(N) ≈ 2O(N)

→ because we used (2) different stack

Implementation (using 2 stack)

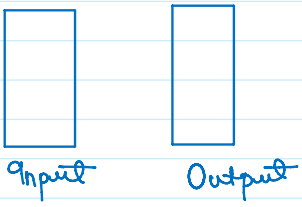
Implementation (using 2 stack)

```
# lc: https://leetcode.com/problems/implement-queue-using-stacks/
# gfg: https://practice.geeksforgeeks.org/problems/queue-using-stack/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article
# todo implement queue using 2 stack
class MyQueue:

    def __init__(self):

        self.stack1 = []
        self.stack2 = []

    def push(self, x):

        while self.stack1:
            self.stack2.append(self.stack1.pop())

        self.stack1.append(x)

        while self.stack2:
            self.stack1.append(self.stack2.pop())

    def pop(self) -> int:

        if self.stack1:
            return self.stack1.pop()
        return -1

    def peek(self) -> int:

        if self.stack1:
            return self.stack1[-1]
        return -1

    def empty(self) -> bool:
        return len(self.stack1) == 0

# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

→ Now these take many more space and time so we get optimized over Approach (again)

→ Now again we required ② stack but different Manner

push(2)
push(5)
push(3)
top()
pop()



Here we also used ② stack but different Approach
→ And reduced time complexity

top()
pop()
push(6)
pop()
pop()
top()

Input Output

Approach
→ And reduced time complexity

Steps :-
push(x)

→ add x → input

pop()

if output (Not Empty)
return output.pop()

else:
input → output
→ that means put all the element from input to output (element by element)

return output.pop()

→ Same process follow into the top() method

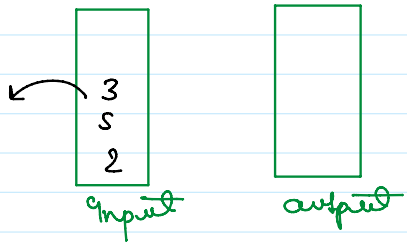
top()
Same process follow that we follow in pop() Operation

let's Understand

push(2) O(1)
push(5) O(1) → first put all the element into the input stack
push(3) O(1)

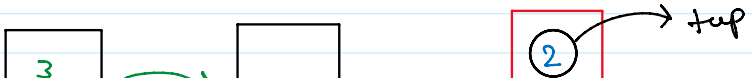
top() O(N)
pop() O(1)
push(6) O(1)
pop() O(1)
pop() O(1)
top() O(N)

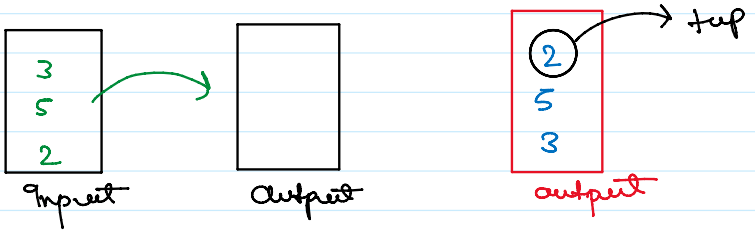
all push done



top()
→ top and pop all the Approach are the same So According to the steps

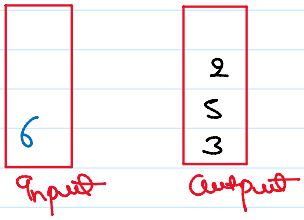
if output is Empty ✓
→ first all the element by element put into the output





top() → (2) after swapping Input to Output and same Approach follow top() and pop() both Options

push(6)
→ first we put the element into the Input



pop() or top()/peek()
we pop and top Operation perform still our Output became Empty

→ if once our output became Empty then again follow Same Approach/steps follow that we follow into the top() Operation

Time Complicity = O(1) + O(N) + O(1) ≈ O(1) (amortised Complicity)

- O(1) → for push Operation
- O(1) → for top and push Operation
- O(N) → for the top and pop if output is Empty
  - → for only that time our output is Empty()

Space Complicity =
O(N) + O(N) ≈ O(N)
→ using (2) different stack

Optimized Approach

```
# todo optimized Approch
class MyQueue:

    def __init__(self):

        self.input = []
        self.output = []
```

Implementation Java

```
class MyQueue {

    Stack<Integer>input;
    Stack<Integer>output;

    public MyQueue() {

        input = new Stack <Integer> ();
```

```
        self.input = []
        self.output = []

    def push(self, x):
        self.input.append(x)

    def pop(self):

        if len(self.output) == 0:
            while self.input:
                self.output.append(self.input.pop())

        return self.output.pop()

    def peek(self):

        if len(self.output) == 0:
            while self.input:
                self.output.append(self.input.pop())

        return self.output[-1]

    def empty(self):

        return len(self.output) + len(self.input) == 0

# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

```
    public MyQueue() {

        input = new Stack <Integer> ();
        output = new Stack <Integer> ();
    }

    public void push(int x) {

        input.push(x);

    }

    public int pop() {

        if(output.size() == 0){
            while(input.size() != 0){
                output.push(input.pop());
            }
        }

        return output.pop();

    }

    public int peek() {

        if(output.size() == 0){
            while(input.size() != 0){
                output.push(input.pop());
            }
        }

        return output.peek();

    }

    public boolean empty() {

        return input.size() + output.size() == 0;

    }
}

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue obj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * boolean param_4 = obj.empty();
 */
```