SQP3 Implementation Stack Using Queue

Saturday, November 12, 2022 5:35 PM

Now in this problem we implement stack using the Queue Data Structure

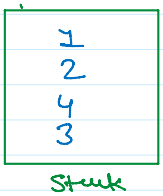
Stack → LIFO

Queue → FILO

→ If we talk about the stack

push(3)
push(4)
push(2)
push(1)

top() → 1
pop()
top() → 2

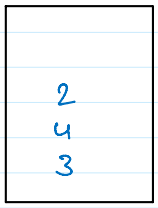


Stack work on the principle of (Up to down) approach

→ if we talk about the queue

push(3)
push(4)
push(2)

top() → 3
pop()
top() → 4



Queue work on the principle of (down to up Approach)

→ Now here the question is how to implement stack using queue data Structure let's understand

Thought Process

In Order to implement stack using Queue we required (2) queue data Structure

Steps :—

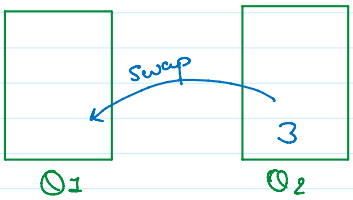
push(x)

→ Add x → Q_2
→ $Q_1 \rightarrow Q_2$ (element)
→ Swap ($Q_1 \rightleftharpoons Q_2$)

pop()

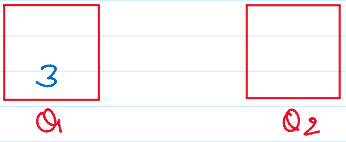
→ Remove the top element by Q_1

push(3) ← (x)
push(4)
push(2)
push(1)



push(3)

→ Now during the pushing element follows the steps that we learn previously



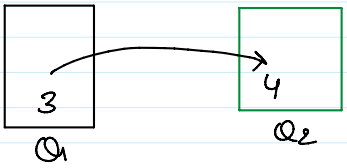
push(4)

→ again follow same process and same steps follow

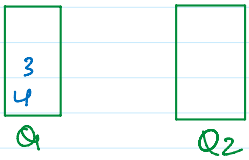
→ after putting all the element into the Q2

→ Now time to swap both queue to each other

$Q_1 \rightleftharpoons Q_2$



$Q_1 \rightarrow Q_2$ (element by element) put

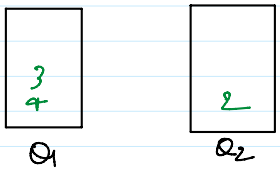


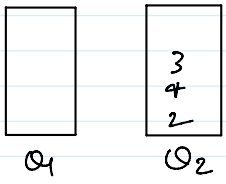
push(2)

→ again follow the same process and follow the same every step that we learn

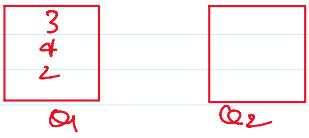
→ that steps we learned previously

→ Now moved the all element Q_1 to Q_2 (element by element)



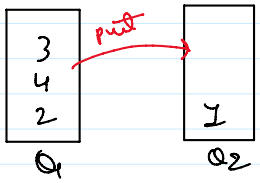


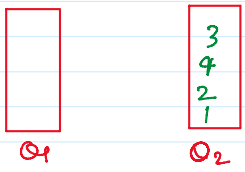
→ Swap the $Q_1 \rightleftharpoons Q_2$



push(1)

→ again same process follow just we do in previously ③ pushing process

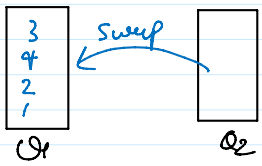




→ Now after moved all the element Q_1 to Q_2 time to swap to each other

Q_1 Q_2

→ Now after moved all the element Q_1 to Q_2 time to swap to each other



→ That is the way to How to working of push Operation using ② Queue data structure

top() → ①

that is the last element that we insert into the Queue So that is the current top() element

pop()

Removed the element from upward (↑) direction for pop() Operation

Time Complexity = O(N)

→ because push, pop can swap take at worst case O(N) time of operations

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

→ because I used ② Queue data structure for Implementation

Implementation

```
# todo Implement Stack using 2 Queues
# lc: https://leetcode.com/problems/implement-stack-using-queues/
from collections import deque

class MyStack:

    def __init__(self):

        self.queue1 = deque()
        self.queue2 = deque()

    def push(self, x):
        self.queue2.append(x)

        while len(self.queue1) != 0:
            self.queue2.append(self.queue1.popleft())

        self.queue1, self.queue2 = self.queue2, self.queue1

    def pop(self):
        if self.queue1:
            return self.queue1.popleft()
        return -1

    def top(self):
        if self.queue1:
            return self.queue1[0]
        return -1

    def empty(self):
        return len(self.queue1) == 0


ans = MyStack()
ans.push(3)
ans.push(4)
ans.push(2)
ans.push(1)

print(ans.top())
print(ans.pop())
print(ans.empty())
```

Useing 1 Queue

Now previously we used (2) different queues but here we used (1) queue data structure for more optimization

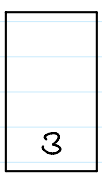
→ basically there is only one change and that change into the push() operation function

→ In Every push Operation we do

→ len(queue) - 1 all the elements pop again into the queue at the top of the (current pushed element)

Example

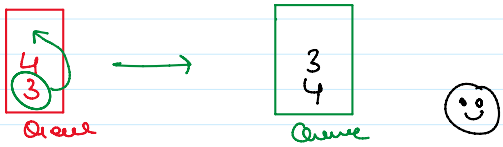
push(3)
push(4)
push(2)
push(1)



Que

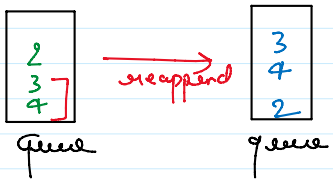
push(4)

→ Now again len(queue) - 1 element ko push its top of the current element



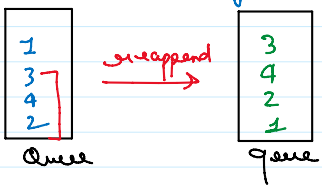
push(2)

→ push 2 into the Queue and len(queue) - 1 ko append into the top of the current element



push(1)

→ push 1 into the queue and again follow same Approach that we follow previously



that is the optimized way to we do this Implementation using only (1) Queue

Time Complexity = O(N)

→ linear Operation

Space Complexity = O(N)

→ because we used a only (1) queue data structure

Implementation (using 1 Queue)

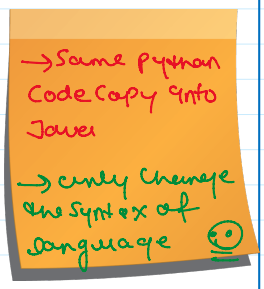
```
# todo Implement Stack using 1 Queues
class MyStack:

    def __init__(self):
        self.queue = deque()

    def push(self, x: int) -> None:
        self.queue.append(x)
        for i in range(len(self.queue) - 1):
            self.queue.append(self.queue.popleft())

    def pop(self) -> int:
        return self.queue.popleft()

    def top(self) -> int:
        return self.queue[0]

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

only push is change and remaining code will be same

Implementation Java (1 Queue)

```
//todo using 2 queue
class MyStack {

    Queue<Integer> q1;
    Queue<Integer> q2;

    public MyStack() {

        q1 = new LinkedList<Integer>();
        q2 = new LinkedList<Integer>();

    }

    public void push(int x) {

        q2.add(x);

        // Push all the remaining
        // elements in q1 to q2.
```

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

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




```
        q2.add(x);

        // Push all the remaining
        // elements in q1 to q2.
        while (!q1.isEmpty()) {
            q2.add(q1.peek());
            q1.remove();
        }

        // swap the names of two queues
        Queue<Integer> q = q1;
        q1 = q2;
        q2 = q;

    }

    public int pop() {

        if (q1.isEmpty()){
            return -1;
        }
        return q1.remove();

    }

    public int top() {

        if (q1.isEmpty()){
            return -1;
        }
        return q1.peek();

    }

    public boolean empty() {

        return q1.size() == 0;

    }
}

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

//todo using 1 queue
class AnotherMyStack {

    Queue<Integer> q;
    public MyStack() {

        q = new LinkedList<Integer>();
    }

    public void push(int x) {
        q.add(x);
        for(int i = 0;i < q.size()-1;i++){
            q.add(q.remove());
        }

    }

    public int pop() {

        if (q.isEmpty()){
            return -1;
        }
```

```
        }
        return q.remove();

    }

    public int top() {

        if (q.isEmpty()){
            return -1;
        }
        return q.peek();
    }

    public boolean empty() {

        return q.size() == 0;
    }
}
```