Previous Next

225. Implement Stack using Queues

April 13, 2016 | 132.7K views

Average Rating: 4.77 (102 votes)

Implement the following operations of a stack using queues.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

Example:

```
MyStack stack = new MyStack();

stack.push(1);
stack.push(2);
stack.top();   // returns 2
stack.pop();   // returns 2
stack.empty(); // returns false
```

Notes:

- You must use *only* standard operations of a queue -- which means only `push to back`, `peek/pop from front`, `size`, and `is empty` operations are valid.
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
- You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

Summary

This article is for beginners. It introduces the following ideas: Stack, Queue.

Solution

Approach #1 (Two Queues, push - $O(1)$, pop $O(n)$)

Intuition

Stack is **LIFO** (last in - first out) data structure, in which elements are added and removed from the same end, called `top`. In general stack is implemented using array or linked list, but in the current article we will review a different approach for implementing stack using queues. In contrast queue is **FIFO** (first in - first out) data structure, in which elements are added only from the one side - `rear` and removed from the other - `front`. In order to implement stack using queues, we need to maintain two queues `q1` and `q2`. Also we will keep top stack element in a constant memory.

Algorithm

Push

The new element is always added to the rear of queue `q1` and it is kept as `top` stack element

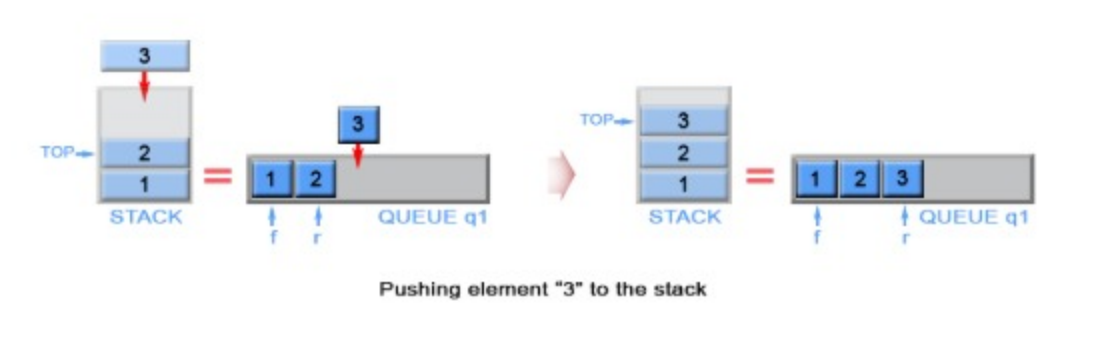

Figure 1. Push an element in stack

Java

```java
private Queue<Integer> q1 = new LinkedList<>();
private Queue<Integer> q2 = new LinkedList<>();
private int top;

// Push element x onto stack.
public void push(int x) {
    q1.add(x);
    top = x;
}
```

Complexity Analysis

- Time complexity : $O(1)$. Queue is implemented as linked list and `add` operation has $O(1)$ time complexity.
- Space complexity : $O(1)$

Pop

We need to remove the element from the top of the stack. This is the last inserted element in `q1`. Because queue is FIFO (first in - first out) data structure, the last inserted element could be removed only after all elements, except it, have been removed. For this reason we need to maintain additional queue `q2`, which will serve as a temporary storage to enqueue the removed elements from q1. The last inserted element in `q2` is kept as top. Then the algorithm removes the last element in `q1`. We swap `q1` with `q2` to avoid copying all elements from `q2` to `q1`.

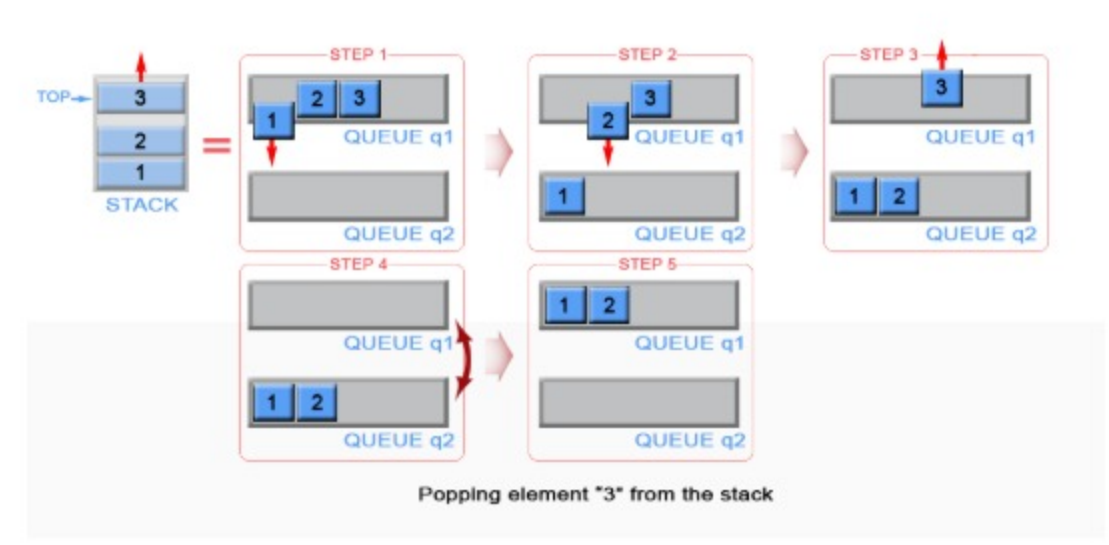

Figure 2. Pop an element from stack

Java

```java
// Removes the element on top of the stack.
public void pop() {
    while (q1.size() > 1) {
        top = q1.remove();
        q2.add(top);
    }
    q1.remove();
    Queue<Integer> temp = q1;
    q1 = q2;
    q2 = temp;
}
```

Complexity Analysis

- Time complexity : $O(n)$. The algorithm dequeues n elements from `q1` and enqueues $n-1$ elements to `q2`, where n is the stack size. This gives $2n-1$ operations.
- Space complexity : $O(1)$.

Approach #2 (Two Queues, push - $O(n)$, pop $O(1)$)

Algorithm

Push

The algorithm inserts each new element to queue `q2` and keep it as the `top` element. In case queue `q1` is not empty (there are elements in the stack), we remove all elements from `q1` and add them to `q2`. In this way the new inserted element (`top` element in the stack) will be always positioned at the front of `q2`. We swap `q1` with `q2` to avoid copying all elements from `q2` to `q1`.

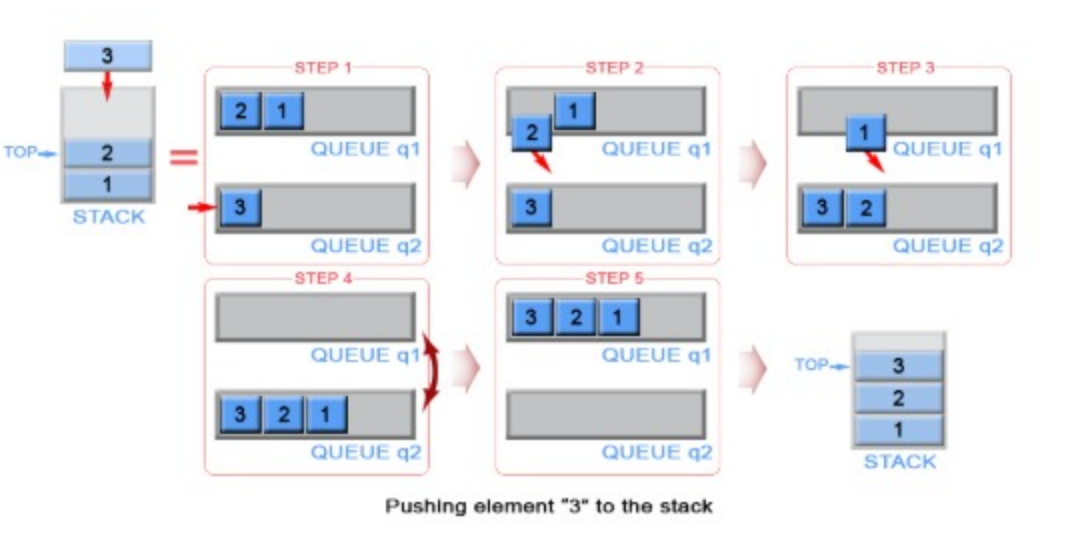

Figure 3. Push an element in stack

Java

```java
public void push(int x) {
    q2.add(x);
    top = x;
    while (!q1.isEmpty()) {
        q2.add(q1.remove());
    }
    Queue<Integer> temp = q1;
    q1 = q2;
    q2 = temp;
}
```

Complexity Analysis

- Time complexity : $O(n)$. The algorithm removes n elements from `q1` and inserts $n+1$ elements to `q2`, where n is the stack size. This gives $2n+1$ operations. The operations `add` and `remove` in linked lists has $O(1)$ complexity.
- Space complexity : $O(1)$.

Pop

The algorithm dequeues an element from queue `q1` and keeps front element of `q1` as `top`.

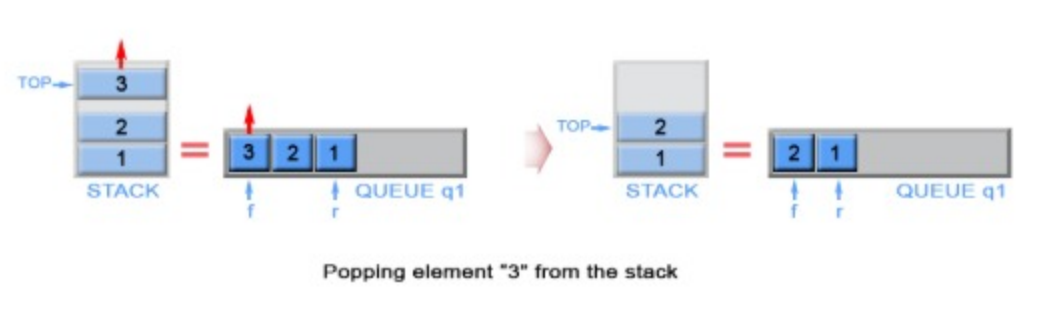

Figure 4. Pop an element from stack

Java

```java
// Removes the element on top of the stack.
public void pop() {
    q1.remove();
    if (!q1.isEmpty()) {
        top = q1.peek();
    }
}
```

Complexity Analysis

- Time complexity : $O(1)$.
- Space complexity : $O(1)$.

In both approaches `empty` and `top` operations have the same implementation.

Empty

Queue `q1` always contains all stack elements, so the algorithm checks `q1` size to return if the stack is empty.

```java
// Return whether the stack is empty.
public boolean empty() {
    return q1.isEmpty();
}
```

Time complexity : $O(1)$.

Space complexity : $O(1)$.

Top

The `top` element is kept in constant memory and is modified each time when we push or pop an element.

```java
// Get the top element.
public int top() {
    return top;
}
```

Time complexity : $O(1)$. The `top` element has been calculated in advance and only returned in `top` operation.

Space complexity : $O(1)$.

Approach #3 (One Queue, push - $O(n)$, pop $O(1)$)

The mentioned above two approaches have one weakness, they use two queues. This could be optimized as we use only one queue, instead of two.

Algorithm

Push

When we push an element into a queue, it will be stored at back of the queue due to queue's properties. But we need to implement a stack, where last inserted element should be in the front of the queue, not at the back. To achieve this we can invert the order of queue elements when pushing a new element.

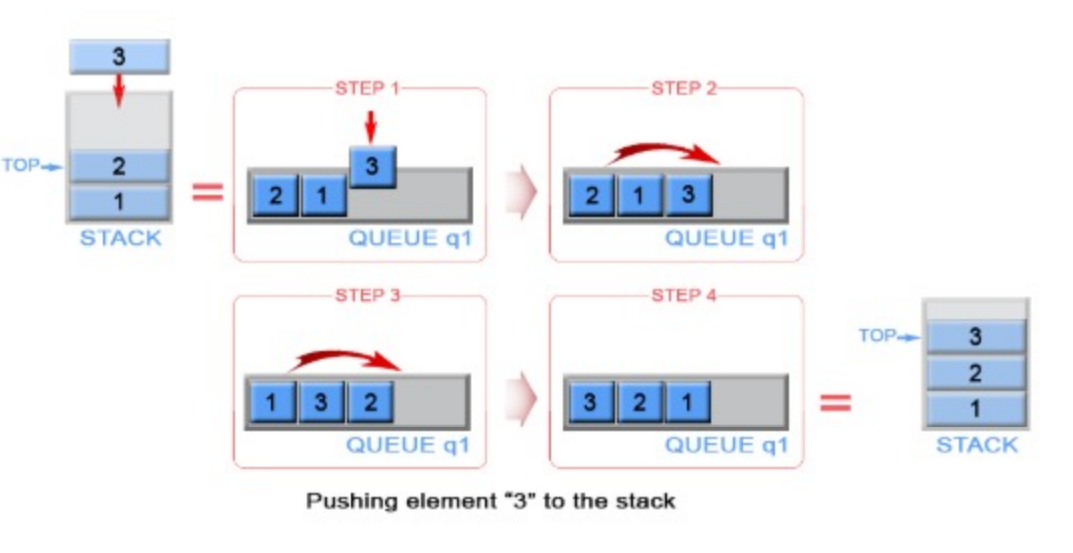

Figure 5. Push an element in stack

Java

```java
private LinkedList<Integer> q1 = new LinkedList<>();

// Push element x onto stack.
public void push(int x) {
    q1.add(x);
    int sz = q1.size();
    while (sz > 1) {
        q1.add(q1.remove());
        sz--;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. The algorithm removes n elements and inserts $n+1$ elements to `q1`, where n is the stack size. This gives $2n+1$ operations. The operations `add` and `remove` in linked lists has $O(1)$ complexity.
- Space complexity : $O(1)$.

Pop

The last inserted element is always stored at the front of `q1` and we can pop it for constant time.

Java

```java
// Removes the element on top of the stack.
public void pop() {
    q1.remove();
}
```

Complexity Analysis

- Time complexity : $O(1)$.
- Space complexity : $O(1)$.

Empty

Queue `q1` contains all stack elements, so the algorithm checks if `q1` is empty.

```java
// Return whether the stack is empty.
public boolean empty() {
    return q1.isEmpty();
}
```

Time complexity : $O(1)$.

Space complexity : $O(1)$.

Top

The `top` element is always positioned at the front of `q1`. Algorithm return it.

```java
// Get the top element.
public int top() {
    return q1.peek();
}
```

Time complexity : $O(1)$.

Space complexity : $O(1)$.

Analysis written by: @elmirap.

Rate this article: ★★★★★

Previous Next

Comments: 21 Sort By

Type comment here... (Markdown is supported)

Preview Post

terrible_whiteboard ★633 May 19, 2020 7:23 AM
I made a video if anyone is having trouble understanding the solution (clickable link)
https://youtu.be/t3OvlcsaXjk
Read More
20 Share Reply

milanocookie ★111 July 29, 2018 4:37 AM
For Approach #1, the pop method returns void when it should return the top. Code below puts the top in a temporary variable called theTop and then it gets removed after. Finally, theTop is returned.
/** Removes the element on top of the stack and returns that element. */
Read More
10 Share Reply

vanni1212 ★21 July 4, 2018 2:28 PM Report
Absolutely loved the article and its complete illustrations. Thanks
14 Share Reply

sergez ★2 April 10, 2018 1:34 PM
In the Approach #3 here is a wrong interface for pop(), the correct one should be:
// Removes the element on top of the stack.
public int pop() {
Read More
2 Share Reply
SHOW 1 REPLY

sirykd ★3 September 10, 2016 6:37 PM
We can also make a one-queue solution with push ~ O(1) and pop ~ O(n)
// Push element x onto stack.
void push(int x) {
_top = x;
Read More
2 Share Reply
SHOW 1 REPLY

kdkhd ★1 December 7, 2019 12:16 AM Report
Easy python solution. Faster than 99.11 % and less space that 100%
class MyStack:
Read More
1 Share Reply
SHOW 2 REPLIES

uluk ★1 July 27, 2019 5:45 AM
This is weirdest problem I have ever seen
1 Share Reply
SHOW 1 REPLY

s961206 ★751 July 8, 2019 6:36 PM
1 Queue / push O(1) / pop O(n) solution:
Queue<Integer> q;
int top = -1;
Read More
1 Share Reply
SHOW 1 REPLY

yh32 ★36 December 28, 2018 11:40 PM
The 3rd approach is a linkedlist! How can this be called a queue?
1 Share Reply
SHOW 7 REPLIES

kharikrishna2004 ★-1 August 28, 2018 6:38 PM
Hi,
Can anyone give me the code implement in c language ?
Kindly upload in the comment box .
Read More
0 Share Reply

‹ 1 2 3 ›