Intermediate Software Engineering

Intermediate-Software-Engineering-Summer-2022 (@ Intermediate)
Personal Member ID#: 52751

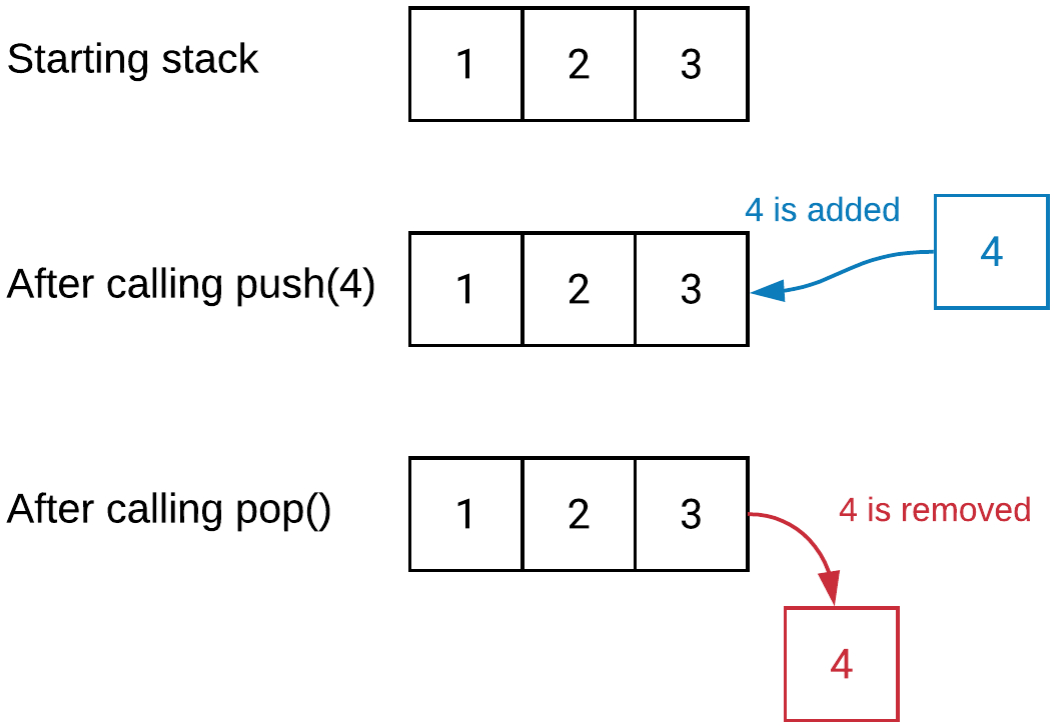
Stacks and Queues UMPIRE Cheat Sheet

Definition

Stacks and queues are foundational data structures that are useful when adding and removing in particular orders. It's important to be comfortable with these two data structures.

Stacks

A **stack** is a data structure that stores objects in which the most recently stored objects are the first ones to be removed, (LIFO: last in, first out). An example to help you remember the mechanics of a stack is to associate it with stacks in real life. With a stack of plates, the plates that are placed on top of a stack will be the first ones that are removed from the top!

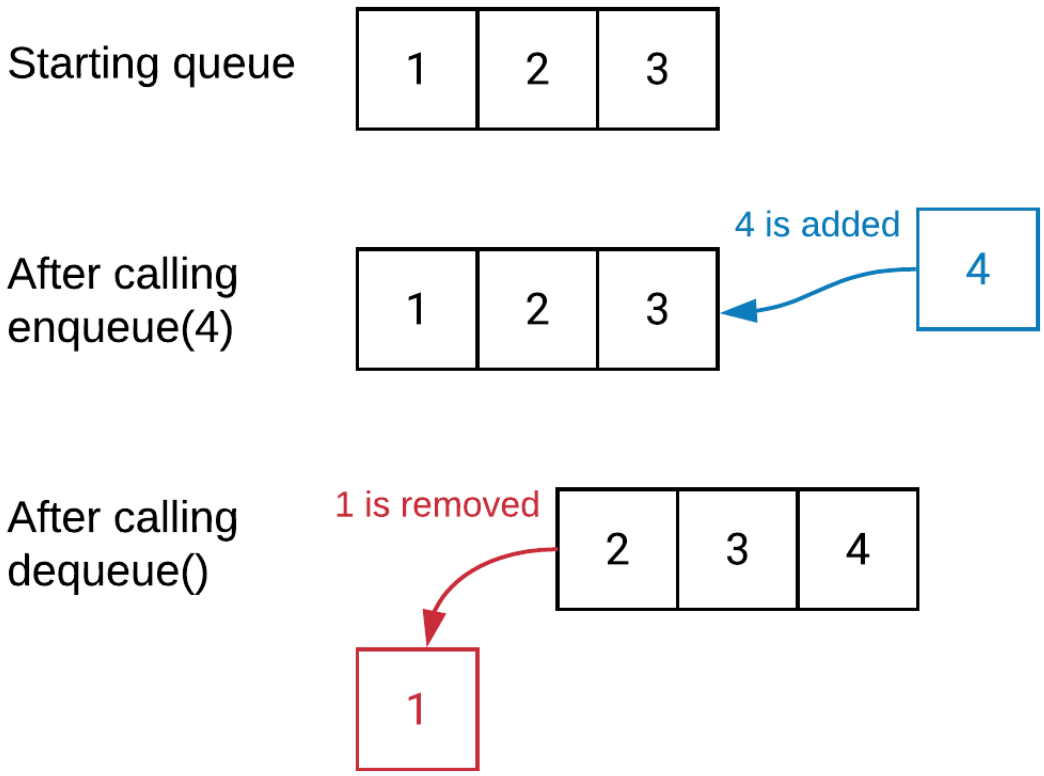


It's important to know the common operations of a stack. The two key stack operations are:

1. `pop()` : removing an item from the stack in a last in, first out order (LIFO)
2. `push(item)` : adding an item to the stack

Queues

A **queue** is a data structure that stores objects in which the most stored objects are the first ones to be removed. A helpful acronym associated with queues is FIFO, first in first out. An example to help you remember the mechanics of a queue is to associate it with queues in real life. With a queue of people waiting to get a seat in a restaurant, the first people to get in the queue will be the first people seated at that restaurant.



It's important to know the common operations associated with a queue. The two important queue operations are:

1. dequeue(): removing an item from the queue in a first in, first out order (FIFO)
2. enqueue(item): adding an item to the queue

Key takeaways

- Stacks are very useful for it's backtracking features. For example, parsing questions tend to use stacks because of the LIFO property.
- Stacks can be used to implement recursive solutions iteratively.
- Queues are useful when the ordering of the data matters as it preserves that ordering. For example, they're used for caching.

Check out CodePath guide on Stack and Queues.

U-nderstand

What are some common questions we should ask our interviewer?

- Are there memory constraints?
- What's the required time complexity?
- What kind of data will the inputs be?
- Can I assume all the inputs will be valid?
- What if the input is empty?
- What should we return if there is no solution to the problem?
- What should we return if there are multiple solutions to the problem?
- Can stack or queue be empty?
- Can you pop from an empty stack?

M-atch

Are there any special techniques that we can use to help make this easier?

Stacks

If recursion is banned, then use stacks. Because of its last-in-first-out (LIFO) property, it has many advantages over other data structures, such as it can be used in cases where we only need to access or remove elements from one end.

A stack can be implemented using linked lists. We can use a list instead of a dynamic array. The head of the linked list will be the top element of the stack. We can dynamically push elements in it and also pop elements from it using delete a node and insert new node operations in linked lists.

Queues

A queue data structure is generally used in scenarios where the FIFO approach (First In First Out) has to be implemented. The following are some of the most common applications of queue in data structure:

- Managing requests on a single shared resource such as CPU scheduling and disk scheduling
- Handling hardware or real-time systems interrupts
- Handling website traffic
- Routers and switches in networking
- Maintaining the playlist in media players

P-lan/Pseudocode

- Can you create any **magic** helper methods that would simplify the solution?
- Talk through different approaches you can take, and their tradeoffs
- Be able to verbally describe your approach and explain how an example input would produce the desired output

Tips:

- Try to avoid nested loops
    - This is usually a brute force solution and is $O(n^2)$ time complexity
- For stacks, look out for corner cases: Empty stack, Popping from an empty stack, Stack with one item, Stack with two items
- For queues, look out for corner cases: Empty queue, Queue with one item, Queue with two items

E-valuate

Stacks Time Complexity

	Average Case	**Worst Case**
Access	O(n)	O(n)
Search	O(n)	O(n)
Insertion	O(1)	O(1)
Deletion	O(1)	O(1)

Queues Time Complexity

	Average Case	**Worst Case**
Access	O(n)	O(n)
Search	O(n)	O(n)
Insertion	O(1)	O(1)
Deletion	O(1)	O(1)