Stacks and Queues

Stacks and queues are two higher-order data structures that store a list of elements.

Stacks

Stacks use a “last-in, first-out” (LIFO) approach that allows elements to be retrieved in the opposite order of their arrival (like a stack of pancakes).

A stack is similar to an array with one significant difference: elements are only accessible from one end, the top.

This means we **can’t randomly access** elements.

We can add elements to the stack and we can remove elements from the stack. If we want to access an element mid-way in the stack, we need to pop the elements above it off the stack.

Stacks are usually thought of as vertical data structures, unlike link lists and arrays, which are horizontal.

Stacks are used:

- To backtrack to the previous task/state, for example in recursive code.

- To store a partially completed task, for example, when you are exploring two different paths on a Tree from a point while figuring out the smallest path to the target

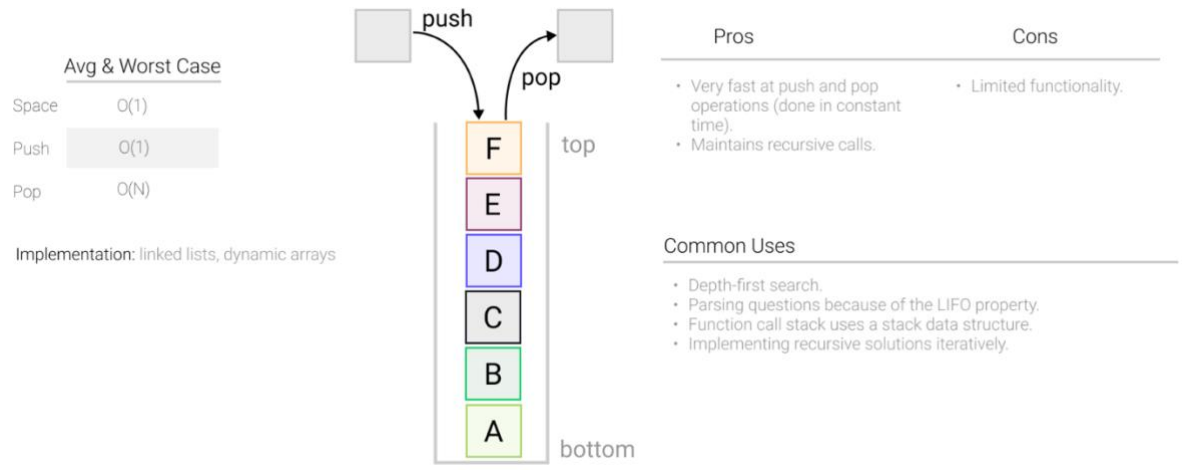
Benefits of Stacks

Stacks allow for constant-time adding and removing of the top item.

Downside of Stacks

Stacks, don't offer constant-time access to the **nth item** in the stack, unlike an array.

If we want to access the third item, we would have to pop off each item in our stack until we reach the 3rd item, and if we want to access the first item in the stack (the bottom element) we have to iterate through each book on top of it and pop it off; this has a worst-case runtime of O(n) where n is the number of items in the stack.



Linked List implementation:

```
1  class Node {
2    constructor(value, next) {
3      this.value = value;
4      this.next = next;
5    }
6  }
7
8  class Stack {
9    constructor() {
10     this.top = 0;
11   }
12
13   push(value) {
14     this.top = new Node(value, this.top);
15     return this;
16   }
17
18   pop() {
19     const popped = this.top;
20     this.top = popped.next;
21     return popped.value;
22   }
23 }
```

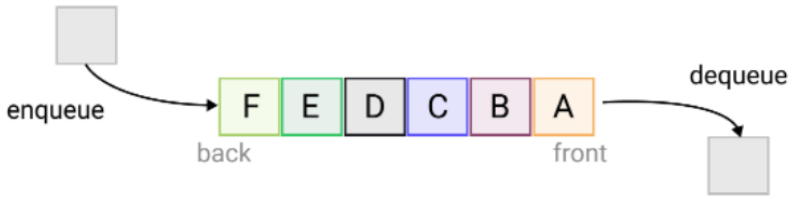
What is a stack overflow? Stack underflow is when we try to pop an item from an empty stack. Stack overflow happens when we try to push more items on a stack than its max capacity.

Array implementation:

```
class Stack {

  constructor() {
    this.store = [];
    this.top = 0;
  }

  push(elem) {
    return this.store.push(elem)
  }

  pop() {
    return this.store.pop()
  }

  getTop() {
    if (this.store.length === 0) return null;
    return this.top;
  }

  isEmpty() {
    return this.store.length === 0;
  }

  size() {
    return this.store.length;
  }
}
```

Queues

Queues obey a “first-in, first-out” (FIFO) behavior in which elements are enqueued and dequeued in the same order they arrive (like a line at the bank.)



Avg & Worst Case	
Space	O(1)
Queue	O(1)
Dequeue	O(N)

Implementation: linked lists

Pros

- Very fast at queue and dequeue operation (done in constant time).
- Keeps track of the order things are added.

Cons

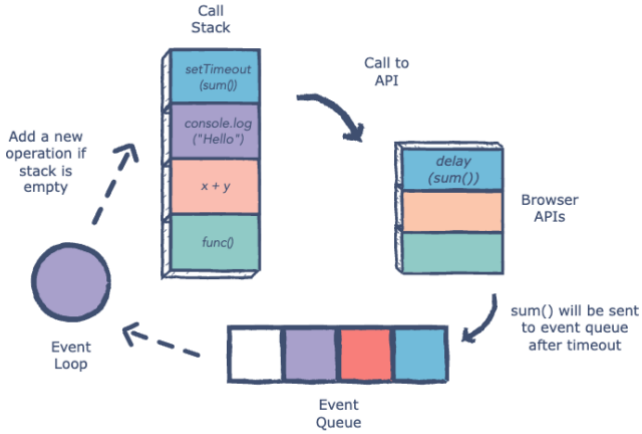
- Limited functionality.

Common Uses

- Breadth-first-search.
- Printers, managing print jobs in order that they were received.
- Web-servers, so that requests can be fulfilled in order that they occurred.

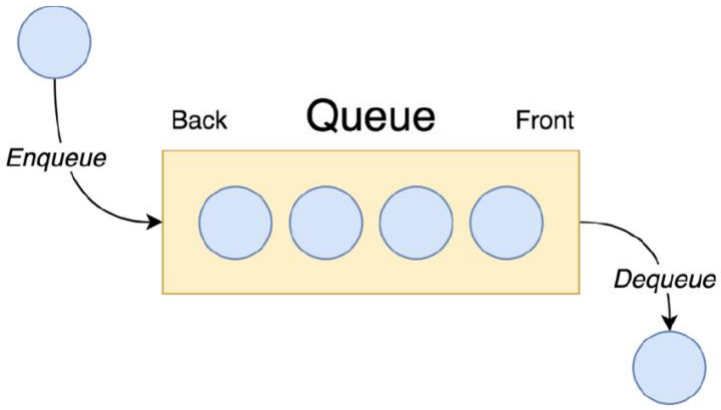
An example of a queue is the fast-food service at McDonald's. You line up, and service is provided in the order that you (and everyone else) lined up. If you are first to line up, you get served first.

A more practical example of a queue is the event loop of a web browser. As different events are being triggered (for example, the click of a button), they are added to an event loop's queue and handled in the order they entered the queue.



The main functions of a Queue include:

- enqueue(data): Adds data to the back (insertion) - **back**
- dequeue(): Removes oldest data added (deletion) - **front**



Note: Operations such as searching or updating elements are not common for queues or stacks.

Linked List implementation:

```
1  class Node {
2    constructor(value) {
3      this.value = value;
4      this.next = null;
5    }
6  }
7
8  class Queue {
9    constructor() {
10     this.first = null;
11     this.last = null;
12   }
13
14   enqueue(value) {
15     const newNode = new Node(value);
16
17     if (this.first) {
18       this.last.next = newNode;
19     } else {
20       this.first = newNode;
21     }
22
23     this.last = newNode;
24   }
25
26   dequeue() {
27     const dequeued = this.first;
28
29     this.first = dequeued.next;
30
31     if (dequeued === this.last) {
32       this.last = null;
33     }
34
35     return dequeued.value
36   }
37 }
38
```

Note: Stacks and queues can either use an array or a linked list as storage. But many favor a linked list because an array requires a set amount of memory to be set aside before they are created. Whereas the memory for a linked list can be dynamic. Arrays should be used to implement queues or stacks when the maximum size of each is known beforehand as to limit the memory requirements.

Some common stack problems

Both stacks and queues are often used in conjunction with a linked list, and it's very good for fast computation since an item will always be added to or removed from the top or bottom of the list no matter how long the list is.

An example of using a queue for an algorithm is tree breadth-first search. We can use a queue to store the nodes that need to be processed later.

Valid Parentheses

a = (){}
b = {}(

parens(a) #=> true
parens(b) #=> false

https://leetcode.com/problems/valid-parentheses/