Chapter 3 Stacks and Queues

Fei Tan

Chaifetz School of Business
Saint Louis University

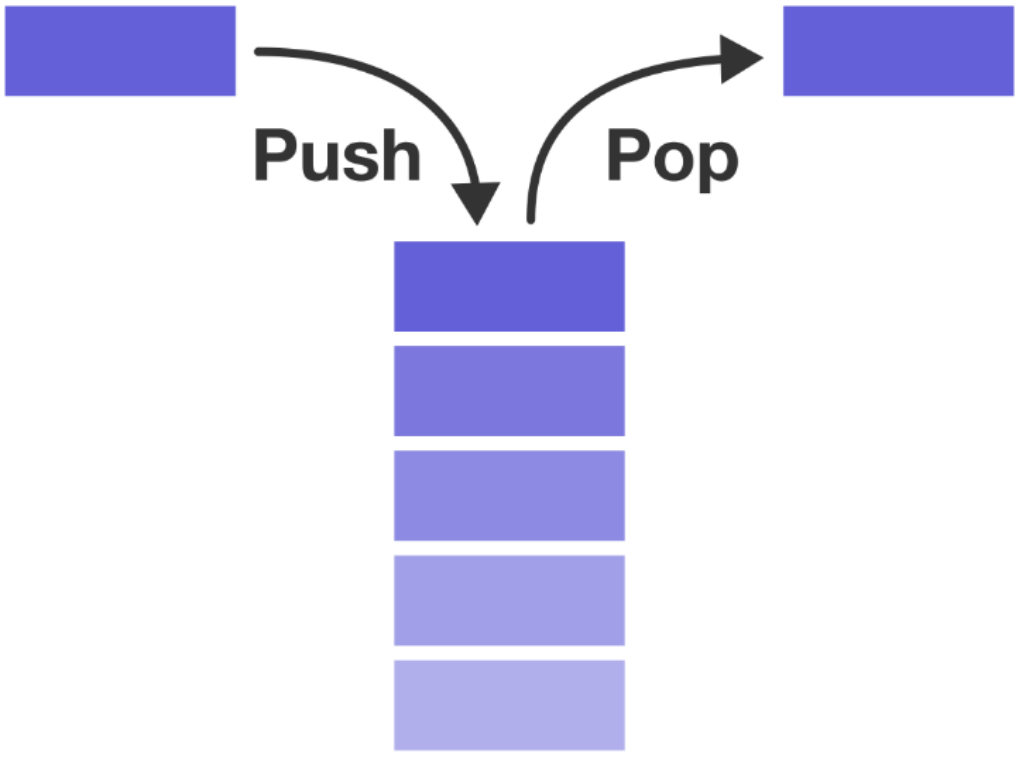
Computer Science Fundamentals
(Source: brilliant.org)

December 2, 2018

The Road Ahead...

- Only the top can come off a stack, while elements must wait their turn in a queue; both data structures have their place in computer science
- What we'll accomplish
  - pop() and push() values to manipulate stacks
  - implement queues efficiently with circular arrays

Data Structure: Stacks



- LIFO: last-in, first-out, e.g., reverse order of elements, 'undo' command, reverse polish
- Stack overflow (push beyond space) vs. stack underflow (pop empty stack)

Python Code

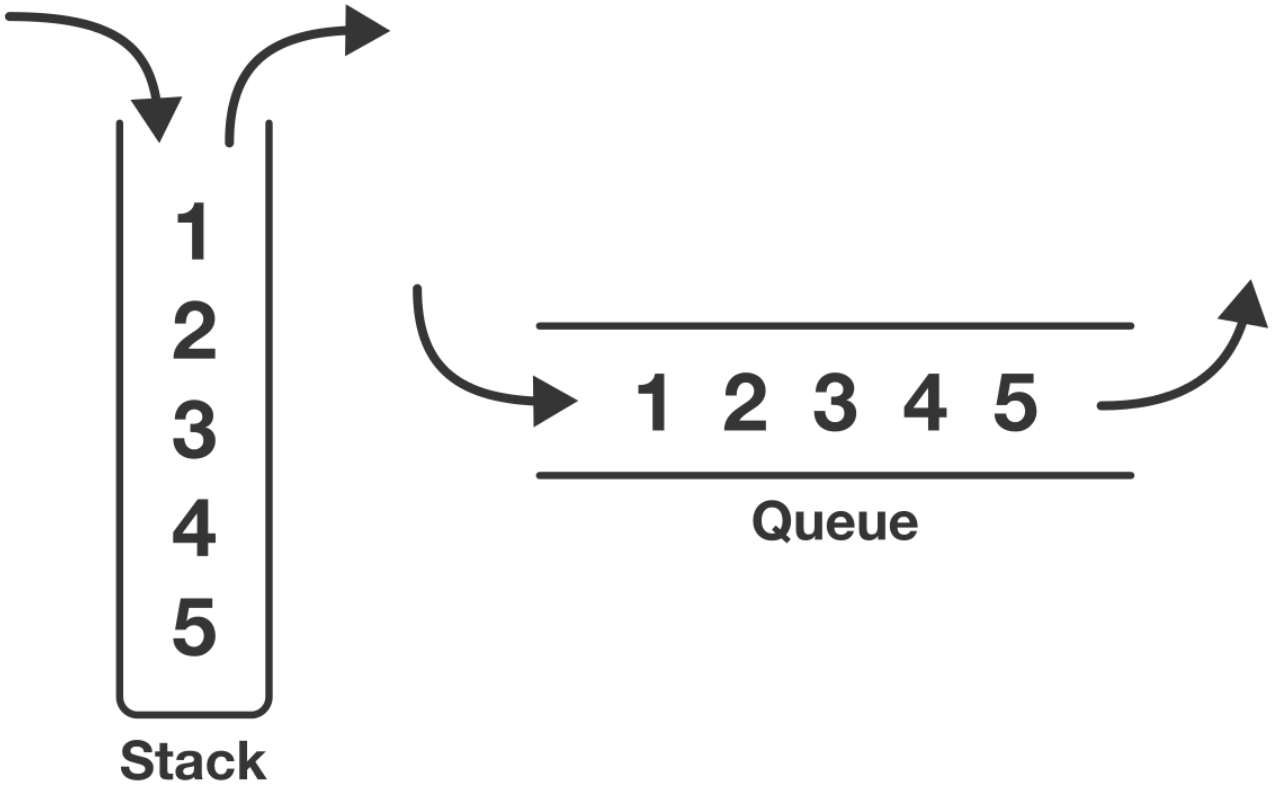
Implementation using array

```
class Stack:
    self.Array = []
    self.size = 0

    def __init__(self,Array=None):
        if Array != None:
            self.Array = Array
            self.size = len(Array)
        else:
            self.Array = []
            self.size = 0

    #continue below...
```

Python Code (Cont'd)

```
class Stack:
    #continue above...

    def push(self,item): #insert item on top
        self.Array.append(item)

    def pop(self): #remove item from top
        Deleted = self.Array[-1]
        self.Array = self.Array[0:-1]
        return Deleted

    def peek(self): #return top element
        return self.Array[-1]

    def size(self): #return stack size
        return self.size
```

Data Structure: Queues



- FIFO: first-in, first-out, e.g., CPU scheduling, elevators, keyboard buffering
- Head (where to remove element) & tail (where to add element)

Python Code

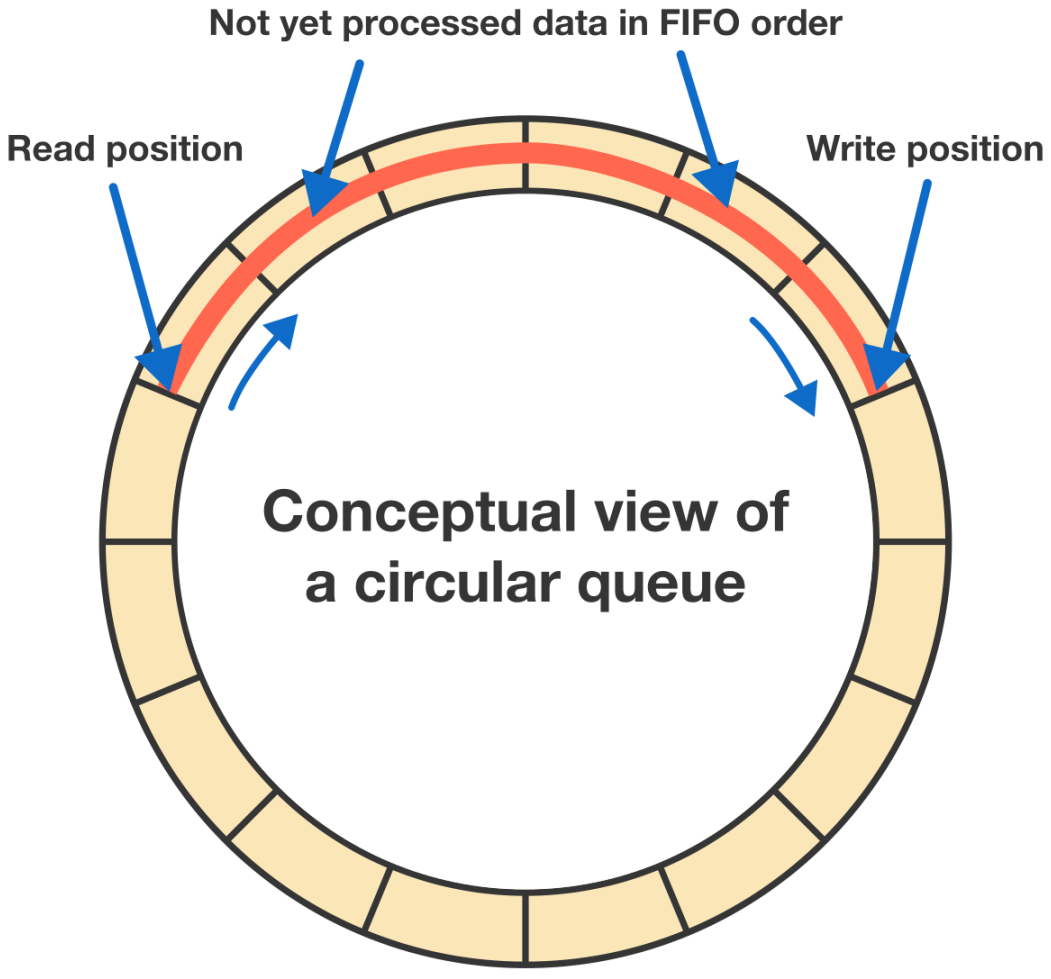
Implementation using array

```
class Queue:
    def __init__(self):
        self.qlist = []
        self.head = -1
        self.tail = -1
    def size(self): # return queue size
        return len(self.qlist)
    def peek(self): # return head element
        if self.head == -1:
            print 'queue is empty'
        else:
            return self.qlist[self.head]

    #continue below...
```

Python Code (Cont'd)

```
class Queue:
    #continue above...

    def enqueue(self, item): #insert item at tail
        if self.head == -1:
            self.head += 1
        self.tail += 1
        self.qlist.append(item)
    def dequeue(self): #remove item from head
        if self.head== -1:
            print 'queue underflow'
        elif self.head == self.tail:
            p = self.qlist.pop(0)
            self.head -= 1
            self.tail -= 1
            return p
        else:
            self.tail -= 1
            return self.qlist.pop(0)
```

Data Structure: Circular Queues



- Dequeue: linear ($O(N)$) vs. circular ($O(1)$)

Python Code

Implementation using array

```
class circularQueue:
    def __init__(self, size):
        self.qlist = [None]*size
        self.head = 0
        self.tail = 0
        self.size = size

    def enqueue(self, item):
        if (self.head-self.tail)%self.size == 1:
            print 'The queue is full'
        else:
            if self.qlist[self.head] != None:
                self.tail += 1
                self.tail = self.tail % self.size
            self.qlist[self.tail] = item

    #continue below...
```

Python Code (Cont'd)

```
class circularQueue:
    #continue above...
    def dequeue(self):
        if self.qlist[self.head] == None:
            print 'queue underflow'
        elif self.head == self.tail:
            p = self.qlist[self.head]
            self.qlist[self.head] = None
            self.head += 1
            self.head = self.head % self.size
            self.tail += 1
            self.tail = self.tail % self.size
            return p
        else:
            p = self.qlist[self.head]
            self.qlist[self.head] = None
            self.head += 1
            self.head = self.head % self.size
            return p
```

Python Code (Cont'd)

```
class circularQueue:
    #continue above...

    def getSize(self):
        count = 0
        for element in self.qlist:
            if element != None:
                count += 1
        return count

    def peek(self):
        if self.qlist[self.head] == None:
            print 'queue is empty'
        else:
            return self.qlist[self.head]
```