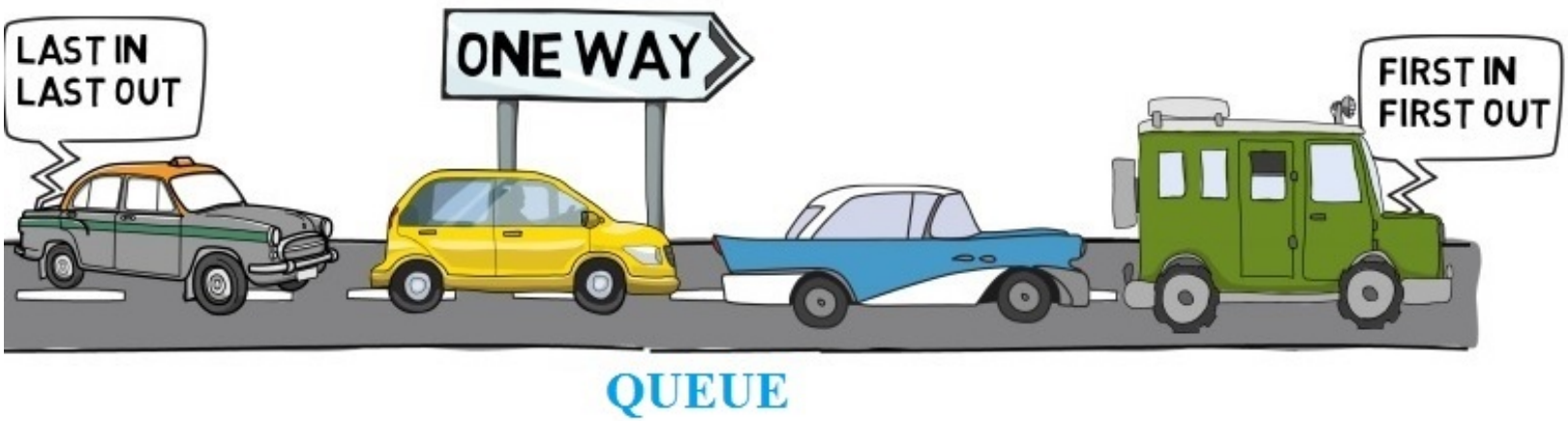
CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

Outline



QUEUE

A Queue ADT

A Container Class for First-In-First-Out Access

A queue is a **first-in-first-out** structure, i.e. it is a list-like container with access restricted to both ends of the list.

Items are added at the back of the queue and removed from the front. In real life: "waiting in line" or "taking a number".

- The enqueue method puts an item at the back of the queue.
- The dequeue method returns the item at the front, and removes it from the queue.
  (precondition: queue is not empty—`size > 0`)
- The front method returns that item
  (precondition: queue is not empty—`size > 0`)
- The size method returns the number of items in the queue.

A Queue ADT

Specification for a typical queue

```
class Queue:
   def __init__(self):
      """ post:  creates an empty FIFO queue"""
   def enqueue(self,x):
      """post:  adds x at the back of the stack"""
   def dequeue(self):
      """pre:  self.size()>0
      post:  removes and returns the front item"""
   def front(self):
      """pre:  self.size()>0
      post:  returns the first intem in the queue"""
   def size(self):
      """post:  returns number of elements in the queue"""
```

A Queue ADT

Queues applications

Queues used in computer programming as a sort of buffer between different phases of a computing process, for example:

- when printing documents
  (job requests are placed on a queue in OS),
- when compiling/interpreting a single program
  (**lexical analysis** splits the program into its meaningful pieces, the **tokens** that can be stored in a queue for subsequent processing by next phase, which is often some sort of grammar-based syntactic analysis)
- when determining whether or not a phrase is a palindrome
  (i.e. has the same sequence of letters when read either forward or backward.

Simple Queue Application: A Palindrome Recognizer

A Palindrome Examples

race car
Madam I'm Adam
I prefer PI
Never odd or even
tricky part: the palindromeness of a phrase is determined only by the letters; spaces, punctuation, and capitalization don't matter.

A Palindrome Recognizer - specification

```
def isPalindrome(phrase):
   """pre:  phrase is a string
      post:  return True if the alphabetic characters
in phrase form the same sequence reading either
left-to-right or right-to-left."""
```

Simple Queue Application: A Palindrome Recognizer

A Palindrome Recognizer

```
from MyQueue import Queue
from Stack import Stack

def isPalindrome(phrase):
    forward = Queue()
    reverse = Stack()
    extractLetters(phrase, forward, reverse)
    return sameSequence(forward, reverse)
```

Simple Queue Application: A Palindrome Recognizer

Phase I: Extract Letters

```
def extractLetters(phrase, q, s):
    for ch in phrase:
        if ch.isalpha():
            ch = ch.lower()
            q.enqueue(ch)
            s.push(ch)
```

Simple Queue Application: A Palindrome Recognizer

Phase II: Same Sequence

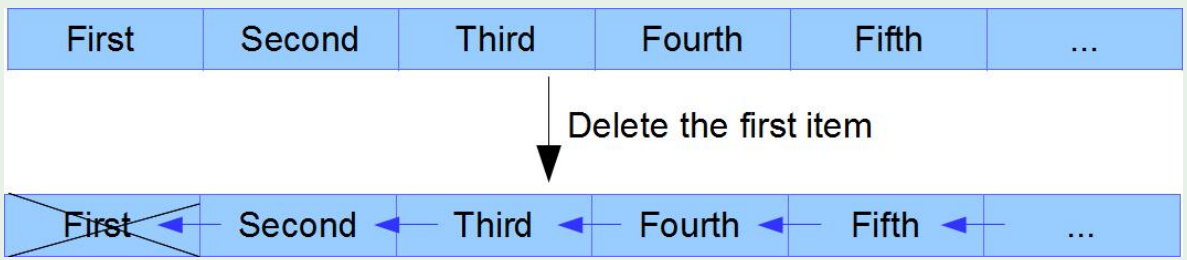
```
def sameSequence(q, s):
   while q.size() > 0:
      ch1 = q.dequeue()
      ch2 = s.pop()
      if ch1 != ch2:
         return False
   return True
```

Queue Implementations

A Python List Is Not An Efficient Queue

Implementation of **queue** with Python's built-in list, implemented as an array, is straightforward: we need to insert at one end, and remove from the other one. If we decide that the first item is at the:

- beginning of the list, then to remove an item from the queue (dequeue) is $\Theta(n)$ – every item must be moved down to delete the first item,

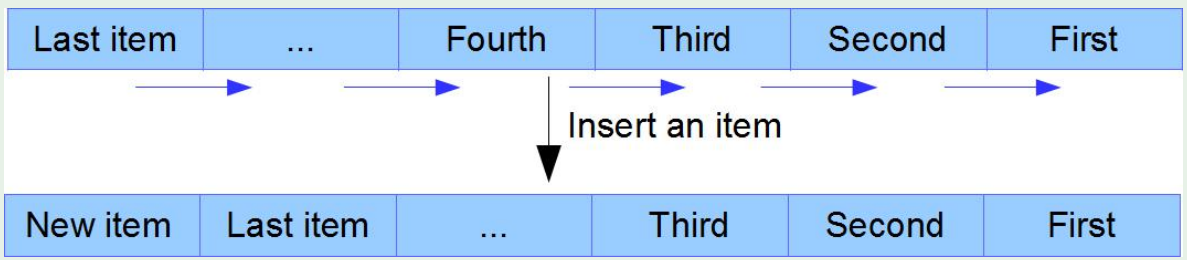


Queue Implementations

A Python List Is Not An Efficient Queue

Implementation of **queue** with Python's built-in list, implemented as an array, is straightforward: we need to insert at one end, and remove from the other one. If we decide that the first item is at the:

- end/back of the list, then to insert an item into the queue (`enqueue`) is $\Theta(n)$ – every item must be moved up to insert the item.



Queue Implementations

Linked List

An alternative: to use linked implementation.

This is the most flexible representation of a Queue ADT.

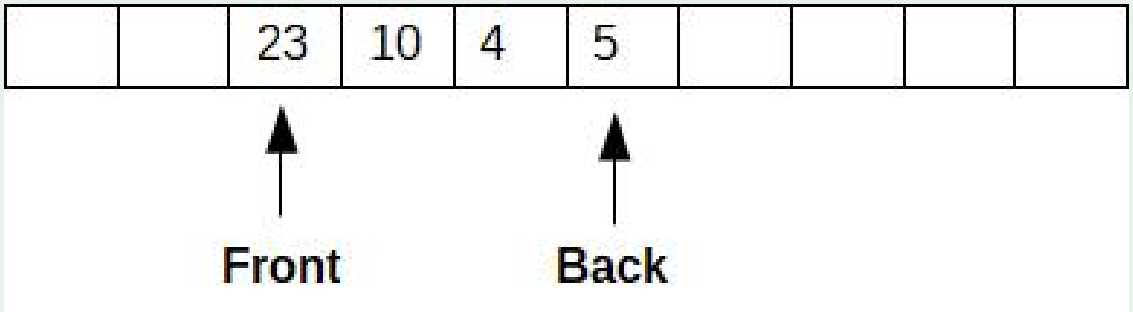
Use references to the first node (front) and last node (back), a singly-linked list can perform `enqueue` and `dequeue` in constant time ($\Theta(1)$).

However, the links take up extra memory space.

Queue Implementations

Circular Array

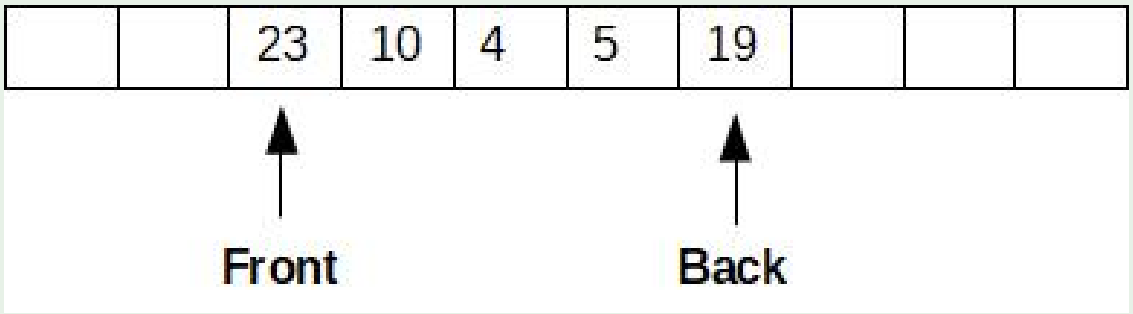
A circular array avoids both the space inefficiency of links and the time inefficiency of the Python list (array) representation by not moving items. Instead, the front and back of the queue move, using changing indexes as markers.



Queue Implementations

Circular Array : enqueue

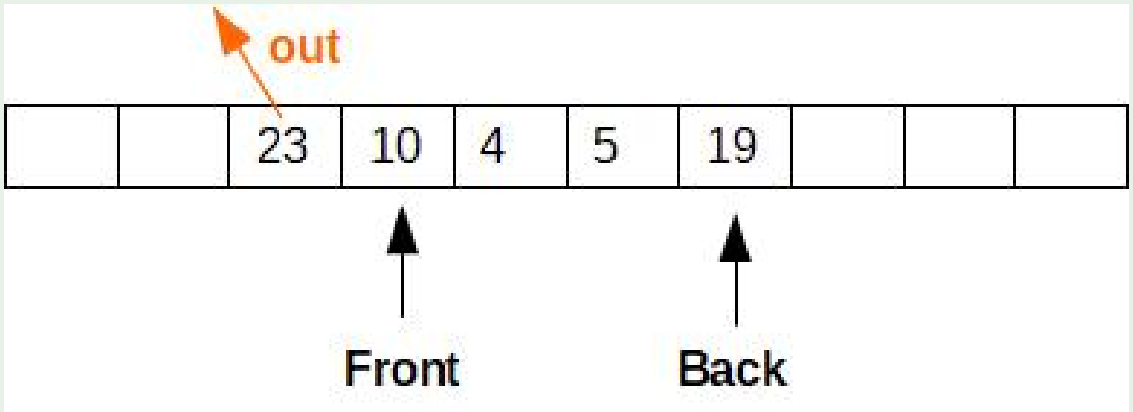
To enqueue a new item: the index marking the back of the queue is increased by one.
The item that was in the back stays in the same position, but the new item goes behind it, into the new "back" position.



Queue Implementations

Circular Array : dequeue

To dequeue an item: the front of the queue is moved to the next item behind it.

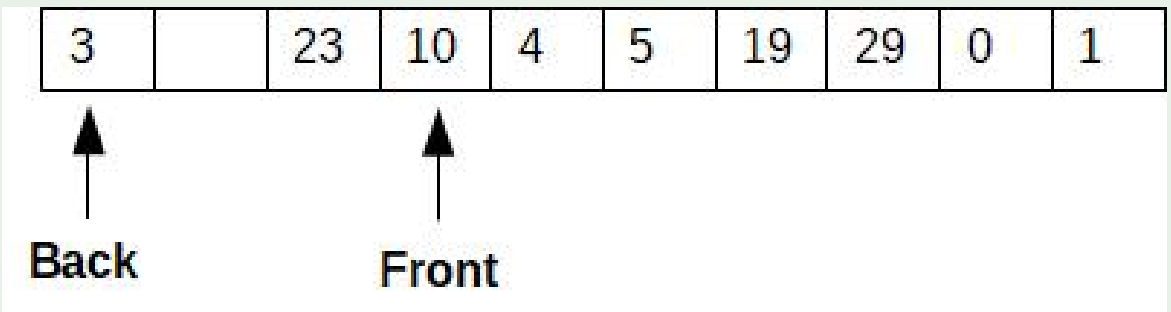


Queue Implementations

Circular Array

The array is "circular" because when either marker goes past the end of the array, it is put back at index zero

(`tail = (tail+1) % size`).



check out this place: http://www.yashcode.com/2017/11/queue-in-data-structure.html

Simulation of Retail Store with One Checkout Register

A Typical Queuing Simulation

Queues can model the behavior of real-world queues. Consider a grocery store with one check-out register. To measure efficiency of service delivery, one runs a program that simulates these events:

- Random arrival times
  (customers finish shopping;
  they don't arrive at a constant rate)
- Waiting for service (grocery checkout register)
- Random time to be serviced
  (depends on number and kind of items)

Simulation of Retail Store with One Checkout Register

Self-study:
Check out the simulation in the book (Section 5.5).

This particular approach is called time-driven simulation, i.e. we increment the clock one tick at a time and do whatever has to be done in that tick.
Any events in the `arrivalQueue` that occur at a given tick are moved into the line.

drawback: many of the cycles around the tick-loop will essentially be idle time.

alternative approach: use event-driven simulation, i.e. we don't model each tick, but simply "jump ahead" to the next event to be processed.

Cab Company

In-class assignment

see the handout or the CabCompany.pdf on our web-site.