CSE 201: Data Structures and Algorithms

Lecture 5:Queues

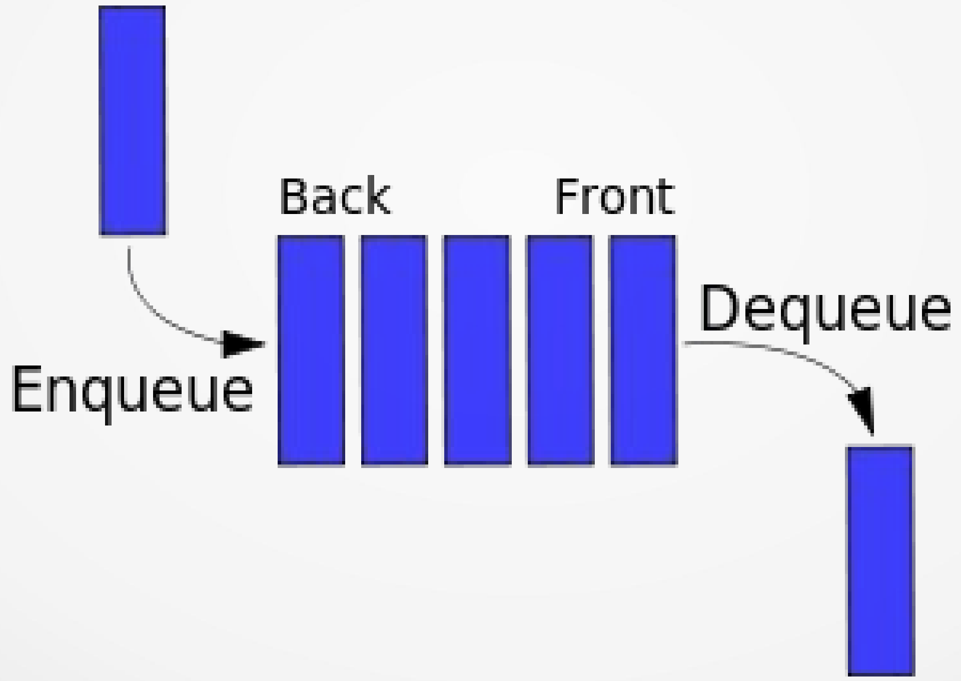
Dr. Vidhya Balasubramanian

Queues

- It is a first-in-first-out abstract data type
- Applications in Real World
  - Transportation
  - Operations Research
  - Acts as buffer in many applications

Queues: An Overview

- Element that is inserted first is removed first
  - Insertions at the rear of the queue and deletions at the front of the queue



http://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/300px-Data_Queue.svg.png

Queue ADT: Main Operations

- enqueue(o)
  - Inserts an object o at the end of the queue
  - Input: object; Output: None
- dequeue()
  - Removes and returns the first element in the queue
  - Input: none; Output: object
  - Error occurs if queue is empty

Other Queue Operations

- size()
  - Returns the number of objects in the queue
- isEmpty()
  - Returns a Boolean indicating if the queue is empty
- front()
  - Return first element of the queue without removing it. Error occurs if queue is empty
  - Input: None; Output: Object

Queue Exceptions

- Some operations may cause an error causing an exceptions
- Exceptions in the Queue ADT
  - QueueEmptyException
    - dequeu() and front() cannot be performed if the queue is empty
  - QueueFullException
    - Occurs when the queue has a maximum size limit ie implemented with an array
    - enqueue(o) cannot occur when the queue is full

Queue Example

Operation	Output	Queue Contents
enqueue(5)	-	(5)
enqueue(3)	-	(5,3)
enqueue(7)	-	(5,3,7)
dequeue()	5	(3,7)
size()	2	(3,7)
enqueue(4)	-	(3,7,4)
dequeue()	3	(7,4)
dequeue()	7	(4)
size()	1	(4)
dequeue()	4	()
dequeue()	“error”	()

Queue Interface (Java)

```
public interface Queue<E>
    int size(): //returns number of objects in the queue
    boolean isEmpty() //returns true if queue is empty, false otherwise
    E front();
        //returns front object in queue, throws exception if queue empty
    void enqueue(E element): //inserts object at rear of queue
    E dequeue();
        //throws(Queue EmptyException)
}
```

Exercise

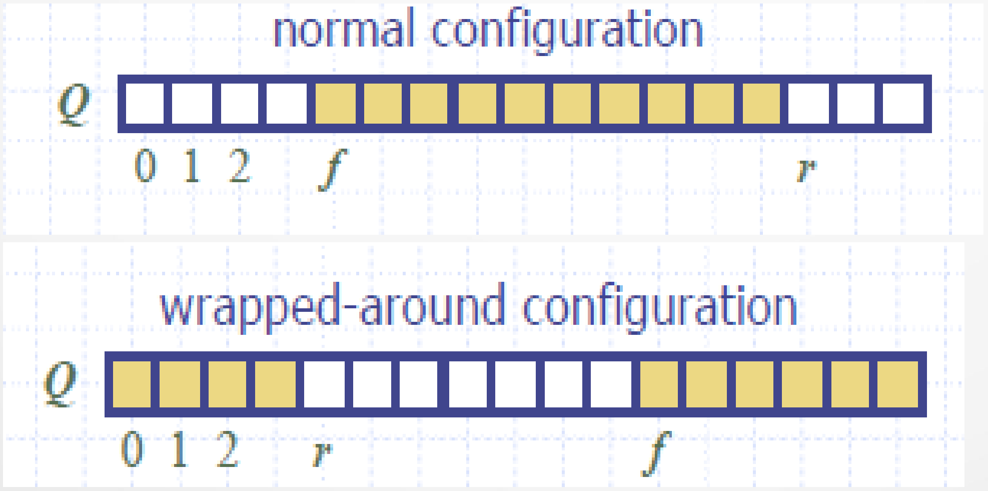
- Describe the output of the following series of queue operations
  - enqueue(5),enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), isEmpty(), enqueue(9), enqueue(1), size(), dequeue(), enqueue(7), enqueue(6), front(), dequeue(), dequeue(), enqueue(4), dequeue(), dequeue()
- Show how to implement a queue using two stacks. Analyze the running time of the queue operations.

Array based implementation of a Queue

- A Queue may be implemented by using a simple array
  - An N-element array
    - Queue is limited by the size of the array
  - Two variable to keep track of front and rear
    - Integer f denotes the index of the front element
    - Integer r denotes the position immediately past the rear element
- Strategy
  - Elements are added left to right

Circular Array Implementation

- After repeated enqueue and dequeue operations the rear part may reach end of queue
  - There may be place at the beginning of array
- Circular Queue
  - Allow f and r to wrap around to end of queue



src: Goodrich notes

Queue ADT Functions

- **Algorithm** size()

  return $(N - f + r) \bmod N$ *#may use a variable __len_ to keep track of size*

- **Algorithm** isEmpty()

  return (*size() ==0*) # return (f=r) and ($|r - f| = N - 1$) indicates queue is full

- **Algorithm** front()

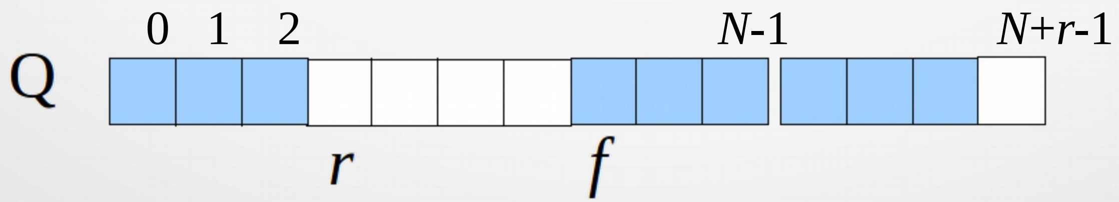
```
if isEmpty() then
    throw a QueueEmptyException
return Q[f]
```

Queue ADT Functions

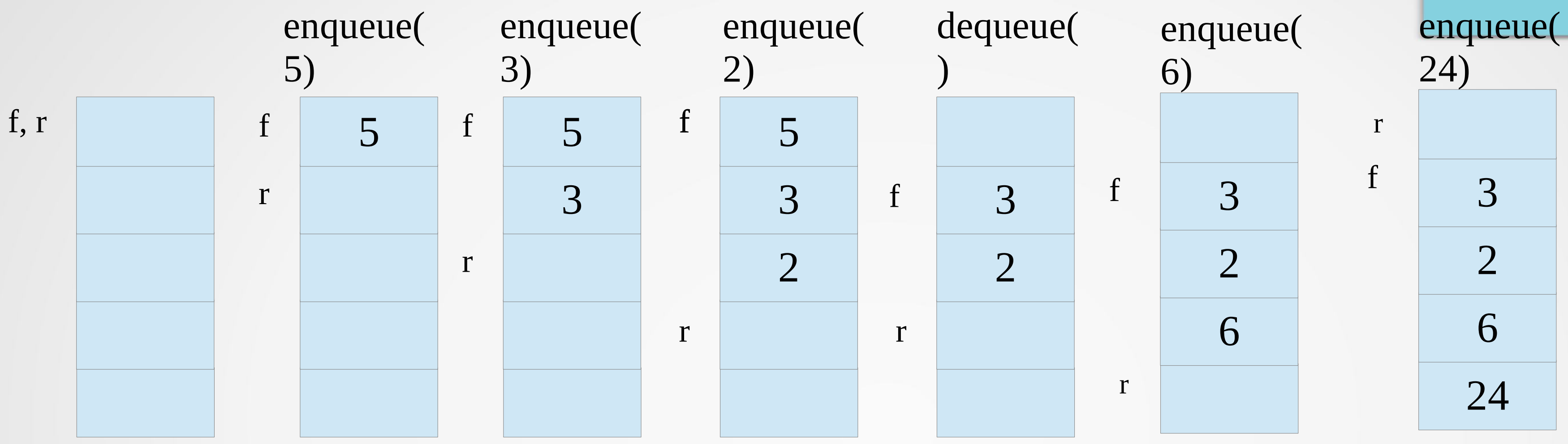
```
• Algorithm enqueu(o)
    if size() = $N-1$ then
        throw a QueueFullException
    $Q[r] \leftarrow o$
    $r \leftarrow (r+1) \bmod N$
• Algorithm dequeu()
    if isEmpty() then
        throw a QueueEmptyException
    $o \leftarrow Q[f]$
    $f \leftarrow (f+1) \bmod N$
    return o
```

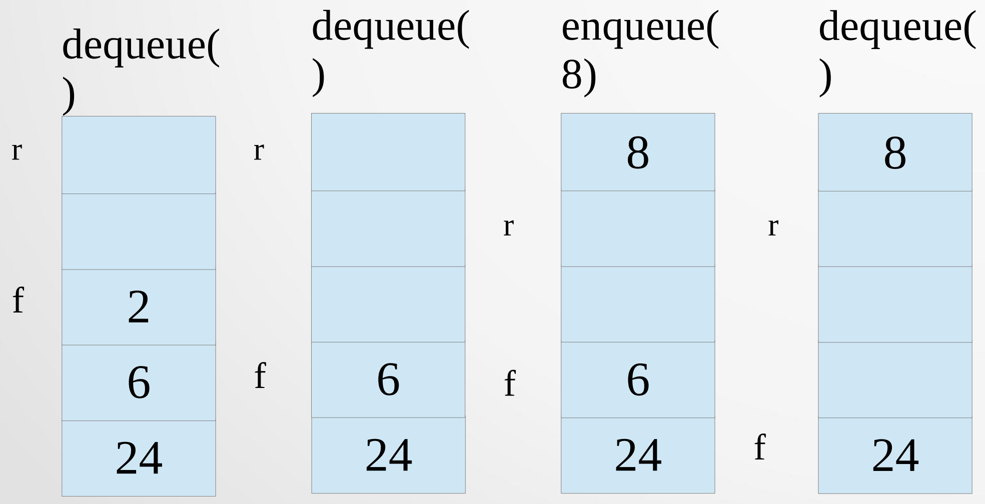
Circular Array

- Size Function
  - size(): return the number $(N - f + r) \bmod N$
  - *If $r \geq f$, then $r - f \geq 0$*
    - $N + (r - f) \geq N$
    - $(N - f + r) \bmod N = - f + r = r - f$
  - *If $r < f$, then $f - r > 0$*
    - $N - f + r = N - (f - r) < N$
    - $(N - f + r) \bmod N = N - f + r$



Circular Queue: Example





- enqueu(24)
  - $r \leftarrow (r+1) \bmod N$
  - $= (4+1) \bmod 5 = 0$
- *dequeu() (fig 11)*
  - $f \leftarrow (f+1) \bmod N$
  - $= (4+1) \bmod 5 = 0$

Complexity Analysis

- Time Complexity
  - size – $O(1)$
  - isEmpty – $O(1)$
  - front – $O(1)$
  - enqueue – $O(1)$
  - dequeue – $O(1)$
- Space Complexity
  - $O(N)$

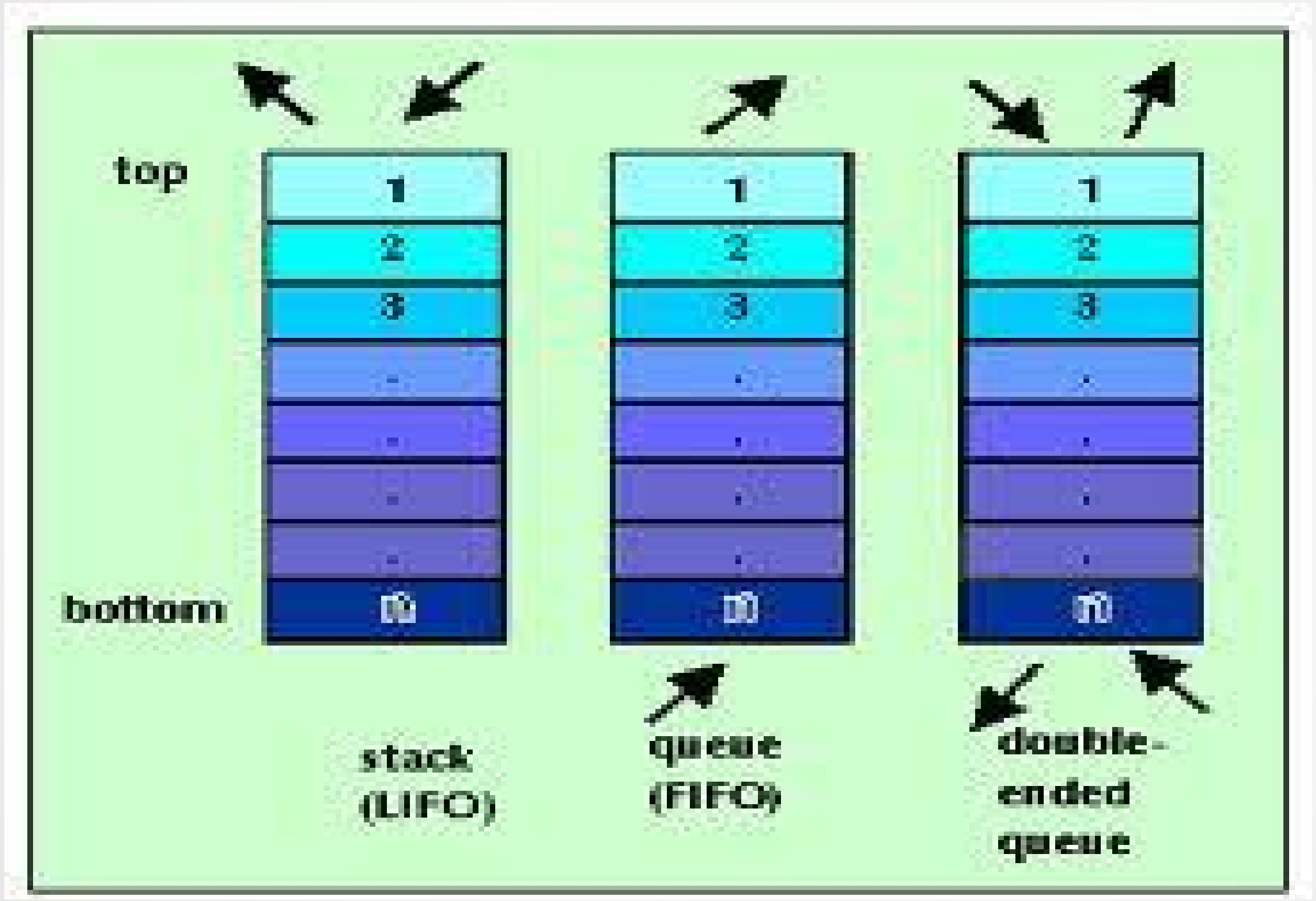
Growable Array-based Queue

- In an enqueue operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one
- Similar to what we did for an array-based stack
- The enqueue operation has amortized running time
  - O(n) with the incremental strategy
  - O(1) with the doubling strategy

Exercises

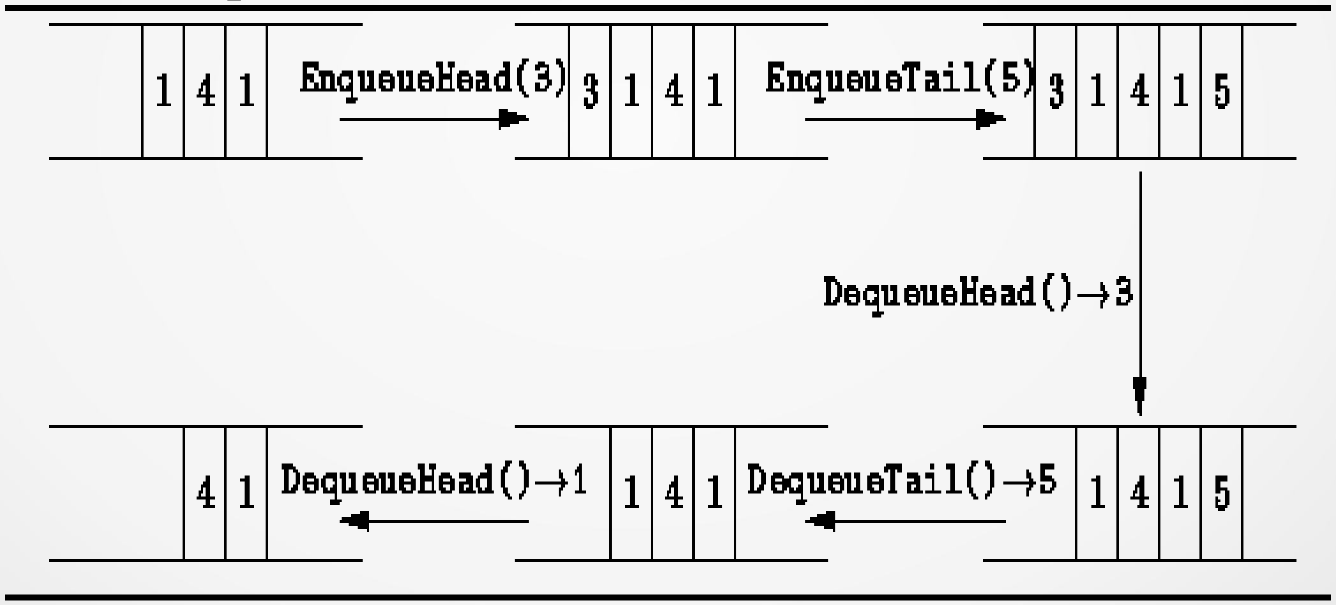
- Reverse the order of elements in a stack S using one additional queue
- Using additional non-array variables, order all elements on a queue using
  - Two additional queues
  - One additional queue
- Is it possible to keep 2 stacks in a single array, if one grows from position one of the array, and the other grows from the last position. Write a procedure PUSH(x,s) that pushes element x onto stack S, where S is one or the other of these two stacks. Include all necessary error checks.

Double-Ended Queue



Double-Ended Queues

- It is a queue like data structure that supports insertion and deletion at both ends
  - Front and rear of the queue
- Also known as deque



http://www.brpreiss.com/books/opus4/html/img751.gif

Deque Abstract Data Type

- insertFirst(*o*)
  - Insert a new object *o* at the beginning of D
  - Input: Object; Output: None
- insertLast(*o*)
  - Insert a new object *o* at the end of D
  - Input: Object; Output: None
- RemoveFirst()
  - Remove the first object of D. Error occurs if D is empty
  - Input: None; Output: None
- RemoveLast()
  - Remove the last object of D. Error occurs if D is empty
  - Input: None; Output: None

Additional Deque functions

- first()
  - Return first object of D; error occurs if D is empty
  - Input: None; Output: Object
- last()
  - Return last object of D; error occurs if D is empty
  - Input: None; Output: Object
- size()
  - Returns the number of objects in D
- isEmpty()
  - Returns a Boolean indicating if D is empty

Implementation

- Can use a dynamic array
  - uses a variant of a dynamic array that can grow from both ends, sometimes called array deques
  - Can also use circular arrays
- Dynamic Arrays
  - Works like regular arrays
  - No limit on size
    - array is discarded and replaced by a bigger array whenever necessary
    - Can use either linear or doubling strategy