

CSE 332 Autumn 2026

Lecture 1: Intro to ADTs, Queues, Stacks

Ruth Anderson

<http://www.cs.uw.edu/332>

Slides based on Nathan Brunelle's

CSE 332: Data Structures and Parallelism

Topics covered:

- Data Structures
 - Specific “classic” data structures
- Introduction to Algorithms and Analysis
- Parallelism and Concurrency
 - Parallelism: Use multiple processors to finish sooner
 - Concurrency: Correct access to shared resources

Plan for Today

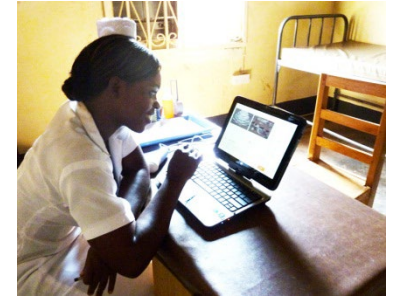
- Introductions
- Administrative Info
- Review: Queues & Stacks

Course Staff

- Instructor:
 - Ruth Anderson (rea@cs.washington.edu)
- TAs:
 - <http://www.cs.uw.edu/332/staff.html#tas>

Ruth Anderson

- Born: Winston-Salem, NC, USA
- **Grad Student at UW** in Programming Languages, Compilers, Parallel Computing
- **Taught Computer Science** at the University of Virginia for 5 years
- **Grad Student at UW**: PhD in Educational Technology, Pen Computing
- **Recent Research**: Computing and the Developing World, Computer Science Education
- **Recently Taught**: CSE 332, CSE 351, CSE 391, CSE 160
- **Activities**: Hiking, Running, getting outside



Course Info

- Text (optional):
 - Data Structures & Algorithm Analysis in Java, (Mark Allen Weiss), 3rd edition, 2012
(2nd edition also o.k.)
- Course Page:
 - <http://www.cs.uw.edu/332>

Communication

- Ed STEM Discussion board
 - Your first stop for questions about course content & assignments

Course Meetings

- Lecture
 - Materials posted (slides usually before class)
 - Recorded using Panopto
 - Ask questions, focus on key ideas (rarely coding details)
- Section
 - Practice problems!
 - Answer Java/homework questions, etc.
 - Occasionally may introduce new material
 - An important part of the course (not optional)
- Office hours
 - Use them: *please visit us!*

Grading

- 13 Weekly-ish homework exercises (all worth the same, 42% total)
 - Sometimes 2 exercises in one week
 - The lowest 1 will be dropped (Except for EX12, which cannot be dropped)
- Participation (3% total)
 - Surveys, Meet the staff, In class activities
- Midterm and final exam (55% total)
 - In-person
 - Midterm (20%): afternoon/evening on Monday 11/09
 - Final (35%): 12:30pm on Thursday 12/17

Collaboration

- Try it yourself first
- Collaborate with classmates
 - Collaboration is “whiteboard only”
 - Looking for a collaborator?
 - Post on the Ed Discussion board
 - Go to the CSE study room (Gates Center CSE2 128, there’s a table specifically for CSE 332!)
- [Allen School’s “AI as a Reference Tool” Policy](#)

Terminology

- Abstract Data Type (ADT)
 - Mathematical description of a “thing” with set of operations on that “thing”
- Algorithm
 - A high level, language-independent description of a step-by-step process
- Data structure
 - An organization of data and family of algorithms for implementing an ADT
- Implementation of a data structure
 - The data organization and algorithms written in a programming language

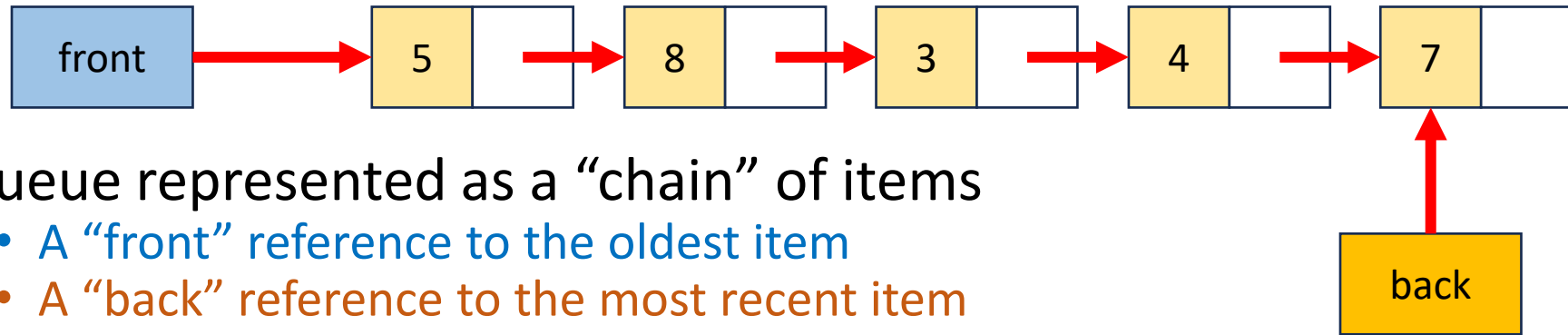
ADT: Queue

- What is it?
- What operations do we need?
- Suggested data structures?

ADT: Queue (Answers)

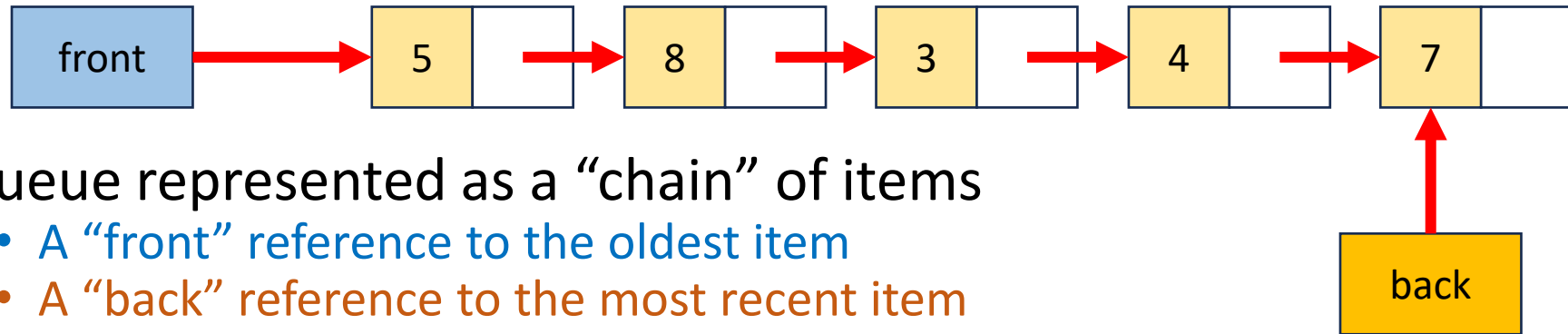
- What is it?
 - A collection of items that we interact with in a “First In First Out” (FIFO) way
- What operations do we need?
 - Enqueue
 - Add a new item to the queue
 - Dequeue
 - Remove the “oldest” item from the queue
 - IsEmpty
 - Indicate whether or not there are items still on the queue

Linked Queue Data Structure (Idea)



- Queue represented as a “chain” of items
 - A “front” reference to the oldest item
 - A “back” reference to the most recent item
 - Each Node references the item enqueued after it
- enqueue Procedure:
- dequeue Procedure:
- isEmpty Procedure:

Linked Queue Data Structure (Algorithms)



- Queue represented as a “chain” of items
 - A “front” reference to the oldest item
 - A “back” reference to the most recent item
 - Each Node references the item enqueued after it

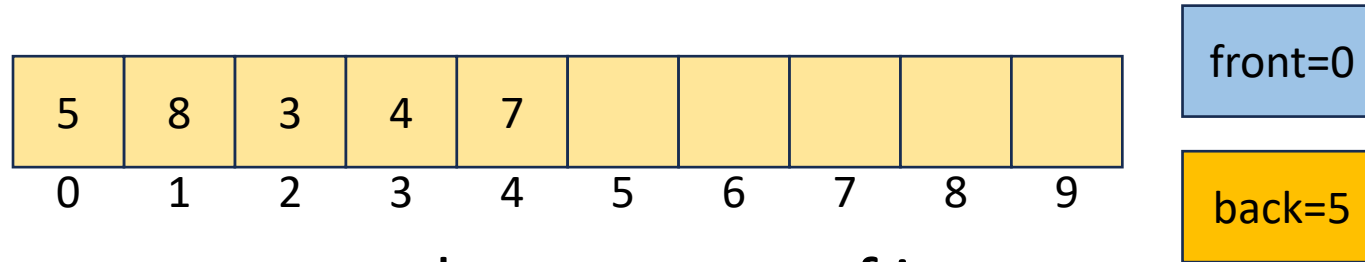
- enqueue Procedure:

```
enqueue(x){
    last = new ListNode(x);
    back.next = last;
    back = last;
}
```
- dequeue Procedure:

```
dequeue(){
    first = front.value;
    front = front.next;
    if (front == null) {back = null;}
    return first
}
```
- isEmpty Procedure:

```
isEmpty(){
    return front == null;
}
```

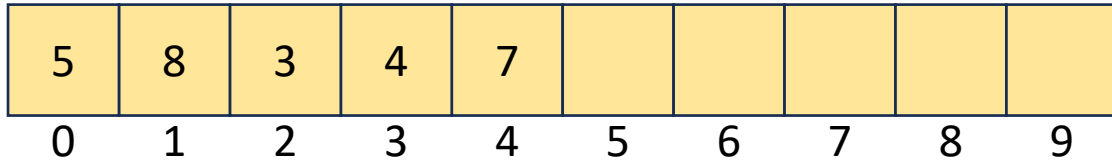
“Circular” Array Queue (Idea)



- Queue represented as an array of items
 - A “front” index to indicate the oldest item in the queue
 - A “back” index to indicate the most recent item in the queue
 - Actually, the first “open” slot in the array
- enqueue Procedure:
- dequeue Procedure:
- isEmpty Procedure:

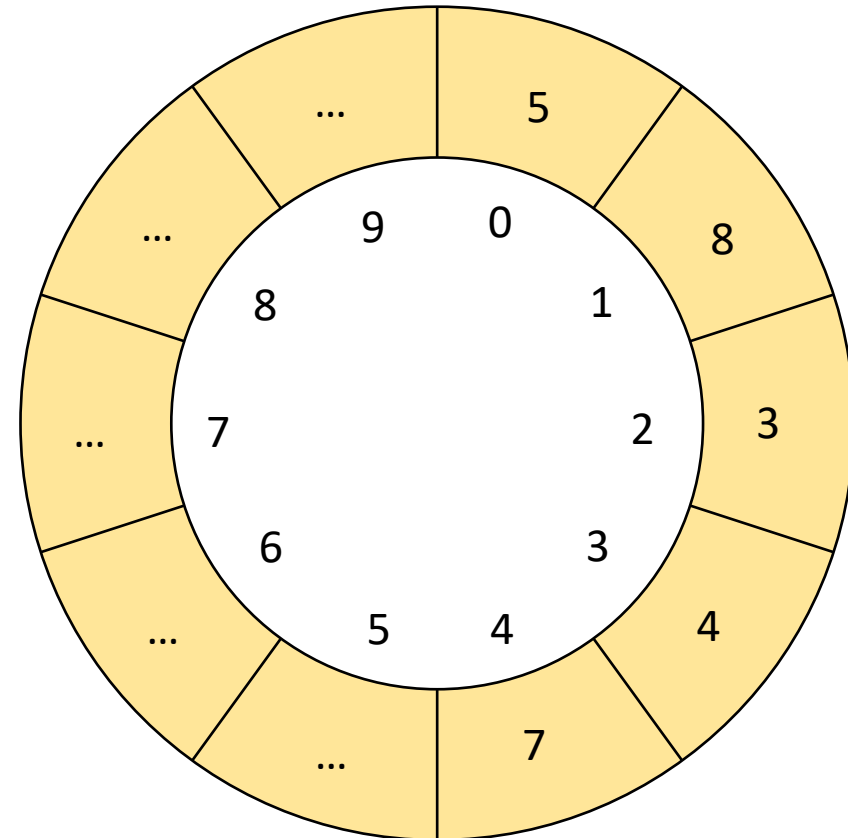
“Circular” Array

- Intuitively, An array of values arranged in a “circle” rather than a line
 - If you go beyond the last index, to wrap back around to 0

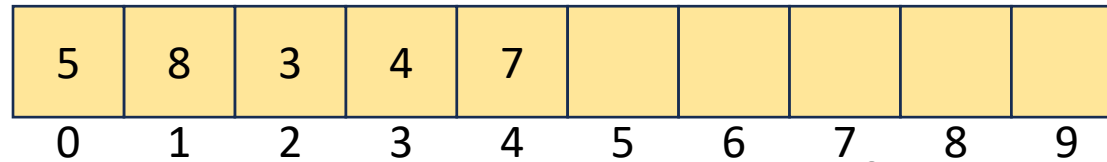


front=0

back=5



“Circular” Array Queue (Algorithms)



front=0

back=5

- Queue represented as an array of items
 - A “front” index to indicate the oldest item in the queue
 - A “back” index to indicate the most recent item in the queue

- enqueue Procedure:

```
enqueue(x){  
    queue[back] = x;  
    back = (back + 1) % queue.length;  
    size++;  
}
```

- dequeue Procedure:

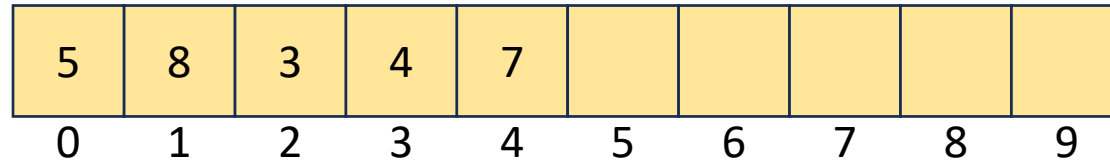
```
dequeue(){  
    // Assumes queue is not empty  
    first = queue[front];  
    front = (front + 1) % queue.length;  
    size--;  
    return first;  
}
```

- isEmpty Procedure:

```
isEmpty(){  
    return size == 0;  
}
```

What if we run
out of space?!

Resizing “Circular” Array Queue



front=0

back=5

- Queue represented as an array of items
 - A “front” index to indicate the oldest item in the queue
 - A “back” index to indicate the most recent item in the queue

How do you resize?

That’s for Exercise 0!

- enqueue Procedure:

```
enqueue(x){  
    if (size == queue.length-1) {resize();}  
    queue[back] = x;  
    back = (back + 1) % queue.length;  
    size++;  
}
```

- dequeue Procedure:

```
dequeue(){  
    // Assumes queue is not empty  
    first = queue[front];  
    front = (front + 1) % queue.length;  
    size--;  
    return first;  
}
```

- isEmpty Procedure:

```
isEmpty(){  
    return size == 0;  
}
```

Linked List vs. Circular Array

- Let's Summarize the benefits and drawbacks of each

ADT: Stack

- What is it?
 - A “Last In First Out” (LIFO) collection of items (sometimes called FILO)
- What operations do we need?
 - push
 - Add a new item onto the stack
 - peek
 - Return the value of the most recently pushed item
 - pop
 - Return the value of the most recently pushed item and remove it from the stack
 - isEmpty
 - Indicate whether or not there are items still on the stack