CSE 332
Data Structures & Parallelism

Intro, Stacks & Queues

Melissa Winstanley
Spring 2024

Welcome!

We have 10 weeks to learn fundamental data structures and algorithms for organizing and processing information

- “Classic” data structures / algorithms and how to analyze rigorously their efficiency and when to use them
- Queues, dictionaries, graphs, sorting, etc.
- Parallelism and concurrency (!)

Today

- Introductions
- Administrative Info
- What is this course about?
- Review: Queues and stacks

CSE 332 Course Staff

Instructor:

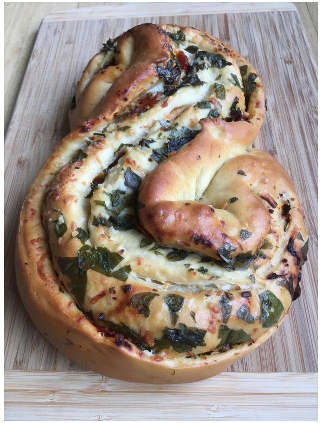
Melissa Winstanley

Teaching Assistants:

- Amanda Yuan
- Arya GJ
- Chandni Rajasekaran
- Hans Easton
- Hitesh Boinpally
- Mohamed Awadalla
- Nile Camai
- Xunmei Liu
- Yijia Zhao

Me (Melissa)

- Undergrad and 5th year masters at UW
  - Research in NLP & computing in the developing world
  - TA-ing (intro, networks) and teaching my own class
- Teaching Computer Science at UW
  - CSE 143 (Intro II) / CSE 374 (Systems Programming) / DATA 515 (Software Design)
- Software Engineer at Meta, Convoy, & Snowflake
  - Focus on large-scale backend infrastructure and platforms
  - Extensive work with intern programs
  - Passionate about making students into *software engineers*



Credit for the course

Many thanks to:

- Ruth Anderson
- Winston Jodjana
- Nathan Brunelle

Today

- Introductions
- Administrative Info
- What is this course about?
- Review: Queues and stacks

Course Information

- **Instructor:** Melissa Winstanley, CSE 214
  - Office Hours: see course web page, and by appointment (mwinst@cs.washington.edu)
- **Course Web page:**
  - http://www.cs.uw.edu/332
- **Text (optional):**
  Data Structures & Algorithm Analysis in Java
  Mark Allen Weiss, 3rd edition, 2012
  (2nd edition also o.k.)

Communication

- **Course email list:** cse332a_sp24@uw
  - You are already subscribed
  - You must get and read announcements sent there
- **Ed STEM Discussion board**
  - Your first stop for questions about course content & assignments
- **Anonymous feedback link**
  - For good and bad: if you don’t tell us, we won’t know!

Course Meetings

- **Lecture**
  - Materials posted (sometimes afterwards), but take notes
  - Ask questions, focus on key ideas (rarely coding details)
  - No participation points - we’re all adults here
- **Section**
  - Practice problems!
  - Answer Java/project/homework questions, etc.
  - Occasionally may introduce new material
  - An important part of the course (not optional)
  - ESPECIALLY THIS WEEK!
- **Office hours**
  - Use them: please visit us

Course Materials

- Lecture and section materials will be posted
  - But they are visual aids, not always a complete description!
  - If you have to miss, find out what you missed
- Textbook: Weiss 3rd Edition in Java
  - Good read, but only responsible for lecture/section/hw topics
- Parallelism / concurrency units in separate free resources designed for 332

Course Work

- ~13 Weekly individual homework exercises (25%)
- 3 programming projects (with phases) (35%)
  - Use Java and IntelliJ, Gitlab
  - Done individually
- Midterm and final exam (40%)
  - In-person, in this room (CSE2 G20)
  - Dates:
    - Midterm: Friday April 26, during lecture
    - Final Exam: Thursday June 8, 8:30-10:20am

Collaboration policy

- Attempt the work on your own
- Then you can talk with classmates
  - “Whiteboard collaboration”
  - Do NOT take any photos/notes away from the session
  - Wait 30 minutes before writing your own solution
  - ONLY people from this iteration of the class
- Search the internet for *general concepts only*
- Cite any sources / collaboration
  - When in doubt, cite!

Homework for today!!!

1. **Preliminary Survey**: due Thursday
2. **Project #1**: Checkpoint 0 due Friday
3. **Review** Java & install IntelliJ
4. **Reading** (optional) in Weiss (see course web page)

Today

- Introductions
- Administrative Info
- What is this course about?
- Review: Queues and stacks

Data Structures + Parallelism

- About 70% of the course is a “classic data-structures course”
  - Timeless, essential stuff
  - Core data structures and algorithms that underlie most software
  - How to analyze algorithms
- Plus a serious first treatment of programming with *multiple threads*
  - For *parallelism*: Use multiple processors to finish sooner
  - For *concurrency*: Correct access to shared resources
  - Will make many connections to the classic material

What 332 is about

- Deeply understand the basic structures used in all software
  - Understand the data structures and their trade-offs
  - Rigorously analyze the algorithms that use them (math!)
  - Learn how to pick "the right thing for the job"
- Experience the purposes and headaches of multithreading
- Practice design, analysis, and implementation
  - The elegant interplay of "theory" and "engineering" at the core of computer science

Goals

- You will understand:
  - what the tools are for storing and processing common data types
  - which tools are appropriate for which need
- So that you will be able to:
  - make good design choices as a developer, project manager, or system customer
  - justify and communicate your design decisions

One view of this course

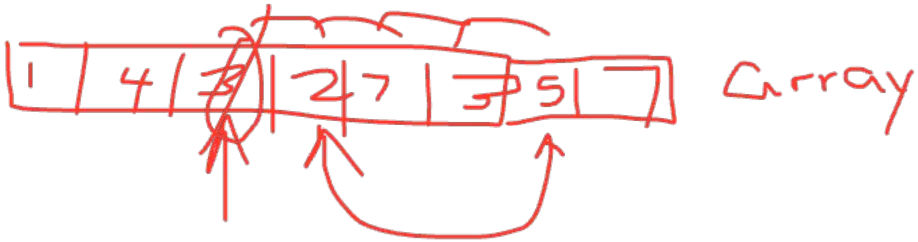
This is the class where you begin to think like a computer scientist

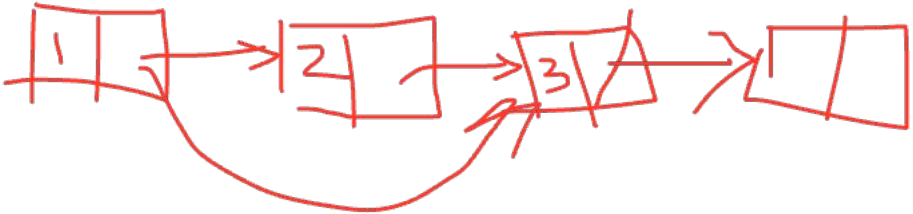
- You stop thinking in Java code
- You start thinking that this is a hashtable problem, a stack problem, etc.

Data Structures?

"Clever" ways to organize information in order to enable efficient computation over that information.

Example Trade-Offs





Trade-Offs

- A data structure strives to provide many useful, efficient operations
- But there are unavoidable trade-offs:
  - Time vs. space
  - One operation more efficient if another less efficient
  - Generality vs. simplicity vs. performance
- That is why there are many data structures and educated CSEers internalize their main trade-offs and techniques
  - And recognize logarithmic < linear < quadratic < exponential

Getting Serious: Terminology

List?

Abstract

- 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
  - A specific organization of data and family of algorithms for implementing an ADT
- Implementation of a data structure
  - A specific implementation in a specific language

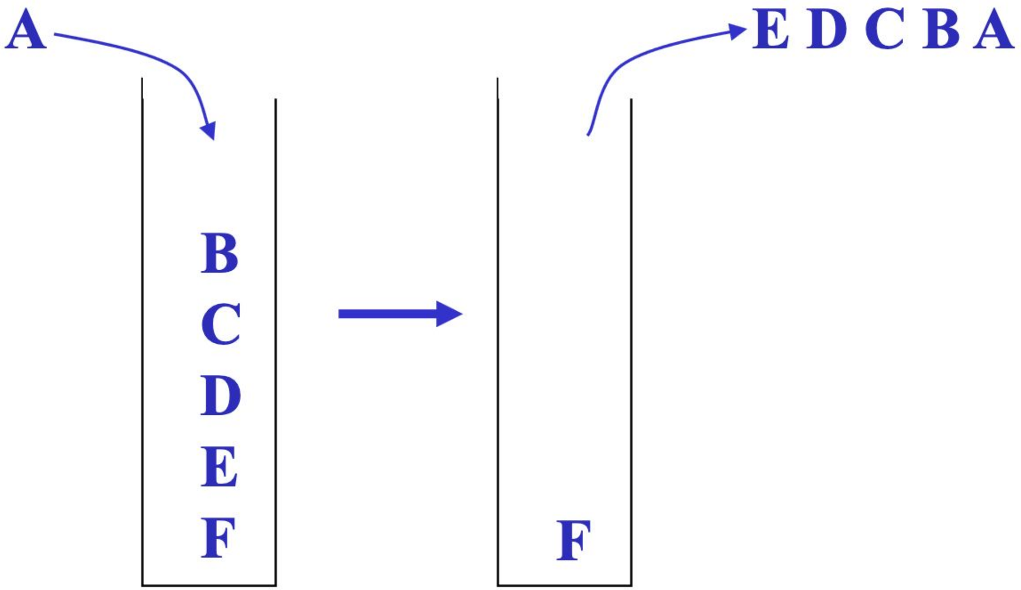
Concrete

The Stack ADT

Last in First Out

Stack operations:

- `push`
- `pop`
- `top/peek`
- `is_empty`



Terminology Example: Stacks

- The ***Stack*** ADT supports operations:
  - **`push`**: adds an item
  - **`pop`**: raises an error if isEmpty, else returns most-recently pushed item not yet returned by a pop
  - **`isEmpty`**: initially true, later true if there have been same number of pops as pushes
  - … (Often some more operations)
- A Stack data structure could use a linked-list or an array or something else, and associated algorithms for the operations
- One implementation is in the library **`java.util.Stack`**

Why useful

The ***Stack*** ADT is a useful abstraction because:

- It arises all the time in programming (see Weiss for more)
  - Recursive function calls
  - Balancing symbols (parentheses)
  - Evaluating postfix notation: 3 4 + 5 *
  - Clever: Infix ((3+4) * 5) to postfix conversion (see Weiss)
- We can code up a reusable library
- We can communicate in high-level terms
  - "Use a stack and push numbers, popping for operators…"
  - Rather than, "create a linked list and add a node when…"

Today

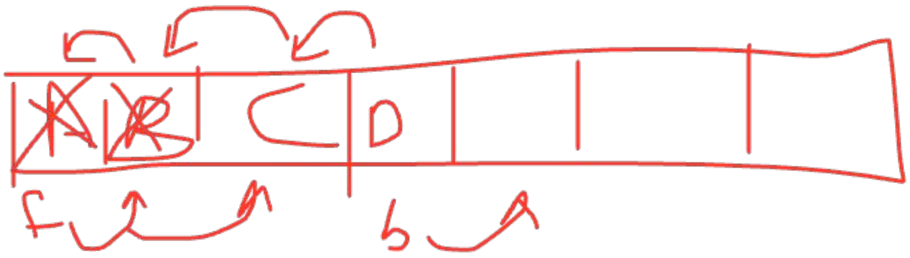
- Introductions
- Administrative Info
- What is this course about?
- Review: Queues and stacks

The Queue ADT

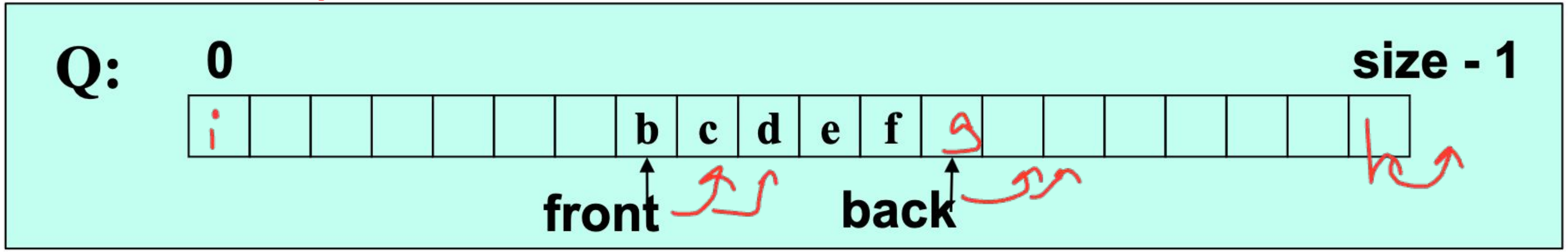
FIFO

Queue operations:

- `enqueue`
- `dequeue`
- `is_empty`



Circular Array Queue Data Structure



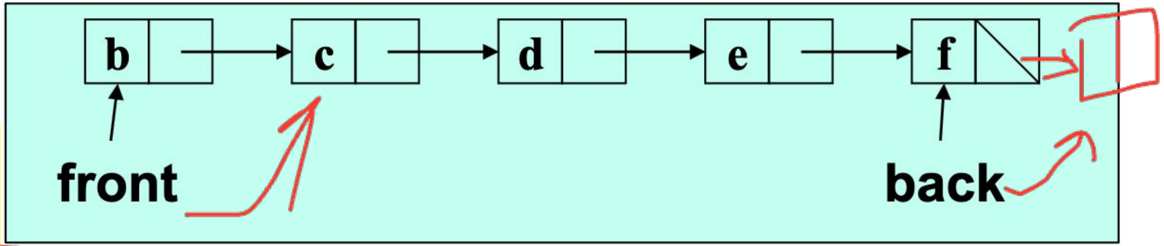

```
// Basic idea only!
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size
}
```

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- What if ***array*** is full?

```
// Basic idea only!
dequeue() {
  x = Q[front];
  front = (front + 1) % size;
  return x;
}
```

- How to *test* for empty?
- What is the *complexity* of the operations?

Linked List Queue Data Structure




```
// Basic idea only!
enqueue(x) {
  back.next = new Node(x);
  back = back.next;
}
```

```
// Basic idea only!
dequeue() {
  x = front.item;
  front = front.next;
  return x;
}
```

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- Can ***list*** be full?
- How to *test* for empty?
- What is the *complexity* of the operations?

Circular Array vs Linked List

Good about

circular arra
y:
- if we do w
ant
to modify - l
ess
error prone
- storing onl
y
one piece o
f
data

Good about the linked list?
- no set size - don't have define size at the start
- add elements in between easily

Circular Array vs Linked List

Array:

- May waste unneeded space or run out of space
- Space per element excellent
- Operations very simple / fast

Operations not in Queue ADT, but also:

- Constant-time “access to kth element”
- For operation “insertAtPosition”, must shift all later elements

List:

- Always just enough space
- But more space per element
- Operations very simple / fast

Operations not in Queue ADT, but also:

- No constant-time “access to kth element”
- For operation “insertAtPosition” must traverse all earlier elements

Homework for today!!!

1. **Preliminary Survey**: due Thursday
2. **Project #1**: Checkpoint 0 due Friday
3. **Review** Java & install IntelliJ
4. **Reading** (optional) in Weiss (see course web page)