CSE 373: Data Structures and Algorithms

Lecture 1: Introduction; ADTs; Stacks/Queues

Riley Porter

Winter 2017

Welcome!

This course is about *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.

Today in class:

- Introductions and course mechanics
- What this course is about
- Start *abstract data types* (ADTs), *stacks*, and *queues*

Overloading

Go to :

Overload Form link:

https://catalyst.uw.edu/webq/survey/cseadv/321439

or

http://tinyurl.com/hz9sxzd

Code Word: Given in Lecture

Do this by 1 hour after this lecture!

Course staff

Instructor: Riley Porter, CSE 450, rileymp2@cs.washington.edu

TA: Zelina Chen : zelinac@cs.washington.edu
TA: Paul Curry : paulmc@cs.washington.edu
TA: Josh Curtis : curtijd@cs.washington.edu
TA: Chloe Lathe : lathec@cs.washington.edu
TA: Trung Ly : trungly@cs.washington.edu
TA: Matthew Rockett : rockettm@cs.washington.edu
TA: Kyle Thayer : kthayer@cs.washington.edu
TA: Raquel Van Hofwegen : raqvh@cs.washington.edu
TA: Pascale Wallace Patterson : pattersp@cs.washington.edu
TA: Rebecca Yuen : rebyuen@cs.washington.edu
TA: Hunter Zahn : hzahn93@cs.washington.edu

Concise to-do list

In next 24-48 hours:

- Take homework 0 (worth 0 points) as Catalyst quiz
- Read/skim Chapters 1 and 3 of Weiss book
  - Relevant to Homework 1, due next week
- Set up your Java environment for Homework 1
- Check out the Course Website and read all the course policies:

https://courses.cs.washington.edu/courses/cse373/17wi

Communication

- Course email list: **cse373_wi17@u.washington.edu**
  - Students and staff already subscribed
  - Fairly low traffic
- Course staff: **cse373-staff@cs.washington.edu** plus individual emails
- Discussion board
  - For appropriate discussions; TAs will monitor
  - Encouraged, but won't use for important announcements
- Anonymous feedback link
  - For good and bad: if you don't tell me, I don't know

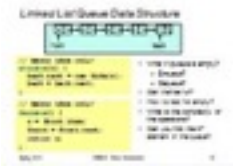
Course meetings

- Lectures
  - Materials posted on the website
  - Ask questions, focus on key ideas
  - Not recorded
- Sections on Thursdays
  - Programming practice, homework prep
  - Math review and example exam problems
  - Would be a really bad idea not to go, won't always post all materials on the website

Office Hours

- Riley: Tuesday 1:30 – 3:20pm in CSE 450
  - Use them: *please visit me... I have candy...*
  - Ideally not *just* for homework questions (but that's OK too)
- TA's: To be determined – will be posted on the website today/tomorrow

Course materials



- All 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/hw topics
  - 3rd edition improves on 2nd, but we'll support the 2nd

- A good Java reference of your choosing
  - Google is only so helpful

Computing

- College of Arts & Sciences Instructional Computing Lab
  - http://depts.washington.edu/aslab/
    - Communications building
  - Or your own machine
- Will use Java 8 for the programming assignments
- Eclipse is recommended programming environment

Course Work

- ~6 homework assingments (50%)
  - Most involve programming, but also written questions
  - Higher-level concepts than "just code it up"
  - First programming assignment released soon and due next week
- Midterm (20%): TBD. Will announce more about this in the coming week.
- Final (30%): Tuesday, March 14th, 2:30-4:20

Collaboration and Academic Integrity

- Read the course policy very carefully
  - Explains quite clearly how you can and cannot get/ provide help on homework and projects

- Always explain any unconventional action on your part
  - When it happens, when you submit, not when asked

Honest work is the most important feature of a university

Academic Honesty Details

- You are expected to do your own work
  - Exceptions (group work), if any, will be clearly announced
- Sharing solutions, doing work for, or accepting work from others is cheating
- Referring to solutions from this or other courses from previous quarters is cheating
- But you can learn from each other: see the policy

Moar Academic Honesty

You spend at least 30 minutes on each and every problem (or programming assignment) alone, before discussing it with others.

Cooperation is limited to group discussion and brainstorming. No written or electronic material may be exchanged or leave the brainstorming session.

You write up each and every problem in your own writing, using your own words, and fully understanding your solution (similarly you must write code on your own).

You identify each person that you collaborated with at the top of your written homework or in your README file.

What 373 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"
  - More thorough and rigorous take on topics introduced in CSE143 (plus more new topics)
- Practice design, analysis, and implementation
  - The elegant interplay of "theory" and "engineering" at the core of computer science
- More programming experience (as a way to learn)

Goals

- Be able to make good design choices as a developer, project manager, etc.
  - Reason in terms of the general abstractions that come up in all non-trivial software (and many non-software) systems
- Be able to justify and communicate your design decisions

Dan Grossman's take:

- Key abstractions used almost every day in just about anything related to computing and software
- It is a vocabulary you are likely to internalize permanently

In CSE 143 (Assumed Background)

- Fundamentals of computer science and object oriented programming
  - Variables, conditionals, loops, methods, fundamentals of defining classes and inheritance, arrays, single linked lists, simple binary trees, recursion, some sorting and searching algorithms, basic algorithm analysis (e.g., $O(n)$ vs $O(n^2)$ and similar things)
  - What other data structures were in 143?

143 vs 373

- 143: Showed you how to use data structures (be the Client vs the Implementor)

- 373:
  - Provide you with the tools to understand when and why one would use certain data structures/ algorithms over others
    - And to be able to implement your own!
  - problem solving and thinking critically

Topics Outline

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues
- Trees, Hashing, Dictionaries
- Heaps, Priority Queues
- Sorting
- Disjoint Sets
- Graph Algorithms
- *May have time for other brief exposure to topics, maybe parallelism, technical interview questions, dynamic programming*

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of a "thing" with set of operations
- 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

Data structures

(Often highly *non-obvious*) ways to organize information to enable *efficient* computation over that information

A data structure supports certain *operations*, each with a:

- Meaning: what does the operation do/return
- Performance: how efficient is the operation

Examples:

- ***List*** with operations **`insert`** and **`delete`**
- ***Stack*** with operations **`push`** and **`pop`**

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

We ask ourselves questions like:

- Does this support the operations I need efficiently?
- Will it be easy to use, implement, and debug?
- What assumptions am I making about how my software will be used? (E.g., more lookups or more inserts?)

Array vs Linked List

Array:

- May waste unneeded space or run out of space
- Space per element excellent
- Constant-time access to k^{th} element
- For operation insertAtPosition, must shift all later elements

List:

- Always just enough space
- Slightly more space per element
- No constant-time access to k^{th} element
- For operation insertAtPosition must traverse all earlier elements

ADT vs. Data Structure vs. Implementation

"Real life" Example (not perfect)

ADT: Automobile

- Operations: Accelerate, decelerate, etc…

Data Structure: Type of automobile

- Car, Motorcycle, Truck, etc…

Implementation (of Car):

- 2009 Honda Civic, 2001 Subaru Outback, …

Example: Stacks

- The ***Stack*** ADT supports operations:
  - **`isEmpty`**: have there been same number of pops as pushes
  - **`push`**: takes an item
  - **`pop`**: raises an error if empty, else returns most-recently pushed item not yet returned by a pop
  - … (possibly more operations)

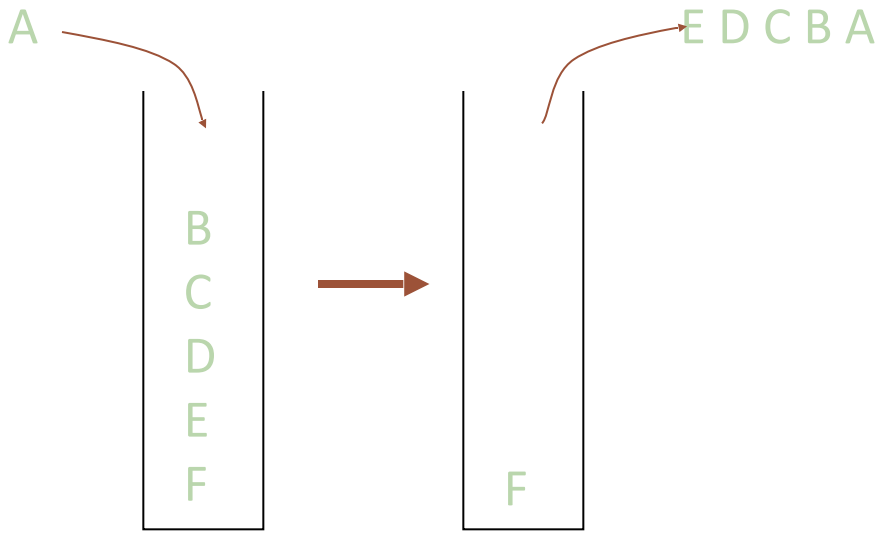
- A Stack data structure could use a linked-list or an array or something else, and associated algorithms for the operations

- One implementation is in the library **`java.util.Stack`**

The Stack ADT

Operations:

- `create`
- `destroy`
- `push`
- `pop`
- `top`
- `is_empty`



Can also be implemented with an array or a linked list

- This is Homework 1 (released soon, due next week)
- Type of elements is irrelevant

Why Stack ADT is useful

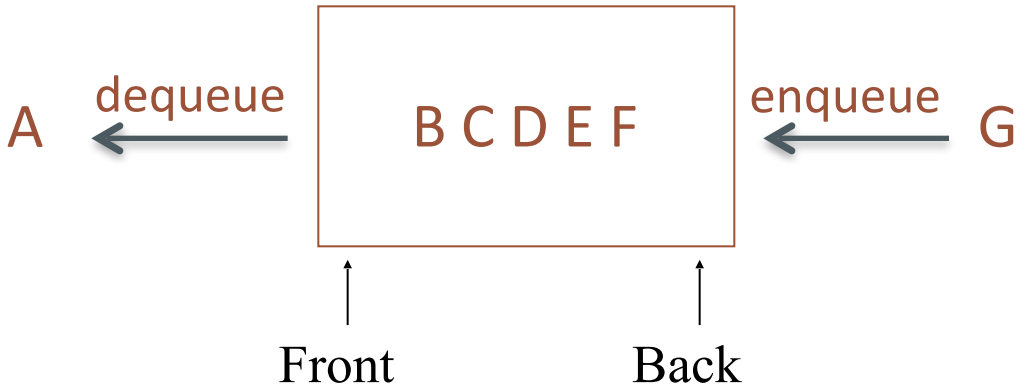
- It arises all the time in programming (e.g., see Weiss 3.6.3)
  - Recursive function calls
  - Balancing symbols (parentheses)
  - Evaluating postfix notation: 3 4 + 5 *
  - Clever: Infix ((3+4) * 5) to postfix conversion (see text)

- 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…"

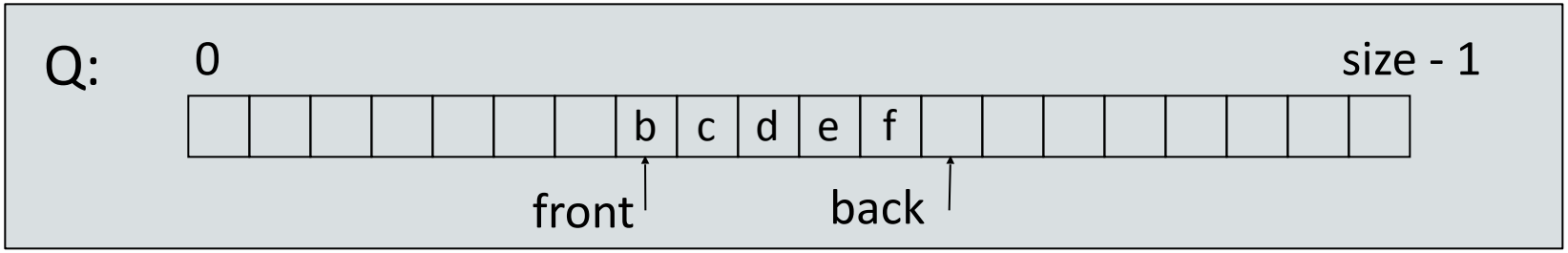
The Queue ADT

- Operations
  `create`
  `destroy`
  `enqueue`
  `dequeue`
  `is_empty`



- Just like a stack except:
  - Stack: LIFO (last-in-first-out)
  - Queue: FIFO (first-in-first-out)
- Just as useful and ubiquitous

Circular Array Queue Data Structure



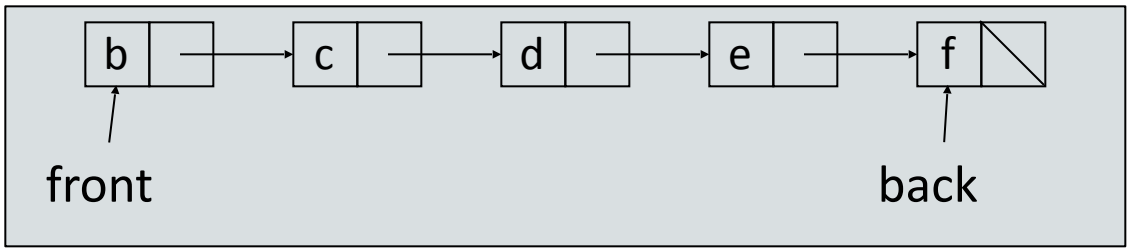

```
// Basic idea only!
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size
}
```

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

Considerations:

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- What if ***array*** is full?
- How to *test* for empty?
- What is the *complexity* of the operations?
- Can you find the k^{th} element in the queue?

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;
}
```

Considerations:

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- Can ***list*** be full?
- How to *test* for empty?
- What is the *complexity* of the operations?
- Can you find the k^{th} element in the queue?

Data Structure Analysis Practice

For each of the following, pick the best Data Structure (**Stack, Queue, either, or neither**) and Implementation (**Array, Linked List, either, or neither**):

- Maintain a collection of customers at a store with a relatively constant stream of customers at all times
- Keep track of a ToDo list
- Maintain a sorted student directory
- Manage the history of webpages visited to be used by the "back" button
- Store data and access the *kth* element often

Pseudocode

Describe an algorithm in the steps necessary, write the shape of the code but ignore specific syntax.

Algorithm: Count all elements in a list greater than x

Pseudocode:

```
int counter // keeps track of number > x
while list has more elements {
    increment counter if current element is > than x
    move to next element of list
}
```

Pseudocode Example 2

Algorithm: Given a list of names in the format "firstName lastName", make a Map of all first names as keys with sets of last names as their values

Pseudocode:

```
create the empty result map
while list has more names to process {
    firstName is name split up until space
    lastName is name split from space to the end
    if firstName not in the map yet {
       put firstName in map as a key with an empty
       set as the value
    }
    add lastName to the set for the first name
    move to the next name in the list
}
```

Review Java Programming

- Course content from most recent 143 offerings:
  - Stuart's Calendar (includes videos of lectures)
  - Zorah's Calendar
  - Adam's Content

- Adam's 14X Unofficial Style Guide although, we will be less strict than the 14X courses here

- Practice-It to practice sample problems from 14X