CSE 326: Data Structures

Ruth Anderson
Winter Quarter 2010
Lecture 1

CSE 326 Course Staff

Instructor: Ruth Anderson

Teaching Assistants:
- Patrick Healy
- Daniel Jones
- Tyler Robison

UNIVERSITY of VIRGINIA

Me (Ruth Anderson)

- Grad Student at UW (Programming Languages, Compilers, Parallel Computing)
- Taught Computer Science at the University of Virginia for 5 years
- Grad Student at UW (Educational Technology, Pen Computing)
- Defended my PhD in fall 2006
- Computing and the Developing World
- Recently taught cse142, cse143, cse326, cse373, compilers, programming languages, architecture

Today's Outline

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

Course Information

- **Instructor**: Ruth Anderson, CSE 360
  Office Hours: M & W 3:30-4:30 and by appointment, (rea@cs.washington.edu)
- **Text**: *Data Structures & Algorithm Analysis in Java*, (Mark Allen Weiss), 2nd Edition, 2007
- **Course Web page**: `http://www.cs.washington.edu/326`

Communication (1)

Instructors
- cse326-instr@cs.washington.edu DO not use until Tuesday!
- (or our individual addresses)

Announcements
- cse326a_wi10@u.washington.edu
- (you are automatically subscribed @u)
- You are responsible for traffic on this list
- Will be archived on the course web page

Communication (2)

Discussion

- Go-Post Discussion board linked off course webpage
- Use your real name and picture

Feedback Always Welcome

- See anonymous link on webpage

Course Mechanics

- Written Homeworks (8 total)
  - › Due at the start of class on due date (Fridays)
  - › No late homeworks accepted
  - › Lowest homework grade dropped
- Programming Projects (3 total)
  - › In Java
  - › Turned in electronically (dates vary) and on paper
  - › Once per quarter: use your "late day" for extra 24 hours – **Must email TA**
- Work in teams only on explicit team projects
  - › Appropriate *discussions* encouraged – see website

Course Mechanics(2)

- Approximate Grading

25% - Written Homework Assignments

25% - Programming Projects

20% - Midterm Exam

25% - Final Exam

5% - Best of the four items above.

Project/Homework Guides

On the website - note especially:

- Gilligan's Island rule applies.

Homeworks: Use pseudocode, not code.

- A human being is reading your homeworks.
- See website for pseudocode example.

Projects: correctness of code is only 40% of your grade!

- Spend time commenting your code as you write - it will help you be a better programmer.

Section

What happens there?

- Answer questions about current homework
- Previous homeworks returned and discussed
- Discuss the project (getting started, getting through it, answering questions)
- Finer points of Java
- Reinforce lecture material
- Occasionally introduce new material

Homework for Today!!

1) **Project #1:** (released by Wednesday) bring questions to section on Thursday
2) **Preliminary Survey**: fill out by evening of Friday January 8th
3) **Information Sheet**: bring to lecture on Friday January 8th
4) **Reading** in Weiss (see handout)

Reading

- Reading in *Data Structures and Algorithm Analysis in Java*, 2nd Ed., 2007 by Weiss
- For this week:
  - › Chapter 1 – (review) Mathematics and Java
  - › Chapter 3 – (Project #1) Lists, Stacks, & Queues
  - › Chapter 2 – (Topic for Wednesday) Algorithm Analysis

Bring to Class on Friday:

- Name
- Email address
- Year (1,2,3,4)
- Hometown
- Interesting Fact or what I did over summer/break.

Today's Outline

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

Class Overview

Introduction to many of the basic data structures used in computer software
- › Be exposed to a variety of data structures
- › Know when to use them
- › Practice mathematical techniques for analyzing the algorithms that use them
- › Practice implementing and using them by writing programs

Goal:
- be able to make good design choices as a developer, project manager, or system customer
- be able to justify and communicate your design decisions

Data Structures

"Clever" ways to organize information in order to enable efficient computation

- › What do we mean by clever?
- › What do we mean by efficient?

Picking the best Data Structure for the job

- The data structure you pick needs to *support* the operations you need
- Ideally it supports the operations you will use most often in an *efficient* manner
- Examples of operations:
  - › List ADT with operations **`insert`** and **`delete`**
  - › Stack ADT with operations **`push`** and **`pop`**

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of an object with set of operations on the object. Useful building block.
- Algorithm
  - A high level, language independent, description of a step-by-step process
- Data structure
  - A specific family of algorithms for implementing an abstract data type.
- Implementation of data structure
  - A specific implementation in a specific language

Terminology examples

- A stack is an ***abstract data type*** supporting push, pop and isEmpty operations
- A stack ***data structure*** could use an array, a linked list, or anything that can hold data
- One stack ***implementation*** is found in java.util.Stack

Why So Many Data Structures?

Ideal data structure:

"fast", "elegant", memory efficient

Generates tensions:

- time *vs.* space
- performance *vs.* elegance
- generality *vs.* simplicity
- one operation's performance *vs.* another's

The study of data structures is the study of tradeoffs. That's why we have so many of them!

Today's Outline

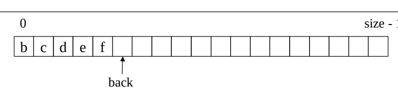
- Introductions
- Administrative Info
- What is this course about?
- Review: Queues and stacks

First Example: Queue ADT

- Queue operations

  create
  destroy
  enqueue
  dequeue
  is_empty

Array Queue Data Structure



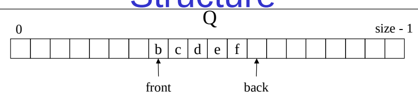

```
enqueue(Object x) {
   Q[back] = x
   back = (back + 1)
   }
dequeue() {
   x = Q[0]
   shiftLeftOne()
   return x
   }
```

What's missing?

How to find K-th element in the queue?

What is complexity of these operations?

Circular Array Queue Data Structure




```
enqueue(Object x) {
   Q[back] = x ;
   back = (back + 1) % size
   }
dequeue() {
   x = Q[front] ;
   front = (front + 1) % size;
   return x ;
   }
```

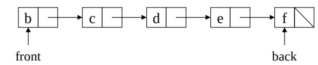
How test for empty list?

How to find K-th element in the queue?

What is complexity of these operations?

Limitations of this structure?

Linked List Queue Data Structure



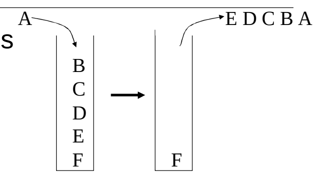

```
void enqueue(Object x) {
  if (is_empty())
       front = back = new Node(x)
  else
       back->next = new Node(x)
       back = back->next
}
bool is_empty() {
  return front == null
}
```

```
Object dequeue() {
   assert(!is_empty)
   return_data = front->data
   temp = front
   front = front->next
   delete temp
   return return_data
}
```


Circular Array vs. Linked List

Second Example: Stack ADT

- Stack operations
  - › create
  - › destroy
  - › push
  - › pop
  - › top
  - › is_empty



Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating Postfix Notation

Homework for Today!!

1) **Project #1:** (released by Wednesday) bring questions to section on Thursday
2) **Preliminary Survey**: fill out by evening of Friday January 8th
3) **Information Sheet**: bring to lecture on Friday January 8th
4) **Reading** in Weiss