CSE 326: Data Structures

Spring 2008

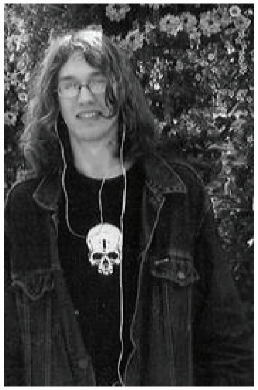
Brian Curless

Lecture 1

CSE 326 Crew

- Instructor: Brian Curless, CSE 664
- TAs:

Laura Effinger-Dean



Ray Smith

Today's Outline

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

Course Information

Web page:

`http://www.cs.washington.edu/326`

Text: Weiss, *Data Structures & Algorithm Analysis in Java*, 2nd Edition, 2007.

Errata: handout.

Communication

Instructors

- cse326-instr@cs.washington.edu
- (or our individual addresses)

Announcements

- cse326a_sp08@u.washington.edu
- (you are automatically subscribed @u)

Discussion

- Discussion board linked off home page

Written homeworks

Written homeworks (8 total)

- Due at the **start of class** on due date
- Typically assigned/due on Fridays
- Latex encouraged
- No late homeworks accepted

Projects

- Programming projects (3 total, with phases)
  - In Java
  - Eclipse encouraged
  - Turned in electronically
  - Can have one "late day" for extra 24 hours
    Must email TA in advance
- Work in teams only on explicit team projects
  - Appropriate *discussions* encouraged – see website

Overall grading

Grading

25% - Written Homework Assignments
25% - Programming Assignments
20% - Midterm Exam (in class, fifth week)
25% - Final Exam
5% - Best of Programming or Exams

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: code is only 40% of your grade!
- Spend time commenting your code as you write - it will help you be a better programmer.

Section

Run by Laura:

- AA – Thurs 12:30 - 1:20 – EEB 025
- AB – Thurs 1:30 - 2:20 – EEB 003

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 and environs
- Reinforce lecture material

Homework for Today!!

Reading in Weiss

Chapter 1 – (Review) Mathematics and Java

Chapter 2 – (Next lecture) Algorithm Analysis

Chapter 3 – (Project #1) Lists, Stacks, & Queues

Today's Outline

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

Data…

Music
Images >2B
Backups
Dictionaries
Documents/Databases
Software/code
Email 300k, 8GB
Social networks

Networks (internet) ~1B
Usage stats
Web ~10B
Graphics
DNA ~3B
Fingerprints
Astronomy 10^{21}
Linux 5-10M, Windows 50M

…Structures…

Queue
Trees
Hash (tables)
Linked lists
Arrays
Associative arrays
Graphs

(…and Algorithms)

Sort
Search
Insert
Remove

Class Overview

- Introduction to many of the basic data structures used in computer software:
  - Understand the data structures.
  - Analyze the algorithms that use them.
  - Know when to apply them.
- Practice design and analysis of data structures.
- Practice using these data structures by writing programs.
- Make the transformation from programmer to computer scientist.

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 can
  - make good design choices as a developer, project manager, or system customer
- You will be able to
  - *Justify* your design decisions via formal reasoning
  - *Communicate* ideas about programs clearly and precisely

Goals

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

Linus Torvalds, 2006

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:
  - A ***List*** with operations **`insert`** and **`delete`**
  - A ***Stack*** 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 organization of the data to accompany algorithms for 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 java.util.Stack; another is java.util.LinkedList

Concepts vs. Mechanisms

Concepts

- Abstract
- Pseudocode
- Algorithm
  - A sequence of high-level, language independent operations, which may act upon an abstracted view of data.
- Abstract Data Type (ADT)
  - A mathematical description of an object and the set of operations on the object.

Mechanisms

- Concrete
- Specific programming language
- Program
  - A sequence of operations in a specific programming language, which may act upon real data in the form of numbers, images, sound, etc.
- Data structure
  - A specific way in which a program's data is represented, which reflects the programmer's design choices/goals.

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

- FIFO: First In First Out
- Queue operations

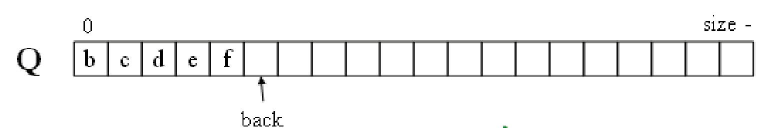
  create
  destroy
  enqueue
  dequeue
  is_empty

Queues in practice

- Print jobs
- File serving
- Phone calls and operators

(Later, we will consider "priority queues.")

Array Queue Data Structure



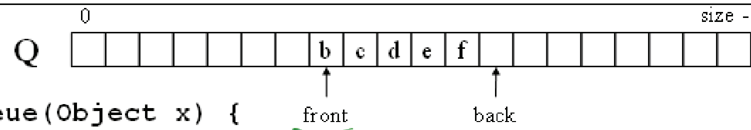

```
enqueue(Object x) {    assert(!is_full())
   Q[back] = x
   back = (back + 1)
}

dequeue() {    assert(!is_empty())
   x = Q[0]
   shiftLeftOne()
   return x
}
```

What's missing in these functions?

How to find K-th element in the queue? Q[K]

What is complexity of these operations?

Circular Array Queue Data Structure




```
enqueue(Object x) {
   assert(!is_empty())
   Q[back] = x
   back = (back + 1)%size
}

dequeue() {
   if (is_full()))
     do something...
   x = Q[front]
   front = (front + 1)%size
   return x
}
```

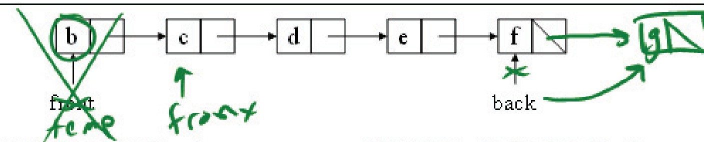
How test for empty/full list?

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

Fast random access

Fixed size (or extra work)

Simpler?

Less memory
(Not so easy)

Might have better
memory performance

Slow..

Grows adaptively

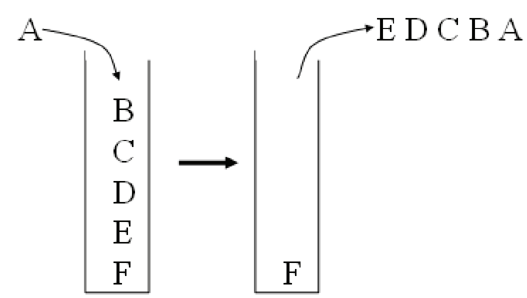
Trickier/more complex?

More memory

Insert/del. easy

Second Example: Stack ADT

- LIFO: Last In First Out
- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is_empty



Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating postfix or “reverse Polish” notation

Reminder: homework

Reading in Weiss

Chapter 1 – (Review) Mathematics and Java

Chapter 2 – (Next lecture) Algorithm Analysis

Chapter 3 – (Project #1) Lists, Stacks, & Queues