CSE 326: Data Structures

James Fogarty

Autumn 2007

Lecture 1

CSE 326 Crew

- Instructor:
  - James Fogarty, CSE 666
- TAs:
  - Peter Henry
  - Bo Qin
- Email:
  - cse326-instr@cs.washington.edu
- Office Hours:
  - TBA

Today's Outline

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

Course Information

- **Text**: *Data Structures & Algorithm Analysis in Java*, (Mark Allen Weiss), 1999
- **Web page**:

  http://www.cs.washington.edu/326
- **Mailing Lists:**
  - announcement list: *cse326-announce@cs.washington.edu*

  Subscribe to this using web interface
- **Discussion list: link on course home page**

Bring to Class on Wednesday:

- Name
- Email address
- Year (1,2,3,4)
- Major
- Hometown
- Interesting Fact or "What I did on my summer vacation"

Course Mechanics

- Written homeworks (7 total)
  - Due at the start of class on due date
  - No late homeworks accepted
- Programming homeworks (3 total, with phases)
  - In Java
  - Turned in electronically and on paper
  - 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
  - Gilligan’s Island rule applies

Course Mechanics

- Approximate Grading

  25% - Written Homework Assignments

  25% - Programming Assignments

  20% - Midterm Exam (in class, fifth week)

  25% - Final Exam (last day of class)

  5% - Best of Programming or Exams

Project/Homework Guides

- On the website - note especially
  - Homeworks: Use pseudocode, not code. A human being is reading your homeworks
  - See website for pseudocode examples
  - Projects: code is only 40% of your grade!
  - Spend time commenting your code as you write - it will help you be a better programmer

Homework for Today!!

1) **Sign up for mailing list (immediately)**
2) **Project #1:** Implement stacks (due in 2 weeks)
3) **Reading** in Weiss

   Chapter 1 – (Review) Mathematics and Java

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

   Chapter 2 – (Topic for Friday) Algorithm Analysis
4) **Homework #1**: based off reading, out next class

Project 1

- Soundblaster! Reverse a song
- Implement a stack and a queue to make the “Reverse” program work
- **Read the website**
  - Detailed description of assignment
  - Detailed description of how programming projects are graded

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
  - › 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

Goals

"Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious."

Fred Brooks, 1975

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 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 java.util.Stack; another is java.util.LinkedList

Concepts vs. Mechanisms

- 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.

- 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

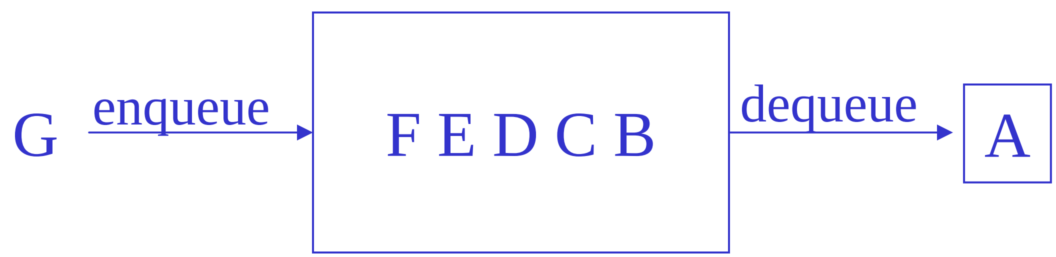
  create

  destroy

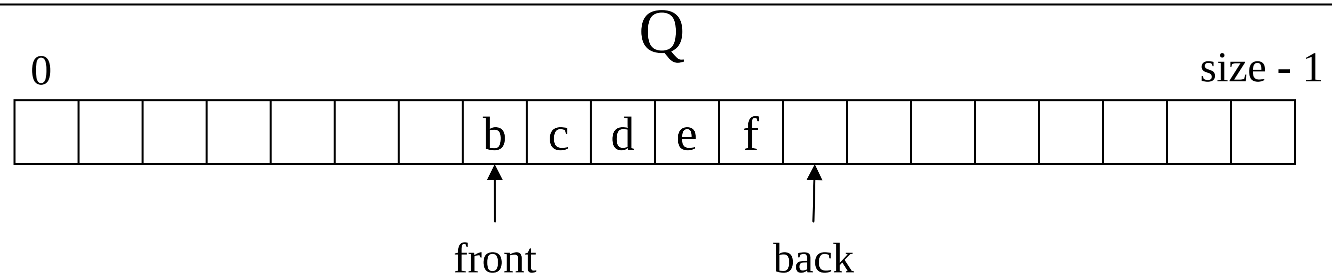
  enqueue

  dequeue

  is_empty



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 ;
   }
```

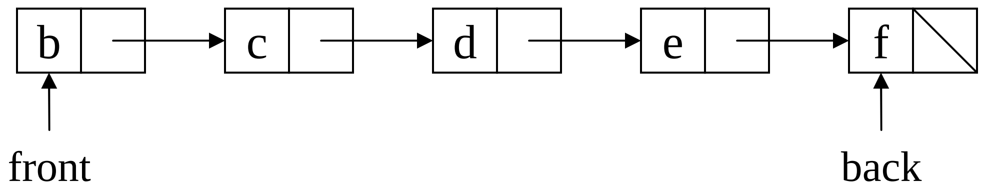
How test for empty list?

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
}
```

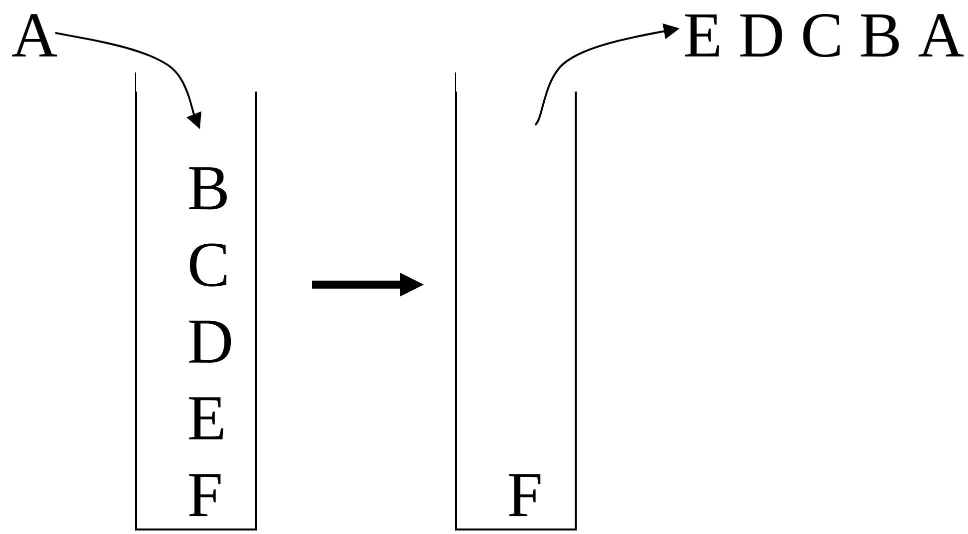
Circular Array vs. Linked List

- Too much space
- Kth element accessed “easily”
- Not as complex
- Could make array more robust

- Can grow as needed
- Can keep growing
- No back looping around to front
- Linked list code more complex

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 Reverse Polish Notation

Homework for Today!!

1) **Sign up for mailing list (immediately)**
2) **Information Sheet**:
   Bring to lecture on Friday
3) **Reading** in Weiss