CSE 326: Data Structures

Neva Cherniavsky
Summer Quarter 2006
Lecture 1

Introduction

• Me: 4th year graduate student at UW
  nchernia@cs.washington.edu
  Office hours: Th. 12:30-1:30 CSE 210
• Gary Yngve
  gyngve@cs.washington.edu
  Office hours: Tu. 9:30-10:30 CSE 220

Today's Outline

• Introductions
• What is this course about?
• Administrative Info
• 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

Goal

• 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
Course Information

- **Text**: Data Structures & Algorithm Analysis in Java, (Mark Allen Weiss), 2006
- **Web page**: http://www.cs.washington.edu/326
- **Mailing Lists**: cse326-announce@cs.washington.edu, cse326@cs.washington.edu
- **EPost Message Board**: Follow link from homepage

Course Mechanics

- **Written homeworks** (8 total)
  - Due at the start of class on due date
  - Pseudocode, no code!
- **Programming homeworks** (3 total, with phases)
  - In Java
  - Turned in electronically and on paper
- Work in teams only on explicit team projects
  - Appropriate discussions encouraged – see website
  - Anytime you use someone else’s work, it’s cheating

(Approximate) Grading

- **25%** - Written Homework Assignments
  - Due every Wednesday at the start of class
- **25%** - Programming Assignments
  - 3 projects, broken into phases
- **20%** - Midterm Exam
  - In class July 17
- **30%** - Final Exam
  - In class August 18 (last day)

Project and homework guidelines

- On the website - note especially
  - Homeworks: use pseudocode, not code. A human being is reading your homeworks
  - See website for pseudocode examples
  - Projects: execution 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 lists (immediately)
2) **Project #1**: Implement Stacks and Queues.
   - Due in one week.
3) **Reading** in Weiss
   1) Chapter 1 (review): Mathematics and Java
   2) Chapter 3 (Project #1): Stacks and Queues
   3) Chapter 2 (Homework #1): Algorithm Analysis
4) **Homework #1** is based off of reading and will be released 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
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
› Abstract Data Type (ADT) - A data object and a set of operations for manipulating it
  › 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

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.
• 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
- What is this course about?
- Administrative Info
- 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

```
<table>
<thead>
<tr>
<th>enqueue(Object x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q[back] = x ;</td>
</tr>
<tr>
<td>back = (back + 1) % size</td>
</tr>
</tbody>
</table>
```

Linked List Queue Data Structure

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

Circular Array vs. Linked List

<table>
<thead>
<tr>
<th>Too much space</th>
<th>Can keep growing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kth element accessed in O(1)</td>
<td>No back going around to front</td>
</tr>
<tr>
<td>Not as complex</td>
<td>Linked list code more robust</td>
</tr>
</tbody>
</table>

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

Algorithm Analysis: Why?

- Correctness:
  - Does the algorithm do what is intended.
  - How well does the algorithm complete its goal.
- Performance:
  - What is the running time of the algorithm.
  - How much storage does it consume.
- Different algorithms may correctly solve a given task
  - Which should I use?

Iterative Algorithm for Sum

- Find the sum of the first n integers stored in an array v.

```pseudocode
sum(integer array v, integer n) returns integer
let sum = 0
for i = 1...n
    sum = sum + i
return sum
```

Note the use of pseudocode

Programming via Recursion

- Write a recursive function to find the sum of the first n integers stored in array v.

```pseudocode
sum(integer array v, integer n) returns integer
if n = 0 then
    sum = 0
else
    sum = n + sum of first n-1 numbers
return sum
```

Proof by Induction

- **Basis Step**: The algorithm is correct for a base case or two by inspection.
- **Inductive Hypothesis (n=k)**: Assume that the algorithm works correctly for the first k cases.
- **Inductive Step (n=k+1)**: Given the hypothesis above, show that the k+1 case will be calculated correctly.

Program Correctness by Induction

- **Basis Step**: sum(v,0) = 0.
- **Inductive Hypothesis (n=k)**: Assume sum(v,k) correctly returns sum of first k elements of v, i.e. v[0]+v[1]+...+v[k-1]
- **Inductive Step (n=k+1)**: sum(v,n) returns v[k]+sum(v,k) = (by inductive hyp.) v[k]+(v[0]+v[1]+...+v[k-1]) = v[0]+v[1]+...+v[k-1]+v[k]
Algorithms vs Programs

• Proving correctness of an algorithm is very important
  › a well designed algorithm is guaranteed to work correctly and
    its performance can be estimated
• Proving correctness of a program (an implementation)
  is fraught with weird bugs
  › Abstract Data Types are a way to bridge the gap between
    mathematical algorithms and programs