Introduction

CSE 373
Data Structures & Algorithms
Ruth Anderson
Spring 2010

Staff

- Instructor
  - Ruth Anderson, rea at cs.washington.edu
- TA's
  - Shih-Yen Liu, sysliu at cs.washington.edu
  - Patrick Healy, healy77 at cs.washington.edu
  - Saptarshi Bhattacharya, saptre at cs.washington.edu

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
- Last year taught compilers, programming languages, data structures, architecture, cse143

Web Page

- All info is on the web page for CSE 373
  - http://www.cs.washington.edu/373
  - also known as
    - http://www.cs.washington.edu/education/courses/373/10sp
- Look there for schedules, contact information, assignments, links to discussion boards and mailing lists, etc.

Office Hours

- Ruth Anderson– 360 CSE (Allen Center)
  - M 12:30-1:30pm, W 1:30-2:30pm or by appointment

CSE 373 E-mail List

- If you are registered for the course, you will be automatically subscribed.
- The E-mail list is used for posting announcements by instructor and TAs.
- You are responsible for anything sent here

CSE 373 Discussion Board

- The course will have a Catalyst Go-Post message board
- Use
  - › General discussion of class contents
  - › Hints and ideas about assignments (but **not** detailed code or solutions)
  - › Other topics related to the course.

Computer Lab for homework and Help sessions

- College of Arts & Sciences Instructional Computing Lab
  - › http://depts.washington.edu/aslab/
- We'll be using Java for the programming assignments.
- Eclipse is recommended programming environment.

Textbook

- *Data Structures and Algorithm Analysis in Java,* by Mark Allen Weiss, 2nd edition, Addison-Wesley, 2007.

Grading

Estimated Breakdown:

- Assignments 50%
  - › Weights may differ to account for relative difficulty of assignments
  - › Assignments will be a mix of shorter written exercises and longer programming projects
- Midterms 30% (Two, 15% each)
- Final Exam 20%
  - › 2:30-4:20pm Wednesday, June 9, 2010.

Deadlines & Late Policy

- Assignments generally due Thursday evenings via the web
  - › Exact times and dates will be given for each assignment
- Late policy: 25% off per 24hrs late
  - › Note: ALL parts of the assignment must be received by that time (may require you to make an electronic version of written assignments).
    - (Talk to the instructor if something truly outside your control causes problems here)

Academic (Mis-)Conduct

- 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 will be penalized
- Referring to solutions from this or other courses from previous quarters is cheating.
- Integrity is a fundamental principle in the academic world (and elsewhere) – we and your classmates trust you; don't abuse that trust

Policy on collaboration

- "Gilligan's Island" rule:
  - › You may discuss problems with your classmates to your heart's content.
  - › After you have solved a problem, *discard all written notes* about the solution.
  - › Go watch TV for a ½ hour (or more). Preferably *Gilligan's Island*.
  - › *Then* write your solution.

Homework for Today!!

0) **Review Java & Explore Eclipse**
1) **Assignment #1:** (posted soon)
2) **Preliminary Survey**: fill out by evening of Tuesday March 30th
3) **Information Sheet**: bring to lecture on Wednesday 31st
4) **Reading** in Weiss (see next slide)

Reading

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

Bring to Class on Wednesday:

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

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.
- Data structures are the plumbing and wiring of programs.

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 will be able to
  - › make good design choices as a developer, project manager, or system customer

Data Structures

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

Course Topics

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues
- Trees, Hashing, Dictionaries
- Heaps, Priority Queues
- Sorting
- Disjoint Sets
- Graph Algorithms

Background

- Prerequisite is CSE 143
- Topics you should have a basic understanding of:
  - Variables, conditionals, loops, methods (functions), 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)
- We can fill in gaps as needed, but if any topics are new, plan on some extra studying

Data Structures: What?

- Need to organize program data according to problem being solved
- 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`**
- Note similarity to Java classes
  - private data structure and public methods

Data Structures: Why?

- Program design depends crucially on how data is structured for use by the program
  - Implementation of some operations may become easier or harder
  - Speed of program may dramatically decrease or increase
  - Memory used may increase or decrease
  - Debugging may be become easier or harder

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 *organization of data* and 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

Algorithm Analysis: Why?

- Correctness:
  - › Does the algorithm do what is intended?
- 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 **num** integers stored in an array **v**.

```
sum(v[ ]: integer array, num: integer): integer {
    temp_sum: integer ;
    temp_sum := 0;
    for i = 0 to num – 1 do
        temp_sum := v[i] + temp_sum;
    return temp_sum;
}
```

Note the use of pseudocode

Programming via Recursion

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

```
sum (v[ ]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
```


Pseudocode

- In the lectures algorithms will often be presented in pseudocode.
  - › This is very common in the computer science literature
  - › Pseudocode is usually easily translated to real code.
  - › This is programming language independent

Proof by Induction

- **Basis Step:** The algorithm is correct for the base case (e.g. n=0) by inspection.
- **Inductive Hypothesis (n=k):** Assume that the algorithm works correctly for the first k cases, for any k.
- **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)` which is the sum of first k+1 elements of v. ✓

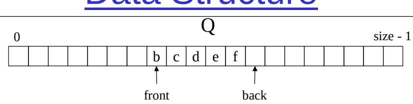
Algorithms vs Programs

- 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

First Example: Queue ADT

- 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 ;
   }
```

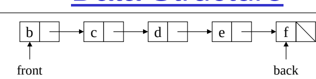
How test for empty list?

How to find K-th element in the queue?

What is complexity of these operations?

Limitations of this structure?

Linked List Queue Data Structure



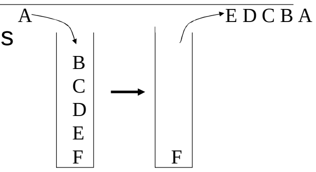

```
void enqueue(Object x) {
  if (is_empty())
      front = back = new Node(x)
  else
      back->next = new Node(x)
      back = back->next
}
bool is_empty() {
  return front == null
}
```

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

Circular Array vs. Linked List

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 Reverse Postfix Notation

Homework for Today!!

0) **Review Java & Explore Eclipse**
1) **Assignment #1:** (posted soon)
2) **Preliminary Survey**: fill out by evening of Tuesday March 30th
3) **Information Sheet**: bring to lecture on Wednesday 31st
4) **Reading** in Weiss (see next slide)