Today’s Outline

• Administrative Info
• Survey
• Overview of the Course
• What is an algorithm? ADT? Data structure?
• Review: Stacks and queues

Course Information

• Instructor: Luke McDowell, Sieg 226C
  lucasm@cs.washington.edu
  Office hours: Tuesday 11:00-12:00, Wed 1-2 p.m., or by appt.
• Teaching Assistants:
  AA: Aiman Erbad  erbad@cs.washington.edu  Mon 1-2 p.m.
  AB: Steven Martin  stevaroo@cs.washington.edu  Fri 2-3 p.m.
  Office hours in Sieg 226A
  Go to any office hours you like.
• Text: Data Structures & Algorithm Analysis in Java, 2nd edition
  (Mark Allen Weiss)
  or
  Data Structures & Algorithm Analysis in C++ (Weiss)

Course Policies

• Written homeworks
  – Due at the start of class on due date
  – No late homeworks accepted
• Programming homeworks
  – Turned in electronically before 11pm on due date
  – 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
• Approximate Grading
  – Weekly assignments: 35%
  – Midterm: 20% Friday July 25, in class
  – Final: 30% Friday Aug. 22 in class
  – Best of above 3: 10%
  – Participation: 5%

Course Mechanics

• 326 Web page: http://www.cs.washington.edu/326
• 326 mailing lists
  – announcement list: cse326-announce@cs.washington.edu
  – discussion list: cse326@cs.washington.edu
  – subscribe to these using web interface, see homepage
• Course laboratories are 232 and 329 Sieg Hall
  – labs have NT machines w/X servers to access UNIX
• All programming projects graded on UNIX
  – OK to develop using other tools (e.g. under Windows) but make
    sure you test under UNIX
  – Program in Java, or talk to the instructor

That Survey Thing

• Why are you taking my picture?
• What if I forgot everything?
• What if I know this all already?
• What if I’m the famous one?
What is this Course About?

Clever ways to organize information in order to enable efficient computation

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

Used Everywhere!

Mastery of this material separates you from:

• Perhaps the most important course in your CS curriculum!
• Guaranteed non-obsolescence!

E.g. 1: Representing Course Prerequisites

Nodes = courses
Directed edge = prerequisite

E.g. 2: Representing Expressions in Compilers

Nodes = symbols/operators
Edges = relationships

E.g. 3: Information Transmission in a Network

Nodes = computers
Edges = transmission rates
E.g. 4: Six Degrees of Separation from Kevin Bacon

Asymptotic Complexity

Our notion of efficiency:
How the running time of an algorithm scales with the size of its input
- several ways to further refine:
  - worst case
  - average case
  - amortized over a series of runs

Specific Goals of the Course

- Become familiar with some of the fundamental data structures in computer science
- Improve ability to solve problems abstractly
  - data structures are the building blocks
- Improve ability to analyze your algorithms
  - prove correctness
  - gauge (and improve) time complexity
- Become modestly skilled with the UNIX operating system (you’ll need this in upcoming courses)

One Preliminary Hurdle

1. Recall what you learned in CSE 321 …
   - proofs by mathematical induction
   - proofs by contradiction
   - formulas for calculating sums and products of series
   - recursion

Know Sec 1.1 – 1.3 of text by heart!

A Second Hurdle

- Unix
  - Experience 1975 all over again!
    - Try to login, edit, and compile your favorite “hello world” program right away
    - Get help at the UNIX tutorial tomorrow!
    - Programming Assignment 1 due next Monday
    - Bring your questions and frustrations to Section on Thursday!

A Third Hurdle: Java

```
Public class Set_of_ints {
    Public void insert( int x );
    Public void remove( int x ); ...
}
```

Review the syntax (see chapter 1)
Run your first program (assignment 1)
Java? Data Structures

One of the all time great books in computer science:

*The Art of Computer Programming (1968-1973)*

by Donald Knuth

Examples in assembly language (and English)!

American Scientist

says: in top 12 books of the CENTURY!

Very little about Java in class.

Weiss textbook’s code – don’t get bogged down!

Today’s Outline

- Administrative Info
- Survey
- Overview of the Course
- What is an algorithm? ADT? Data structure?
- Stacks and queues

What is an Algorithm?

- ???

According to …

- According to Mirriam-Webster, an *algorithm* is …
  - a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation
  - *(broadly)* a step-by-step procedure for solving a problem or accomplishing some end especially by a computer

- So …
  - What’s the difference between an “algorithm” and a “program”?

Concepts vs. Mechanisms

- 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.

- 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.

ADT’s vs Data Structures

- List ADT
  - Stack ADT
  - Queue ADT
- Collection ADT
  - Stores objects without duplicates
  - Dictionary ADT
    - Stores (Key, Value) pairs
    - Alternatively: Maps Keys to Values
  - Priority Queue ADT
    - Stores objects, and supports efficient removal of objects based upon some kind of ordering
- Graph ADT
- … and lots more!

So … which ADT’s do these data structures implement?
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!

ADT Presentation Algorithm

• Present an ADT
• Motivate with some applications
• Repeat until it’s time to move on:
  – develop a data structure and algorithms for the ADT
  – analyze its properties
    • efficiency
    • correctness
    • limitations
    • ease of programming
• Contrast strengths and weaknesses

First Example: Queue ADT

• Queue operations
  – create
  – destroy
  – enqueue
  – dequeue
  – is_empty

• Queue property: if x is enQed before y is enQed, then x will be deQed before y is deQed
FIFO: First In First Out

Applications of the Q

• Hold jobs for a printer
• Store packets on network routers
• Make waitlists fair
• Breadth first search

Circular Array Q Data Structure

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

Linked List Q Data Structure

void enqueue(Object x) {
  if (!is_empty())
    front = new Node(x)
  else
    back->next = new Node(x)
    back = back->next
}
Object dequeue() {
  assert(!is_empty)
  return_data = front->data
  temp = front
  front = front->next
  delete temp
  return temp->data
}
bool is_empty() {
  return front == null
}
Circular Array vs. Linked List

Second Example: Stack ADT

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

- Stack property: if x is on the stack before y is pushed, then x will be popped after y is popped
  LIFO: Last In First Out

Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating Reverse Polish Notation
- Depth first search

Array Stack Data Structure

void push(Object x) {
    assert(!is_full())
    S[back] = x
    back++
}

Object top() {
    assert(!is_empty())
    return S[back - 1]
}

Object pop() {
    assert(!is_empty())
    return_data = back->data
    temp = back
    back = back->next
    return return_data
}

bool is_empty() {
    return back == null
}

Linked List Stack Data Structure

void push(Object x) {
    temp = back
    back = new Node(x)
    back->next = temp
}

Object top() {
    assert(!is_empty())
    return_data = back->data
    temp = back
    back = back->next
    return return_data
}

Object pop() {
    assert(!is_empty())
    return_data = back->data
    temp = back
    back = back->next
    return return_data
}

bool is_empty() {
    return back == null
}

Data structures you should already know

- Arrays
- Linked lists
- Queues
- Stacks
To Do

• Return your survey before leaving!
• Check out the web page
• Come to the Unix tutorial **tomorrow (Tuesday)**, Sieg 232, 12-2 p.m.
• Sign up on the cse326 mailing lists
• Log on to the PCs in rooms 232 or 329 and access an instructional UNIX server
  – If you don’t have a CSE account, **sign up today**!
• Read 1.1-1.3, Chapters 2 and 3 in the book
  – Don’t worry, it gets better!
• **HW 1 due this Monday, June 30**!