Welcome to CSE 326!
Data Structures

Pick up…
• First day survey
• Copy of lecture slides
• Textbook errata
• Course syllabus

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

Who am I?
Ashish Sabharwal

n\textsuperscript{th} year grad student, CSE [currently n = 6]
Please don’t call me “professor”… yet!

• Research
  Theory: Algorithms, complexity
  Applications: Solvers for AI, model checking, etc.
• Teaching
  TA’ed several courses
  My first full class!

What’s this tabletPC thing
I’m walking around with?

Classroom Presenter system

Allows cool stuff such as
• Presenter and viewer modes
• Writing on slides

Still under construction.
Will it fail? I don’t know…

Course Staff and Textbook
• Instructor: Ashish Sabharwal, Allen Center 214, ashish@cs
  Office hours: TBD
• Teaching Assistants:
  Ethan Phelps-Goodman enhanpe@cs  Sections, concepts
  Albert J. Wong awong@cs  Programming guru, special tutorials (eg. unix)
  Office hours: TBD
• Textbook: Data Structures & Algorithm Analysis in Java
  - by Mark Allen Weiss

Course Mechanics
• Web page: http://www.cs.washington.edu/326
• Mailing aliases
  – announcement list  cse326-announce@cs
  – discussion list  cse326@cs
  – staff alias  cse326-staff@cs
  – subscribe to the lists using web interface; see webpage
• Course laboratories are 002, 006, 022 Allen Center
  – labs have NT machines with X servers to access UNIX
  – All programming projects graded on UNIX server attcres
  – OK to develop using other tools (e.g. under Windows) but make sure you test under UNIX
Course Policies

- Written assignments
  - Due at the start of class on due date; late homeworks not accepted!
- Programming assignments
  - Turned in electronically before 11 pm 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
  - Assignments: 30%
  - Midterm: 20% Monday Nov 3, in class
  - Final: 30% Monday Dec 15, in class (2 hours)
  - Best of above 3: 10%
  - Participation & quizzes: 10%

A quick break before we delve into course material!

- Fill out the survey
- Tell me times that are BAD for office hours

Today’s Outline

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

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?

Clever? Efficient?

- Lists, Stacks, Queues
- Heaps
- Binary Search Trees
- AVL Trees
- Hash Tables
- Graphs
- Disjoint Sets

Data Structures Algorithms

Used Everywhere!

Mastery of this material separates you from:

- Perhaps the most important course in your CS curriculum!
- Guaranteed non-obsolescence!
Example 1: Representing Course Prerequisites

```
<table>
<thead>
<tr>
<th>142</th>
<th>143</th>
</tr>
</thead>
<tbody>
<tr>
<td>321</td>
<td>322</td>
</tr>
<tr>
<td>378</td>
<td>341</td>
</tr>
<tr>
<td>40</td>
<td>421</td>
</tr>
</tbody>
</table>
```

Nodes = courses
Directed edge = prerequisite

Example 2: Representing Expressions in Compilers

```
x1 = q + y*z
x2 = y*z - q
```

Naive:
```
\[ yz \text{ calculated twice!} \]
```

Common Subexpression Eliminated:

Nodes = symbols/operators
Edges = relationships

Example 3: Information Transmission in a Network

```
<table>
<thead>
<tr>
<th>128</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>181</td>
<td>130</td>
</tr>
<tr>
<td>14</td>
<td>30</td>
</tr>
</tbody>
</table>
```

Nodes = computers
Edges = transmission rates

Efficiency: Asymptotic Complexity

Run the program and measure time:
- Typically insufficient!
- How big is the sample input? What about other inputs?

Our notion of efficiency:
How does the running time of an algorithm scale
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

\[ \text{Know Sec 1.1 - 1.3 of text by heart!} \]
A Second Hurdle

2. Unix
   Experience 1975 all over again!
   - Try to login on attu.cs, edit, and compile your
     favorite “hello world” program right away
     Get help at the UNIX tutorial (tomorrow?)
   - Programming Assignment 1 (to be released Wed)
   - 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)!

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
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.

ADT’s vs Data Structures

- List ADT
  - Stack 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 even more!

- Linked List
- Circular Array
- Binary Search Tree
- Splay Tree
- Hash Table
- leftist Heap
- Skew Heap
- Adjacency Matrix
- ... 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

1. Present an ADT
2. Motivate with some applications
3. Repeat until it’s time to move on:
   a. analyze its properties
   b. develop a data structure and algorithms for the ADT
      i. efficiency
      ii. correctness
      iii. limitations
      iv. ease of programming
4. 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

enq(x) {  
    Q[back] = x;  
    back = (back + 1) % size  
}

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

How test for empty list?
How to find kth element in the queue?
What is complexity of these operations?
Limitations of this structure?

Linked List Q Data Structure

void enq(x) {  
    if (is_empty())  
        front = back = new Node(x)  
    else  
        back->next = new Node(x)  
        back = back->next  
}

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

bool is_empty() {  
    return front == null  
}

Circular Array vs. Linked List

Stacks in Practice

- Function call stack
- Converting recursion to iteration
- Balancing symbols (parentheses)
- Evaluating Reverse Polish (postfix) Notation
- Depth first search

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

Array Stack Data Structure

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

Object pop() {  
    assert(!is_empty())  
    return S[back]  
}

bool is_empty() {  
    return back == 0  
}

bool is_full() {  
    return back == size  
}
Linked List Stack Data Structure

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

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 - TBD
- Sign up for the cse326 mailing lists
- Log on to the PCs in rooms 002, 006 or 022 and access instructional UNIX server attu.cs
  - 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!