CSE 326: Data Structures
Introduction

Hannah Tang and Brian Tjaden
Summer Quarter 2002

Today’s Outline

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

Course Information

• Instructors: Hannah Tang and Brian Tjaden
  226C Sieg Hall
  hctang@cs.washington.edu and tjaden@cs.washington.edu
  Hannah’s office hours: Tuesday 10-11:00, Friday 1-2:00
  Brian’s office hours: Monday 1-2:00, Thursday 1-2:00
• Grader/Consultant extraordinaire: Albert Wong
  awong@cs.washington.edu
• Text: Data Structures & Algorithm Analysis in C++, 2nd edition, by Mark Allen Weiss
  or
  Data Structures & Algorithm Analysis in Java, by Mark Allen Weiss

C++ or Java...

you make the call!

Course Assessment

• Homeworks or projects due each week
• Weekly written homework due at the start of class on the due date
• Projects due by 10PM on the due date
• Quizzes each Thursday in section!!!
• Final Exam: August 23 in class!!!
• Grading
  – homework: 20%
  – projects: 30%
  – quizzes: 20%
  – final: 25%
  – participation: 5%

Course Mechanics

• 326 Web page: http://www.cs.washington.edu/326
• 326 course directory: /cse/courses/cse326/02su
• 326 mailing lists
  – announcement list: cse326announce
  – discussion list: cse326
  – extra topics list: cse326beyond
  – subscribe to the mailing list 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
What is an Algorithm?

- 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.
  - Each program must decide how to store its data, and these choices influence the program at every level:
    - Execution speed
    - Memory requirements
    - Maintenance (debugging, extending, 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
- 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

- Linked List
- Circular Array
- Binary Search Tree
- Splay Tree
- Hash Table
- L 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!

Goals of the Course

- Learn some of the fundamental data structures in computer science
  - And understand their tradeoffs!

- Learn to see and solve problems abstractly
  - Be able to see the intrinsic problem behind real-world scenarios, or vice versa, be able to realize an abstract solution in the real world
- Data structures are your problem-solving building blocks?
- Learn to analyze and improve algorithms
  - Prove correctness
  - Gauge and improve time complexity

- Become modestly skilled with the UNIX operating system
- Appreciate that all languages are not created equal...
Learning Concepts vs. Learning Code

CSE 326 balances concepts with mechanisms
- Grade is 65% concepts and plans, 30% coding skill, but …
- Coding greatly improves grasp of concepts and plans

Different approaches
- Weiss is code-centric: emphasizes mechanisms
- Introduction to Algorithms by Cormen, Leiserson, Rivest is pseudocode-centric: emphasizes concepts
- The Art of Computer Programming (1968-1973) by Donald Knuth emphasizes concepts and mechanisms
  - Examples in assembly language (and English)!
  - American Scientist ranks in top 12 books of century!
- Many, many more!

Translating Concepts Into Mechanisms

- In a perfect world …
  - An interface (or abstract base class) describes ADT
  - Inherited classes implement data structures
  - Can change data structures transparently (to client code)
- In the real world …
  - Different implementations sometimes suggest different interfaces (generality vs. simplicity)
  - Performance of a data structure may influence form of client code (time vs. space, one operation vs. another)

Data Structure Presentation Algorithm

- Present data structure
- Motivate with some applications
- Repeat until you see visions of the data structure in your sleep
  - Determine which ADT’s this data structure can implement
  - Analyze its properties
    - Efficiency
    - Correctness
    - Limitations
    - Ease of programming
  - Contrast data structure’s strengths and weaknesses
    - Understand when to use each one

And now, the moment you’ve been waiting for: Your first ADT!

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
- Hold memory “freelists”
- Make waitlists fair
- Breadth first search

Circular Array Q Data Structure

- Present data structure
- Motivate with some applications
- Repeat until you see visions of the data structure in your sleep
  - Determine which ADT’s this data structure can implement
  - Analyze its properties
    - Efficiency
    - Correctness
    - Limitations
    - Ease of programming
  - Contrast data structure’s strengths and weaknesses
    - Understand when to use each one

And now, the moment you’ve been waiting for: Your first ADT!

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
- Hold memory “freelists”
- Make waitlists fair
- Breadth first search

Circular Array Q Data Structure

- Present data structure
- Motivate with some applications
- Repeat until you see visions of the data structure in your sleep
  - Determine which ADT’s this data structure can implement
  - Analyze its properties
    - Efficiency
    - Correctness
    - Limitations
    - Ease of programming
  - Contrast data structure’s strengths and weaknesses
    - Understand when to use each one

And now, the moment you’ve been waiting for: Your first ADT!
**Q Example**

enqueue R
enqueue O
dequeue
denqueue
tenqueue T
tenqueue A
tenqueue T
dequeue
tenqueue E
dequeue

**Linked List Q Data Structure**

```java
void enqueue(Object x) {
    if (is_empty())
        front = back = 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**

- 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 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

**Stacks in Practice**

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

**Array Stack Data Structure**

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

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

Object pop() {
    back--
    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 top() {
    assert(!is_empty())
    return back->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

- Check out the web page
- Come to the Unix tutorial tomorrow (Tuesday, June 25), Sieg 322, 4:30-5:30
- Sign up on the cse326 mailing lists
- Log on to the PCs in rooms 232 or 329 and access an instructional UNIX server
- Read Chapters 2 and 3 in the book
- Project 1 due this Monday, July 1!