CSE332: Data Abstractions

Lecture 1: Introduction; Stacks/Queues

Dan Grossman
Spring 2010

Welcome!

We have 10 weeks to learn *fundamental data structures and algorithms for organizing and processing information*

- "Classic" data structures / algorithms and how to analyze rigorously their efficiency and when to use them
- Queues, dictionaries, graphs, sorting, etc.
- Parallelism and concurrency (new!)

Today in class:

- Course mechanics
- What this course is about ("the new 326")
  - And how it fits into the revised curriculum
- Start (finish?) stacks and queues (largely review)

Concise to-do list

In the 24-48 hours:

- Adjust class email-list settings
- Email homework 0 (worth 0 points) to me
- Read all course policies
- Read/skim Chapters 1 and 3 of Weiss book
  - Relevant to Project 1, due next week
  - Will start Chapter 2 on Wednesday

Possibly:

- Set up your Eclipse / Java environment for project 1
  - Thursday's section will help

http://www.cs.washington.edu/education/courses/cse332/10sp/

Who

Course Staff:

Dan Grossman

Tyler Robison

Brent Sandona

Dan: Faculty, "a 341 guy", excited about 332 (& curriculum revision)
Tyler: Grad student, Summer2010 332 instructor, Winter2010 326 TA
Brent: Senior, CompE, veteran TA, including 326

Office hours, email, etc. on course web-page

Staying in touch

- Course email list: cse332a_sp10@u
  - Students and staff already subscribed
  - You must get announcements sent there
  - Fairly low traffic
- Course staff: cse332-staff@cs plus individual emails
- Message Board
  - For appropriate discussions; Brent will monitor
  - Optional, won't use for important announcements
- Anonymous feedback link on webpage
  - For good and bad: if you don't tell me, I don't know

Course meetings

- Lecture (Dan)
  - Material posted (sometimes afterwards), but take notes
  - Ask questions, focus on key ideas (rarely coding details)
- Section (Tyler)
  - Often focus on software (Java features, tools, project issues)
  - Reinforce key issues from lecture (good practice)
  - Answer homework questions, etc.
  - Goal: More important than it was for your friends in 326
- Office hours
  - Use them: *please visit me*
  - Ideally not *just* for homework questions (but that's great too)

Course materials

- All lecture and section materials will be posted
  - But they are visual aids, not always a complete description!
  - If you have to miss, find out what you missed
- Textbook: Weiss 2nd Edition in Java
  - Good reading, but only responsible for lecture/section/hw topics
  - Will assign homework problems from it
- Core Java book: A good Java reference (there may be others)
  - Don't struggle Googling for features you don't understand
  - Same book recommended for CSE331
- Weeks 8-10 not in either book (or any appropriate one)
  - Will write and post more thorough lecture notes, etc.

Course Work

- 8 written/typed homeworks (25%)
  - Due at beginning of class each Friday (not this week)
  - No late homeworks accepted
  - Often covers through Monday before it's due
- 3 programming projects (with phases) (25%)
  - First phase of first project due in 9 days
  - Use Java and Eclipse (see this week's section)
  - One 24-hour late-day for the quarter
  - Projects 2 and 3 will allow partners
- Midterm Friday April 30 (20%)
- Final Tuesday June 8 (25%)

Collaboration and Academic Integrity

- Read the course policy very carefully
  - Explains quite clearly how you can and cannot get/provide help on homework and projects
- Always explain any unconventional action on your part
- Your instructor has promoted and enforced academic integrity since he was a freshman
  - Believes in great trust with little sympathy for violations
  - Honest work is the most important feature of a university

Unsolicited advice

- Get to class on time!
  - Instructor pet peeve (I will start and end promptly)
  - First 2 minutes are *much* more important than last 2!
  - April 30 will prove beyond any doubt you are capable
- Learn this stuff
  - You need it for so many later classes/jobs anyway
  - Falling behind only makes more work for you
- Embrace being a "charter member" of CSE332
  - Things may be a little rough around the edges
  - But we make up for it with excitement and a nice class size

Today in Class

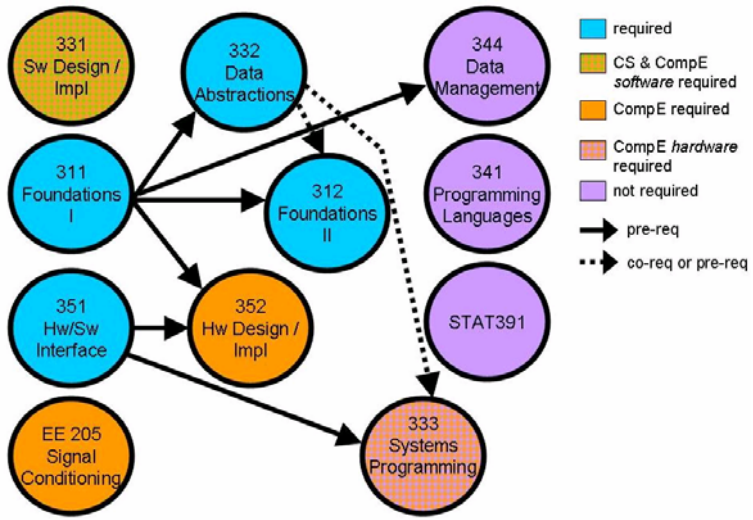
- Course mechanics – Did I forget anything?
- What this course is about ("the new 326")
  - And how it fits into the revised curriculum
- Start stacks and queues (largely review)

Last word about 326

- Many new courses have no one obvious counterpart in the old
- But 332 is about *70%* of the material from 326
  - First 7 weeks or so, obviously cutting out some topics
    - and a little moving to 312
  - Timeless, essential stuff
  - And a good chance to clean up some 326 "cruft"
- Biggest new topic: a serious treatment of programming with *multiple threads*
  - For *parallelism*: To use multiple processors to finish sooner
  - For *concurrency*: Allow properly synchronized access to shared resources

Where 332 fits

Note: This picture was the story of my life in 2009



Okay, so what is 332 about already!

- Introduction to many (not all) of the basic data structures used in computer software
  - Understand the data structures and the trade-offs they make
  - Rigorously analyze the algorithms that use them (math!)
  - Learn how to pick "the right thing for the job"
  - More thorough and rigorous take on topics introduced in 143
    - And more
- Practice design and analysis of data structures / algorithms
- Practice implementing and using these data structures by writing programs
- Experience the purposes and headaches of multithreading

Goals

- To be able to make good design choices as a developer, project manager, etc.
  - Reason in terms of the general abstractions that come up in all non-trivial software (and many non-software) systems
- To be able to justify and communicate your design decisions

Dan's take:

3 years from now this course will seem like it was a waste of your time because you can't imagine not "just knowing" every main concept in it

- Key abstractions computer scientists and engineers use almost every day
- A big piece of what separates us from others

Data structures

(Often highly *non-obvious*) ways to organize information in order to enable *efficient* computation over that information

- Key goal over the next week is introducing asymptotic analysis to *precisely* and *generally* describe efficient use of time and space

A data structure supports certain *operations*, each with a:

- Meaning: what does the operation do/return
- Performance: how efficient is the operation

Examples:

- ***List*** with operations **`insert`** and **`delete`**
- ***Stack*** with operations **`push`** and **`pop`**

Trade-offs

A data structure strives to provide many useful, efficient operations

But there are unavoidable trade-offs:
- Time vs. space
- One operation more efficient if another less efficient
- Generality vs. simplicity vs. performance

That is why there are many data structures and educated CSEers internalize their main trade-offs and techniques
- And recognize logarithmic < linear < quadratic < exponential

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of a "thing" with set of operations on that "thing"
- Algorithm
  - A high level, language-independent description of a step-by-step process
- Data structure
  - A specific family of algorithms for implementing an ADT
- Implementation of a data structure
  - A specific implementation in a specific language

Example: Stacks

- The ***Stack*** ADT supports operations:
  - **`isEmpty`**: initially true, later have there been same number of pops as pushes
  - **`push`**: takes an item
  - **`pop`**: raises an error if isEmpty, else returns most-recently pushed item not yet returned by a pop
  - … (Often some more operations)
- A Stack data structure could use a linked-list or an array or something else, and associated algorithms for the operations
- One implementation is in the library **`java.util.Stack`**

Why useful

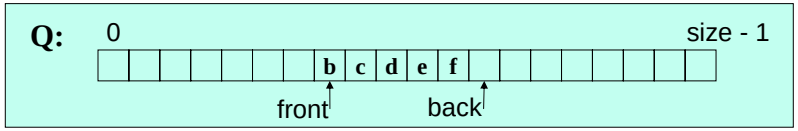
The Stack ADT is a useful abstraction because:
- It arises all the time in programming (see text for more)
  - Recursive function calls
  - Balancing symbols (parentheses)
  - Evaluating postfix notation: 3 4 + 5 *
  - Clever: Infix ((3+4) * 5) to postfix conversion (see text)
- We can code up a reusable library
- We can communicate in high-level terms
  - "Use a stack and push numbers, popping for operators…"
  - Rather than, "create a linked list and add a node when…"

The Queue ADT

- Operations
  create
  destroy
  enqueue
  dequeue
  is_empty

- Just like a stack except:
  - Stack: LIFO (last-in-first-out)
  - Queue: FIFO (first-in-first-out)
- Just as useful and ubiquitous

Circular Array Queue Data Structure

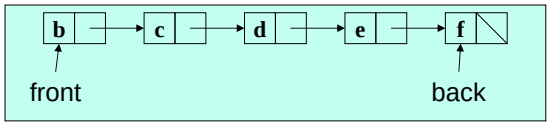



```
// Basic idea only!
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size
}
```

```
// Basic idea only!
dequeue() {
  x = Q[front];
  front = (front + 1) % size;
  return x;
}
```

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- What if ***array*** is full?
- How to *test* for empty?
- What is the *complexity* of the operations?
- Can you find the k^{th} element in the queue?

Linked List Queue Data Structure




```
// Basic idea only!
enqueue(x) {
  back.next = new Node(x);
  back = back.next;
}
```

```
// Basic idea only!
dequeue() {
  x = front.item;
  front = front.next;
  return x;
}
```

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- Can ***list*** be full?
- How to *test* for empty?
- What is the *complexity* of the operations?
- Can you find the k^{th} element in the queue?

Circular Array vs. Linked List

(This is something every trained computer scientist knows in his/her sleep – it's like knowing how to do arithmetic.)

Array:

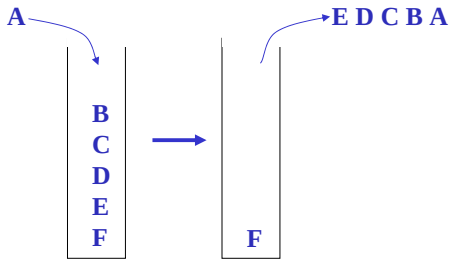
- May waste unneeded space or run out of space
- Space per element excellent
- Operations very simple / fast
- Constant-time access to k^{th} element
- For operation insertAtPosition, must shift all later elements
  - Not in Queue ADT

List:

- Always just enough space
- But more space per element
- Operations very simple / fast
- No constant-time access to k^{th} element
- For operation insertAtPosition must traverse all earlier elements
  - Not in Queue ADT

The Stack ADT

- Operations
  `create`
  `destroy`
  `push`
  `pop`
  `top`
  `is_empty`



- Can also be implemented with an array or a linked list
  - This is Project 1!
  - Like queues, type of elements is irrelevant
    - Ideal for Java's generic types (section and Project 1B)