CPSC 221: Algorithms and Data Structures Lecture #0: Introduction

Alan J. Hu
(Borrowing some slides from Steve Wolfman)

Webpage: http://www.ugrad.cs.ubc.ca/~cs221

Rule #1: Ask questions!

Who I Am

Alan J. Hu (You can call me Alan or Prof. Hu.)
ajh@cs.ubc.ca
ICICS 325
office hours: TBD

- I will hang out after class if people have questions (sometimes I'll need to leave earlier).
- Feel free to stop by my office at other times, but I may be busy or not there.

Other Introductions

- Two Sections: MWF 10-11 in DMP 110 and MWF 4-5pm in SCRF 100.
- TAs: See website.
- TA Office hours: TBA; see website

Textbooks

- Texts: Epp Discrete Mathematics, Koffman C++ (But… feel free to get Epp 3rd ed or alternate texts)

Course Policies (Subject to Change)

- No late work; *may* be flexible with advance notice
- Programming projects (~3) typically due 9PM on due date
  - **All programming projects graded on Linux/g++**
- Written homework (~3) typically due 5PM on due date
- Labs (roughly) every week. Please attend assigned lab section for now. Labs start next week.
- Exams: **Must pass final exam to pass the course!**
- Grading: See webpage for detailed breakdown.

Midterm Date

- The midterm will be in the evening, combined with the other section.

Collaboration

READ the collaboration policy on the website.

You have **LOTS** of freedom to collaborate!

Use it to learn and have fun while doing it!

Don't violate the collaboration policy. There's no point in doing so, and the penalties are severe.

Course Mechanics

- 221 Web page: www.ugrad.cs.ubc.ca/~cs221
- We will be using Piazza as the primary way to answer questions, have discussions, etc.
  - You MUST have an account, but you may use a fictitious email if you wish (this option is to comply with BC privacy laws).
- We will also use Connect, mainly so you can see your marks.
- **You are responsible for keeping up with all of these sources!**

What This Course Is About

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

What's an Algorithm?

- Some Classic $\boxed{\text{Algorithms}}$
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- Algorithm (Typical Definition)
  - A high-level, language-independent description of a step-by-step process for solving a problem

What's an Algorithm?

- Some Classic $\boxed{\text{Algorithms}}$
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- Algorithm (Street Definition)
  - A smarter way to solve the problem!

What’s a Data Structure?

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- Data Structure (Street Definition)
  - How to organize your data to get the results you want, along with the supporting algorithms

Why Study Classic Examples?

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- Reason #1: They are useful!
  - Like pre-packaged intelligence in a can!
  - Don't have to work hard to come up with your own solution

Why Study Classic Examples?

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- Reason #2: They let you abstract away details!
  - These are "power tools" for programming.
  - Let you focus on solving bigger problems, ignore details.

Why Study Classic Examples?

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- Reason #3: You learn general solution ideas!
  - This will help you solve new, unexpected problems.
  - Great masters in any field study the classic examples from their field.

Why the Theory and Math?

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

- This gives you the tools to determine:
  - what's good or bad
  - what trade-offs are being made

and explain clearly why.

What this Course Is About

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)

Overall, this course is a major step that separates you from being just some schmoe who learned a bit of programming!

What this Course Is About

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - (Also, a bit of leftover theory that should be in 121)
- Some Basics of Parallelism and Concurrency

These were always supposed to be part of 221, but neglected in past, and increasingly important on modern computers!

Goals of the Course

- Become familiar with some of the fundamental data structures and algorithms in computer science
- Improve ability to solve problems abstractly
  - data structures and algorithms are the building blocks
- Improve ability to analyze your algorithms
  - prove correctness
  - gauge, compare, and improve time and space complexity
- Become modestly skilled with C++ and UNIX, but this is largely on your own!

Fun Example

- We’ll look at a simple example, to see how different choices affect performance:
  - Fibonacci Numbers

Fun Example

- We'll look at a simple example, to see how different choices affect performance:
  - Fibonacci Numbers

- Does performance matter in practice?

Fun Example

- We'll look at a simple example, to see how different choices affect performance:
  - Fibonacci Numbers

- Does performance matter in practice?
  - Massive load on web applications: Anyone use Cuil instead of Google?
  - Huge amounts of data
  - Efficient algorithms allow lower power, longer battery life, cheaper processors, etc.

Fibonacci Numbers

- Common example in CS
- Some applications, pops up in unusual places (art, nature, algorithm analysis)
- Mainly, just a convenient, small example that illustrates important CS points.

- 1, 1, 2, 3, 5, 8, 13, …
- Each number is sum of previous two

Obvious Recursive Fibonacci

- Base Case:

  fib(1) = 1

  fib(2) = 1

- General Case:

  fib(n) = fib(n-1) + fib(n-2)

- We’ll use the GNU Multiple Precision Arithmetic Library to handle big numbers.
  - Compile like this: (those are lowercase L, not the digit 1

    g++ bigfib.cpp –lgmpxx -lgmp

Faster Iterative Fibonacci

- Just iterate up from beginning

  fib(1) = 1

  fib(2) = 1

  fib(3) = 2

  fib(4) = 3

  … etc.

Faster Iterative Fibonacci

- Just iterate up from beginning

  fib(1) = 1

  fib(2) = 1

  fib(3) = 2

  fib(4) = 3

  … etc.

- Can we do better?

Matrix Multiplication

- Consider this matrix equation:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x+y \\ x \end{bmatrix}$$

Aside: Matrices

- A “matrix” is just a rectangular array of numbers:

$$\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \end{bmatrix}$$

Aside: Matrix Multiplication

- There's a standard, special way to define the multiplication of two matrices:

$$\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} = ?$$

Aside: Matrix Multiplication

- There's a standard, special way to define the multiplication of two matrices:

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \end{bmatrix} \begin{bmatrix} ? & ? & 15 & ? \\ ? & ? & ? & ? \end{bmatrix}$$

Matrix Multiplication

- Consider this matrix equation:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + y \\ x \end{bmatrix}$$

Matrix Multiplication

- Consider this matrix equation:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x+y \\ x \end{bmatrix}$$

- Hey! That's one iteration of Fibonacci!

Matrix Fibonacci

- Repeated matrix multiplication computes Fibonacci numbers…

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

$$\vdots$$

Multiplication is associative.

- Associative Law: (xy)z = x(yz)
- Therefore,

$$x^n = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot \ldots$$

$$= ((((x \cdot x) \cdot x) \cdot x) \cdot x) \cdot x \cdot x \cdot x \cdot \ldots$$

$$= (x \cdot x) \cdot (x \cdot x) \cdot (x \cdot x) \cdot (x \cdot x) \cdot \ldots$$

$$= x \cdot ((x \cdot x) \cdot (x \cdot x)) \cdot x \cdot (x \cdot x) \cdot \ldots$$

$$= \ldots$$

Matrix multiplication is associative, too!

- Associative Law: (XY)Z=X(YZ)
- Therefore,

$$
\begin{aligned}
T^n &= T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot \ldots \\
&= ((((T \cdot T) \cdot T) \cdot T) \cdot T) \cdot T \cdot T \cdot T \cdot \ldots \\
&= (T \cdot T) \cdot (T \cdot T) \cdot (T \cdot T) \cdot (T \cdot T) \cdot \ldots \\
&= T \cdot ((T \cdot T) \cdot (T \cdot T)) \cdot T \cdot (T \cdot T) \cdot \ldots \\
&= \ldots
\end{aligned}
$$

Matrix Fibonacci

- Repeated matrix multiplication computes Fibonacci numbers…

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

$$\vdots$$

Matrix Fibonacci

- Repeated matrix multiplication computes Fibonacci numbers…

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

$$\vdots$$

Repeated Multiplication is Exponentiation!

$$T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = T^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Exponentiation by Iterative Squaring

- Imagine you have an old calculator and need to compute 2^{32}.
- You could type **`2 x 2 x 2 x 2 x ….`**
- Or, you could type:
  - **`2 x 2 =`** and get 2^2.
  - **`x =`** and get 2^4.
  - **`x =`** and get 2^8.
  - **`x =`** and get 2^{16}.
  - **`x =`** and get 2^{32}.

Iterative squaring works for matrices, too!

$$T^2 = T \cdot T$$

$$T^4 = T^2 \cdot T^2$$

$$T^8 = T^4 \cdot T^4$$

$$T^{16} = T^8 \cdot T^8$$

$$T^{32} = T^{16} \cdot T^{16}$$

$$T^{64} = T^{32} \cdot T^{32}$$

$$\vdots$$

Iterative Squaring Example

$$T^{100} = T^{64} \cdot T^{32} \cdot T^{4}$$

Do only 8 (matrix) multiplications instead of 99!