Unit #0: Introduction

CPSC 221: Algorithms and Data Structures

Lars Kotthoff[1]
larsko@cs.ubc.ca

[1]With material from Will Evans, Steve Wolfman, Alan Hu, Ed Knorr, and Kim Voll.

Course Information

Instructor

Lars Kotthoff, `larsko@cs.ubc.ca`, ICCS X569

Course website

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

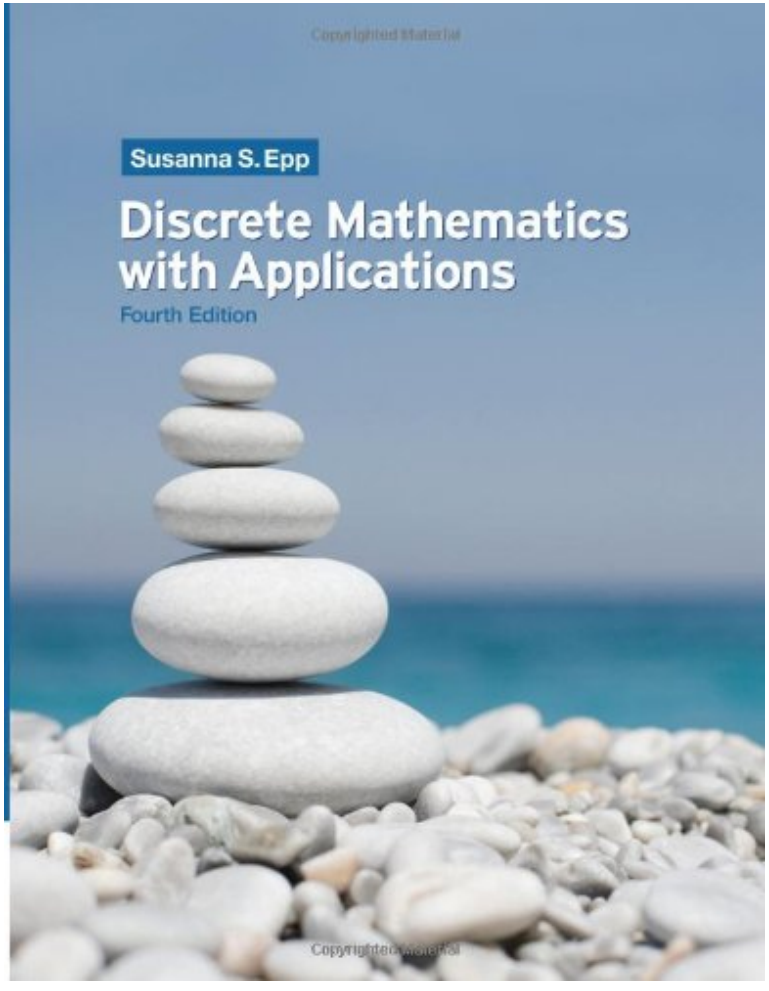
Office hours

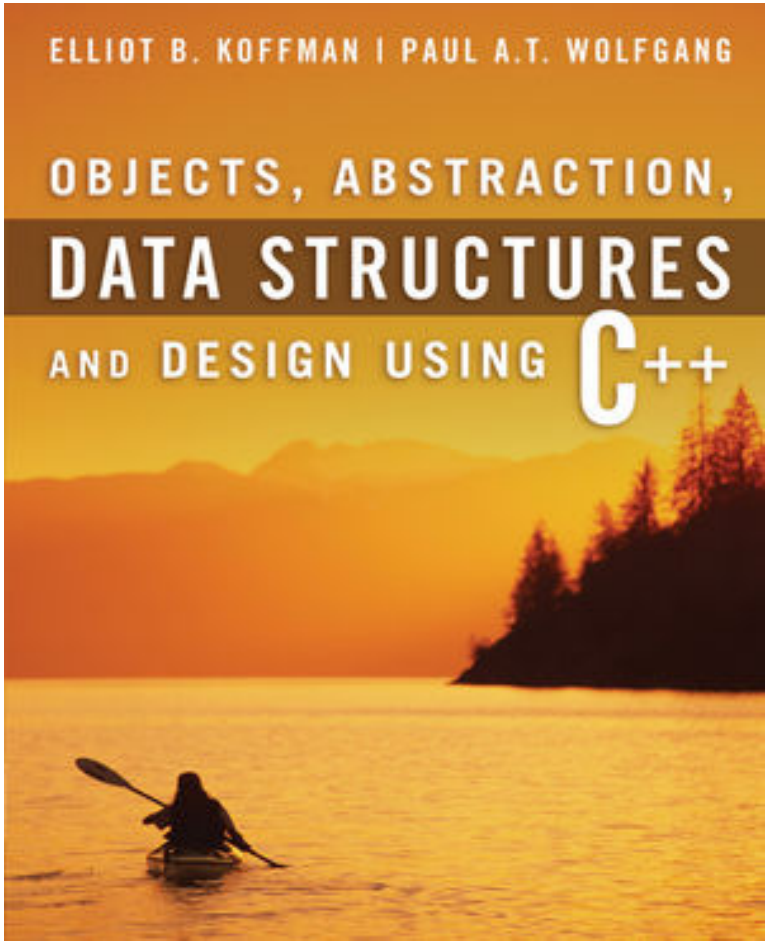
TBD

TAs

see website

Textbooks


Susanna S. Epp
Discrete Mathematics with Applications
Fourth Edition


ELLIOT B. KOFFMAN | PAUL A.T. WOLFGANG
OBJECTS, ABSTRACTION,
DATA STRUCTURES
AND DESIGN USING C++

Course Policies

No late work; may be flexible with advance notice

10%	Labs
15%	Programming projects ($\approx$3)
15%	Written homework ($\approx$3)
20%	Midterm exam
40%	Final exam

Must pass the final and combo of labs/assignments to pass the course.

Collaboration

You may work in groups of two people on:

- ▷ labs
- ▷ programming projects
- ▷ written homework

You may also collaborate with others as long as you follow the rules (see the website) and **acknowledge** their help on your assignment.

Don't violate the collaboration policy.

Course Mechanics

- Web page, `http://www.ugrad.cs.ubc.ca/~cs221`
- Piazza, `https://piazza.com/ubc.ca/winterterm22015/cpsc221`
- UBC Connect, `www.connect.ubc.ca`
- Labs start next week, (roughly) every week
- Programming projects will be graded on Linux and g++ (CS ugrad machines)

Help

- other students
- Piazza
- TAs, instructors
- the interwebs (e.g. Stackoverflow for programming questions, see `https://stackoverflow.com/help/how-to-ask`)

Your degree is your responsibility.

Algorithms and Data Structures

- What is an algorithm?

Algorithms and Data Structures

- What is an algorithm? High-level, language-independent description of step-by-step process for solving a problem.

Algorithms and Data Structures

- What is an algorithm? High-level, language-independent description of step-by-step process for solving a problem.
- What is a data structure?

Algorithms and Data Structures

- What is an algorithm? High-level, language-independent description of step-by-step process for solving a problem.
- What is a data structure? Specialized format for organizing and storing data efficiently.

Algorithms and Data Structures

- What is an algorithm? High-level, language-independent description of step-by-step process for solving a problem.
- What is a data structure? Specialized format for organizing and storing data efficiently.

Particular algorithms may work (better) with particular data structures.

Observations

- programs manipulate data
  - programs process, store, display, gather data
  - data can be text, numbers, images, sound
- programs must decide how to store and manipulate data
- choice affects behaviour of the program
  - execution speed
  - memory requirements
  - maintenance (debugging, extending, etc.)

Being able to analyze this behaviour is what separates good programmers from bad programmers.

Goals of the Course

- become familiar with some of the fundamental data structures and algorithms in computer science and learn when to use them
- improve your ability to solve problems abstractly with algorithms and data structures as the building blocks
- improve your ability to analyze 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)

Analysis Example

Fibonacci Numbers

- first two numbers are 1, each subsequent number sum of two preceding it
- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...
- common example in CS
- applications in many areas (e.g. bee ancestry, branching of trees, arrangement of leaves on a stem)

Recursive Fibonacci

Calculate the nth Fibonacci number.

Recursive definition:

$$fib_n = \begin{cases} 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ fib_{n-1} + fib_{n-2} & \text{if } n \geq 3 \end{cases}$$

C++ code:

```
int fib(int n) {
   if(n <= 2) return 1;
   else       return fib(n-1) + fib(n-2);
}
```

Recursive Fibonacci

Calculate the nth Fibonacci number.

Recursive definition:

$$fib_n = \begin{cases} 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ fib_{n-1} + fib_{n-2} & \text{if } n \geq 3 \end{cases}$$

C++ code:

```
int fib(int n) {
   if(n <= 2) return 1;
   else       return fib(n-1) + fib(n-2);
}
```

Too slow!

Iterative Fibonacci

Idea: Save result of previous computations instead of computing the same values over and over again.

```
int fib(int n) {
  int F[n+1];

  F[0]=0; F[1]=1; F[2]=1;
  for(int i=3; i<=n; ++i) {
    F[i] = F[i-1] + F[i-2];
  }
  return F[n];
}
```

Iterative Fibonacci

Idea: Save result of previous computations instead of computing the same values over and over again.

```
int fib(int n) {
  int F[n+1];

  F[0]=0; F[1]=1; F[2]=1;
  for(int i=3; i<=n; ++i) {
    F[i] = F[i-1] + F[i-2];
  }
  return F[n];
}
```

Can we do better?

Fibonacci by formula

Idea: Use a formula (a *closed form solution* to the recursive definition).

$$fib_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$$

where $\varphi = (1 + \sqrt{5})/2 \approx 1.61803$.

```
#include <cmath>
int fib(int n) {
   double phi = (1 + sqrt(5))/2;
   return (pow(phi, n) - pow(-phi,-n))/sqrt(5);
}
```

Sadly, it's impossible to represent $\sqrt{5}$ exactly on a digital computer.

Fibonacci with Matrix Multiplication

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+1 \\ 1 \end{bmatrix} = \begin{bmatrix} fib_3 \\ fib_2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} fib_4 \\ fib_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} fib_n \\ fib_{n-1} \end{bmatrix}$$

How do we calculate $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2}$?

Repeated Squaring

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

$$\begin{aligned} A \times A &= A^2 \\ A^2 \times A^2 &= A^4 \\ A^4 \times A^4 &= A^8 \\ A^8 \times A^8 &= A^{16} \\ A^{16} \times A^{16} &= A^{32} \\ A^{32} \times A^{32} &= A^{64} \\ &\vdots \end{aligned}$$

Repeated Squaring Example

$$A^{100} = A^{64} \times A^{32} \times A^{4}$$

$\hookrightarrow$ instead of 99 multiplications only 8 (matrix) multiplications

Is this better than iterative Fibonacci?