CSC 421: Algorithm Design & Analysis

Spring 2016

See online syllabus: http://dave-reed.com/csc421   (also on BlueLine)

Course goals:

- To appreciate the role of algorithms in problem solving and software design, recognizing that a given problem might be solved with a variety of algorithms.
- To be capable of selecting among competing algorithms and justifying their selection based on efficiency.
- To be capable of selecting and utilizing appropriate data structures in implementing algorithms as computer programs.
- To develop programs using different problem-solving approaches (divide-and-conquer, backtracking, dynamic programming), and be able to recognize when one approach is a better fit for a given problem.
- To design and implement programs to model real-world systems, and subsequently analyze their behavior.

1

Your programming evolution…

221: programming in the small
- focused on the design & implementation of small programs
- introduced fundamental programming concepts
  - ✓ variables, assignments, expressions, I/O
  - ✓ control structures (if, if-else, while, for), strings, lists
  - ✓ functions, parameters, OO philosophy

222: programming in the medium
- focused on the design & analysis of more complex programs
- introduced object-oriented approach
  - ✓ classes, objects, fields, methods, object composition, libraries
  - ✓ interfaces, inheritance, polymorphism, system modeling
  - ✓ searching & sorting, Big-Oh efficiency, recursion, GUIs

321: programming in the larger
- focus on more complex problems where data structure choices matter
- introduce standard data structures, design techniques, performance analysis
  - ✓ stacks, queues, sets, maps, linked structures, trees, graphs, hash tables
  - ✓ algorithm design, data structure selection/comparison/analysis
  - ✓ algorithm analysis, recurrence relations, counting & proof techniques

you should be familiar with these concepts (we will do some review next week, but you should review your own notes & text)

2

421: programming in the even larger

still not developing large-scale, multi-programmer systems
- see CSC 548, CSC 599

we will tackle medium-sized (3-10 interacting classes) projects in which
- there may be multiple approaches, with different performance characteristics
- the choice of algorithm and accompanying data structure is important
- the wrong choice can make a solution infeasible

- we will consider multiple design paradigms and problem characteristics that suggest which paradigm to apply
  - ✓ brute force, decrease & conquer, divide & conquer, transform & conquer
  - ✓ greedy algorithms, backtracking, dynamic programming, space/time tradeoffs

- we will also study the notions of computability and feasibilty
  - ✓ P vs. NP, NP-hard problems, approximation algorithms

3

When problems start to get complex…

…choosing the right algorithm and data structures are important
- e.g., phone book lookup, checkerboard puzzle

- must develop problem-solving approaches (e.g., divide&conquer, backtracking)
- be able to identify appropriate data structures (e.g., lists, trees, sets, maps)

		1			2			
	3	7	8					2
2	4						7	3
4						7	1	
			6		8			
	8	2						9
9	5						3	7
6				5	4	8		
			4		5			

EXAMPLE: solving a Sudoku puzzle
- need to be able to represent the grid
- devise an algorithm to fill in the blanks so that every row, column & subsquare contains 1-9

- how do you solve Sudoku puzzles?
- should the computer use the same approach?
- how complicated/fast would it be?

4

Another example

word ladders are puzzles in which you are to bridge from one word to another by changing only one letter

BUS

CAR

- need to be able to read and store a dictionary
- need to find a (shortest?) sequence between two words
- in doing so, need to recognize when words differ by 1 letter

- how do you complete a word ladder?
- should the computer use the same approach?
- how complicated/fast would it be?

NICK → … → WINS

DAVE → … → REED

5

Another example

suppose you are given a set of integers (positive and negative)

{ 4, -9, 3, 4, -1, -5, 8 }

is there a subset of integers that add up to 0?

- this is known as the *subset sum problem*

- it turns out that there is no known efficient algorithm to solve this
- may have to exhaustively try every possible subset of numbers

- how many subsets of N items can there be?

6

OOP and code reuse

when solving large problems, code reuse is important

- designing, implementing, and testing large software projects is HARD
  whenever possible, want to utilize existing, debugged code

- reusable code is:
  clear and readable (well documented, uses meaningful names, no tricks)
  modular (general, independent routines – test & debug once, then reuse)

OOP is the standard approach to software engineering

philosophy: modularity and reuse apply to data as well as functions

- when solving a problem, must identify the objects involved
  e.g., banking system: customer, checking account, savings account, …

- develop a software model of the objects in the form of abstract data types (ADTs)
  a program is a collection of interacting software objects
  can utilize inheritance to derive new classes from existing ones

7

HW1: Java & DS review

review exercise– NO LATE ASSIGNMENTS ACCEPTED

GOAL 1: get you back up to speed with programming
GOAL 2: identify gaps in your skills/knowledge & address them
GOAL 3: get you into the habit of coming to my office for guidance/assistance

idea: want to process viewer reactions to a political debate

- each viewer is given a series of dials, one for each candidate
- can turn a dial, positive or negative, in reaction to the candidate's statement
- e.g., 3 viewer logs:

```
Tromp -2          Hollery -1.5   Flanders 3.3
Rudio 2.5         Rudio 4        Tromp 1.0
Hollery 3         Tromp 1.1      Tromp -0.5
Hollery -3.2      Hollery 1.5    Flanders -1.5
Flanders -0.9     Rudio -2.0     Hollery 1.8
Rudio -1                         Rudio 0.1
Tromp -1
Hollery 2.6
Tromp 1
```

8

4

HW1 details

you are to design and implement a program to :

- read in a series of log files
- produce a list of candidates with total points, # pos/neg reactions
- average points & totals

follow good OO practices:

- cohesive & loosely coupled classes
- Javadoc comments & Java naming conventions
- follow MVC pattern: all I/O should occur in the driver class

```
Tromp -2
Rudio 2.5
Hollery 3
Hollery -3.2
Flanders -0.9
Rudio -1
Tromp -1
Hollery 2.6
Tromp 1
```

```
Hollery -1.5
Rudio 4
Tromp 1.1
Hollery 1.5
Rudio -2.0
```

```
Flanders 3.3
Tromp 1.0
Tromp -0.5
Flanders -1.5
Hollery 1.8
Rudio 0.1
```

```
Enter the log file name(s) on a single line:
log1.txt log2.txt log3.txt

CANDIDATE      TOTAL    # POS    # NEG
Hollery        4.2        4        2
Rudio          3.6        3        2
Flanders       0.9        1        2
Tromp         -0.4        3        3
------------------------------------
               2.1       11        9
```