CSC 421: Algorithm Design & Analysis
Spring 2013

See online syllabus: http://dave-reed.com/csc421 (also on BlueLine2)

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.

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
421: programming in the even larger

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

we will tackle medium-sized (3-8 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 feasibility
  - P vs. NP, NP-hard problems, approximation algorithms

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)

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?
Another example

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

- 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?

DAVE $\rightarrow$ ... $\rightarrow$ REED
ALEX $\rightarrow$ ... $\rightarrow$ REED

Another example

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

\{ 4, -9, 3, 4, -1, -4, 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?
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

NetBeans

in class, we will be using the NetBeans IDE (Java SE v7.0.1)
- freely available from http://netbeans.org/downloads for download & installation instructions

nice features:
- can easily set preferences & defaults
- can automatically generate javadocs
- code completion
- code refactoring