EE360C: Algorithms

Course Introduction

Spring 2018

Department of Electrical and Computer Engineering
University of Texas at Austin

Course Information

Who am I?

- Dr. Vallath Nandakumar
- Email: `vallathn@austin.utexas.edu`
- Office: EER 4.826
- Office hours: TBD
- Other section time and location: TTh 12:30-2pm, EER 1.516

Who teaches the other section?

- Dr. Christine Julien
- Email: `c.julien@utexas.edu`
- Office: EER 7.806
- Office hours: TTh 11am-12:20pm (right before class) or by appointment

Coordination of two sections

- **You need to take weekly quizzes in the section you are registered (the section quizzes will be different). Participation points may also depend on your attending your registered section.**
- You are welcome to attend other section also subject to available seats (the courses are close to room capacity)
- You are welcome to attend either professor's office hours
- The sections will share seven TAs
- The sections will have the same homeworks, programming assignments, midterms, and final exam
- The sections will share the same Piazza forum
- You are encouraged to both post and answer other student questions: this will count towards your participation grade

Teaching Assistants

- Cassidy Burden (`burdencassidy@utexas.edu`)
    - Office hours: TBD
- Guibing Cai (`guibing.cai@gmail.com`)
    - Office hours: TBD
- Jie Hua (`mich94hj@utexas.edu`)
    - Office hours: TBD
- Isidoro Tziotis (`isidoros_13@hotmail.com`)
    - Office hours: TBD
- Ka Tai Ho (`kataihoo@gmail.com`)
    - Office hours: TBD
- Guarav Nagar (`gnagar1996@gmail.com`)
    - Office hours: TBD
- Aravind Srinivasan (`aravindsrinivasan@utexas.edu`)
    - Office hours: TBD

Communicating with Us

- The best way to ask questions about lecture or assignments is through the discussion boards on Piazza (follow Piazza link on Canvas)
- We will all monitor the discussion board, and that way others can benefit from the answer to your question
- **Do not post partial problem solutions or code to the discussion board!**

Course Logistics

- Time and Location
    - TTh 12:30-2pm
    - EER 1.516
- Course description
    - We will study combinatorial algorithms, with a focus on theoretical style in lectures and practical application through assignments
- Prerequisites
    - M325K or PH313K: Discrete Mathematics
    - You should be comfortable writing, compiling, and debugging Java programs of moderate complexity (i.e., EE422C System Design and Implementation II isn't going to hurt)

Course Logistics (cont.)

- Textbook:
  - J. Kleinberg and E. Tardos. Algorithm Design. Addison Wesley, 2005.
- Recommended Texts:
  - T. H. Cormen, C. E. Leiserson, R. H. Rivest, and C. Stein. Introduction to Algorithms. McGraw-Hill, 2009 (Third Edition).
  - B. Eckel. Thinking in Java. Prentice Hall, 2006 (Fourth Edition).

Evaluation and Grading

- Evaluation
  - Weekly quizzes: 25% of grade
  - Programming assignments: 20% of grade
  - Exams: 50% of grade (2 in-class exams: 15% each; final exam: 20%)
  - In Class Participation: 5% of grade
- Grading Scale
  - 90-100%: A
  - 80-89%: B
  - 70-79%: C
  - 60-69%: D
  - 0-59%: F

Assignments

- Homework
  - given out weekly
  - not collected or graded
  - a weekly quiz similar to the homework will be graded
  - if you do the homework, the quiz should be straightforward
  - HW1 posted; first quiz in two weeks
- Programming Assignments
  - 3 programming assignments
  - due (approximately) two weeks after assigned **electronically** at 11:59pm (via Canvas)
  - **no late assignments will be accepted**

Exams

- **Exam** dates (tentative):
  - **Wednesday** February 22, 2018 at 7:00pm-8:30pm in TBD
  - **Monday** March 29, 2018 at 7:00pm-8:30pm in TBD
  - Final TBD–awaiting date and time assignment for uniform exam from Registrar's office
- All exams are cumulative.
- The lectures immediately prior to midterms are optional (see Course Plan on Canvas)

Course Expectations

- Attendance
  - You should attend class. Lecture notes will be made available, but they should not be considered a substitution for attending class.
  - You can’t get participation points if you don’t attend
- Collaboration
  - You can discuss both homework problems and programming assignments with other students *at a conceptual level*
  - Do **not** write or program while talking to a fellow student
  - Do **not** use any other resources without citation

Course Overview

Course Overview

- Review of Discrete Math and Proof Techniques
- Algorithm Analysis
- Graphs and Graph Algorithms
- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flow
- NP-Completeness

Reading Assignments

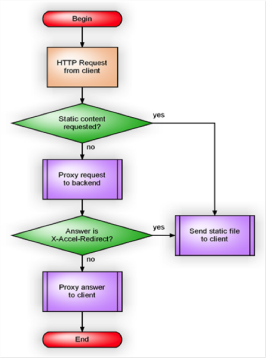
There's no way for me to enforce this, but I recommend you read the relevant chapters in advance of the class lecture. I will try to remind you what to read.

In that spirit, start reading Chapter 1.

Why Study Algorithms

What is an Algorithm?

- A step-by-step procedure to solve a problem
- Every program is the instantiation of some algorithm



http://blog.kovyrin.net/wp-content/uploads/2006/05/algorithm_c.png

A Canonical Example: Sorting

An algorithm solves a general, well-specified problem

Given a sequence of n keys, $a_1, \ldots, a_n$ as input, produce an output reordering (i.e., a *permutation*) $b_1, \ldots, b_n$ of the keys so that $b_1 \leq b_2 \leq, \ldots, b_n$.

The problem has specific instances

[Dopey, Happy, Grumpy] or [3, 5, 7, 1, 2, 3]

An algorithm takes any possible instance and produces output that has the desired properties

e.g., insertion sort, quicksort, heapsort ...

So What's the Challenge?

It is hard to design algorithms that are:

- correct
- efficient
- (easily) implementable

To do so effectively, we need to know about:

- algorithm design and modeling techniques
- existing resources (i.e., don't reinvent the wheel!)

Algorithm Correctness

How do you know an algorithm is correct?

It produces the correct output on every possible input (!)

- Since there are usually infinitely many inputs, ensuring this is not trivial
- Saying “it’s obvious” can be dangerous
- Often one’s intuition can be tricked by a particular type of input

The Tour Finding Problem

Given a set of n points in the plane, what is the shortest tour that visits each point and returns to the beginning?

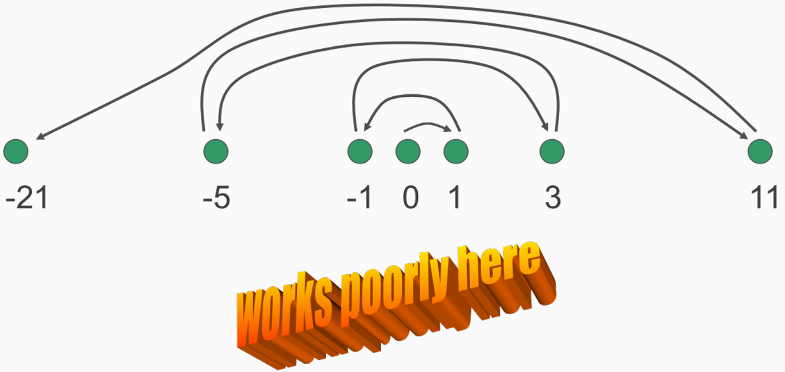
An Application

Consider a robot arm that solders contact points on a circuit board; we want to minimize the movement of the robot arm.

Finding a Tour: Nearest Neighbor

- Start by visiting any point
- While all points are not visited, choose an unvisited point closest to the last visited point and visit it
- Return to the first point

Nearest Neighbor Counter Example



So How Do We Prove Correctness?

- There exist formal methods (even automated tools), but they're largely beyond this course
- In this course, we'll primarily rely on more informal reasoning about correctness
- Often seeking counter-examples to proposed algorithms as an important part of the design process

Algorithm Efficiency

- Software and hardware are both continually advancing; advancing software constantly demands a faster CPU, more memory, etc.
- Given a problem:
  - What is an efficient algorithm?
  - What is the most efficient algorithm?
  - Does there even exist an algorithm?

Measuring Efficiency

We generally focus on machine-independent measures

Measuring Efficiency

- We analyze a “pseudocode” version of the algorithm
- We assume an idealized model of a machine in which one instruction takes one unit of time

“Big-O” notation

- Analyze the order of magnitude changes in efficiency as the problem size increases

We often focus on worst-case performance

- This is safe, the worst case often occurs frequently, and the average case is often just as bad

Some Caveats

- There's no point in finding the fastest algorithm for parts of the program that are not bottlenecks.
- If the program will only be run a few times or time is not an issue (e.g., the program will be left to run overnight), the there's no real point in finding the fastest algorithm.

Making it Easier

Cast your application in terms of well-studied data structures

Concrete	*Abstract*
arrangement, tour, ordering, sequence	permutation
cluster, collection, committee, group, packaging, selection	subsets
hierarchy, ancestor/descendants, taxonomy	trees
network, circuit, web, relationship	graph
sites, positions, locations	points
shapes, regions, boundaries	polygons
text, characters, patterns	strings

Some Real World Applications

- Hardware design: VLSI chips
- Compilers
- Computer graphics: movies, video games
- Routing messages in the Internet
- Searching the Web
- Distributed file sharing
- Computer aided design and manufacturing
- Security: e-commerce, voting machines
- Multimedia: CD player, DVD, MP3, JPG, HDTV
- DNA sequencing, protein folding
- ... and many more!

Some Important Problem Types

- Sorting (a set of items)
- Searching (among a set of items)
- String processing (text, bit strings, gene sequences)
- Graphs (modeling objects and their relationships)
- Combinatorial (find desired permutation, combination, or subset)
- Geometric (graphics, imaging, robotics)
- Numerical (continuous math, solving equations, evaluating functions)

Algorithm Design Techniques

- Brute force and exhaustive search
  - follow definition / try all possibilities
- Divide and conquer
  - break problem into distinct subproblems
- Transformation
  - convert one problem into another one
- Dynamic programming
  - break problem into overlapping subproblems
- Greedy
  - repeatedly do what is the best right now
- Iterative improvement
  - repeatedly improve current solution
- Randomization
  - use random numbers

Plan for This Course

- Cover a variety of fundamental algorithm design techniques as applied to a number of basic problems
- Along the way, infuse discussions with different types of algorithm analysis
- Study some lower bounds, indicating inherent limitations in finding efficient algorithms
- Learn about undecidability: some problems are simply unsolvable

Questions