6.854 — Advanced Algorithms

Erik Demaine and David Karger

Handout #1, September 3, 2003 — Class Outline

Summary: The need for efficient algorithms arises in nearly every area of computer science. But the type of problem to be solved, the notion of what algorithms are "efficient," and even the model of computation can vary widely from area to area. In this second class in algorithms, we will survey many of the techniques that apply broadly in the design of efficient algorithms, and study their application in a wide range of application domains and computational models. Techniques to be covered include amortization, randomization, fingerprinting, word-level parallelism, bit scaling, dynamic programming, network flow, linear programming, fixed-parameter algorithms, and approximation algorithms. Domains include string algorithms; network optimization; parallel algorithms; computational geometry; online algorithms; external memory, cache, and streaming algorithms; and data structures.

Goals: Our hope is that this class will confer

- some familiarity with several of the main thrusts of work in algorithms—sufficient to give you some context for formulating and seeking known solutions to an algorithmic problem;
- sufficient background and facility to let you read current research publications in the area of algorithms;
- a set of tools for design and analysis of new algorithms for new problems that you encounter.

Class webpage: `http://theory.lcs.mit.edu/classes/6.854/`

Lecturers: Erik Demaine `edemaine@mit.edu`. URL: `http://theory.lcs.mit.edu/~edemaine`
David Karger, `karger@mit.edu`. URL: `http://theory.lcs.mit.edu/~karger`

TA: Grant Wang, `gjw@mit.edu`. URL: `http://theory.lcs.mit.edu/~gjw`

Content: The goal is for the class to be broad rather than deep. Our plan is to touch upon the following areas. This is a tentative list of topics that *might* be covered in the class; we will select material adaptively based on background, interests, and rate of progress.

String Algorithms: Rabin-Karp fingerprinting algorithm. Suffix trees.

Data Structures: More advanced solutions to basic data structuring problems: Fibonacci heaps. Van Emde Boas priority queues. Dynamic data structures for graph connectivity/reachability.

Bit Tricks: Word-level parallelism. Transdichotomous model. $o(n \log n)$ integer sorting.

Maximum Flows: Augmenting paths and push-relabel methods. Minimum cost flows. Bipartite matching.

Linear Programming: Formulation of problems as linear programs. Duality. Simplex, interior point, and ellipsoid algorithms.

Online Algorithms: Ski rental. River search problem. Paging. The k-server problem. List ordering and move-to-front.

Approximation Algorithms: One way of coping with NP-hardness. Greedy approximation algorithms. Dynamic programming and weakly polynomial-time algorithms. Linear programming relaxations. Randomized rounding. Vertex cover, wiring, and TSP.

Fixed-Parameter Algorithms: Another way of coping with NP-hardness. Parametrized complexity. Kernelization. Vertex cover. Connections to approximation.

Parallel Algorithms: PRAM. Pointer jumping and parallel prefix. Tree contraction. Divide and conquer. Randomized symmetry breaking. Maximal independent set.

External-Memory Algorithms: Accounting for the cost of accessing data from slow memory. Sorting. B-trees. Buffer trees. Cache-oblivious algorithms for matrix multiplication and binary search.

Computational Geometry: Convex hull. Line-segment intersection. Sweep lines. Voronoi diagrams. Range trees. Seidel's low-dimensional LP algorithm.

Streaming Algorithms: Sketching. Distinct and frequent elements.

Prerequisites: We assume that the reader has had an undergraduate class in Algorithms (6.046) and some exposure to probability (6.041 or 6.042 are more than sufficient). Complexity Theory (6.045) is a bonus.

Requirements:

Scribing and/or grading (5%). Help scribe a lecture in LaTeX and/or help grade a problem set.

Homework (70%). Weekly problem sets, with collaboration usually allowed.

Reading/Research/Implementation project (25%). You will read a new (not yet textbook) algorithm from the recent research literature, and improve upon it via some mixture of the following:

- Write a description *of greater clarity* than the original publication, or
- Devise an improved solution to the problem under consideration, and write up your improvement (with appropriate discussion of the original algorithm).
- Implement the algorithm in order to study its performance in practice. Considerations include choice of algorithm, design of good tests, interpretation of results, and design and analysis of heuristics for improving performance in practice.

Ideally, the algorithm you choose will be motivated by a computational problem you need to solve. If the project is large, it may be tackled by a group.

Collaboration Policy. Collaboration is encouraged on all aspects of the class, except where explicitly forbidden. Note:

1. All collaboration (who and what) must be clearly indicated in writing on anything turned in.
2. Homeworks may be solved collaboratively except as explictly forbidden, but solutions must be written up independently. Groups should be small enough that each member plays a significant role.
3. For projects every collaborator must contribute significantly to reading, implementation, and writeup. To allow this, groups should limit their size to 3 unless the project is unusually large.

Textbooks. There are no textbooks covering a majority portion of the material we will be studying in this class. Roughly 20% of the material is covered in Goemans' lecture notes (see below). Our lectures will often draw from the following (optional) texts, all of which we consider nice to have.

1. Michel Goemans. Class notes for *Advanced Algorithms*, at `http://theory.lcs.mit.edu/~goemans`.
2. Cormen, Leiserson, Rivest, and Stein. *Introduction to Algorithms.* MIT Press. 2001.
3. Ahuja, Magnanti, and Orlin. *Network Flows.* Prentice Hall, 1993.
4. Motwani and Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
5. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
6. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.
7. Robert Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983. A classic—no longer up to date, but outstanding writing.
8. Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer Verlag, 2000.
9. Dorit Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* Brooks Cole, 1996.