Introduction to Analysis of Algorithms — Final Exam Review

CS4820 Spring 2013 — Sunday, May 12, 2013

The general comments from the Prelim 1 and Prelim 2 reviews, available on the handouts page, still apply. You should review these materials thoroughly.

Topics

The final exam is cumulative. It will cover the following general topics:

1. Basic techniques (graph searching, matching, minimum spanning trees, greedy algorithms, divide and conquer, dynamic programming, solution of recurrences): K&T Chs. 1, 4–6;
2. Flow problems and algorithms: K&T Ch. 7 and the handouts on the Edmonds–Karp algorithm and the Dinic and MPM algorithms;
3. NP-completeness: K&T Ch. 8 and the handouts on reductions and NP-completeness and tbe Cook–Levin theorem;
4. Turing machines and general computability: the handouts on Turing machines and clocked diagonalization;
5. Approximation algorithms: K&T Ch. 11 §1–3, 8.

The following is a list of things you will need to know for each of these general topics.

Basic Techniques

- graph searching with DFS and BFS, strongly connected components of a directed graph, connected components of an undirected graph, topological sort;
- the definition of a matching;
- the Gale–Shapley algorithm for stable matchings;
- the Kruskal and Prim algorithms for minimum-weight spanning trees;
- divide-and-conquer algorithms for searching, sorting, and median finding;
- dynamic programming algorithms for string matching, weighted interval scheduling, shortest paths (Bellman–Ford algorithm);
- techniques for the analysis of dynamic programming algorithms;
- pseudo-polynomial time dynamic programming algorithms for Subset Sum, Knapsack, Partition;
- techniques for solving recurrences.

You may be asked to come up with a dynamic programming algorithm for some problem. The first thing to do is to determine how an optimal solution for a given instance of the problem depends on optimal solutions for smaller instances. Then write down a recurrence, give an slgorithm for solving it, and analyze the running time of your algorithm.

Flow Problems and Algorithms

- the formal definition of flow graph and flow;
- the formal definition of a circulation graph and circulation, including circulations with lower bounds;
- what a residual graph is and how to construct one;
- what an augmenting path is and how to find one;
- what a level graph is;
- the various flow algorithms and their complexities, including Ford–Fulkerson and Edmonds–Karp;
- how to find a max flow in a given flow graph;
- how to find a min s, t-cut in a given flow graph;

You may be given a problem in English that can be solved by constructing a flow or circulation graph and finding a max flow, min cut, or circulation.

Reductions and NP-Completeness

- the formal definition of polynomial-time reduction between decision problems;
- the definition of the class NP;
- the definition of NP-hardness and NP-completeness;
- the definition of various NP-complete problems studied in class, including
  - satisfiability problems: Boolean satisfiability, the definition of conjunctive normal form, CNFSAT, 3CNFSAT (aka 3SAT);
  - graph problems: Clique, Independent Set, Vertex Cover, Dominating Set, Colorability, Planar 3-colorability;
  - covering problems: Set Cover, Exact Cover (XC), Exact Cover by 3-sets (X3C), 3-dimensional matching (3DM);
  - tour problems: directed and undirected Hamiltonian circuit (HC), Traveling Salesperson (TSP);
  - numerical problems: Subset Sum (SS), Partition, Knapsack;
  - other applications: Bin Packing, Linear and Integer Programming (LP, IP).
- how to do a reduction.

For the last, you may be given a problem B and asked to prove that it is NP-complete. To do this, there are two things you have to do.

1. Show that the problem is in NP by giving a guess-and-verify algorithm for it.
2. Show that the problem is NP-hard by selecting a known NP-hard problem A and reducing A to B. This involves
   (a) describing a polynomial-time transformation σ that constructs an instance $\sigma(x)$ of problem B from a given instance x of problem A;
   (b) showing that $\sigma(x)$ is a "yes" instance of problem B if and only if x is a "yes" instance of problem A.

Turing Machines and General Computability

- the statement of Church's thesis;
- the formal definition of a one-tape deterministic Turing machine, including the definition of configurations and acceptance;
- what it means for a Turing machine to be total;
- what it means for a problem to be decidable;
- what it means for a set to be r.e. or recursive;
- how a universal Turing machine works;
- the definition of the halting and membership problems for Turing machines;
- diagonalization proof of the undecidability of the halting problem;
- clocked diagonalization to construct a decidable problem not contained in a given complexity class;
- how to do a reduction to show that a problem is not decidable or not r.e.

You may be asked to construct a Turing machine to perform some simple task, or to show that a problem is undecidable or not r.e. by constructing a reduction from a problem that is known not to be decidable or r.e. You may be given a list of problems and asked to determine whether they are decidable, r.e. but not decidable, or not r.e. Look at the list of problems in the handout on Turing machines, §5, for examples.

Approximation Algorithms

- approximation techniques for load balancing problems;
- scaling approximation techniques for pseudo-polynomial time problems.

You may be asked to give an approximation algorithm for a variant of one of the problems on Homework 8.

Sample Problems

Here are some suggested problems to give you some practice with good exam-level questions. Exam-level questions are questions that you should need to think about, but not for very long. The first things that comes to mind should be pretty close to the intended solution.

Besides these suggested problems, the first ten or so problems of each chapter are generally good practice problems.

Do not take the number of decidability problems suggested to be indicative of the degree of emphasis on the exam. I have just tried to included a few extra problems because there are no sample decidability problems in K&T.

Greedy: K&T Ch. 4 Ex. 1, 2, 3, 6

Divide and Conquer: K&T Ch. 5 Ex. 2, 3

Dynamic Programming: K&T Ch. 6 Ex. 2, 3, 9, 13

Network Flow: K&T Ch. 7 Ex. 6, 9, 14, 15[1]

NP-Completeness: K&T Ch. 8 Ex. 4[2], 6, 10(a), 18

[1] For 15(a), you may want to write out a full solution (reduction, runtime, proof) explaining any relevant details from the bipartite matching problem.

[2] An exam question consisting of all four parts of this problem would be considered too lengthy. Realistically, to deal with time constraints, a final exam question would ask you to solve at most two of the four parts of this question.

Decidability: Determine whether each of these languages is decidable, r.e. but not decidable, or not r.e. Prove your answer. It may be good to go through an iteration of trying to think what each should be without proof, checking your intuition with us if you are not sure, then flushing out full proofs.

1. The set of all TMs with input alphabet $\{a, b\}$ that accept the string aab.
2. The set of all TMs with input alphabet $\{a, b\}$ that reject the string aab.
3. The set of all TMs with input alphabet $\{a, b\}$ that loop on the string aab.
4. The set of pairs of TMs (M, N) such that $L(M) = L(N)$.
5. The set of all TMs M such that $L(M)$ contains at least 4820 elements.
6. The set of all TMs M such that $L(M)$ contains at most 4820 elements.
7. The set of all TMs M such that $L(M)$ is infinite.
8. The set of all TMs M such that $L(M)$ is finite.
9. The set of all TMs M with input alphabet $\{a, b\}$ such that $L(M) = \{a, b\}^*$.

Approximation: K&T Ch. 11 Ex. 7, 9, 10