CSE 101: Review topics

In preparation for final: June 8, 2005

Review:

- chapters 1-8 (skip 4.8, 4.9, 5.6, 7.4, 7.7, 7.9, 7.13, 8.9)
- Midterms.
- Slides.
- Edmonds book, chapter 8.

1. Introduction
   (a) Stable matching problem and algorithm
2. Basics of Algorithm Analysis
   (a) Definition of O, Θ, Ω.
   (b) Proving a function is in a complexity class
   (c) Proof by induction
   (d) Preconditions of an algorithm
   (e) Postconditions of an algorithm
   (f) Loop invariants: summarizing the essence of the result of the loop
3. Graphs
   (a) Graphs
   (b) Vertices
   (c) Edges
   (d) Directed Graphs
   (e) Breadth-first Search
   (f) Depth-First Search

   (g) Connected Graph
   (h) Acyclic Graph
   (i) Tree
   (j) Connected Components

4. Greedy algorithms
   (a) Optimal substructure (optimal solution has subproblem with optimal subsolution)
   (b) Greedy: There exists an optimal solution that uses the greedy choice
   (c) Minimal Spanning Trees
      i. Prim's algorithm
      ii. Kruskal's algorithm
   (d) Shortest Path (Dijkstra's algorithm)

5. Divide and Conquer
   (a) Divide into smaller problems
   (b) Conquer the smaller problems
   (c) Combine the solutions
   (d) Show runtime
   (e) Solving recurrence relations:
      i. Master method
      ii. Tree method
      iii. by induction
   (f) Quicksort. Worst-case: $O(n^2)$ time. Average-case: $O(n \log n)$ time. In-place, divides input into two (possibly unequal) sets based on splitter value. Recursively subsorts two sets.
   (g) Mergesort: Worst-case: $O(n \log n)$ time. Divides input into left-half and right-half. Merges recursively sorted subsets.
   (h) Lower bound on comparison-based sorting
      i. Decision tree for comparison based sorting algorithm shows each comparison made.

ii. Tree must have at least $n!$ leaves (to distinguish all possible input permutations).
iii. Height of tree must be at least $\log(n!) = \Omega(n \log n)$.
iv. At least one path from root to leaf must be at least $\Omega(n \log n)$ long.
v. Any comparison-based sorting algorithm has at least one input that takes $\Omega(n \log n)$ comparisons.

6. Dynamic Programming
   (a) Define notation (what does the recurrence and parameters mean?)
   (b) Show recurrence with Optimal substructure (optimal solution has subproblem with optimal subsolution).
   (c) Explain why recurrence is correct
   (d) Show memoization (top-down), or build up array (bottom-up) to eliminate exponential calculations
   (e) Show runtime (number of cells in array * time to compute each cell).
7. Backtracking
   (a) Systematic search through decision tree
   (b) Don't search farther down tree if solutions in that subtree are infeasible
   (c) Branch and bound (for optimization problem)
      i. Keep value of best solution so far.
      ii. Don't search farther down tree if solutions in that subtree can't beat best solution
8. Network Flow
   (a) Definition of flow network
      i. Directed graph G=(V, E) with designated source, s, and sink, t, nodes
      ii. No edges go into s or out of t

iii. Positive capacities, $c(e)$, per edge, e

iv. Flow, f

   A. Assigns a flow, $f(e)$ to every edge, e

   B. $0 \leq f(e) \leq c(e)$ (*capacity constraint*)

   C. $\forall v \in V - s, t, \sum_{\text{edges e into v}} f(e) = \sum_{\text{edges e out of v}} f(e)$ (*conservation*)

v. The value of a flow $v(f)$ = $\sum_{\text{edges e out of s}} f(e)$

(b) Cut (A, B) is a partition of nodes A and B, with $s \in A$ and $t \in B$.

(c) Capacity of a cut $cap(A, B)$, is the capacity of the edges leaving A and entering B.

(d) Net flow across a cut, $f(A, B)$, is the flow of the edges from A to B minus the flow of the edges from B to A

(e) Given a flow f, net flow across any cut is equal to $v(f)$

(f) The max flow, $v(f^*)$ is equal to the minimum capacity of any cut.

(g) Floyd-Fulkerson algorithm. Runtime: $O(E \cdot V \cdot C)$ (where C = maximum capacity of an edge).

(h) Residual graph

(i) Augmenting paths

(j) A flow is maximum if no augmenting path can be found in the residual graph.

(k) Edmonds-Karp algorithm: choose augmenting path with fewest number of edges. Runtime: $O(V \cdot E^2)$.

(l) Show solving a problem using network flow:

   i. How you build the flow network G (a picture often helps, but is not normally sufficient).

   ii. State that you run Edmonds-Karp algorithm on G to determine f, the maximum flow in $O(V \cdot E^2)$ time.

   iii. Give a proof that there's a solution to the problem iff the graph has a particular flow.

      $\rightarrow$ If there's a solution to the problem, a network flow can be constructed.

$\leftarrow$ If there's a particular network flow, that leads to a solution

iv. Runtime

9. NP

(a) A decision problem X is in NP if:

- there exists a certifier that can verify an instance I using a certificate C (polynomial in the input), in polynomial time (certifier can verify that I is in X).
- For every instance I in X, there exists a certificate C

(b) Problem A is polynomial-time reducible to problem B ($A \leq_P B$) if there is an algorithm for A that:

- Runs in polynomial time plus
- Makes polynomially many calls to an oracle for B

A problem A is NP-hard if all problems in NP are polynomial-time reducible to A

A problem A is NP-complete if A is NP-hard and A is in NP.

If any NP-complete problem has a polynomial-time algorithm, then all NP problems have polynomial-time algorithms (P=NP).

If some NP-problem can't be solved in polynomial time, then no NP-complete problem can be solved in polynomial time.

How to show a problem Y is NP-complete:

- Choose an NP-complete problem X
- Show $X \leq_P Y$
  - Show algorithm with polynomial number of steps
  - Show polynomial calls to problem Y
  - Show correctness of algorithm (iff)
- Show X is in NP

Halting problem is undecidable

Other problems are undecidable. For example, does program return 0 when given input I? If we could determine that, then we could decide whether a program halts when given input I. "Does A(I) halt?" is equivalent to "Does A'(I) return 0?"

```
A′(I)
  A(I)
  return 0
```