Basic Algorithms (V22.0310); Fall 2005; Yap

STUDY QUESTIONS for FINAL EXAM (with HINT)

Dec 14, 2005

Instructions

- Please study these questions and hints carefully.
- They are designed to help you prepare for the FINAL EXAM.
- They will form the basis for some questions (but I will ask some unrelated questions as well).
- The final Exam will be based on EVERYTHING in the reading assignments as well as what was covered in lectures. But we will emphasize material after midterm, and those found in homework and lectures.
- Do not forget to prepare two sheets of notes (8"x11") to bring to our closed book final.

What to Study

Besides the study questions below, what else should you study?

- Check our Reading List found in the homework directory (this list has been updated for the final exam). Try to do exercises from the assigned reading (we can help if you get stuck in them).
- Read up all our solutions to homework and quizes. Try to reconstruct the solutions on your own first!
- Review your lecture notes.
- The Study Questions we prepared for Quiz2 is still valid.

Master Theorem

You must know the Master Theorem (not in the text, but covered in lectures). It is important to know how to bound OTHER kinds of recurrences using the Master Theorem.

(a) Consider $T(n) = 2T(n/2) + 3T(n/3) + n$. This is NOT a Master recurrence. You have no chance if you tried the Rote method (but please go ahead and try to see what happens). Nevertheless, the Master Theorem is very useful in bounding such recurrences.

Give upper and lower bounds on $T(n)$ using the Master theorem.

(b) Here is another one: $T(n) = 9T(n/3) + n^2 \log n$.

(c) Here is another one: $T(n) = 2T(n/3) + 3T(n/4) + n^2 \log n$.

(d) You should also know how to estimate (without calculators) logarithms. E.g., consider the following two recurrences:

$$\begin{aligned} T_2(n) &= 12T_2(n/2) + n \\ T_3(n) &= 36T_3(n/3) + n \\ T_4(n) &= 61T_4(n/4) + n \end{aligned}$$

Order these complexity functions in increasing O-order.

SKETCH SOLUTION:

(a) Upper bound by $T_1(n) = 5T_1(n/2) + n$, and lower bound by $T_0(n) = 5T_0(n/3) + n$.

(b) Upper bound by $T_1(n) = 9T_1(n/3) + n^{2+\varepsilon})$ for any $\varepsilon > 0$, and lower bound by $T_0(n) = 9T_0(n/3) + n^2)$.

(d) $\log_2(12) > 3$, $\log_3(36) > 3$, $\log_4(61) < 3$. So T_4 is smallest. $\log_2(12) = 2 + \log_2(4)$ and $\log_3(36) = 2 + \log_3(4)$. But $\log_2(4) > \log_3(4)$. So clearly T_2 is largest.

Geometric Series

The basic properties of geometric series should be mastered. You should the how to sum the following: $\sum_{i=1}^{n} x^i$ and also $\sum_{i=1}^{\infty} x^i$ (when $|x| < 1$).

(a) What are these values? Give the Θ-order of these sums.

(b) Prove $x^2/2^x \leq 1/c^x$ when x is sufficiently large (i.e., eventually).

(c) Suppose we have a binary tree with $2^n - 1$ nodes and height $n - 1$. Suppose that at *each* node, you need to do some amount of work. Let the work done at a node of height i be $O(i^2)$. What is the total amount of work summed over all the nodes? HINT: Recall the solution to hw2, question 6 on heaps.

SKETCH SOLUTION:

(a) Standard stuff. For $|x| < 1$, just remember that the sum is $\Theta(1)$.

(b) We use the fact that $\lg x \leq Cx$ (eventually) for any $C > 0$. Now $x^2 \leq (2/c)^x$ iff $2 \lg x \leq x \lg(2/c) = x - x \lg c$. But if we choose any c with $1 < c < 2$, then $0 < \lg c < 1$, and so the inequality is eventually true.

(c) Look for the geometric series in here! There are at most 2^{n-1} nodes of height 0, and at most 2^{n-2} nodes of height 1, etc. In general there are at most 2^{n-h} nodes of height $h - 1$. Thus the work done among nodes of height h is at most $h^2 2^{n-h} = 2^n(h^2/2^h)$. Summed over all all h, we get a total work of $2^n \sum_h (h^2/2^h)$. But we saw in (b) that this sum is $\Theta(1) + 2^n \sum_h (1/c)^h$ which is $\Theta(1) + 2^n \Theta(1) = \Theta(2^n)$.

Divide and Conquer

Suppose that instead of finding the closest pair of points in an input set $P = \{p_1, \ldots, p_n\}$ of points, we want to find the minimum triangle. That is, we want to find a triple $\{p, q, r\} \subseteq P$ such that the distance

$$d(p, q, r) := d(p, q) + d(q, r) + d(r, p)$$

is minimum where $d(p, q)$ is the usual Euclidean distance. For simplicity, we just

(a) Give a naive solution and its complexity.

(b) If $d(p, q, r) = D$, how big can $d(p, q)$ be as a function of D?

(c) Can you generalize the divide-and-conquer approach of finding closest pair?

SKETCH SOLUTION:

(a) This is a classic triple loop to find all triples of points, and keeping track of the mimimum found so far. You already saw this for the double loop case.

(b) $d(p, q) \leq D/2$. This equality is achieved when r is lies between p and q on a straightline.

Dynamic Programming

In text editing, we have three kinds of operations: insert a new character, delete a new character, replace a new character. The **edit distance** $E(X, Y)$ is the minimum number of edit operations to transform X to Y. E.g., $X = good$ and $Y = bad$. You can transform X to Y in 3 operations: delete g, and replace oo by ba. It is clear that you cannot do this in two operations. Hence $E(good, bad) = 3$.

(a) Give the precise relation between this problem and the Alignment Problem.

(b) Give the dynamic programming principle for computing $E(X, Y)$.

(c) Using (b), compute $E(nice, super)$. Organize your working in a suitable table.

(d) Write the pseudo-Java code for implementing (b).

SKETCH SOLUTION:

(a) It is the Alignment Problem, with $\delta = 1$ and $\alpha_{x,y} = 0$ if $x = y$ and $\alpha_{x,y} = 1$ otherwise.

(b) Let $X = X'x$ and $Y = Y'y$. Then $E(X, Y) = 1 + E(X', Y'$ if $x = y$. Otherwise, $E(X, Y) = 1 + \min\{E(X', Y), E(X, Y')\}$.

(c) This is organized in a matrix form, as in the alighment problem.

(d) Straightforward.

Reducibility

A basic skill in understanding NP-Completeness is the ability to do reducibility among computational problems. You also need to know some basic NP-Complete Problems. Often, these problems are closely related. For example, we mention the following pairs of related problems:
Satisfiability vs. 3-SAT,
Vertex Cover vs. Set Cover,
Independent Set vs. Set Packing,
Hamiltonian Cycle vs. Traveling Salesman.

Let us consider the Traveling Salesman Problem (TSP), you are given a bigraph $G = (V, E)$ with edge weights $W : E \to \mathbb{R}_{>0}$ (i.e., edges have positive weight). Write $G = (V, E; W)$ for this weighted graph. A **tour** is a path $\pi = (v_0, v_1, \ldots, v_n)$ that visits all the $n = |V|$ vertices in V exactly once, and returns to the starting city (so $v_0 = v_n$). The cost of π is just the sum of the edge weights: $C(\pi) = \sum_{i=1}^{n} W(v_{i-1}, v_i)$. Consider the following 3 versions of the Traveling Salesman Problem:

- TSP: Given $G = (V, E; W)$, compute a minimum tour π, i.e., π such that $C(\pi)$ is minimum.
- TSC: Given $G = (V, E; W)$, compute the cost $c^* = C(\pi)$ of a minimum tour.
- TSD: Given $G = (V, E; W)$ and an integer $k > 0$, determine whether the cost $C(\pi)$ of a minimum tour is at most k.

(a) The following should be easy: show that

$$TSD \leq_P TSC \leq_P TSP.$$

(b) This is harder: show that $TSP \leq_P TSC$? HINT: Note that it does not matter what the starting vertex v_0 is. But how can you use TSC to help you determine whether there is an optimum tour that begins with the edge (v_0, v_1)?

(c) How do results such as (a) and (b) help to justify our formal definition of the classes P and NP to be sets of languages (a language is just a set of strings over some alphabet).

(d) Do Exercise 1, p. 505.

(e) Exercise 2, p. 505.

SKETCH SOLUTION:

(a) To reduce TSD to TSC: on input G, k, we just compute TSC on G to get the minimum cost c^*. Return yes iff $c^* \leq k$. The other reduction is just as simple.

(b) Suppose you already know that $P_i = (v_0, v_1, \ldots, v_i)$ is the initial part of a minimum cost tour. Let G_i be obtained from the original graph by deleting any edge (u, v) where $v \in \{v_1, v_2, \ldots, v_i\}$, unless (u, v) is an edge in the path P_{i+1}.

Our goal is to find some v_{i+1} such that $P_{i+1} = (v_0, \ldots, v_i, v_{i+1})$ is also the initial part of a minimum cost tour. To do this, we take the graph G_i and start to modify it as follows: pick any edge of the form (v_i, u). Delete it. Then call TSC on this graph, and check if the optimum cost has increased with this modification. If so, we can set v_{i+1} to u. If not, we pick another edge (v_i, u') and delete it as well (note that (v_i, u) is NOT put back into the graph). We keep doing this until we find one edge (v_i, u') whose deletion increases the TSC solution. We have thus extended P_{i+1}.

Eventually, we would have reconstructed the optimum tour and solved TSP. Note that we call TSC at most $|E|$ times in this reduction.

(c) Such results tells us that, for studying polynomial solvability, it is enough to study decision problems. So P and NP stands a simple surrogates for the corresponding optimization problems.

(d) Exercise 1, p. 505. The Interval Scheduling decision problem is clearly in P. Hence the answer is "Yes" to the first part. The answer to second part is clearly "Unknown" since that would imply $P = NP$.

(e) Exercise 2, p. 505. This is just the set packing problem in disguise! So it is NP-complete.

Local Search

Local Search has this abstract framework. Assume (p.663) that we have the set $\mathcal{C}$ of all possible solutions and a cost function $c : \mathcal{C} \to \mathbb{R}$. We want to find a **optimum solution** $S^* \in \mathcal{C}$, i.e., one whose cost $c(S^*)$ is minimum. Let us simplify the problem and assume we just want to compute the **optimum value**, $c(S^*)$.

(a) The **Metropolis Algorithm** for local search requires a **neighborhood structure** N, namely for each $S \in \mathcal{C}$, define $N(S) \subseteq \mathcal{C}$ with the property that for all S and S', (1) $S \in N(S)$ and (2) $S \in N(S')$ iff $S' \in N(S)$. Here is another rendition of the Metropolis Algorithm (p.668) with some input parameters:

```
MetropolisSearch(T, N, S_0, n)
    Input: T = temperature,
                N=neighborhood structure, n=number of steps
▷ Initialize
    c* ← +∞    ◁ optimal value
    S ← S_0    ◁ current solution
▷ Loop
    for i = 1 to n
    1.  Pick S' ∈ N(S) uniformly at random.
    2.  if c(S') < c(S)
            c* ← c(S'), S ← S'
    3.  else with probability e^{-(c(S')-c(S))/k_0 T}    ◁ k_0 > 0 is constant
            S ← S'
    return(c*)
```

Modify the above algorithm to give the corresponding Steepest Descent Algorithm: use the input parameters N, S_0, n as in the Metropolis Algorithm (but we no longer need the Temperature variable or use probability). In Steepest Descend, you can also before n steps if you already reach a local minima.

(b) How can you experimentally adjust the parameters T, N, S_0, n and call this algorithm repeatedly to find a good c^*?

(c) Consider the complete bipartite graph $K_{m,n}$ which has two disjoint sets of vertices $V = \{v_1, \ldots, v_m\}$ and $U = \{u_1, \ldots, u_n\}$. The edges are (v_i, u_j) for all i and j. Let S be a vertex cover of $K_{m,n}$. Prove that $|S| \geq \min\{m, n\}$.

(d) Let $G = (V, E)$ be an input instance for the vertex cover problem. Let $\mathcal{C}(G)$ be the set of vertex covers of G and $c(S) = |S|$. We define a **k-neighbor** of a vertex cover $S \subseteq V$ to be vertex cover $S' \subseteq V$ such that

$$|S \oplus S'| \leq k.$$

Here, $A \oplus B = (A \setminus B) \cup (B \setminus A)$ denotes the symmetric difference of two sets. Write $N_k(S)$ to be the set of k-neighbors of S. The text just considered the case where $k = 1$. When $B = \{b\}$, we can write $A \oplus b$ for $A \oplus B$.

What is the behavior of Steepest Descent when $\mathcal{C} = \mathcal{C}(K_{1,n})$ when the neighborhood structure is $N = N_1$? What if $N = N_2$?

(e) Generalize your answers to (d) to say how we should find the optimum solution for $\mathcal{C}(K_{m,n})$.

SKETCH SOLUTION:

(a) For Steepest Descent, instead of picking a random $S' \in N(S)$ we pick the best solution (minimal $c(S')$). There is no temperature (or set $T = 0$).

(b) This is a heuristic. You begin with a low temperature, and slowly increase it to see if you get any better solution. You keep doing it as long as you improve. You can also use larger and larger neighborhoods (like $N_k(S)$ for $k = 1, 2, \ldots$) then you can also begin with a small neighborhood, and slowly increase the size of neighborhoods until no improvements are seen. Note that increasing either temperature or neighborhoods slows down your algorithm. You can try a combination of both in the Metropolis Algorithm.

(c) If S does not contain the entire set V and also does not contain the entire set U, then any $v \in V \setminus S$ and $u \in U \setminus S$ yields an edge (u, v) that is not covered by S. Hence $V \subseteq S$ or $U \subseteq S$. Thus $|S| \geq \min\{|V|, |U|\}$.

(d) We discussed this in class: with N_1, we could get stuck in a local minimum (see also the text). With N_2, we could never get stuck.

(e) With $K_{m,n}$ where $m < n$, then we could get stuck using neighborhoods of N_m: To see this, Steepest descent will try to reduce the current cover by removing m vertices. These m vertices might all come from V, as $|V| = m$. This instantly gives a local minima that is not global. Next consider N_{m+1} neighborhoods: In the first step, we can remove $m+1$ points all from U, but cannot remove $m+1$ points which include any element from V. Hence after the first step, the cover will have size $n-m-1$. After this first step, any cover must include all of V. This means that we will eventually get the global minimum cover, viz., V.