Fundamental Algorithms (G22.1170); Fall 2005; Yap

STUDY QUESTIONS for FINAL EXAM

Dec 16, 2005

INSTRUCTIONS:

- Please study these questions. First try to solve it yourself without using using the hints.
- 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.

Pure Graph Algorithms

In HW3, we defined the concept of a bottle neck in a DAG. Some of you noted that our definition of bottleneck is not quite the intuitive one. Let us explore this: an edge (v, w) in a DAG G is called a **by-pass** for a node u if v is an proper ancestor of u, and w is a proper descendent of u. Let us define u to be a **real bottleneck** if, in addition to being a bottleneck, there does not exist any by-passes for u. Give an efficient algorithm to determine if a DAG has any real bottlenecks; state its complexity.

HINT: Let $D[v]$ denote the number of descendents of $v \in V = \{1, \ldots, n\}$. Note that v is considered its own descendent, but an improper descendent. How can the array $D[1..n]$ help determine if an edge is a by-pass for a vertex?

MORE HINT: First compute $D[1..n]$. Now you can check if each u is a real bottle neck in $O(m + n)$ time. Overall complexity to compute $D[1..n]$ and to see if there is a real bottleneck is $O(n(m + n))$.

Pure Graph Algorithms

Let $N[v]$ be the number of distinct paths originating from each vertex v of a DAG G. E.g., let G be the "straightline graph" with vertices $v_1, \ldots, v_n$ with edges (v_i, v_{i+1}) for $i = 1, \ldots, n-1$. Then $N[v_i] = n - i + 1$ for $i = 1, \ldots, n$. Give an efficient algorithm to compute $N[v]$ for all vertices v, and state its complexity.

HINT: Let the vertices adjacent to u be $v_1, \ldots, v_k$. Give a formula for $N[u]$ in terms of $N[v_i]$ $(i = 1, \ldots, k)$.

MORE HINT: Use the POSTVISIT routine to compute the formula $N[u] = 1 + N[v_1] + \cdots + N[v_k]$.

Huffman Code

Let $s \in \Sigma^*$ be a string. Suppose $C : \Sigma \to \{0, 1\}^*$ is the static Huffman code used to encode s. Then we need to transmit both $C(s) \in \{0, 1\}^*$ as well as some representation of C. Let us see how to construct such a representation $r(C) \in \{0, 1\}^*$ of C.

Assume $\Sigma^* \subseteq \{0, 1\}^t$. E.g., $t = 8$ for ASCII code. The receiver knows t and that Σ is a subset of $\{0, 1\}^t$, but knows nothing else about Σ. Note that C may be represented by a full binary tree T whose leaves are labeled by elements of Σ. If $n = |\Sigma|$, then T has n leaves and

$n-1$ internal nodes. So we just transmit a representation $r(T)$ of T followed by the sequence of labels of the leaves, listed in a left-to-right manner. The number of bits transmitted is then $|r(T)| + nt$.

(a) Give a representation for $r(T)$ which uses as few bits as possible. Be sure that we can reconstruct T from $r(T)$. HINT: Use $r(T)$ to describe a systematic traversal of the edges of T, where a 0 bit represents going down an edge, and 1 bit represents going up an edge. This leads to a $4n-4$ bit solution. Try to improve this.

(b) In HW4, Question 1, the Huffman code of the ASCII string "$hello, world$!" is 42 bits. How many bits would you need to transmit the corresponding Huffman code C in Figure 1 of HW4? How does the total transmission cost for static Huffman code compare to the cost for dynamic Huffman code?

(c) Using your scheme of (a), write out the string $r(C)$ for the Huffman code in part (b). You can just write 'h' instead of the ASCII code for h, etc.

(d) If you managed to find the $2n-1$ scheme, then please reconstruct the following code C: $r(C) = 0010111abcd$ where a, b, c, d stands for the appropriate ASCII code.

HINT:

(a) There is a scheme using $2n-1$ bits. A straightforward scheme uses $4n-4$ bits; you can then improve this to $3n-2$, and finally to $2n-1$.

MORE HINT:

(a) Note that going up, you don't need to say how far to go up since you know T is a full binary tree. This saves $n-2$ bits. You can further save another $n-1$ bits by not having to specify the 0 bit that follows a 1 bit.

(b) For the Huffman code in Question 1, HW4, $n = 10$ and $r(T) = 0010110001101100111$ has 19 bits.

(d) This code is found in Lecture V (Figure 1, page 8).

Dynamic Programming

We said that $LCS(X, Y)$ can be exponential in the $m = |X|$ and $n = |Y|$. Construct two families of strings $\{X_n : n \in \mathbb{N}\}$ and $\{Y_n : n \in \mathbb{N}\}$ such that for each n, $|LCS(X_n, Y_n)| \geq 2^n$. Also, we require $|X_n| = O(n)$ and $|Y_n| = O(n)$.

HINT: suppose $X_n = 01a01a01\ldots = (01a)^n$ and $Y_n = 10b10b10\ldots = (10b)^n$. To construct a common subseqeunce of X_n, Y_n, for each 01-block in X_n, it can choose to match either 0 or 1 with the corresponding letter in 10-block in Y_n. This gives 2^n common subsequences of length n. Is this argument correct?

MORE HINT: The problem with the first hint is that the LCS of X_n, Y_n is actually $2n-1$ and not n, as suggested by the HINT! Can you see how to get $2n-1$? To fix this problem, we keep X_n but modify Y_n to $Y_n = 10b10b10\ldots = (10b)^n$. Then $L(X_n, Y_n) = 2n$ (why?), and the argument in the HINT is correct.

Dynamic Programming

Let us generalize the string edit distance problem as follows: let Σ be the alphabet and $*$ be a symbol not in Σ. Let $\delta : (\Sigma \cup \{*\})^2 \to \mathbb{R}$ be any cost function. We interpret $\delta(x, y)$ to be the cost of replacing x by y in a string. If $x = *$, it means inserting y. If $y = *$, it means deleting x. Let $\delta(X, Y)$ denote the minimum cost of transforming X to Y using the this δ cost function. Assume that $\delta(x, y) = \delta(y, x)$ for all x, y.

(a) Let $\Sigma = \{a, b, c, \ldots, x, y, z\}$ (English alphabet) and suppose $\delta(x, y) = 0$ if $x = y$ and $\delta(x, y) = 1$ if x, y are both vowels or both consonents. Let $\delta(x, y) = 2$ otherwise. What is $\delta(good, bad)$?

(b) Describe the exact relationship between the string edit distance problem and this $\delta(X, Y)$ function.

(c) Give the dynamic programming principle for computing $\delta(X, Y)$.

(d) Can you think of reasons to allow negative cost, $\delta(x, y) < 0$?

HINT:
(a) $\delta(good, bad) = 4$.
(b) String edit distance is a special case of the δ measure.
(c) Relate $\delta(X, Y)$ to $\delta(X', Y'), \delta(X', Y), \delta(X, Y')$ where X', Y' are defined as in LCS.
(d) Can you think of applications where $\delta(good, goof)$ and $\delta(on, of)$ should not be the same?
MORE HINT:
(c) $\delta(X, Y) = \min\{\delta(x, y) + \delta(X', Y'), \delta(x, *) + \delta(X', Y), \delta(*, y) + \delta(X, Y')\}$.
(d) If we want to find the closest pair of words between two databases, then one with more matching letters should be better. So we want $\delta(good, goof) < \delta(on, of)$. To achieve this, we may want $\delta(x, x) < 0$ for all x.

Dynamic Programming

Here is a Google problem: let F be a file, viewed as a seqence of N words (ignoring punctuations, etc). Each occurrence of a word in F can be identified by a position $i \in 1, \ldots, N$. We have a dictionary which contains a set of "keywords", and with each key word w, there is a sorted list $P(w)$ of positions indicating where w occurs in F. E.g., if $P(dynamic) = (7, 16, 42, 101, 125)$, it means the keyword *dynamic* occurs 4 times in F at positions 7, 16, etc.

The problem is this: suppose we are given k keywords, $w_1, \ldots, w_k$. An interval $J = [s, t]$ is called a **cover** if each w_i occurs at least once within the positions in J. Our task is to compute a cover $J = [s, t]$ of minimum size $t - s$. E.g., continuing the previous example, suppose $P(programming) = (9, 43, 300)$. You want to find a cover for the key words *dynamic* and *programming* (so $k = 2$). In this case, $J = [42, 43]$ would be the solution.

(a) If $k = 2$, give a linear time algorithm that finds all the minimum covers. Let the lists $P(w_i)$ have n_i positions; then linear time means $O(n)$ where $n = n_1 + \cdots + n_k$.

(b) Give an $O(n \log n)$ time recursive algorithm for $k = 3$.

HINT:
(a) Just merge $P(w_1)$ with $P(w_2)$ in linear time. Scan the resulting list.
(b) Merge $P(w_1), P(w_2), P(w_3)$. Let $A[1..n]$ be the merged list. Recursively, compute the minimum cover of $A[1..n/2]$ and $A[(n/2)+1..n]$. We now need to find the minimum cover that straddles $\frac{A[n/2]+A[(n/2)+1]}{2}$: there are 6 possibilities.
MORE HINT:
(a) If $A[1..n]$ is the resultant list, then the minimum cover can be found as the pair $A[i], A[i+1]$ where $A[i] - A[i+1]$ has minimum absolute value, and where $A[i] \in \P(w_1)$ iff $A[i+1] \in P(w_2)$.
(b) Let $J = [A[i], A[j]]$ be the minimum cover where $i \leq n/2$ and $j \geq (n/2) + 1$. One case is $[A[i], A[n/2]]$ contains only w_1 while $[A[(n/2) + 1], A[j]]$ contains w_2 and w_3. This can be computed $O(n)$ time. Similarly for the other 6 cases.

Amortization

Let C be a binary counter, initially with value $\mathrm{Val}(C) = 0$. We want to increment and decrement counters. Consider the amortized cost of an arbitary sequence of n operations of $Inc(C)$ and $Dec(C)$. The cost for $Dec(C)$ from a value m is the same as the cost to $Inc(C)$ from a value of $m - 1$.

(a) Assume that we never decrement a value below 0. Show that it is no longer possible to have $O(1)$ amortized cost for each operation.

(b) Show how we can achieve $O(1)$ amortized cost for part (a) if we use pair of counters to represent each number. You need to define a suitable potential function.

(c) In (b), the counter value C can now be negative. Suppose you want to implement a new operation, $sign(C)$. Show how this can be implemented in amortized constant time.

Shortest Paths

The input for Dijkstra's algorithm is a digraph $G = (V, E; s, C)$ where s is the source node and $C(e) > 0$ are costs of each edge $e \in E$.

(a) Show how to implement Dijkstra's algorithm using Fibonacci heaps.

(b) Analyze the complexity of your algorithm in (a).

(c) Suppose the cost $C(e)$ may be negative. So we cannot use Dijkstra's algorithm on G. Consider the following idea: for each edge $e \in E$, redefine $C'(e) = C(e) + M$ where M is a large positive constant. Now $G' = (V, E; s, C')$ has only positive costs, and so we can run Dijkstra's algorithm on G'. If $d'[i]$ is the minimum cost from the source s to vertex i in G'. CLAIM: if the minimum path from s to i has k edges, then $d[i] = d'[i] - kM$ is the minimum cost from source s to i in the original graph $G = (V, E; s, C)$. Prove this or show a counter example.

(d) Modify the Floyd-Warshall algorithm so that it detects if there is any negative cycle in a graph. A negative cycle is cycle whose cost is negative. Your algorithm should terminate as soon as it detects a negative cycle.

(e) Do the hand simulation of Dijkstra's algorithm using the organization described in §3 of the Lecture Notes on Minimum Cost Path (Lecture 14). For input, use the graph in Figure 2, but change the edge costs as follows: if the cost is 10, keep it unchanged; if the cost is < 10, increase it by 10; if the cost is > 10, decrement it by 10.

HINT: (a),(b) standard material. (c) The shortest path may have lots of edges, but the modified cost functions favors those paths with few edges. So the shortest paths may now be different. (d) Let $C^{(k)}(i, j)$ be the shortest path from i to j in the kth stage. What is the cost of the minimum cycle through i at this stage?

MORE HINT: (a),(b) read the texts. (d) Negative cycle detected as soon $C^{(k)}(i, i) < 0$ for any k, i.

NP-Completeness and Reducibility

In the Traveling Salesman Problem (TSP), you are given a bigraph $G = (V, E)$ with edge costs, $C : E \to \mathbb{N}$ (i.e., edge costs are natural numbers). If $(u, v) \notin E$ then $C(u, v) = \infty$. Write $G = (V, E; C)$ for this weighted graph. A **tour** is a path that visits each of the $n = |V|$ vertices in V exactly once, and returns to the starting vertex. Consider the following 3 versions of the Traveling Salesman Problem:

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

(a) Show that $TSD \leq_P TSC$ and $TSC \leq_P TSP$, where $A \leq_P B$ means that A can be solved in polynomial-time if there is a polynomial time algorithm for B.

(b) Show that $TSP \leq_P TSC$ and $TSC \leq_P TSD$.

(c) The classes P and NP represent decision problems. How is the study of P and NP justified when we are interested in optimization problems, not decision problems?

(d) The problem of Hamiltonian Circuit (HAM) is this: given a bigraph $G = (V, E)$, we want to know if there exists a tour (also called "Hamiltonian circuit" in this context). Show that $HAM \leq_P TSP$.

(e) Show that if $A \leq_P B$ and $B \leq_P C$ then $A \leq_P C$.

(f) Suppose $A \leq_P B$ and A is NP-complete. What can you say about B?

HINT:

(a) Straightforward.

(b) $TSC \leq_P TSD$ follows by binary search. For $TSP \leq_P TSC$, 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) Look at parts (a) and (b).
(e), (f) are standard results.

MORE HINTS:

(b) If you know that (v_0, v_1) is the start of an optimum tour, you can now check if (v_0, v_1, v_2) is also the case.
(c) There are closely connected decision and optimization problems with the property one is solvable in polynomial time iff the other is. E.g., TSP and TSD.