Ph.D. Qualifying Exam
Design and Analysis of Algorithms and Data Structures

Computer Science Department
Indiana University

Fall 1997

This exam has two parts.

Part I. The *string edit* problem is to find the minimum cost for modifying two strings so that they are the same. The permitted operations are deletions, insertions, and substitutions. Let the two strings be $a_1 a_2 \dots a_n$ and $b_1 b_2 \dots b_m$, where each a_i and each b_j is a character from the character set Σ. If s_i and s_j are two characters of Σ, then the cost of deleting s_i is $D_{s_i} > 0$, the cost of inserting s_i is $I_{s_i} > 0$, and the cost of changing s_i to s_j is $C_{s_i s_j} \geq 0$. Assume that $C_{s_i s_j} = C_{s_j s_i}$ and that $C_{s_i s_j} = 0$ if and only if $s_i = s_j$. The following algorithm computes the minimum cost for editing two strings of lengths i and j, respectively:

$se(i,j)$ [where $i, j \geq 0$] :=
  if $i = 0$ **and** $j = 0$ **then** 0
  else if $j = 0$ **then** $se(i-1, 0) + D_{a_i}$
  else if $i = 0$ **then** $se(0, j-1) + I_{b_j}$
  else $\min(se(i-1, j-1) + C_{a_i b_j}, se(i-1, j) + D_{a_i}, se(i, j-1) + I_{b_j})$

1. Argue briefly that the given algorithm is correct.
2. Give the recurrence equation that computes the running time of the given algorithm if it executes exactly as stated. Give a worst-case upper bound on the running time.
3. Present an algorithm that computes $se(i,j)$ much more efficiently than the given algorithm.
4. Analyze the complexity of your algorithm.

Part II. The numbered paragraphs give what you need to do. The other paragraphs explain the problem.

This question concerns a directed acyclic graph (DAG) . Each node in the graph has a name; zero, one, or two parents; and zero or more children. This data structure is suitable for showing the family relationships among a collection of humans.

A node can belong to one or more of the following categories. Each category is a function of the pointer P that points to some node in the DAG. To save writing, we call the node that P points to "node P". The categories are:

a. *self*: the node P.

b. *ancestor*: the node P and a parent of any *ancestor*.

c. *relative*: any *ancestor* of P and a child of any *relative*.

d. *connector*: Consider each pair of nodes A and B such that A and B are both *relatives* of node P. They can be related in other ways as well. Consider any path x_0, x_1, $\ldots$, x_n such that A is x_0, B is x_n. In addition require that there exists an index i such that, for $0 \leq j < i$, x_i is a child of x_{i+1}, and, for $i < j \leq n$, x_{i-1} is a parent of x_i. For any such A, B, path, and j in the range $0 \leq j \leq n$, node x_j is a *connector*. In other words, the *connectors* are those nodes needed to show all the various ways in which the *relatives* of node P are related to each other. Note that *connectors* depend on only P and the DAG.

e. *other*: any node.

When deciding which algorithms to chose for this problem, select algorithms that are both simple and fast. Avoid algorithms that are complex and difficult to understand or program. It is desirable that the algorithm run in time that is bounded by a constant times the number of nodes, but it may not be possible in some cases. It is desirable for the algorithm to run faster than its worst-case time in favorable cases. This question calls for *simple* fast algorithms.

1. Describe the data structure that you actually use for this problem. Your data structure needs to be suitable for efficient algorithms for computations described below. Your description should be complete and concise.

2. For each category give an algorithm to decide which nodes belong to the category.

In the end, we prefer to have a *single* category for each node. We wish for each node to be assigned to the first category for which it qualifies. Thus, if a node belongs to the *self* category, we do not wish to assign it to the other four categories.

3. Describe how to combine (or modify) your algorithms so that each node is assigned to the appropriate *single* category.

   Once you have described your algorithms, if you notice some complex algorithm that runs even faster. If so, comment briefly. (This is a minor part of the question, so don't spend much time to spend on it.)

4. Give the worst-case running time for each algorithm. Briefly justify your answer.