Algorithms – CS-37000 Homework 2 – January 5, 2006

Instructor: László Babai Ry-164 e-mail: `laci@cs.uchicago.edu`

ADVICE. Take advantage of the TA's office hours.

READING KT, Chapters 4.1, 4.2. (KT = Kleinberg - Tardos text), Binary Search handout. Study PSEUDOCODE conventions in the handout and in Homework 1.

REVIEW from Discrete Math: Graphs and Digraphs, Asymptotic notation.

HOMEWORK. Please **print your name on each sheet.** Exceptionally, you do not need to put every solution on a separate sheet; each solution should occupy a few lines only. Please try to make your solutions easily readable.

This homework is due on **Tuesday, January 10** at the **beginning of the class.**

In the problems below, the digraph G is either given by an *array of adjacency lists* ("adjacency list representation" for short) or by the list of vertices and the list of edges ("edge list representation"). If there are no multiple items on these lists (a tacit assumption we always make unless the opposite is expressly stated), then the length of each representation is $\Theta(n+m)$ where n is the number of vertices and m is the number of edges. An algorithm runs in *linear time* if its cost is big-Oh of the length of the input, so it is $O(n+m)$ under our tacit assumption. Unit cost is associated with the following operations: looking up and copying an integer between 1 and n, looking up an entry at a given location in an array, moving to the next item in a linked list, and similar basic bookkeeping operations.

Unless expressly stated otherwise, **all algorithms must be described in pseudocode**. IMPORTANT: **Define and explain your variables** and put comments on the lines of your pseudocode. (Otherwise your code will be unintelligible. You lose credit if understanding your solution requires unreasonable amount of work.)

All problems below have VERY SIMPLE solutions (just a few lines of pseudocode). ELEGANCE COUNTS! Hint: The typical cycle structure of a linear time graph algorithm based on an adjacency list representation is this:

```
1 for $i = 1$ to $n$ do    (visit every vertex)
2     for $j \in$ adj[$i$] do    (visit every edge out of vertex $i$)
3         (instructions)
4     end(for)
5 end(for)
6 return    (value)
```

Things might get slightly more complicated, but often enough, this is the basic structure. If line 3 takes constant time then the overall time is linear.

For each problem below, state your answer in the form of an elegant pseudocode. Add definitions of your variables and brief comments explaining what is happening. No explanation of the correctness and the cost of your algorithms is required (as long as they are indeed correct and run in linear time).

2.1 (6 points) Given an edge-list representation of a digraph with possibly repeated edges, turn it into a multiplicity-free edge-list representation (no repeated edges). Do this in linear time.

2.2 (2 points) Given an edge-list representation of a digraph, turn it into an adjacency list representation in linear time.

2.3 (2 points) Given an adjacency list representation of a digraph, turn it into an edge-list representation in linear time.

2.4 (6 points) The *reverse* of the digraph G has the same vertex set but all edges are reversed. Given an adjacency list representation of a digraph, construct an adjacency list representation of its reverse in linear time.

2.5 (6 points) An adjacency list is *monotone* if the neighbors are listed in increasing order. Given an adjacency list representation of a digraph, turn it into a monotone adjacency list representation in linear time.

2.6 (6 points) A digraph is *undirected* if it is its own reverse. Given an adjacency list representation of a digraph, decide in linear time whether or not it is undirected.

2.7 (6 points) An edge in a digraph is *undirected* if its reverse is also an edge. Given a digraph by an edge-list, count its undirected edges in linear time.

2.8 (6 points) Given a digraph by an edge-list, sort the vertices by their out-degrees in linear time, smallest degree first. (You should return an array which contains a permutation of the numbers 1 to n. Ties are resolved arbitrarily.)