Algorithms – CMSC-27200
http://alg15.cs.uchicago.edu
Homework set #2   due January 21, 2015

Read the homework instructions on the website. The instructions that follow here are only an incomplete summary.

Hand in your solutions to problems marked "HW." Do not hand in problems marked "DO." If you hand in homework marked "XC" (extra credit), do so on *separate and separately stapled sheets.* PRINT YOUR NAME ON EVERY SHEET you submit. We ask you to use LaTeX to typeset your solutions; starting January 28, this will be required. Hand in your solutions on paper, do not email.

When writing pseudocode, **explain the meaning of your variables.** Use comments to explain what is happening in each line. Also, give a short explanation of the idea behind the algorithm. **Describe all algorithms in this problem set in pseudocode. Elegance of your code matters.**

Carefully study the policy (stated on the website) on collaboration, internet use, and academic integrity.

In this problem set, every digraph is given in an *adjacency-list* representation, unless expressly stated otherwise. Recall that an adjacency-list representation consists of an array of vertices, where vertex u is the start of the linked list Adj$[u]$ that lists all the out-neighbors of u in some order. An algorithm on a digraph $G = (V, E)$ is said to run in ***linear time*** if the number of steps is $O(|V| + |E|)$.
By "graph" we mean an undirected graph without self-loops and without parallel edges. A graph is a digraph satsifying the condition that for all $u, v \in V$, if $(u, v) \in E$ then $(v, u) \in E$ and $(u, u) \notin E$.

2.1 DO **(Conversion of representations)** An *edge-list representation* a digraph is a list of the edges in some order.

(A) Design an algorithm that converts an edge-list representation into an adjacency-list representation in linear time. Describe your algorithm in elegant pseudocode.

(B) Design an algorithm that converts an adjacency-list representation into an edge-list representation in linear time. Again, describe your algorithm in elegant pseudocode.

2.2 **HW (3 points) (Transpose)** The *transpose* of a digraph $G = (V, E)$, denoted G^T, has vertex set V, and edge set $E^T = \{(u, v) \mid (v, u) \in E\}$.

(i) Given an adjacency-list representation of the digraph G, produce an adjacency list representation of G^T in linear time. Describe your algorithm in elegant pseudocode.

(ii) Reason why your algorithm runs in linear time.

2.3 DO **(Monotone adjacency lists)** Consider a digraph $G = (V, E)$ where $V = \{1, \dots, n\}$. We say that an adjacency list representation of G is *monotone* if for every vertex i the vertices adjacent to i are listed in increasing order. Given an adjacency list representation of G, produce a monotone adjacency list representation of G in linear time. Describe your algorithm in elegant pseudocode.

2.4 **HW (3 points) (Recognizing undirected graphs)** Given a digraph, decide in linear time whether or not it is an undirected graph. Note: If you wish to refer to "DO" exercises above as subroutines, you need to describe those subroutines as well.

2.5 DO **(Removing multiplicities of neighbors)** Consider an adjacency-list representation of a digraph, where in the adjacency list of vertex v a vertex w can appear multiple times. In linear time produce an adjacency list representation of the same digraph without repetitions. Describe your algorithm in elegant pseudocode. Here "linear time" means at most a constant times the length of time it takes to make a single pass through the input, i. e., $O(|V| + |E'|)$ wghere E' is the "multiset" of edges, each edge (u, v) being repeated as many times as v appears in $\text{Adj}[u]$.

2.6 DO **(In-degrees)** Given a digraph, compute the in-degree of every vertex in linear time.

2.7 **XC (6+8 points) (Graph properties)** Consider the *adjacency matrix* representation of an n-vertex graph G. We will count the number of times our algorithm accesses an element of the adjacency matrix.

(a) Prove that a deterministic algorithm that tests whether G is connected must make $\Omega(n^2)$ accesses. (Hint: produce an "adversary" strategy that, as long as it can, "feeds" the algorithm information that is not sufficient to decide connectedness.)

(b) A *graph property* is a predicate on graphs that does not depend on the numbering of the vertices, i.,e., if it is true for a graph G then it is also true for all graphs that are isomorphic to G. Examples: connectedness, planarity, being regular of degree 3, 3-colorability, being isomorphic to a given graph, etc. Find a graph property that can be decided in $O(n)$ accesses to the adjacency matrix. The property should not be trivial (must be true for some graphs on n vertices and false for others).

2.9 **HW (8 points) (Topological sort)**

Given an n-vertex digraph G, either

(a) find a directed cycle of G; or

(b) compute a relabeling of the vertices as $v_1, v_2, \ldots, v_n$ such that $(v_i, v_j) \in E(G)$ then $i < j$).

Your algorithm should run in linear time. Do NOT use depth-first search (DFS).

Reason the correctness and running time of your algorithm.

2.10 DO **(BFS tree)** (i) Show by example that the BFS tree with source s of an undirected graph G *can* depend on the ordering of vertices within the adjacency lists. Make your example small.

(ii) Prove that the distance to the source s, computed by BFS, does *not* depend on the ordering of vertices within adjacency lists.

2.11 DO **(BFS edge classification)** Consider a classification of edges in a BFS tree, using the criteria used to classify edges in depth-first search. Let $u.dist$ denote the directed distance from the source s.

(i) Prove that in the BFS of an undirected graph

1. There are no back edges and no forward edges.
2. For each tree edge (u, v) $v.dist = u.dist + 1$
3. For each cross edge (u, v), $v.dist = u.dist$) or $v.dist = u.dist + 1$)

(ii) Prove that in the BFS of a digraph

1. There are no forward edges
2. For each tree edge we have $v.dist = u.dist + 1$
3. For each cross edge (u, v) we have $v.dist \leq u.dist + 1$
4. For each back edge (u, v) we have $0 \leq v.dist \leq u.dist$.

2.12 **HW (4 points) (Strongly connected)** Recall that a digraph is *strongly connected* if every vertex is accessible from every vertex. Decide in linear time whether or not a digraph is strongly connected.

Do NOT use DFS. You may use BSF. Your algorithm should be very simple.

2.13 DO **(DFS counterexample)** Draw a counterexample to the conjecture that if a directed graph G contains a path from u to v, and if $u.d < v.d$ in a DFS of G then v is a descendant of u in the DFS tree produced.

2.14 **HW (4 points) (DFS counterexample)** Draw a counterexample to the conjecture that if a digraph G contains a directed path from vertex u to vertex v then any depth-first search must result in $v.d \leq u.f$. Make sure to label the vertices (assign the numbers $1, \ldots, |V|$ to them). Indicate the discover and finish time of each vertex. Make your example as small (have as few edges) as possible. Do not prove that it is smallest.

2.15 **(DFS without long paths)**

(a) DO: Let $G = (V, E)$ be a connected undirected graph and $u \in V$. Give a counterexample to the conjecture that the DFS tree constructed by DFS-VISIT(G, u) will include a longest path starting from u. Make sure to label the vertices.

(b) **XC (7 points)** For infinitely many values of n construct an undirected graph G with n vertices (appropriately numbered) such that G is Hamiltonian (has a Hamilton cycle) but the length of the longest path in a DFS tree is only $O(\log n)$.

2.16 **HW (4 points) (Alternating paths)** Consider a digraph $G = (V, E)$ whose edges are marked red or blue. An *alternating path* is a path whose edges alternate in color (red-blue-red-. . . , or blue-red-blue-. . .). The *alteranting distance* from vertex u to vertex v is the length of the shortest alternating path from u to v. Determine in linear time the alternating distance from a source vertex s to all vertices.

2.17 DO **(Counting alternating paths)** Consider a DAG (directed acyclic graph) $G = (V, E)$ whose edges are marked red or blue. For $u \in V$, let $A[u]$ be the number of alternating paths starting at u. In linear time, compute the array A.

2.18 DO **(Longest path in DAG)** In a DAG (directed acyclic graph), find a longest path in linear time.

2.19 **HW (6 points) (All-ones squares)** Given an $n \times n$ array A of zeros and ones, find the maximum size of a contiguous square of all ones. (You do not need to locate such a largest all-ones square, just determine its size.) Solve this problem in *linear time.* "Linear time" means the number of steps must be O(size of the input). In the present problem, the size of the input is $O(n^2)$. Manipulating integers between 0 and n counts as one step; such manipulation includes copying, incrementing, addition and subtraction, looking up an entry in an $n \times n$ array. Name the method used.

The pseudocode should be *very simple,* no more than a few lines. **Elegance counts.**

Example:

$$\begin{array}{|cccccc|}
\hline
1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 \\
\hline
\end{array}$$

In this example, the answer is 3. There are three contiguous 3×3 square subarrays of all-ones. One is indicated below by $\underline{\text{underlines}}$, another is shown in a $\boxed{\text{box,}}$ the third one is indicated by *Italics.*

$$\begin{array}{|cccccc|}
\hline
1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & \mathit{1} & \mathit{1} & \mathit{1} \\
\cline{3-5}
1 & 0 & \multicolumn{1}{|c}{1} & \mathit{1} & \multicolumn{1}{c|}{\mathit{1}} & \mathit{1} \\
\underline{1} & \underline{1} & \multicolumn{1}{|c}{\underline{1}} & \mathit{1} & \multicolumn{1}{c|}{\mathit{1}} & \mathit{1} \\
\underline{1} & \underline{1} & \multicolumn{1}{|c}{\underline{1}} & 1 & \multicolumn{1}{c|}{1} & 0 \\
\cline{3-5}
\underline{1} & \underline{1} & \underline{1} & 0 & 1 & 1 \\
\hline
\end{array}$$