

## Homework 2 – Due Friday, September 12, 2008

Please refer to the general information handout for the full homework policy and options.

### Reminders

- Your solutions are due before the lecture. Late homework will not be accepted.
- Collaboration is permitted, but you must write the solutions *by yourself without assistance*, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. *Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.*
- To facilitate grading, please write down your solution to each problem on a separate sheet of paper. Make sure to include all identifying information and your collaborators on each sheet. Your solutions to different problems will be graded separately, possibly by different people, and returned to you independently of each other.
- For problems that require you to provide an algorithm, you must give a precise description of the algorithm, together with a proof of correctness and an analysis of its running time. You may use algorithms from class as subroutines. You may also use any facts that we proved in class or from the book.

**Reading** Review chapters 2.1-2.4, read chapter 3 in Kleinberg Tardos.

**Exercises** These should not be handed in, but the material they cover may appear on exams:

1. (**Graph representations**) Questions in this problem refer to the adjacency list and adjacency matrix representations defined on pages 87–89 of KT.

You are given a directed graph  $G = (V, E)$  with  $|V| = n$  vertices and  $|E| = m$  edges. Let  $G^R$  be the graph obtained by reversing the directions of all the edges in  $G$ . For each of the following, give an efficient algorithm and analyze its running time.

If  $G$  is given in the adjacency list representation,

- (a) compute the out-degree of each vertex;
- (b) compute the in-degree of each vertex;
- (c) compute the adjacency list representation of  $G^R$ .

If  $G$  is given in the adjacency matrix representation,

- (d) compute the adjacency matrix of  $G^R$ .

2. Give two algorithms to detect whether a given undirected graph has a cycle. If the graph contains a cycle, your algorithms should output one (not all of them, just one). Base the first algorithm on BFS and the second, on DFS. The running time of your algorithms should be the same as the running times of BFS and DFS. Explain why your algorithms are correct and run in the required time.

3. A directed graph  $G = (V, E)$  is **singly connected** if for all vertices  $u, v$  in  $V$  there is at most one simple path from  $u$  to  $v$ . (Recall that a path is simple if all vertices on the path are distinct.) Give an efficient algorithm to determine whether or not a directed graph is singly connected. *Hint:* Run DFS from each vertex.

### Problems to be handed in

**Page limits:** The answer to each problem should fit in **2 pages (or one double-sided sheet)** of paper. Longer answers will be penalized.

1. **(More Asymptotics)**

- (a) Prove that for any positive numbers  $a, b > 0$ , we have  $n^b = \omega(\log^a(n))$ .  
 There are several ways to approach this. One way is to use L'Hospital's rule for evaluating limits together with the following fact:  $f(n) = \omega(g(n))$  if and only if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = +\infty$ .
- (b) Prove or disprove: There exist two *strictly increasing* functions  $f(n), g(n)$ , such that *neither*  $f(n) = O(g(n))$  nor  $g(n) = O(f(n))$  holds.
- (c) Prove or disprove: If  $h(n) = \lceil n \log(n) \rceil$ , then  $n = \Theta(h(n)/\log h(n))$ .

2. **(Basic proof techniques)**

- (a) **(Induction)** Chapter 3, problem 5.  
 (b) **(Contradiction)** Chapter 3, problem 7.

3. **(Shortest cycles containing a given edge)** Give an algorithm that takes as input a (undirected) graph  $G = (V, E)$  and an edge  $e_0 \in E$ , and outputs a shortest cycle that contains  $e$  (if no cycle containing  $e$  exists, the algorithm should output "no cycle").

(Note: See the reminder above on problems that require an algorithm. Remember to prove correctness of your algorithm. You may use algorithms from class as subroutines.)

4. **(Articulation points)** A vertex  $v \in V$  is an *articulation point* in an undirected graph if removing  $v$  from the graph increases the number of connected components (that is, removing  $v$  breaks up one of the graph's connected components).

Give as efficient an algorithm as you can for finding all the articulation points of a graph  $G = (V, E)$  given in adjacency list format. State the running time of your algorithm in terms of  $m = |E|$  and  $n = |V|$ .

5. \* **(Extra credit)** Given a graph  $G$ , we say that paths  $(u, a_1), (a_1, a_2), \dots, (a_k, v)$  and  $(u, b_1), (b_1, b_2), \dots, (b_t, v)$  are vertex disjoint if they are both valid paths in  $G$  and they do not pass through any of the same vertices, that is, the sets  $\{a_1, a_2, \dots, a_k\}$  and  $\{b_1, b_2, \dots, b_t\}$  are disjoint.

Prove that a graph  $G$  has no articulation points *if and only if* for every pair of vertices  $u$  and  $v$ , there exist two vertex-disjoint paths from  $u$  to  $v$ .