COMS 4231: Analysis of Algorithms I, Fall 2016

Problem Set 5, due Tuesday November 22, 11:59pm in pdf format on Courseworks.

Please follow the homework submission guidelines posted on courseworks.

As usual, for each of the algorithms that you give, include an explanation of how the algorithm works and show its correctness. Make your algorithms as efficient as you can, state their running time, and justify why they have the claimed running time. All time bounds below refer to worst-case time.

Problem 1. We are given a directed graph G=(N,E) with node set N={1,…,*n*}, and we are also given a partition of the set N of nodes into *k* parts, numbered 1,…,*k*. The graph is given by an adjacency list representation and the partition is given by an array *C* indexed by the nodes, which specifies for each node the part to which it belongs, i.e., *C*[*u*]=*i* iff node *u* belongs to part *i*. The *quotient graph* Q of G with respect to the partition is a simple graph (i.e. no multiple edges) with set of nodes *K*={1,…,*k*} and with set of edges $\{(i,j) \mid \exists (u,v) \in E$ such that $C[u]=i, C[v]=j\}$. For example, if the partition given by the array *C* is the partition of the nodes into strongly connected components (scc's), then the corresponding quotient graph Q is the DAG of scc's.
Give an efficient algorithm that constructs an adjacency list representation of the quotient graph. The algorithm should run in time O(|N|+|E|).
(Hint: You may find Radix Sort useful here. The algorithm does not involve any graph searching.)

Problem 2. An undirected graph *G*=(*N*,*E*) is called *bipartite* if its set *N* of nodes can be partitioned into two subsets N_1, N_2 ($N_1 \cap N_2 = \varnothing$, $N_1 \cup N_2 = N$) so that every edge connects a node of N_1 with a node of N_2.

a. Prove that if a graph contains a cycle of odd length then it is not bipartite.
b. Give a O(*n*+*e*)-time algorithm that determines whether a given graph is bipartite, where *n* is the number of nodes and *e* is the number of edges; the graph is given by its adjacency list representation.
If the graph is bipartite, then the algorithm should compute a bipartition of the nodes according to the above definition.
If the graph is not bipartite then the algorithm should output a cycle of odd length.
c. We are given a set of *non-equality* constraints of the form $x_i \neq x_j$ over a set of Boolean variables $x_1, x_2, ..., x_n$. We wish to determine if there is an assignment of Boolean values 0,1 to the variables that satisfies all the constraints, and compute such a satisfying assignment if there is one. Show that this problem can be solved in time O(*n*+*m*), where *n* is the number of variables and *m* is the number of constraints.

Problem 3. Do Problem 22-2, parts a-d in CLRS (page 622) on articulation points. (If you have time, do for yourself the remaining parts also (e-h), but you do not have to turn them in.) We reproduce parts a-d below for convenience.

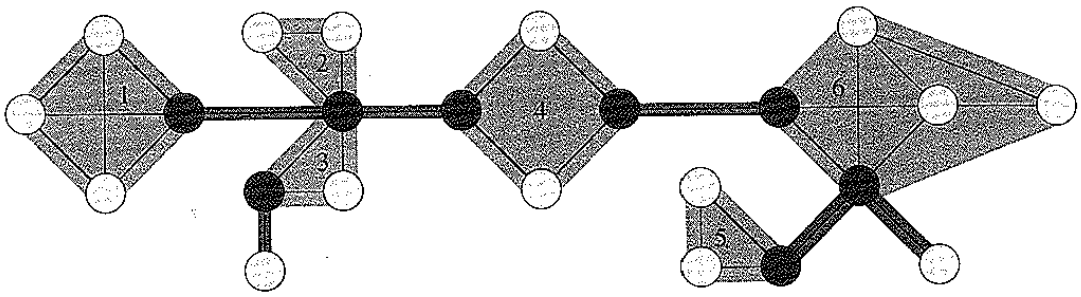

Figure 22.10 The articulation points, bridges, and biconnected components of a connected, undirected graph for use in Problem 22-2. The articulation points are the heavily shaded vertices, the bridges are the heavily shaded edges, and the biconnected components are the edges in the shaded regions, with a *bcc* numbering shown.

22-2 Articulation points, bridges, and biconnected components

Let $G = (V, E)$ be a connected, undirected graph. An ***articulation point*** of G is a vertex whose removal disconnects G. A ***bridge*** of G is an edge whose removal disconnects G. A ***biconnected component*** of G is a maximal set of edges such that any two edges in the set lie on a common simple cycle. Figure 22.10 illustrates these definitions. We can determine articulation points, bridges, and biconnected components using depth-first search. Let $G_\pi = (V, E_\pi)$ be a depth-first tree of G.

a. Prove that the root of G_π is an articulation point of G if and only if it has at least two children in G_π.

b. Let v be a nonroot vertex of G_π. Prove that v is an articulation point of G if and only if v has a child s such that there is no back edge from s or any descendant of s to a proper ancestor of v.

c. Let

$$v.low = \min \begin{cases} v.d\,, \\ w.d : (u, w) \text{ is a back edge for some descendant } u \text{ of } v\,. \end{cases}$$

Show how to compute $v.low$ for all vertices $v \in V$ in $O(E)$ time.

d. Show how to compute all articulation points in $O(E)$ time.

Problem 4. We are given a set *V* of *n* variables $\{x_1, x_2, \ldots, x_n\}$ and a set *C* of *m* weak and strict inequalities between the variables, i.e., inequalities of the form $x_i \le x_j$ or $x_i < x_j$. The set *C* of inequalities is called *consistent* over the positive integers Z^+ iff there is an assignment of positive integer values to the variables that satisfies all the inequalities.
For example, the set $\{ x_1 \le x_3,\ x_2 < x_1 \}$ is consistent, whereas $\{ x_1 \le x_3,\ x_2 < x_1,\ x_3 < x_2 \}$ is not consistent.
a. Give an efficient algorithm to determine whether the set *C* of inequalities is consistent over the positive integers. State precisely the asymptotic running time of your algorithm in terms of *n* and *m*.
b. If the set of inequalities has a solution, then it has a unique minimum solution, i.e. a solution in which every variable has the minimum value among all possible solutions. Give an efficient algorithm to compute the minimum solution.
(Note: Both parts can be solved in O(*n*+*m*) time.
Hint: Construct a suitable graph and use appropriate algorithms.)

Problem 5. In this problem you will use Kruskal's algorithm to show some properties of minimum spanning trees.
a. Let G=(N,E) be an undirected graph with weight function *w* on the edges. In general, there may be multiple minimum weight spanning trees. In this part you will show that *every* MST can be produced by Kruskal's algorithm for some sorted listing of the edges. Let *T* be any minimum weight spanning tree. Let L=<e_1, e_2 , …, e_m > be a sorted listing of the edges of G in nondecreasing order of weight with ties broken in favor of the edges of *T* in case of edges with the same weight, i.e. if $w(u,v)= w(x,y)$ and $(u,v)\in T$, $(x,y)\notin T$, then edge (u,v) is ordered before edge (x,y) in L (edges of *T* with the same weight are ordered arbitrarily, and the same for edges that are not in *T*). Suppose that we run Kruskal's algorithm with the edges processed in the order of L, and let T_K be the tree computed by Kruskal's algorithm. Show by induction on *i* that $e_i \in T_K$ if and only if $e_i \in T$, for all i=1,…,*m*. Conclude that $T_K = T$.

b. Show that if in a weighted graph all edge weights are different, then there is a unique minimum spanning tree.

c. Show that if *T* is a minimum weight spanning tree of a graph G=(N,E) with weight function *w* and we change the weight function to *w'* where $w'(e_i) = (w(e_i))^3$ for every edge $e_i \in$E, then *T* is also a minimum weight spanning tree of G with the new weight function *w'*.

Exercises for you practice (do not turn in):

Practice the graph algorithms that we learn on some graphs of your choice.
For example, draw an arbitrary directed or undirected graph, pick a source node and apply Breadth-First Search. Compute the BFS tree, the distances, and the partition of

nodes into layers. Does the tree depend on the order in which nodes appear in the adjacency lists? (Recall that the adjacency list of a node contains the adjacent nodes in arbitrary order.) How about the distances and the layers?

Similarly with Depth-First Search. Compute the DFS tree, the discovery and finish times of the nodes, classify the edges (tree, forward, back, cross). Compute the strongly connected components for a directed graph. For example, do exercises 22.3-2 and 22.5-2.

Draw a weighted undirected graph, and apply Prim's and Kruskal's algorithm to it.

Similarly, practice all the other graph algorithms that we do in class.