Algorithms Final Examination
Jan. 11, 2017

1. (10%) Write an algorithm that takes an integer n as input and determine the total number of solutions to the n-Queen problem. (You need to show the time complexity of your algorithm.)

2. (10%) Given an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(|V|)$ time, independent of $|E|$.

3. (10%) Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree. Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

4. (10%) Describe Dijkstra's single-source shortest path algorithm. Explain why Dijkstra's algorithm does not allow negative edges.

5. (10%) Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

6. (10%) Given a directed graph $G = (V, E)$, we want to find a shortest from a given source vertex $s \in V$ to each vertex $v \in V$. Please find the algorithm and its corresponding time complexity for the single-source shortest-paths problem with each of the following assumptions.

   a) Assume each edge of the graph G has unit weight.

   b) Assume the graph G is a directed acyclic graph.

7. (10%) Give an efficient algorithm to find the length (number of edges) of a minimum-length negative-weight cycle in a graph. (You need to give the time complexity of your algorithm and prove your algorithm is correct.)

8. (10%) Suppose that we run Johnson's algorithm on a directed graph G with weight function w. Show that if G contains a 0-weight cycle c, then $\hat{w}(u, v) = 0$ for every edge (u, v) in c.

9. (10%) Let us define two problems as follows. 1) The CLIQUE problem: Given a graph $G = (V, E)$, and an integer k, does the graph contains a complete graph with at least k vertices? 2) The VERTEX-COVER problem: Given a graph $G = (V, E)$, and an integer k, does the graph contain k vertices such that all edges are covered by them? If we know that the VERTEX-COVER problem is NP-complete problem. Please prove that the CLIQUE problem is an NP-complete problem too.

10. (10%) For each of the following statements, determine whether it is true or false. Then give a brief justification for your answer. Your answer will be evaluated based on your justification and not the True/False marking alone. (Note: 只回答True or False沒有分數，答對一小題給2分，此題最多給10分。)

    (1) If a polynomial-time algorithm exists for any NP-hard problem, then NP = P.

    (2) We have found an approximation algorithm for the general Travelling Salesperson Problem.

    (3) The Hamiltonian-path problem can be solved in polynomial time on directed acyclic graphs.

    (4) The longest-simple-path (LSP) problem is the problem of determining a simple path (no repeated vertices) of maximum length in a graph. The LSP problem can be solved in polynomial time.

    (5) 3-CNF (the satisfiability problem, in which each clause has exactly three literals) is polynomial-time reducible to 2-CNF problem.

    (6) We have a dynamic programming (DP) algorithm to solve the 0/1 Knapsack problem that runs in $O(nW)$ time, where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack. Therefore, the DP algorithm is a polynomial time algorithm.

    (7) If Problem A can polynomial-time reduce to problem B, then problem B can also polynomial-time reduce to problem A.

    (8) If a problem is NP-complete, then it is unlikely that we can find a polynomial time algorithm to solve it in average cases.

Happy New Year!