Name: Email: Section:

CSCI2300 – Introduction to Algorithms
Spring 2018, Exam II (100 Points)

Put your name, email and section on front page. No calculators, laptops or tablets allowed. Do not write long and complicated code. Instead describe the steps in plain English, with high-level pseudo-code if absolutely necessary.

1. (25 points) Given a connected, unweighted, and undirected graph G. Answer the following questions:

   (a) (10 points) A *bridge* is an edge whose removal disconnects the graph. Describe a linear time algorithm to determine whether a given edge (x, y) is a bridge.

   Answer: Remove edge (x, y), then run DFS or BFS from x to see if y is reachable from x or not. If not, then the edge is a bridge.

   (b) (15 points) Give a linear time algorithm to find *all* bridges in G. (hint: try to think of what makes an edge a bridge, and then how to identify such edges.)

   Answer: To answer this question, we need to first find all edges involved in a cycle. Any edge not part of a cycle must be a bridge.

   For identifying all cycles, we need to do the following: 1) use DFS to obtain pre-post numbers of each vertex in $O(|V| + |E|)$ time. 2) Next make another DFS traversal in the same DFS order, and add all forward edges onto a stack. We pop an edge from the stack if we backtrack from a node (when all its neighbors have been visited.) Any back edge will create a cycle, which can be used to eliminate all edges part of that cycle. That is, If we encounter a vertex x with an edge (x, y) where the interval of y is larger than the interval of x, then (x, y) is a back edge, and we can then mark all forward edges beginning from the edge that has y as the source vertex up to the current edge as being part of a cycle. If we ever pop an unmarked edge, then this edge is not part of any cycle, and is therefore a bridge edge.

   This method will identify all bridge edges in $O(|V| + |E|)$ time.

2. (25 points) Given a weighted, directed graph G, with $w(u, v)$ denoting a positive weight on edge (u, v). Answer the following questions:

   (a) (10 points) Give a linear time algorithm to find all nodes x, such that x can reach every other node.

   Answer: First determine that the directed graph is connected by treating it as an undirected graph and checking there is only one connected component. If the graph is disconnected then such a x does not exist.

   Now, assuming it is connected, we know that a directed graph is a DAG over its strongly connected components (SCCs). So, we next determine all SCCs via the approach in section 3.4.2, namely 1) compute the reverse graph. 2) run DFS to get post-number in the reverse graph. 3) run DFS on original graph in decreasing post number order and determine SCCs.

   Next, identify the source SCCs via topological sort over the SCCs. If there is only one source SCC then all vertices in that SCC can reach all other vertices. If there is more than one SCC, then there does not exist a vertex that can reach all other vertices.

   (b) (15 points) Give a $O(|V|^2)$ time algorithm to find the shortest path between all pairs of nodes subject to the constraint that it must pass through a fixed node x. You may assume that G is dense.

   Answer: Compute all shortest paths to x via Dijkstra's on G^R. Next find all shortest paths from x to all other nodes via Dijkstra's on G. Now for each pair of vertices (u, v), the shortest path length from u to v that passes through x is simply the sum of the path length from u to x, and from x to v. Since Dijkstra's method takes $O(|V|^2)$ in the worst case, and adding the length for each pair of vertices also takes $O(|V|^2)$ time, the total time is $O(|V|^2)$.

3. (20 points) Given a connected, weighted, undirected graph, where each edge weight lies in the range 1 to C, for some constant C. Answer the following questions.

   (a) (10 points) How fast can you make Prim's MST algorithm run? Describe which steps can be improved (if any) and the time complexity.

   Answer: Since all weights are in the range [1,C], we do not need a priority queue (PQ) to find the minimum weight vertex. Instead we will maintain an array indexed from 1 to C.

   We begin at some vertex u_0, and add each of its neighbors x to the, based on the weight of the edge between u_0 and x.

   Since C is a constant, we can find the minimum weight vertex in O(1) time instead of using a PQ.

   Likewise, each time we process an edge, we can implement the decrease key operation in $O(1)$ time, by moving the vertex to the appropriate cell.

   Thus, we can find the min weight vertex in O(1) time, and decrease key also takes O(1) time.

   Total time is therefore $O(|V| + |E|)$.

   (b) (10 points) How fast can you make the Kruskal's MST algorithm run? Describe which steps can be improved (if any) and the time complexity.

   Answer: For Kruskal's we can only speed up the sorting time to $O(|E| + C) = O(|E|)$, once again by adding each edge into an array indexed from 1 to C.

   The union-find time does not change. Total time is therefore $O(|E| \log |V|)$ or $O(|E| \log^* |V|)$ if we use path compression for the union-find data structure.

4. (20 points) Let the frequency of character i be given as F_i, where F_i is the i-th term of the Fibonacci sequence. Answer the following questions about Huffman codes. Assume that smaller frequency child is always the left child. If two nodes have the same frequency, then the lexicographically higher node should be the left node.

   (a) (10 points) What is the Huffman code for $n = 8$ based on the Fibonacci frequencies below.

char	1	2	3	4	5	6	7	8
freq	1	1	2	3	5	8	13	21

Answer: The greedy method will merge b and a first, next c, next d, and so on, so the Huffman code is:
f_8, h: 0
f_7, g: 10
f_6, f: 110
f_5, e: 1110
f_4, d: 11110
f_3, c: 111110
f_2, b: 1111110
f_1, a: 1111111

   (b) (10 points) Generalize your answer to find the optimal code for n characters.

Answer: Let f_i denote the i-th Fibonacci number. Basically, the n-th character, with frequency f_n, will have code 0, the $(n-1)$-th character, with frequency f_{n-1}, will have code 10, and so on, until the first two characters, with frequency f_2 and f_1, which will have codes 111...10 and 11..11, with codes length $n-1$, i.e.,
f_n: 0
f_{n-1}: 10
f_{n-2}: 110
f_{n-3}: 1110
$\vdots$
f_2: 111...10
f_1: 111...11
A general formula is that the k-th Fibonacci term has $n-k$ 1's followed by a 0, except for the first term, which has $n-1$ 1's.

5. (10 points) Draw a weighted connected undirected graph with 6 nodes and at least 6 edges in which the shortest path between two fixed vertices x and y is *not* part of *any* minimum spanning tree (MST).

   Answer: There are several possible answers, one is a cycle with 6 nodes, with all edges having weight 1, except the edge (x, y) which has weight 2.

```
x -- y -- a
|         |
d -- c -- b
All edges have weight 1, except x--y which has weight 2
```