College of Computer & Information Science
Northeastern University
Fall 2018

CS5800
Algorithms
12 November 2018

Instructions:

- This quiz is closed book and closed notes. Please use both sides of the page.
- Please write clearly and legibly. Grading will be based on both clarity and correctness.

Sample Solution to Quiz 2

Problem 1. (2 + 2 + 6 = 10 points) Supports of a directed graph

We call a vertex x of a directed graph G a *support* of G if for every other vertex v of G, either x has a path to v (i.e., x can reach v) or v has a path to x (i.e., x is reachable from v), or both. For example, in the graph G with vertex set $\{a, b, c\}$, and edge set $\{(a, b), (b, c)\}$, every vertex is a support.

(a) Draw a directed graph with 4 vertices that has exactly one support.

Answer: $G = (V, E)$ where $V = \{a, b, c, d\}$ and $E = \{(a, d), (b, d), (c, d)\}$.

(b) Draw a directed graph with 4 vertices and 4 edges that has no support.

Answer: $G = (V, E)$ where $V = \{a, b, c, d\}$ and $E = \{(a, c), (a, d), (b, c), (b, d)\}$.

(c) Suppose G is a *directed acyclic graph* given in adjacency list format with n vertices and m edges. Give the most efficient algorithm you can to determine *all the supports* of G. State the worst-case asymptotic running time of your algorithm. Your grade will depend on the correctness and the efficiency of your algorithm. No proof of correctness or analysis of running time is required.

Answer: We present two algorithms.

Algorithm 1: We can determine in linear time if a particular vertex v is a support. Then, we can run this for each vertex, and complete in $\Theta(n(n+m))$ time.

```
1. Compute $G'$, the reverse of graph $G$.
2. Let $L$, the list of supports, be initially empty.
3. for each vertex $v$:
   (a) Run DFS from $v$ in $G$ and determine list $S_v$ of vertices reachable from $v$ in $G$.
   (b) Run DFS from $v$ in $G'$ and determine list $S'_v$ of vertices that can reach $v$ in $G$.
   (c) Do a linear scan of the lists $S_v$ and $S'_v$ and determine if every vertex in $G$ is in one
       of the two lists. If yes, then add $v$ to $L$.
```

The above algorithm can be made a bit more efficient, but do not see a way of improving the asymptotic running time. It also does not use any properties of DAGs.

Algorithm 2: We now analyze more carefully what a support in a DAG looks like. Note that a source can be a support if and only if there is exactly one source. Two or more sources cannot reach one another. (Similarly, a sink can be a support if and only if there is exactly one sink.)

This suggests the following idea, an adaptation of topological sort. We begin with all the sources S of G. If there is exactly one, we add it to the support list. If there is a sink in S (i.e., there is a vertex that is both source and sink), then we should stop immediately since no other vertex can be reached from this sink or can reach the sink. Otherwise, we remove all the sources in S.

Now what? If the graph still has vertices, we will find one or more sources. Again if there is exactly one source, we can add it to the support list. Why? Because, every vertex we have encountered thus far has a path to this source (otherwise, this would have become a sink earlier). And this source has a path to every other remaining vertex (otherwise, there would have been more sources). So we repeat the previous step and proceed until the garph is empty.

```
1. Let $L$, the list of supports, be initially empty.
2. While $G$ is not empty:
   (a) Let $S$ be the set of sources in $G$.
   (b) If $S$ has exactly one vertex $v$, then add $v$ to $L$.
   (c) If $S$ has a sink, then return $L$ and exit.
   (d) Remove $S$ and its outgoing edges from $G$. Update the degrees of all affected vertices
       (enabling the easy calculation of $S$ in the next iteration).
3. return $L$
```

The above algorithm is a small adaptation of topological sort, and takes time $\Theta(n+m)$.