Problem Set: Graphs: DFS & BFS

Problem 1. Prove the correctness of the following recursive EXPLORE algorithm, i.e. suppose the algorithm was given a vertex s to explore, then you must prove the following

- If u is reachable from s then $visited[u] = 1$
- If u is not reachable from s then $visited[u] = 0$

Algorithm 1 : Explore

```
$visited \leftarrow$ ZEROS($n$)                ▷ Initialize the visited array to n zeros
function EXPLORE($v$)
    $visited[v] \leftarrow 1$
    for $(v, u) \in E$ do                      ▷ for all neighbors of $v$
        if not visited[$u$] then
            EXPLORE($u$)
```

Problem 2. Write pseudocode of DFS using Stack data structure instead of recursive EXPLORE().

Problem 3. Analyze the runtime of the following algorithms:

1. DFS on a graph $G = (V, E)$ if
   (a) G is given as a binary adjacency matrix.
   (b) G is given as an adjacency list.
   (c) G is a directed connected graph.
   (d) G is a directed disconnected graph.
2. BFS on a connected graph $G = (V, E)$ if
   (a) G is given as a binary adjacency matrix.
   (b) G is given as an adjacency list.

Problem 4. Given an undirected graph $G = (V, E)$, prove that if we run $DFS(s)$ from some $s \in V$, there cannot be a cross edge in the DFS tree.

Problem 5. Let $G = (V, E)$ be a directed graph. Prove that G has a directed cycle iff $DFS(G)$ has a back edge

Problem 6. Given a digraph $G = (V, E)$ suppose we run DFS from $s \in V$. You're given the arrays s and f, containing starting time and finishing time of vertices respectively, after running $DFS(s)$. Then, given an edge $e = (v, u) \in E$ describe how would you determine if e is a tree edge, back edge, forward edge, or a cross edge.

Problem 7. Let $G = (V, E)$ be a graph. For a vertex $v \in V$, let $R(v) := \{u \in V : \exists$ a path from v to $u\}$, i.e. $R(v)$ is the set of vertices that are reachable from v. Let $f[v]$ denote the finishing time of a node during the DFS run. Prove that if we run DFS on G, then for any vertex v, $\forall\ u \in R(v), f[v] \geq f[u]$.

Problem 8. Let G be a directed acyclic graph (DAG). Prove that G must have a source vertex and a sink vertex.

Problem 9. Let $G = (V, E)$ be a directed acyclic graph. Let G' be the transpose graph of G (formed by reversing the edges in G), i.e. $G' = (V, E')$ and $E' = \{(u, v) : (v, u) \in E\}$). Show that all the vertices that were sinks in G become sources in G' and the vertices that were sources in G become sinks in G'.

Problem 10. Show that the strongly connected components graph of any graph G is a DAG. Strongly connected components graph is a graph where each SCC is a vertex and there is a directed edge from SCC C_1 to another SCC C_2, if and only if $\exists u \in C_1,\ v \in C_2,\ (u, v) \in E(G)$.

Problem 11. Show that If a graph G consists of only two strongly connected components C and C', and there is an edge from a node in C to a node in C', then after DFS the largest finish time will be of some vertex in C.

Problem 12. Prove that if we run DFS on $G = (V, E)$, then the largest finishing time will be for a vertex in a source component.

Problem 13. Given an undirected graph $G = (V, E)$, find its connected components. Analyze runtime of your algorithm.

Problem 14. For the topological sort algorithm discussed in class (repeatedly removing source vertices, for details refer to lecture notes) describe how would you find a source vertex in the graph and hence analyze the total runtime complexity of the algorithm.

Problem 15. Give detailed analysis of the topological sort algorithm based on finish times, (not the one discussed above). Compare runtime of this algorithm with the one given above.

Problem 16. Given a graph $G = (V, E)$, prove that for $s \in V$, $BFS(s)$ will visit only those vertices that are reachable from s.

Problem 17. Given an undirected graph G on n vertices, design an algorithm that determines whether G is a tree.

Problem 18. The distance between two vertices a and b, i.e. $d(a, b)$, is defined to be the length of the shortest path from a to b. Let $L(x)$ be the level number of the vertex x in the BFS tree.

1. Suppose we have a $BFS(s)$ tree of a connected graph $G = (V, E)$. Given $(u, v) \in E$, prove that $|L(u) - L(v)| \leq 1$.

2. Given a graph $G = (V, E)$, show that if the shortest path from $s \in V$ to $t \in V$ is $s, v_1, v_2, \ldots, v_k, t$, then the length of the shortest path from s to v_i must be exactly i. This is called *path suboptimality.*

3. Given a graph $G = (V, E)$, suppose we run $BFS(s)$ for $s \in V$. Prove that a vertex $x \in V$ is in level i of the $BFS(s)$ tree if and only if $d(s, x) = i$.

4. Given a connected graph $G = (V, E)$, let $s \in V$, and $u, v \neq s$ such that $(u, v) \in E$. Construct an example graph on $n \leq 6$ vertices such that $d(s, u) \geq d(s, v) + 2$, or prove that such a graph does not exist.

Problem 19. Communication networks can be visualized as graphs. Network nodes such as routers, hubs, etc. are represented using vertices and the links between the network nodes are represented by edges. Subsequently, nodes which can communicate with each other must have at least one path between them on the graph. As such, it is of interest to network designers to identify possible points of failure, which upon failing would leave nodes unable to communicate with each other.

1. Given an n-node network, which is represented by the graph $G = (V, E)$. Suppose we are given two nodes s and t in V, such that all paths between s and t are strictly greater than $n/2$. Show that there is a node v, not equal to s or t, such that removing v from G would eliminate all $s - t$ paths (i.e. it will disconnect s and t).

2. Devise an algorithm with runtime $O(|V| + |E|)$, which can find such a node v.

Problem 20. We have a connected graph $G = (V,\ E)$, and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u, and obtain the same tree T. Prove that $G = T$. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)