ESO207A: Data Structures and Algorithms

Practice Problems *1*

Instructions.

1. Start each problem on a new sheet. For each problem, Write your name, Roll No., the problem number, the date and the names of any students with whom you collaborated.
2. For questions in which algorithms are asked for, your answer should take the form of a short write-up. The first paragraph should summarize the problem you are solving and what your results are (time/space complexity as appropriate). The body of the essay should provide the following:
   (a) A clear description of the algorithm in English and/or pseudo-code, where, helpful.
   (b) At least one worked example or diagram to show more precisely how your algorithm works.
   (c) A proof/argument of the correctness of the algorithm.
   (d) An analysis of the running time of the algorithm.

   Remember, your goal is to communicate. *Full marks will be given only to correct solutions which are described clearly.* Convoluted and unclear descriptions will receive *low marks.*

Problem 1. Bipartite Graphs. A *bipartite* graph is an undirected graph $G = (V, E)$ whose vertices can be partitioned into two sets $V = V_1 \cup V_2$, where, V_1 and V_2 are disjoint subsets, and there are no edges between vertices in the same set (i.e., for any $u, v \in V_1$, there is no edge between u and v.

(a) A graph G is said to be colorable with k-colors, if each vertex is given one of the k colors and no two adjacent vertices are given the same color. (Adjacent vertices are given different colors). Show that a graph is bipartite iff it is 2-colorable.

(b) Show that an undirected graph is bipartite if and only if it contains no cycles of odd length.

(c) Give a linear time $O(|V|+|E|)$ algorithm to determine whether an undirected graph is bipartite or not.

(d) If an undirected graph has exactly one odd cycle, what is the minimum number of colors needed to color it.

Problem 2. Review of DFS basics. Suppose you are running a depth first search of a given directed graph $G = (V, E)$. At some given time, the vertex being explored is u (i.e., u is at the top of the stack) and the edge being considered is (u, v). Assume that $v.d = v.f = \infty$ for all vertices v initially.

1. If v is colored black (i.e., $v.f$ is finite or, equivalently, v has been fully explored) and without checking any further information, which of the conclusions is possible: (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge. Make your arguments clearly.

2. Suppose v is colored black and additionally it is observed that $v.d < u.d$. Which of the conclusions must be made and why? (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge.

3. Suppose v is colored black and additionally it is observed that $v.d > u.d$. Which of the conclusions must be made and why? (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge.

4. If v is colored white, which of the conclusions must be made and why? (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge.

5. If v is colored gray, which of the conclusions must be made and why? (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge.

6. A vertex u is gray iff it is being explored but has not been fully explored. Hence, $u.d < \infty$ and $u.f = \infty$. Argue yes or no.

7. At some instant of DFS, say u and v are both colored gray. Which of the conclusions is possible: (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge.

8. At some instant of DFS of a graph G, say u and v are both colored gray. If there is an edge (u, v) and $u.d < v.d$, then, which of the conclusions is possible: (a) (u, v) is a tree edge, (b) (u, v) is a back edge, (c) (u, v) is a forward edge, or, (d) (u, v) is a tree edge.

9. At some instant of DFS of a graph G, say u and v are both colored gray and $u.d < v.d$. Then, is it correct to say the following:
   (a) At time $u.d$, v is white.
   (b) v is a descendant of u in the DFS tree.
   (c) Consider the DFS stack and say the segment from u to v is $u, w_1, w_2, \ldots, v$. Then, there is a sequence of edges $(u, w_1), (w_1, w_2), \ldots, (w_k, v)$, which were explored consecutively by DFS after u was discovered.

10. Suppose DFS is being run on a graph. At the instant $u.d$, v is a vertex reachable from u and is white. Is it correct to infer that v will become a descendant of u? Show a counterexample.

11. Suppose DFS is being run on a graph. At the instant $u.d$, v is a vertex reachable from u via a path consisting of entirely white vertices. Then, show that every vertex on this path becomes a descendant of u. Hence for every vertex w on this path, $u.d < w.d < w.f < u.f$.

12. Suppose DFS is being run on a graph. At the instant $u.d$, v is a vertex reachable from u via a path consisting of entirely white vertices. Is it true that the consecutive vertices on the path will be consecutively on the DFS stack? Give a counterexample.

13. Design a simple variant of DFS of an undirected graph G that assigns to each connected component C of G a unique number, and for all vertices $u \in C_i$, the variable $ccnum[u]$ is set to this number. The number of these unique numbers should equal the number of connected components in G (i.e., numbering starts at 1 and there are no gaps).

14. Suppose G is an undirected graph and $u, v_1, v_2, \ldots, v_k, u$ is an undirected cycle in G where, $u, v_1, \ldots, v_k$ are all distinct vertices. In the course of DFS, suppose u is the first vertex of this cycle that is visited (becomes gray first). By the white path property, each of $v_1, v_2, \ldots, v_k$ is a descendant of u.

    (a) Can we claim that $\{v_k, u\}$ is a back edge? Give a counterexample.

    (b) Can we claim that $\{v_1, u\}$ is a back edge? Give a counterexample.

    (c) Can we claim that one of $\{v_k, u\}$ and $\{v_1, u\}$ is a back edge. Argue this.

15. Consider DFS of a directed graph. If (u, v) is an edge, is it always true that $u.f > v.f$? Give a counterexample.

16. Consider DFS of a directed acyclic graph. If (u, v) is an edge, argue that $u.f > v.f$.

17. If G is a directed acyclic graph, argue that G has at least one source vertex, that is, a vertex with no incoming edges into it. Similarly argue that G has a sink vertex, that is, a vertex with no outgoing edges from it.

18. Suppose we do a DFS of a directed acyclic graph. Is it true that if s is a source vertex, then it has the highest $u.f$ value among all vertices u. Give a counterexample.

19. Suppose a DFS of a directed acyclic graph is performed. Show that the vertex s with the largest $s.f$ value is a source vertex.

20. Suppose a DFS of a directed acyclic graph $G = (V, E)$ is performed and the vertex s with the largest $s.f$ value is removed from V to get the subgraph $G' = (V - \{s\}, E')$, where, E' contains the edges in E except those that are incident upon or from s. Without repeating the DFS, show that in G', the vertex with the largest $.f$ value is a source vertex *in* G'. This gives another argument for obtaining a topological ordering: 1. Perform DFS. 2. Sort the vertices in decreasing order of $.f$ values.

21. Another way of looking at topological sorting algorithm is as follows. In a directed graph G, if there is an edge (u, v), then, $u.f > v.f$ (why?). Hence sorting in decreasing order of $.f$ values is consistent with edges (u, v).

22. Suppose we wish to find a sink vertex in an acyclic graph. Does the following algorithm work: Start at any vertex. Now visit any neighbor, if there is one. Repeat until you come to a vertex with no neighbor. Is this correct? What is the time complexity of this procedure? Can this be repeated to obtain a topological ordering of the graph? What is the complexity? Would you prefer a DFS-based solution and why?

23. Suppose we wish to find a sink in a directed acyclic graph using DFS. Suppose we construct $G^R = (V, E^R)$, that this, if (u, v) is an edge in E, then, (v, u) is an edge in E^R. Show that a sink in G is a source vertex in G^R. This gives an algorithm to find a sink. A sink is a vertex with the largest finish times in the DFS of G^R. What is the time complexity of this algorithm? Is it preferable than the earlier algorithm described?

24. Recall the definition of "connected" in a directed graph. Two vertices u and v are said to be connected if there is a directed path from u to v and there is a directed path from v to u. Let $G = (V, E)$ be a directed graph. *Show* that "connectedness" of two vertices is an equivalence property: it is reflexive, symmetric and transitive (i.e., if u is connected to v and v is connected to w, then u is connected to w). Let $[u]$ denote the set of all vertices that u is connected with. *Show* that, by property of equivalence relation, for any $u, v \in V$, either $[u] = [v]$ or $[u] \cap [v] = \phi$. The set of vertices in $[u]$ are those vertices that are connected with u and $[u]$ is called the strongly connected component of u. Define $\bar{V}$ to be the set of such strongly connected components, that is, $\bar{V} = \{[u] \mid u \in V\}$. Define the graph $\bar{G} = (\bar{V}, \bar{E})$ to be the graph of strongly connected components of G. There is an edge from $[u]$ to $[v]$ if there is an edge $(w, x) \in E$ such that $w \in [u]$ and $x \in [v]$. *Show* that $\bar{G}$ is an acyclic graph.

Problem 3. DFS basics: II These next few parts are concerned with finding the strongly connected components of $\bar{G} = (\bar{V}, \bar{E})$ of a given directed graph $G = (V, E)$.

1. Is it true that if the *DFS_Explore*(G, u) routine is called, then it will terminate after visiting all (and only those) nodes that are reachable from u? Show this.

2. Suppose C_s is a sink component of $\bar{G} = (\bar{V}, \bar{E})$. Now, if we start *DFS_Explore* (G, u) from any vertex $u \in C_s$, then, all vertices of C_s are discovered and only these are discovered. Show this.

3. So, the first problem is: how do we find a sink component in $\bar{G}$? Second problem: Once we remove all the vertices of this sink component from G and the edges incident to and from them, how do we find the sink component in the smaller graph, so that we can keep the process going? The rest of the parts try to answer this.

4. Suppose we consider the reversed graph $G^R = (V, E^R)$. All edge directions are reversed. Show that the strongly connected components of G^R are the same as G. What is the time taken to compute G^R?

5. Suppose we do a DFS on G and let C and C' be two distinct components in $\bar{V}$ and there is an edge from some vertex in C to a vertex in C'. Then, the highest finish time among the vertices in C is higher than the highest finish time among the vertices in C'. Show this. Alternately, complete the following argument.

   (a) There are two cases. The DFS on G^R touches a vertex of C before it visits any vertex of C'. Show that $u.f$ will be larger than $v.f$ for any vertex $v \in C'$. (At the time the first vertex u of C is visited, all other vertices of C and C' are white and they will become descendants of u and therefore will finish before u finishes. In fact, $u.f$ will be greater than the finish times of every single vertex in $(C \cup C') - \{u\}$).

   (b) Case 2: The DFS of G^R touches a vertex u' of C' before it visits the first vertex u of C. Show that $u.f > u'.f$. (Once u' is visited, then, all vertices of C' will be visited and finished visiting before $u'.f$. At time $u'.f$, none of the vertices of C have yet to be visited, since, there is no path from u' to any vertex in C. In fact there is no path from any vertex in C' to any vertex in C. let u be the first vertex in C to be visited by the DFS. Then, $u.d > u'.f$ and therefore, $u'.f < u.d < u.f$. In fact, $u.f$ is greater than the finish times of all vertices in C.)

6. So now we have a way of finding a vertex in a source component. Do a DFS on G and return the vertex with the largest finish time. Why does this work? (By previous part, suppose the vertex returned is u and belongs to component C. If there is a component C' such that there is an edge from some vertex $v' \in C'$ to $v \in C$, then, by the previous argument, the largest finish time of vertices in C' will be larger than the largest finish time of vertices in C. But we have the largest finish time of all vertices as u. So it is a source component).

7. How do we find a sink component in G ? Reverse G to get G^R. Do a DFS on G^R. We get a vertex u say in a source component C of G^R. Now this source component C in G^R is a sink component of G due to reversal of edges. Do a *DFS_Explore* starting from u in G. This will discover C and only C. Thus we have obtained the first sink component.

8. Having found the first sink component, delete it from G (i.e., the adjacency list representation of G). This takes time $O(|C| + |E_C|)$ where, E_C is the number of edges in this component. Show this.

9. How do we find the sink component in the reduced graph, that is, the graph after the first sink component has been deleted? Consider the finish times in the DFS of G^R and find the vertex u_1 among the remaining vertices in G^R (or G) that has the highest finish time. This belongs to a source component of the reduced (remaining) reverse graph. That is, it belongs to a sink component of G. A *DFS_Explore* from u_1 gives a sink component of the reduced graph.

10. The above procedure can be repeated to obtain the strongly connected components of G. Design an algorithm that runs in linear time.

    (a) Run DFS on G^R and order the vertices in decreasing order of finish times.

    (b) Run DFS on G by considering the vertices in the order of decreasing $u.f$ as above. Output the vertices in each tree of the DFS forest as a separate strongly connected component.

Problem 4. Diameter of a tree A *tree* (also called a free tree in the Appendix of the CLRS book) is an undirected, acyclic and connected graph $T = (V, E)$. Design a linear time algorithm to compute the diameter of a given unweighted tree T, that is,

$$\text{diameter}(T) = \max_{u,v \in V} \delta(u, v)$$

where, $\delta(u, v)$ is the length of the shortest path from u to v. The algorithm should also maintain enough information to report a simple path whose length equals the diameter.