Sample Solution to Problem Set 4

Problem 1. (5 points) Computing a cycle

Design an algorithm that takes as input an undirected graph G and returns any cycle in G, if one exists; if no cycle exists, then your algorithm must indicate so. Analyze the worst-case running time of your algorithm, in terms of the number of vertices n and the number of edges m of G.

Answer: Here is the algorithm.

1. Run DFS with the adaptation given in the step below. Recall that DFS in an undirected graph classifies edges into tree and back edges. For each vertex v, let $\pi(v)$ denote the parent of v in the traversal.
2. During the traversal, if you find a back edge from descendant u to ancestor v, then the path that goes from u to $\pi(u)$ to $\pi(\pi(u))$, so on, to v, followed by the edge (u, v) is a desired cycle.
3. If there is no back edge, then return that the graph has no cycles.

Running time is $\Theta(n + m)$.

Problem 2. (8 points) Trading Places

Alice and Bob are located at two different vertices s and t, respectively, of a directed graph $G = (V, E)$; that is, Alice is at s and Bob is at t. They want to exchange their places in the minimum number of steps. In any step, each can do the following: either stay where they are, or follow an outgoing edge to a neighboring vertex. But there is a catch; at no time can they be on the same vertex and during no step can they be crossing the same edge (in either direction).

Design an algorithm that returns a minimum-step solution for Alice and Bob to trade their places. If there is no way for Alice and Bob to trade their places under the constraint given above, then your algorithm must indicate so. Make your algorithm as efficient as you can. Analyze the worst-case running time of your algorithm, in terms of the number of vertices n and the number of edges m of G.

Answer: We follow the approach in the posted hint. Construct a new graph $G' = (V', E')$ in which the set of vertices is $V \times V$; that is, we have an ordered pair (u, v) for every $u, v \in V$. We have an edge $(u, v) \to (u', v')$ in E' if it is legally possible for Alice to move from u to u' and Bob to move from v to v', both simultaneously in a single step.

1. for any u in V, for any v in V:
   (a) if $u \neq v$, add (u, v) to V';
   (b) for every x in V, for every y in V:

i. if ($x \neq y$ and $(u, x) \in E$ and $(v, y) \in E$ and not ($x = v$ and $y = u$)), then add $((u, x), (v, y))$ to E'.

2. Solve the given problem by doing a BFS in $G' = (V', E')$, starting from source vertex (s, t). We return the shortest path to (t, s). The path gives the entire sequence of moves.

Every sequence of moves in G for Alice and Bob can be captured by a pair of sequences $s = s_0, s_1, \ldots, s_T = t$ and $t = t_0, t_1, \ldots, t_T = s$ with the requirement that for every $0 \leq i < T$, it is legal for Alice to move from s_i to s_{i+1} and Bob to move from t_i to t_{i+1} simultaneously in a single step. (Note that if s_{i+1} is same as s_i, then this move refers to Alice staying at her current location.) This sequence of moves, taking T steps, exactly corresponds to the path

$$(s, t) = (s_0, t_0) \to (s_1, t_1) \to (s_{T-1}, t_{T-1}) \to (s_T, t_T) = (t, s)$$

of length T in G'. Therefore a shortest sequence of legal moves in G that allow Alice and Bob to trade places corresponds to a shortest path in G' from (s, t) to (t, s). If no such path exists, then there is no way for Alice and Bob to trade places.

The running time of the algorithm is $O(n^4)$ since the four nested for-loops each has n iterations. The BFS takes time linear in the size of G', which is $O(n^4)$ since the maximum number of vertices and edges in G' is at most n^2 and n^4, respectively.

We explored some ideas that have the potential to reduce the complexity of the above algorithm (for example, by only constructing as much of G' as needed), but have not been able to achieve a time better than $O(n^4)$.

Problem 3. (2 + 5 = 7 points) Course sequences

You are enrolled in a program whose curriculum consists of n courses, all of them mandatory. Some courses are prerequisites for others, so the courses cannot be taken in any order. You are given a prerequisite graph G, which has a vertex for each course, and an edge from course v to course w if and only if v is a prerequisite for w.

(a) Suppose you can only take one course per semester. Then, it is going to take n semesters for you to complete the program. Design a linear-time algorithm that returns a valid order in which to take the n courses.

Answer: Here is the algorithm.

1. Run Topological Sort on G.
2. Return the courses in the order returned by the Topological Sort.

The running time of Topological Sort is $\Theta(n + m)$ time, where m is the number of edges in G.

(b) Now suppose that you can take any number of courses in one semester. Design a linear-time algorithm that computes the minimum number of semesters necessary to complete the curriculum.

Answer:

1. Initialize semester $T = 1$.

2. If G is empty, then exit.
3. Identify all nodes in G with no incoming edges. Schedule all of them at time T.
4. Remove all the nodes just scheduled. Increment T by 1 and go to step 2.

Correctness: Let the *critical path* of G be the length of the longest path in G. Clearly, any schedule will have to take time at least the length of the critical path. We now claim that our algorithm outputs a schedule that takes time equal to the length of the critical path. To see this, observe that in every iteration, we remove all nodes with in-degree 0; this decreases the length of the critical path by exactly 1. So the number of iterations, which is exactly the length of the schedule, equals the length of the critical path in G.

Running time: The above algorithm is very similar to topological ordering. The same reasoning as in the analysis for topological ordering establishes a $\Theta(n + m)$ running time.