Honors Analysis of Algorithms November 14, 2025

Solutions to Problem Set 4

Name: Daniel Wichs (TA 2016), Srihari Narayanan (Course Assistant 2025) Due: November 2, 2025

Problem 1

1.1

Algorithm to detect whether a given undirected graph contains a cycle, and to output a cycle if there exists one that runs in $O(m+n)$

Solution: Without loss of generality, assume the graph is connected (if not, do the following for each connected component).

Pick an arbitrary vertex as root and do a DFS, while maintaining a DFS tree. The DFS tree initially contains the root and at each point, when you encounter an edge to a new vertex, the new vertex is added to the tree and the edge is added between the new vertex and the parent. If at any point you encounter an edge from a node to a seen vertex, then there is a cycle in the graph, and you can return the cycle by tracing the path from each end point of the latest edge backwards in the DFS tree till they meet (linear time), and displaying it in the correct order. If the DFS ends without incident, there is no cycle. DFS runs in linear time on the edges/vertices, as each edge/vertex is visited only once. □

1.2

Given a graph with edges between vertices that say "same" or "different", determine in $O(m+n)$ if there exists a valid labeling of vertices with 2 labels.

Solution: Without loss of generality, assume the graph is connected (if not, do the following for each connected component).

Pick an arbitrary vertex to be the root, and assign it label A. Now do a BFS from root. For each neighbor, if the neighbor is not-visited, and is connected to the current vertex via a "same" edge, give it the same label as the parent, and if the not-visited neighbor is connected to the current node via a "different" edge, give it the alternate label. If the neighbor is visited and the label it contains is different from the one you would have given it if it were not visited, then return "inconsistent" and stop. If you visit all the nodes and are able to assign labels to all nodes without any inconsistencies, return "consistent". Note that since we assign a label to the root, at any point the current vertex will have a label assigned to it. We visit each edge and vertex only once, so the algorithm is $O(n+m)$. □

1.3

Given an undirected graph on n nodes, and 2 vertices s and t that are a distance strictly greater than $n/2$ apart, show that there is some node that is not s or t such that deleting the node removes all $s-t$ paths. Find the node v.

Solution: To show that such a vertex exists, assume contradiction. This implies that we must have at least 2 vertex disjoint paths from s to t in the graph. But the distance between s and t is strictly greater than $n/2$. Therefore, each of these paths has $\geq n/2$ vertices without including s or t, which would mean there has to be $\geq n+2$ vertices in the graph which is a contradiction. Therefore, such a vertex exists.

To find this vertex, do a BFS from s till you reach t. There are at least $n/2$ layers between s and t, which means there has to be at least one layer which has only 1 vertex, otherwise we end up having more vertices than n, which is a contradiction. We can therefore maintain the level of each vertex seen before t, and return any vertex that is alone in its level, which can be done in linear time.

The time complexity of the algorithm is $O(m+n)$

□

Problem 2

Solve: [Kleinberg Tardos] Chapter 3, problem 12, page 112.

Solution: Create a directed graph which has 2 vertices associated with each person, b_i and d_i, denoting birth date and death date. For each person, add a directed edge from b_i to d_i, implying chronological order. For each fact of the form i died before j was born, add an edge from d_i to b_j. For each fact of the form lives of i and j overlapped, add edges from b_i to d_j and from b_j to d_i. (Their lives overlap if and only if each of them was born before the other died.)

Check if there is a cycle, if so, return "inconsistent". If not, run a topological sort and we get an order of births and deaths that is consistent with the facts. □

Problem 3

Show that starting with the zero flow, the Ford-Fulkerson max-flow algorithm could take upto 2000 augmentation steps.

Solution: The initial residual graph is the same as the flow network. The Ford-Fulkerson algorithm could choose the (s,a,b,t) augmenting path and push a flow of 1 through it. Then it could do the same thing for the path (s,b,a,t). Therefore, after two steps, the residual graph would be similar to the initial one with 999 instead of 1000 for the 4 edges $s-a$, $s-b$, $a-t$ and $b-t$. This sequence could continue for another 1998 steps until the residual capacities of the four peripheral edges become zero, so the running time would be 2000 steps. □

Problem 4

As suggested by Dinitz as well as Edmonds and Karp, during the execution of the Ford-Fulkerson max-flow algorithm, suppose we always select an augmenting path in the residual graph that contains a minimum number of edges. Prove that the Ford-Fulkerson algorithm terminates in $O(mn)$ iterations where n and m are the number of vertices and edges in the network respectively.

Solution: Let R be the residual graph. We show that $\text{dist}_R(s,t)$, the length of the shortest path from s to t in R, a) never decreases, and b) increases by at least 1 after at most $O(m)$ iterations of the algorithm. Since the length of the path from s to t in an s-t connected graph can be at most $n-1$, proving these two facts immediately shows that the Dinitz-Edmonds-Karp modification of Ford-Fulkerson gives an algorithm with at most $O(mn)$ iterations.

Lemma 1. *$dist_R(s,t)$ never decreases.*

Proof. We consider a BFS in R from s with layers $L_0, \ldots, L_{n-1}$. Let $k := \text{dist}_R(s,t)$, so that $t \in L_k$. Let P be a shortest augmenting path from s to t, and let R' be the residual graph obtained by breaking this path. If we have edges in P corresponding to forward edges with flow 0 or fully saturated backward edges, then breaking P could add new edges to the residual graph, of the form of edges f going from L_i to L_{i-1} for $1 \leq i \leq k$. Now, suppose for a contradiction that we have a shorter path P' from s to t in R'. Such a P' must include such a new edge f, since otherwise we could find P' in the original R; however, this would necessitate edges going from L_j to L_{j+3} for some j in the BFS, which is impossible by definition of BFS. □

Lemma 2. *$dist_R(s,t)$ increases by at least 1 after at most $O(m)$ iterations.*

Proof. Notice that we only ever add backward edges to the BFS and always delete at least one forward edge in the layers between L_0 and L_k per iteration. Thus, since the graph R has at most $2m$ edges, $2m = O(m)$ is an upper bound on the number of iterations needed to break all k-length paths between s and t. □

□

Problem 5

Vertex cover and matching.

Solution: 1)Every edge of the matching M must have at least one of its vertices in S. Therefore, every edge in M, can be mapped to some unique vertex in S.
2)A fully connected graph of three vertices has matchings of size 1 but its vertex cover must be of size at least two.
3) Construct a flow network as described in class by adding a source s and a sink t. Compute the max-flow and the corresponding min-cut. Let the min-cut be $\{s\} \cup A \cup B$ and $\{t\} \cup (U \setminus A) \cup (W \setminus B)$ (all cut edges go from A to C). Let $C \subseteq (W \setminus B)$ be the set of neighbors of A in $W \setminus B$. In order to verify that $(U \setminus A) \cup B \cup C$ is a vertex cover, we notice that all possible edges have at least one of their endpoints in this set. All possible edges must either start from $(U \setminus A)$ and end in W (covered by $U \setminus A$) or start from A and end in C (covered by C) or start from U and end in B(covered by B).
We now show that the size of this set is equal to the number of edges that cross the minimum cut. We have exactly $(U \setminus A)$ edges from s to $(U \setminus A)$, exactly B edges from B to t and exactly C edges from A to C. Therefore, this is the capacity of the cut and it is equal to the value of the maximum flow. Since we already know that the value of the maximum flow equals the size of the maximum matching, we conclude that it suffices to just run the Ford-Fulkerson algorithm for the new graph. □

Problem 6

Decomposition into strongly connected components.

Solution: See *Kosaraju's algorithm* for an $O(|V| + |E|)$ algorithm. Here we give a simpler poly-time algorithm.
To begin with, we perform a depth first search from each vertex and we aggregate all the child vertices that can be reached. This way, in polynomial time, for each vertex we have a list of all the vertices that can be reached from it through directed paths.
We now pick some arbitrary vertex $u \in V$. For every vertex v in the reachability list of u, we check if u also exists in the reachability list of v. We put u along with all the vertices of its list which prove to be equivalent with it in a set P_1. We delete all the vertices of P_1 from V and continue by picking an arbitrary next vertex and repeating the above process, defining one new set P_i each time.

Now that we have a decomposition of the graph into strongly directed components, we assume that there exists some directed cycle in the induced graph G'. If we have an edge in G' between two sets P_i and P_j, this means that there exists a directed path from every vertex of P_i to every vertex of P_j. The existence of a directed cycle in G' therefore implies the existence of a directed cycle in G which passes through vertices that are not equivalent. This is a contradiction, since all vertices that belong to a directed cycle must be equivalent with each other. Therefore, G' must be acyclic. □