CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Homework #4 Solutions

Wednesday, April 06

Problem 1. A *walk* in a directed graph $G = (V, E)$ from a vertex s to a vertex t, is a sequence of vertices $v_1, v_2, \ldots, v_k$ where $v_1 = s$ and $v_k = t$ such that for any $i < k$, (v_i, v_{i+1}) is an edge in G. The *length* of a walk is defined as the number of vertices inside it minus one, i.e., the number of edges (so the walk $v_1, v_2, \ldots, v_k$ has length $k - 1$).

Note that the only difference of a walk with a *path* we defined in the course is that a walk can contain the same vertex (or edge) more than once, while a path consists of only distinct vertices and edges.

Design and analyze an $O(n + m)$ time algorithm that given a directed graph $G = (V, E)$ and two vertices s and t, outputs *Yes* if there is a walk from s to t in G whose length is divisible by five, and *No* otherwise.

(25 points)

Solution. The algorithm is by reduction to the graph search problem.

Reduction: We create a graph $G' = (V', E')$ as follows. Vertices of G' are obtained by copying the vertices V of G five times to gets sets $V' = V_1 \cup V_2, \ldots \cup V_5$. For any vertex $v \in V$ and $i \in \{1, \ldots, 5\}$, we use v_i to denote the copy of v inside V_i in G'. For any edge $(u, v) \in E$ inside G, we add the following five edges to E':

$$(u_1, v_2) \quad (u_2, v_3) \quad (u_3, v_4) \quad (u_4, v_5) \quad (u_5, v_1).$$

We then run a graph search algorithm on G' from the vertex $s_1 \in V_1$: if the vertex $t_1 \in V_1$ is reachable in this search, we return *yes* (meaning there is such a walk) and otherwise we return *No*.

Proof of Correctness: We prove that there is a walk with length divisible by 5 in G if and only if t_1 is reachable from s_1 in G'. This then immediately implies the proof by the graph search algorithm we are running on G'.

- A length-5-divisible walk from s to t in G implies t_1 is reachable from s_1 in G': Let

  $$W = s, u, w, z, y, x, \ldots, t$$

  be the walk in G. Consider the sequence

  $$W' = s_1, u_2, w_3, z_4, y_5, x_1, \ldots, t_\ell$$

  in G'. This sequence forms a valid walk in G' by the definition of edges of G' that always go from V_i to V_{i+1} for $i \in \{1, \ldots, 4\}$ and from V_5 to V_1 for $i = 5$. Moreover, as the length of the walk W is divisible by five, we get that the ending vertex of the walk would be $t_\ell = t_1$ which is the vertex we wanted to reach. So there is also a walk from s_1 to t_1 in G' in this case and thus the algorithm answers correctly.

- If t_1 is reachable from s_1 in G' then there is a length-5-divisible walk from s to t in G: Consider the path

  $$P' = s_1, u_2, w_3, z_4, y_5, x_1, \ldots, t_1$$

  in G' which exists as t_1 is reachable from s_1. Given the edges of the graph G' always go from V_i to V_{i+1} for $i \in \{1, \ldots, 4\}$ and from V_5 to V_1 for $i = 5$, the only way this path can end in t_1 is if its length is a multiple of 5. Given that each consecutive pairs of vertices in P' correspond to a pair in G, this path translates to a walk

  $$P = s, u, w, z, y, x, \ldots, t$$

  in G with the same length as P'. Thus, there is also a length-5-divisible walk from s to t in G.

This concludes the proof of correctness.

Runtime Analysis: Creating the graph $G' = (V', E')$ takes $O(n+m)$ time as we are copying each vertex and edge five times. Running graph search, say by DFS/BFS algorithm, on G' also takes another $O(5n+5m) = O(n+m)$ time as it has $5n$ vertices and $5m$ edges. So, total runtime is $O(n+m)$.

Problem 2. We say that an undirected graph $G = (V, E)$ is **2-edge-connected** if we need to remove *at least two* edges from G to make it disconnected. Prove that a graph $G = (V, E)$ is 2-edge-connected if and only if for every cut $(S, V - S)$ in G, there are *at least two cut edges*, i.e., $|\delta(S)| \geq 2$. **(25 points)**

Solution. We prove each part separately:

- If G is 2-edge-connected then every cut in G has at least 2 cut edges. We prove this by contradiction. Suppose G is 2-edge-connected but there exists a cut $(S, V - S)$ with only one cut edge e. Consider the graph $G - e$ obtained by removing the edge e from G; the cut $(S, V - S)$ now has zero cut edges in $G - e$ and as proven in Lecture 9, this means $G - e$ is disconnected. This contradicts the fact that we needed to remove two edges from G to make it disconnected.
- If G is not 2-edge-connected then there exists a cut in G with less than 2 cut edges. Since G is not 2-edge-connected, there is an edge e such that $G-e$ is disconnected; let S be any connected component of $G - e$ and note that $(S, V - S)$ is a cut in G with only one cut edge e, as desired.

Problem 3. Given a connected undirected graph $G(V, E)$ with distinct weights w_e on each edge $e \in E$, a *maximum* spanning tree of G is a spanning tree of G with maximum total weight of edges[1]. Design an $O(m \log m)$ time algorithm for finding a maximum spanning tree of any given graph. **(25 points)**

Hint: You may want to use graph reductions, but remember that you are <u>not</u> allowed to have *negative* weights on edges of graphs (for now at least).

Solution. The algorithm is by reduction to the minimum spanning tree problem.

Reduction: We find the maximum weight edge in G and let W denote this edge weight. We then create new weights $w'_e = W + 1 - w_e$ for every edge $e \in E$. Finally, we run any MST algorithm (say Kruskal's algorithm) for finding a minimum spanning tree of graph G with new weights and return this tree as a maximum spanning tree of the G under original weights.

Note that since all edge weights are *positive* in this new graph, we are allowed to run a MST algorithm on the resulting weights as well (in general, MST is only defined for positive weight graphs and so we are not allowed to run an arbitrary algorithm for MST on a negative-weight graph, even though some of the algorithm's like Kruskal's or Prim's work without any change for graphs with negative weights also. Since our goal is to do a reduction and hence we should be able to use *any* algorithm for MST, we cannot assume that algorithm works with negative weight edges.).

Proof of Correctness: Consider any spanning tree T of G. Let $w(T)$ denote the total weight of T under weight function w_e on edges $e \in E$ and $w'(T)$ denote the weight under w'_e. By our construction, $w(T) = (W+1) \cdot (n-1) - w'(T)$ since each spanning tree has exactly $n-1$ edges. As such, maximizing $w(T)$ (over the choice of T), is equivalent to minimizing $w'(T)$ (again over the choice of T). Thus, a minimum spanning tree under weights w' corresponds to a maximum spanning tree under weights w.

Runtime Analysis: Finding the maximum weight edge and defining weights w'_e takes $O(n+m)$ time to go over the graph. We can then use the best possible algorithm for MST on our graph to solve the problem;

[1] This is exactly the opposite of the minimum spanning tree (MST) problem we studied in the lectures.

even though we do not know what necessary is runtime of that algorithm, we know that it is at most $O(m \log m)$ (since Kruskal's algorithm takes at most that time) and hence runtime of our algorithm is at most $O(m \log m)$ (but it can potentially be lower by using a better MST algorithm).

Problem 4. Let $G = (V, E)$ be an undirected connected graph with maximum edge weight $W_{\max}$. Prove that if an edge with weight $W_{\max}$ appears in some MST of G, then all MSTs of G contain an edge with weight $W_{\max}$. **(25 points)**

Solution. Let T be a MST of G that contains an edge e with weight weight $W_{\max}$. Suppose towards a contradiction that the statement of this problem is not true and there is some MST T' such that *all* edges in T' have weight $< W_{\max}$.

Consider the graph $T - e$. Since we removed an edge from a tree to get $T - e$, we have that there are two connected components S and $V - S$ in $T - e$. Since T' is connected, there is at least one cut edge in the cut $(S, V - S)$ in T' (note that vertices of T, $T - e$, and T' are the same). Let f be such a cut edge.

Define the graph $T'' = T - e + f$. We claim that T'' is also a spanning tree. This is because (1) T'' also has $n - 1$ edges as T had $n - 1$ edges and we removed one edge and added another; (2) T'' is connected as $T - e$ had two connected components S and $V - S$ and we connected them with the edge f in $T - e + f$.

Finally, we have that $w(T'') < w(T)$ where $w(\cdot)$ denotes the weight of the entire tree. This is because $w_e = W_{\max}$ and $w_f < W_{\max}$ by our assumptions. But since T'' is also a spanning tree we now have a contradiction: T is supposed to be a MST of G meaning that there cannot be any other spanning tree in G with weight less than T.
