Foundations of Algorithms, Spring 2023
Homework #5

All members of the collaboration group are expected to participate fully in solving collaborative problems. Note, however, that each student is required to write up their solutions individually. Common solution descriptions from a collaboration group will not be accepted. Furthermore, to receive credit for a collaboration problem, each student in the collaboration group must actively and substantially contribute to the collaboration. This implies that no single student should post a complete solution to any problem at the beginning of the collaboration process.

Problems for Grading

1. [20 points] Suppose 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.)

2. [20 points] Suppose you and your friend Alanis live together with $n-2$ other people at a popular "cooperative" apartment. Over the next n nights, each of you is supposed to cook dinner for the co-op exactly once, so some one cooks on each of the nights. To make things interesting, everyone has scheduling conflicts on some of the nights (e.g., exams, deadlines at work, basketball games, etc.), so deciding who should cook on which night becomes a tricky task. For concreteness, let's label the people $\{p_1, ..., p_n\}$ and the nights $\{d_1, ..., d_n\}$. Then for person p_i, associate a set of nights $S_i \subset \{d_1, ..., d_n\}$ when they are *not* available to cook. A *feasible dinner schedule* is defined to be an assignment of each person in the co-op to a different night such that each person cooks on exactly one night, then there is someone to cook on each night, and if p_i cooks on night d_j, then $d_j \notin S_i$.

   (a) [10 points] Describe a bipartite graph G such that G has a perfect matching if and only if there is a feasible dinner schedule for the co-op.

   (b) [10 points] Your friend Alanis takes on the task of trying to construct a feasible dinner schedule. After great effort, she constructs what she claims is a feasible schedule and then heads off to work for the day. Unfortunately, when you look at the schedule she created, you notice a big problem– $n-2$ of the people at the co-op are assigned to different nights on which they are available (no problem there), but for the other two people p_i and p_j, and the other two days d_k and d_l, you discover she has accidentally assigned both p_i and p_j to cook on night d_k and no one to cook on night d_l. You want to fix this schedule without having to recompute everything from scratch. Show that it is possible, using her "almost correct" schedule, to decide in only $O(n^2)$ time whether there exists a feasible dinner schedule for the co-op. If one exists, your algorithm should also provide that schedule.

3. [30 points] You are helping some security analysts monitor a collection of networked computers, tracking the spread of an online virus. There are n computers in the system, labeled $C_1, ..., C_n$, and as input, you are given a collection of trace data indicating the times at which pairs of computers communicated. Thus the data is a sequence of ordered triples (C_i, C_j, t_k). Such a triple indicates that C_i and C_j communicated at time t_k. Assume there are m triples total.

   Now let us assume that the triples are presented to you sorted by time of communication. For purposes of simplicity, we will assume that each pair of computers communicates at most once during the interval you are observing. The security analysts you are working with would like to be able to answer the following question: If the virus was inserted into computer C_a at time x, could it possibly have infected computer C_b by time y? The mechanics of infection are simple–if an infected computer C_i communicates with an uninfected computer C_j at time t_k, (in other words, if one of the triples (C_i, C_j, t_k) or (C_j, C_i, t_k) appears in the trace data), then the computer C_j becomes infected as well, starting at time t_k. Infection can thus spread from one machine to another across a sequence of communications, provided that no step in this sequence involves a move backwards in time. Thus, for example, if C_i is infected by time t_k and the trace data contains triples (C_i, C_j, t_k)

and (C_j, C_q, t_r), where $t_k \leq t_r$, then C_q will become infected via C_j. (Note that it is okay for t_k to be equal to t_r. This would mean that C_j had open connections to both C_i and C_q at the same time, so a virus could move from C_i from C_q.)

For example, suppose $n = 4$, and the trace data consists of the triples

$$(C_1, C_2, 4), (C_2, C_4, 8), (C_3, C_4, 8), (C_1, C_4, 12),$$

and the virus was inserted into computer C_1 at time 2. Then C_3 would be infected at time 8 by a sequence of three steps–first C_2 becomes infected at time 4, then C_4 gets the virus from C_2 at time 8, and then C_3 gets the virus from C_4 at time 8. On the other hand, if the trace data were

$$(C_2, C_3, 8), (C_1, C_4, 12), (C_1, C_2, 14),$$

and again the virus was inserted into computer C_1 at time 2, then C_3 would not become infected during the period of observation. Observe, however, that although C_2 becomes infected at time 14, C_3 only communicates with C_2 before C_2 becomes infected. There is no sequence of communications moving forward in time by which the virus could get from C_1 to C_3 in this second example.

(a) [12 points] Design an algorithm that answers questions of this type: given a collection of trace data, the algorithm should decide whether a virus introduced at computer C_a at time x could have infected computer C_b by time y.

(b) [12 points] Prove that the algorithm runs in time $O(m)$.

(c) [6 points] Prove the correctness of your algorithm.

4. [30 points] ***Collaborative Problem***: We define the *Escape Problem* as follows. We are given a directed graph $G = (V, E)$ (picture a network of roads.) A certain collection of vertices $X \subset V$ are designated as *populated vertices*, and a certain other collection $S \subset V$ are designated as *safe vertices*. (Assume that X and S are disjoint.) In case of an emergency, we want evacuation routes from the populated vertices to the safe vertices. A set of evacuation routes is defined as a set of paths in G such that (i) each vertex in X is the tail of one path, (ii) the last vertex on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated vertices to "escape" to S without overly congesting any edge in G.

(a) [10 points] Given G,X, and S, show how to decide in polynomial time whether a set of evacuation routes exists.

(b) [10 points] Suppose we have exactly the same problem as in (a), but we want to enforce an even stronger version of the "no congestion" condition (iii). Thus, we change (iii) to say, "the paths do not share any *vertices*." With this new condition, show how to decide in polynomial time whether such a set of evacuation routes exists.

(c) [10 points] Provide an example with the same G, X, and S in which the answer is "yes" to the question in (a) but "no" to the question in (b).