Boston University, CS 330 Fall 2019
Homework 4
Due Wednesday, October 2, 2019

Homework Guidelines

Please make sure you read the collaboration policy before you start working on your homework. Refer to the general information handout for the homework policy and additional options.

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. Understandability of your answer is as desirable as correctness, because communication of technical material is an important skill. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

1. **(Leftover graphs, 10 points)** You are running a student club, and you notice that people are dropping out of the club. You know who is friends with whom in your club, and you observe a curious phenomenon: the people who drop out only know at most one other person in your club! The folks who know two or more other people stick around.

   Suppose you start out with n club members. Their friendships form an undirected graph G, with n vertices (one for each club member). You can assume that no new friendships form in the club—it's not that social.

   At every subsequent meeting, if there is some student still in the club who sees that they have fewer than two other friends still in the club, then they drop. (In graph terms, each vertex of degree 0 or 1 will be removed, along with its edge, if applicable.)

   After a while, the process will stop—either there will be no one left, or there will be a set of students who each know at least two others in the group. Let G' denote the graph of friendships among the students remaining when the process stops. We call G' the *heart* of G.

   For example, if G consisted of just three vertices in a path, then its heart G' would be the empty graph (since all the vertices would get deleted). If G consisted of a cycle, then its heart G' would be the same as G.

   (a) (0.5 pages) After playing with a few small examples, you come with the following guess: For every undirected graph G that is a tree, its heart G' is empty.

   Is this true? Give either a proof (say, by induction on the number of nodes), or a counterexample. [*Hint:* If one of the leaves of a tree is removed, what's left?]

   (b) (1 page) Give an algorithm which takes as input a graph G and returns its heart G' in time $O(m+n)$, where m is the number of edges in G. (See solution format guidelines.) [*Hint:* There are a few natural approaches. One is to run DFS or BFS and traverse the tree starting with the leaves (be careful: the traversal might start at a vertex that ends up getting deleted). Another approach is to keep a list of all the degree-1 vertices, and update it as vertices are deleted.]

2. **(Scheduling Site Visits, 10 points)** You are working for a construction company running a large set of projects in far away Somerville, MA. There are n projects, where project i starts on date s_i and ends on f_i. You wish to visit Somerville at least once during each of the projects, but you also want to visit as few times as possible (it's on Red Line!).

   For a list of project dates $[s_i, f_i]$, $i = 1, ..., n$, a set of visit dates $\{t_1, ..., t_k\}$ is *sufficient* if every project has a visit that happens while it is being done (it is ok if the visit happens on date s_i, date f_i or anywhere in between). In the example of Figure 1, the black vertical lines mark a set of sufficient dates. Removing any one of them would make the set not sufficient. (But it's still not the smallest such set.) You want to design an efficient algorithm that finds as small a sufficient set as possible. In the following, you may assume that (i) all the endpoints $s_1, ..., s_n, f_1, ..., f_n$ are distinct, and (ii) the original set of projects covers a contiguous segment of the real line.

   (a) (Do not hand in) Find a smallest sufficient set of visit dates for the example in Figure 1.

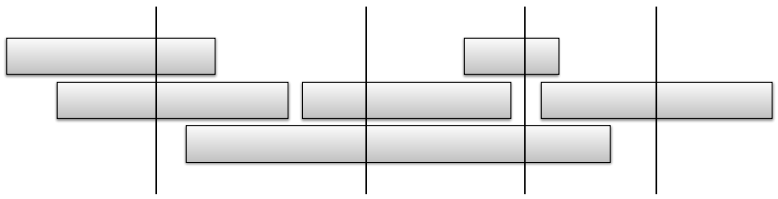

Figure 1: A sufficent set of four site visits.

(b) You discuss the problem with two colleagues. The first one suggests the following greedy approach: at each stage, add a visit date that intersects as many projects as possible from among the projects that are not covered so far. Give an example showing that this approach will not find the smallest sufficient set of visit dates.

(c) Your other colleague thinks there is a connection to the interval scheduling problem we saw in class.[1] She wonders out loud: "if the input contains a set of k mutually disjoint projects, I think every sufficient set has size at least k?" Is she right? Justify your answer briefly.

(d) Give a polynomial-time algorithm that takes the $(s_1, f_1), ..., (s_n, f_n)$ as input and outputs a minimum-size set of acceptable dates. [*Hint:* Modify the interval scheduling algorithm.]

(e) What is the running time of your algorithm in terms of n?

(f) Prove that your algorithm is correct. [*Hint:* Use your second colleague's observation. You need to show that your solution matches her bound.]

3. (Extra exercise, *do not hand in.*) Imagine the process keeps on going in stages: once we there are no more vertices of degree 0 or 1, we perform stage 2 by removing vertices of degree 2 or less, until the graph is empty or all vertices have degree at least 3. In stage 3, we remove vertices of gree three or less. At stage k (if we ever get there), we remove vertices of degree k or less until all remaining vertices have degree $k + 1$ or more.

   We say a student is *k-hardcore* if they get removed at stage k. Give a $O(m+n)$-time algorithm to compute the hardcoreness of every student.

[1] Reminder: in that problem, given a set of intervals, we wanted to find a *maximum*-size set of intervals that are mutually disjoint.