CS 170
Fall 2017

Efficient Algorithms and Intractable Problems
Prasad Raghavendra and Sanjam Garg

Homework 4

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

Special Questions:

- *Shortcut questions*: Short questions are usually easy questions that give you opportunities to practice basic materials. However, we understand that some of you are very familiar with the topics and do not want to spend too much time on easy questions. Therefore, we design shortcut questions for this purpose. A shortcut question usually has multiple parts that build upon each other and are ordered by their difficulty level. You can work on those in order or start from wherever you like. However you only need to submit the last part you are able to solve. For example, if a question has 5 parts (a, b, c, d, e), you are confident about part e, you should submit part e without any of the previous four parts. If you are confident about d but not sure about e, you should submit d for grading purposes. Please clearly indicate in your submission which part you are submitting.
- *Redemption questions*: It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.
- *Extra credit questions*: We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.

Due Wednesday, October 4, at 4:59pm

0. Who did you work with?

List all your collaborators on this homework. If you have no collaborators, please list "none".

1. (★★ level) Kruskal and Prim

Answer the following questions. Justify your answer with a short proof or counterexample. You can assume all edges have distinct costs.

(a) We create a subgraph of G with the first k edges chosen by Kruskal's algorithm. Can there be a strictly cheaper acyclic subgraph of G with k edges?

(b) We create a subgraph of G with the first k edges chosen by Prim's algorithm (starting from some root node r). Can there be a strictly cheaper connected subgraph of G containing the node r along with k other nodes?

2. (★★★ level) Uniqueness of Minimum Spanning Tree

In this problem, we will study the conditions that make a graph have a unique MST and devise an efficient algorithm to recognize such graphs. For all parts of this problem, assume that the input graph is connected.

(a) In general, a graph need not have a unique MST. Give an example of a graph with multiple MSTs.

(b) Recall the *Cut Property*: Suppose edges X are part of at least one MST of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then, $X \cup \{e\}$ is part of some MST.

Prove that, if we additionally assume that e is the unique lightest edge across the partition, then not only *some* MST containing X contains e but *every* MST containing X also contains e. This is called the *Unique Cut Property*.

(c) For any graph G, let E' be the set of all edges e such that e is a (not necessarily unique) lightest edge across some partition. Show that G has a unique MST if and only if E' forms a spanning tree of G.

Hint: you may use part (b).

(d) Design an efficient algorithm to decide whether a graph $G = (V, E)$ has a unique MST. Your algorithm's running time should be at most $O((|V| + |E|) \log |V|)$.

(Please turn in a four part solution to this problem, but, in the "Proof of Correctness" section, you are allowed to provide just a short justification of the algorithm instead of a full formal proof.)

3. (★★★★ level) A Wizardry Party

Unlike magicians, wizards are not known to transport people across universes. Instead, they are known for wearing (often pointy) hats, sporting long white beards, and just being old and grumpy in general. Nevertheless, they still like to socialize and they regularly organize parties. Although you are a Muggle (i.e. No-Maj), you would like to attend these cool parties too. There is one problem regarding this though: these wizards talk a lot and, if you are to attend their parties, you have to blend in well during conversations.

One topic they converse about often is their ages. Not totally unlike us humans, wizards consider asking for someone's age or saying it out loud a bit rude. Instead, wizards ask about their relative ages, i.e., how much older is wizard x_i compared to wizard y_i? During the party, you sometimes overhear other wizards' questions and answers. While other times, some wizards ask you questions. In order to not look too suspicious, you would like to try your best to use the queries you overheard to answer questions directed to you.

More specifically, at each time step i, you are given one of the following as input:

- Three integers x_i, y_i and d_i: this means that you overhear that wizard x_i is older than wizard y_i by d_i years. Note that d_i can be negative if wizard y_i is older than wizard x_i.
- Two integers x_i and y_i: this means that you are asked how much older wizard x_i is compared to wizard y_i. Your answer should be of the following two forms:
  - If you can determine the answer from the conversations you overheard so far, then output a single integer corresponding to the number of years wizard x_i is older than wizard y_i.
  - Otherwise, output "I don't know".

Design an efficient algorithm to solve the problem. To receive full points, your algorithm should run in time at most $O(m \log n)$ where m is the number of questions and n is the number of wizards.

(Please turn in a four part solution to this problem, but, in the "Proof of Correctness" section, you are allowed to provide just a short justification of the algorithm instead of a full formal proof.)

Example Input:
1 2 1
1 3
2 3 -3
1 3
Example Output:
"I don't know"
-2
Explanation: First, you overhear that wizard 1 is one year older than wizard 2. Then, you are asked how much older wizard 1 is compared to wizard 3; at this point, you do not have enough information to answer the question yet, so you just output "I don't know". Next, you learn that wizard 2 is three years younger than wizard 3. You are then asked the same question again. This time you have enough information to deduce that wizard 1 is two years younger than wizard 2 and hence you output -2.

4. (★★★ level) Arbitrage

Shortest-path algorithms can also be applied to currency trading. Suppose we have n currencies $C = \{c_1, c_2, \ldots, c_n\}$: e.g., dollars, Euros, bitcoins, dogecoins, etc. For any pair i, j of currencies, there is an exchange rate $r_{i,j}$: you can buy $r_{i,j}$ units of currency c_j at the price of one unit of currency c_i. Assume that $r_{i,i} = 1$ and $r_{i,j} \geq 0$ for all i, j.

(a) The Foreign Exchange Market Organization (FEMO) has hired Oski, a CS170 alumnus, to make sure that it is not possible to generate a profit through a cycle of exchanges; that is, for any currency $i \in C$, it is not possible to start with one unit of currency i, perform a series of exchanges, and end with more than one unit of currency i. (That is called *arbitrage*.) Give an efficient algorithm for the following problem: given a set of exchange rates $r_{i,j}$ and two specific currencies s, t, find the most advantageous sequence of currency exchanges for converting currency s into currency t. We recommend that you represent the currencies and rates by a graph whose edge lengths are real numbers.

(b) In the economic downturn of 2016, the FEMO had to downsize and let Oski go, and the currencies are changing rapidly, unfettered and unregulated. As a responsible citizen and in light of what you saw in lecture this week, this makes you very concerned: it may now be possible to find currencies $c_{i_1}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \times r_{i_2,i_3} \times \cdots \times r_{i_{k-1},i_k} \times r_{i_k,i_1} > 1$. This means that by starting with one unit of currency c_{i_1} and then successively converting it to currencies $c_{i_2}, c_{i_3}, \ldots, c_{i_k}$ and finally back to c_{i_1}, you would end up with more than one unit of currency c_{i_1}. Such anomalies last only a fraction of a minute on the currency exchange, but they provide an opportunity for profit.

You decide to step up to help out the World Bank. Given an efficient algorithm for detecting the presence of such an anomaly. You may use the same graph representation as for part (a).

5. (★★★★ level) Dogs and Strangers

You're walking home from work, with your city modeled as a directed graph. You start at vertex s with vertex t as your destination. On each road (directed edge), you may encounter a dog, generous stranger, or no one at all. Meeting a dog means you give him one or more pieces of your bacon. A generous stranger will give you a certain number of bacon pieces. And meeting no one means you keep your current amount of bacon. Each road always has the same person/dog who takes or gives the same amount of bacon each time you walk down that road. You leave your starting location without any bacon. At no point on your journey home are you allowed to have negative pieces bacon (meaning you would owe a dog bacon), and you are allowed to go down the same road multiple times. You can assume that every vertex is reachable from s and you can reach t from every vertex. Devise an efficient algorithm to determine whether there exists a path for you to get home or if no such path exists.

6. (??? **level)** **(Optional) Redemption for Homework 3**

Submit your *redemption file* for Homework 3 on Gradescope. If you looked at the solutions and took notes on what you learned or what you got wrong, include them in the redemption file.