CS 170
Spring 2017

Efficient Algorithms and Intractable Problems
Prasad Raghavendra and Sanjam Garg

Homework 5

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 Tuesday, March 7, at 11:59am

1. (★ level) Name Distances

Let A be your first name (yes, you, the student) either reduplicated or trimmed until it is 10 letters long. Let B be your last name, similarly reduplicated or trimmed until it is 10 letters long.

Specifically, if your name is fewer than 10 letters long, reduplicate it as necessary (e.g. "ohio" becomes "ohioohiooh", and "nevada" becomes "nevadaneva"), and if your name is longer than 10 letters long, trim it (e.g. "mississippi" becomes "mississipp").

Find the edit distance between A and B! (You may assume upper-case and lower-case letters are equivalent). Show your work by running the algorithm from the textbook/lecture, drawing the edit distance table, and indicating the path corresponding to the solution.

2. (★★★ level) Weighted Set Cover

In class (and Chapter 5.4) we looked at a greedy algorithm to solve the *set cover* problem, and proved that if the optimal set cover has size k, then our greedy algorithm will find a set cover of size at most $k \log n$.

Here is a generalization of the set cover problem.

- *Input:* A set of elements B of size n; sets $S_1, \ldots, S_m \subseteq B$; positive weights $w_1, \ldots, w_m$.
- *Output:* A selection of the sets S_i whose union is B.
- *Cost:* The sum of the weights w_i for the sets that were picked.

Design an algorithm to find the set cover with approximately the smallest cost. Prove that if there is a solution with cost k, then your algorithm will find a solution with cost $O(k \log n)$. Please provide the Main Idea and Proof of Correctness only. If you think Pseudocode can better convey your idea, add it to the *Main Idea*.

Note: We will accept solutions whose running time is a reasonable (low-order) polynomial in n and m. Do not expend too much effort trying to get the running time as low as possible; we are much more concerned with achieving the required approximation factor.

3. (★★★★ level) Ternary Huffman

Trimedia Disks Inc. has developed ternary hard disks. Each cell on a disk can now store values 0, 1, or 2 (instead of just 0 or 1). To take advantage of this new technology, provide a modified Huffman algorithm for compressing sequences of characters from an alphabet of size n, where the characters occur with known frequencies $f_1, f_2, \ldots, f_n$. Your algorithm should encode each character with a variable-length codeword over the values 0, 1, 2 such that no codeword is a prefix of another codeword and so as to obtain the maximum possible compression. Your proof of correctness should prove that your algorithm achieves the maximum possible compression.

Please provide the main idea and proof of correctness only. If you think pseudocode can better convey your idea, add it to the *Main Idea*.

If you are stuck on the proof of correctness, please see proof of binary huffman from lecture.

4. (★★★★ level) Birthday Surprise

Anakin Skywalker has assigned his droid, C3PO, a series of tasks to complete before Padme's birthday. C3PO has N tasks labeled $1, \ldots, N$. For each task, it gains some Republic credits $V_i \geq 0$ for completing the task, and some operating cost $P_i \geq 0$ per day that accumulates for each day until the task is completed. It will take $R_i \geq 0$ days to successfully complete task i. C3PO needs to accumulate enough Republic credits to have a droid makeover before it requires maintenance.

Each day, C3PO chooses one unfinished task to complete. A task i has been finished if it has spent R_i days working on it. This doesn't necessarily mean it has to spend R_i consecutive days working on task i. C3PO starts on day 1, and wants to complete all tasks and finish with maximum Republic credits. If C3PO finishs task i at the end of day t, it will get reward $V_i - t \cdot P_i$. Note, this value can be negative if it chooses to delay a task for too long.

Given this information, what is the optimal task scheduling policy to complete all of the tasks?
Please provide the main idea and proof of correctness only. If you think pseudocode can better convey your idea, add it to the *Main Idea*.

5. (★★★★ level) Finding Maul

The Jedi Council has heard rumors that Maul has been reborn and amassing an army. Obi-Wan Kenobi has been tasked with finding Maul before he can threaten the republic. In order to do so, Obi-Wan leverages the underworld network and aims to find all of the *crime-lords*. A *crime-lord* is a person in the black market who has a link to at least 20 other crime-lords.

We can formalize this as a graph problem. Let the undirected graph $G = (V, E)$ denote the underworld's relationship graph, where each vertex represents a person who has significant dealings in the black market on Coruscant. There is an edge $\{u, v\} \in E$ if u and v have a known relationship with each other on Coruscant (we will assume that relationships are symmetric). We are looking for a subset $S \subseteq V$ of vertices so that every vertex $s \in S$ has edges to at least 20 other vertices in S. And we want to make the set S as large as possible, subject to these constraints.

Design an efficient algorithm to find the set of crime-lords (the largest set S that is consistent with these constraints), given the graph G. Unlike previous questions in this homework, please use a 4-part algorithm solution.

Hint: There are some vertices you can rule out immediately as not crime-lords.