CS 170 Efficient Algorithms and Intractable Problems
Fall 2017 Prasad Raghavendra and Sanjam Garg

Homework 6

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 18, at 4:59pm

This homework emphasizes dynamic programming problems. In your solutions, note the following:

- When you give the main idea for a dynamic programming solution, you need to explicitly write out a recurrence relation and explain its interpretation.
- When you give the pseudocode for a dynamic programming solution, it is not sufficient to simply state "solve using memoization" or the like; your pseudocode should explain how results are stored.

0. Who did you work with?

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

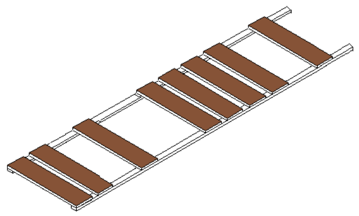
1. **(★★★ level) Longest Palindrome Subsequence**

   A subsequence is *palindromic* if it is the same whether read left to right or right to left. For example, "bob" and "racecar" are palindromes, but "cat" is not. Devise an algorithm that takes a sequence $x[1..n]$ and returns the length of the longest palindromic subsequence. Its running time should be $O(n^2)$. (Recall that a subsequence need not be consecutive, e.g., "aba" is the longest palindromic subsequence of "anbma".)

   For this problem, you should know how to do the proof of correctness, but need not include it in your submission. You should submit the main idea, pseudocode, and runtime.

2. (★★★ level) Bridge Hop

You notice a bridge constructed of a single row of planks. Originally there had been n planks; unfortunately, some of them are now missing, and you're no longer sure if you can make it to the other side. For convenience, you define an array $V[1..n]$ so that $V[i] = \text{TRUE}$ iff the ith plank is present. You're at one side of the bridge, standing still; in other words, your *hop length* is 0 planks. Your bridge-hopping skills are as follows: with each hop, you can increase or decrease your hop length by 1, or keep it constant.



For example, the image above has planks at indices [1, 2, 4, 7, 8, 9, 10, 12], and you could get to the other side with the following hops: [0, 1, 2, 4, 7, 10, 12, 14].

You start at location 0, just before the first plank. Arriving at any location greater than n means you've successfully crossed. Due to your winged shoes, there is no maximum hop length. But you can only hop forward (hop length cannot be negative).

Devise an efficient algorithm to determine whether or not you can make it to the other side.

For this problem, you should know how to do the proof of correctness, but need not include it in your submission. You should submit the main idea, pseudocode, and runtime.

3. (★★★ level) A Sisyphean Task

Suppose that you have n boulders, each with a positive integer weight w_i. You'd like to determine if there is any set of boulders that together weight exactly k pounds. You may want to review the solution to the Knapsack Problem for inspiration.

For this problem, you should know how to do the proof of correctness, but need not include it in your submission. You should submit the main idea, pseudocode, and runtime.

(a) Design an algorithm to do this.

(b) Is your algorithm polynomial in the *size* of the input? Remember that size is in terms of how many bits we need.

4. (★★★★ level) Non-Prefix Code

As we have learned in lecture, the Huffman code satisfies the *Prefix Property*, which states that the bit string representing each symbol is not a prefix of the bit string representing any other symbol. One nice property of such codes is that, given a bit string, there is at most one way to decode it back to a sequence of symbols. However, this is not true anymore once we are working with codes that do not satisfy the Prefix Property. For example, consider the code that maps A to 1, B to 01 and C to 101. A bit string 101 can be interpreted in two ways: as C or as AB. Your task is to, given a bit string s, determine how many ways one can interpret s. The mapping from symbols to bit strings of the code will be given to you as a dictionary d (e.g., in the example, $d = \{A : 1, B : 01, C : 101\}$); you may assume that you can access each symbol in the dictionary in constant time. Your algorithm should run in time at most $O(nm\ell)$ where n is the length of the input bit string s, m is the number of symbols, and ℓ is an upper bound on the length of the bit strings representing symbols.

Please turn in a four-part algorithm solution for this problem.

5. (★★★★ level) Lexicographically smallest subsequence

You are given a string S of length n, and an input k. Give the main idea and runtime (including recurrence relation) for a dynamic programming algorithm to find the smallest (by lexicographical ordering) subsequence of length exactly k. Note that a subsequence must retain characters in the same ordering as in S, but need not be contiguous. For example: in the word "rocket", the smallest subsequence of length 3 is "cet".

Please turn in a four-part algorithm solution for this problem. You may assume that $n > k$, and that concatenating strings together takes constant time. But comparing two strings of length k takes $\Theta(k)$ time.

Extra Practice Question: Can you find a solution that only takes $O(nk)$ time? If you have finished all the other questions, we encourage you to think about this question.

6. (??? level) (Optional) Redemption for Homework 5

Submit your *redemption file* for Homework 5 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.