CSE 417
Algorithms & Computational Complexity
Assignment #3 (rev. b)
Due: Wednesday, 1/30/19

Turnin instructions: this assignment will be turned in (and graded) via gradescope.com, not Canvas. Colleagues who have used it give it four stars, so I am hopeful that it will be convenient for all of us, but I have not used yet, so there may be glitches; please be patient. You should soon get/should have gotten an email from gradescope.com informing you that you have been added to their CSE417 class roster. I added each of you with your @uw email address, i.e., your UW NETID. I think when you go to gradescope.com the first time it will ask you to set a password. If you did not get such an email, check your spam folder, and make sure you are reading your @uw email. If that fails, I believe you can add yourself using your @uw email and the add code M3RX2Y. Once you have successfully logged in, you can upload your assignment by following instructions (from another course) at https://courses.cs.washington.edu/courses/cse401/18au/hw/submitting_hw_guide.pdf. A key step in this is for you to identify which page(s) of your submission contain your answers to which problems. If you have troubles with any of this, please email cse417-staff@cs.washington.edu.

1. [10 points] Prof. Flashbulb proposed the following greedy algorithm for solving the interval partitioning problem (aka interval coloring, aka scheduling all intervals; section 4.1, p 122): apply the greedy algorithm for interval scheduling from the earlier part of section 4.1 (the algorithm on page 118); give all of those intervals one color, remove them from consideration, and recursively apply the same method to the resulting reduced set of intervals (using new colors). Show that this method does not give an optimal solution to the problem by giving a counterexample. (Keep it simple; 4 to 6 intervals should suffice.) Explain why your counterexample is a counterexample.

2. [10 points] Suppose you are designing a new Computer Science building. It has one very long corridor, along which all faculty offices are located, say at positions $x_1 < x_2 < \cdots < x_n$ (arbitrary real numbers, for simplicity). Of course, faculty will die if their offices are not located within 100.0 feet of a combined espresso cart/WiFi hotspot.

   a) Briefly describe a simple greedy algorithm which, given the $x_i$, will find cart/hotspot locations $g_1 < g_2 < \cdots < g_m$ satisfying the above constraint (i.e., for each $1 \leq i \leq n$ there is a $1 \leq j \leq m$ such that $|x_i - g_j| \leq 100$) while minimizing the total number $m$ of carts/hotspots.

   b) Explain why your algorithm is correct. In particular, argue that any solution that differs from the one your algorithm specifies can be replaced by one with no additional carts/hotspots that also satisfies the constraints and is more similar to your algorithm’s solution.

3. [10 points] A quick look at the algorithm on page 124 for the interval partitioning problem suggests an $O(n^2)$ time bound: an outer loop for $j = 1, \ldots, n$, and an implicit inner loop “for each interval $I_i$” that also seems to need order $n$ time.

   a) The algorithm on pg 124 appears to assume that $d$ (the depth of the problem instance; pg 123) is known in advance. However, the very simple variant of it given on slide 29 (approximately) avoids this assumption, and determines $d$ while computing the partition. (How? Basically, assume the depth is 1 until you are faced with 2 overlapping intervals; assume it is 2 until you see 3, etc., and partition accordingly.) Give a slightly more detailed description of an implementation of the later algorithm that will take only $O(dn)$ time. (For your timing analysis, ignore the $\Theta(n \log n)$ time for the initial step that sorts the intervals by start time.)

   b) If $d$ is small, the above method typically will be much faster than the simple ($n^2$) analysis suggests. Show, however, that it is not faster in the worst case. I.e., show that there are instances where it takes time $\Omega(n^2)$. [Note that to show a lower bound like this, it’s not good enough to show one bad example; you really need to show that there are infinitely many bad examples, for larger and larger values of $n$.]

   c) Give a more clever implementation of the algorithm whose running time is $O(n \log n)$, (even if depth is $\Omega(\log n)$) and justify this time bound. [Hint: use a better data structure.]
4. [10 points] The interval partition algorithm sorts intervals by start time. At first glance, it seems possible that sorting by end time would work just as well. Indeed, most of the statements made in the correctness proof (approximately slide 30 in the greedy slides) remain correct when applied to this modified algorithm (after changing “start time” to “end time” and some other obvious things in a few places).

(a) Give a counterexample to the correctness of the modified algorithm and explain why it is a counterexample. (Keep it simple; 4 to 6 intervals should suffice.)

(b) Explain where the proof goes wrong and why. E.g., which statement(s) on slide 30 are false for the modified algorithm (even after fixing the obvious issues like “start time” vs “end time”) and why are these statements important?

5. [10 points]

(a) Simulate the Huffman tree algorithm on the input alphabet a, b, c, d, e with frequencies .15, .30, .50, .02, .03, resp. Show the intermediate “trees” built by the algorithm, as well as the final tree.

(b) What will be the total length in bits of the compressed text built from a sequence of 100 letters having the above frequencies using the Huffman code you calculated?

(c) Suppose I tell you that the frequencies of a and b are .15, .30, resp, as above, but I don’t tell you the other frequencies. I can quickly tell (e.g., without running the full Huffman algorithm) that the Huffman code for this data is not the one shown in figure 4.16 (a) (pg 168). Explain how.

6. Extra Credit: KT Chapter 4, problem 6, page 191. (.tiff image)