CS 170 Algorithms
Fall 2014 David Wagner

HW Instructions

How to write up homework solutions

Here are detailed instructions on how to write your homework solutions, and what we are looking for. This is based on feedback from the readers about common mistakes and shortcomings, so please read these instructions carefully.

Whenever you are asked for an algorithm, your solution should include five elements:

- The **main idea** of your algorithm. This should be short and concise, at most one paragraph—just a few sentences. It does not need to give all the details of your solution or why it is correct. It should be enough that if you were to share it with a classmate, it would be a total giveaway to the problem. This is the single most important part of your solution. If you do a good job here, the readers are more likely to be forgiving of small errors elsewhere.

- The **pseudocode** for your algorithm. The purpose of pseudocode is to *communicate* concisely and clearly, so think about how to write your pseudocode to convey the idea to the reader.

  Note that pseudocode is meant to be written at a high level of abstraction. Executable code is *not* acceptable, as it is usually too detailed. Providing us with working C code or Java code is not acceptable. The sole purpose of pseudocode is to make it easy for the reader to follow along. Therefore, pseudocode should be presented at a higher level than source code (source code must be fit for computer consumption; pseudocode need not). Pseudocode can use standard data structures. For instance, pseudocode might refer to a set S, and in pseudocode you can write things like "add element x to set S." That would be unacceptable in source code; in source code, you would need to specify things like the structure of the linked list or hashtable used to store S, whereas pseudocode abstracts away from those implementation details. As another example, pseudocode might include a step like "for each edge $(u,v) \in E$", without specifying the details of how to perform the iteration. See the pseudocode in the textbook for examples of clear, concise, and precisely specified pseudocode.

- A **proof of correctness**. Your proof needs to be rigorous and convincing, to the level of detail found in the textbook. You must prove that your algorithm work correctly, *no matter what input is chosen*.

  For iterative or recursive algorithms, often a useful approach is to find an invariant. A loop invariant needs to satisfy three properties: (1) it must be true before the first iteration of the loop; (2) if it is true before the ith iteration of the loop, it must be true before the $i+1$st iteration of the loop; (3) if it is true after the last iteration of the loop, it must follow that the

output of your algorithm is correct. You need to prove each of these three properties holds. Most importantly, you must specify your invariant precisely and clearly.

How much detail is needed? Here is a good guideline. Suppose you claim something. Ask yourself whether a fellow CS170 student would immediately see how to prove your claim. If it might not be obvious to your fellow student, you'd better write a careful proof. Or, if a fellow CS170 student might have to think deeply to convince themselves of its correctness, then you'd better prove your claim carefully. If in doubt, prove it out.

If you invoke an algorithm that was proven correct in class or in the textbook, you don't need to re-prove its correctness.

- The asymptotic **running time** of your algorithm, stated using $O(\cdot)$ notation.
- Your **running time analysis**, i.e., the justification for why your algorithm's running time is as you claimed. Often this can be stated in a few sentences (e.g.: "the loop performs $|E|$ iterations; in each iteration, we do $O(1)$ Find and Union operations; each Find and Union operation takes $O(\lg|V|)$ time; so the total running time is $O(|E|\lg|V|)$"). Alternatively, this might involve showing a recurrence that characterizes the algorithm's running time and then solving the recurrence.

Important: clearly **label** your answer to each of these categories. In other words, you should have a prominent label "Main idea:", followed by the main idea. Leave blank space between each section so it is easy for the readers to identify each of the 5 sections of your answer.

The most common mistakes:

1. Failing to include a clearly stated main idea. When you omit the main idea, the readers have to read your pseudocode carefully to try to infer your intent, and this makes them grumpy (and more likely to be picky about slight typos/errors in your pseudocode). If you include the main idea, you're making it easy for the readers to quickly say "Oh, yeah, he/she got the right concept", and then you're more likely to get credit on the question.
2. Failing to label the parts of your answer. This makes it hard to jump to the appropriate part.
3. Handwaving in the proof of correctness. For models of what is considered a sufficient level of detail, you can see the model solutions or the textbook. You should view this as a low watermark: this is likely to be the minimal level of detail necessary, but it's OK to include more detail in your proof of correctness.
4. Attaching a printout with 2 pages of C/Java code, instead of writing clear pseudocode intended for human consumption.

On the next page is a sample of the expected format of your answer. Feel free to use it as a template.

Main idea. To find longest paths in G, we negate every edge length, then run Bellman-Ford to find shortest paths in the negated graph.

Pseudocode.
1. For each edge $(u,v) \in E$:
2. Set $\ell(u,v) := -\ell(u,v)$.
3. Run Bellman-Ford(G,s).

Proof of correctness. Maximizing $\sum_i \ell(v_i, v_{i+1})$ over all paths $v_0, v_1, \ldots, v_k$ is equivalent to minimizing $\sum_i -\ell(v_i, v_{i+1})$, so a longest path in the original graph is equivalent to a shortest path in the negated graph. Bellman-Ford is guaranteed to find the shortest paths from s in the negated graph, as long are no negative cycles in the negated graph. Consequently, our algorithm is guaranteed to find the longest paths from s in the original graph, as long as there are no positive cycles in the original graph.

Running time. $O(|V||E|)$.

Justification. The loop in steps 1–2 takes $O(|E|)$ time, so the running time is dominated by the cost to run Bellman-Ford, which takes $O(|V||E|)$ time.