Introduction to Analysis of Algorithms	Sample Homework Solutions
CS4820 Spring 2013	Thursday, January 24, 2013

Based on the original version by Constadino Moraites and Bobby Kleinberg, CS 4820 S12.

1 The Good, the Bad, and the Ugly

This handout discusses three solutions to Chapter 4, Exercise 15 in K&T. The first solution gives insufficient detail, the second one gives too much detail, and the third one is just right. Certain sentences in the solutions are labeled with numbers in parentheses. A critique is provided after each solution; the numbered items in the critique refer to the correspondingly numbered sentences of the solution.

First let us recall the problem statement.

> The manager of a large student union on campus comes to you with the following problem. She's in charge of a group of n students, each of whom is scheduled to work one *shift* during the week. There are different jobs associated with these shifts (tending the main desk, helping with package delivery, rebooting cranky information kiosks, etc.), but we can view each shift as a single contiguous interval of time. There can be multiple shifts going on at once.
>
> She's trying to choose a subset of these n students to form a *supervising committee* that she can meet with once a week. She considers such a committee to be *complete* if, for every student not on the committee, that student's shift overlaps (at least partially) the shift of some student who is on the committee. In this way, each student's performance can be observed by at least one person who's serving on the committee.
>
> Give an efficient algorithm that takes a schedule of n shifts and produces a complete supervising committee containing as few students as possible.

1.1 Not enough detail

1.1.1 Algorithm

For our algorithm to solve this problem, we take the interval with the earliest finishing time and then, among the ones that overlap it, take the one with the latest finish time.(1) This algorithm clearly terminates as their are only a finite number of intervals.(2) Further, the runtime is $O(n^4)$ because that leaves us a lot of wiggle room and we only need upper bounds for this problem. (3)

1.1.2 Correctness

To prove that our algorithm always returns an optimal solution, we need to show that our algorithm always returns a minimal set that covers all the students. Our algorithm clearly returns a set that covers all the students because our algorithm picks an interval to cover each student. (4) If it didn't cover some student than that would mean that we didn't look at some student, but our algorithm picks the shift with the latest finish time to cover each shift.

Our algorithm obviously produces a set of shifts with minimum cardinality. For each shift we are picking the shift with the latest finish time which overlaps with it so there is no way we would need more shifts than any other algorithm, thus our algorithm returns a set of shifts with minimum cardinality. (5)

1.1.3 Critique

1. This sentence actually isn't as useless as it sounds. The problem is: it doesn't explain any information about preprocessing you might need to fulfill any assumptions you might have about the organization of the input, it isn't precise about what to do after picking intervals (how do I know I am not picking the same interval twice?), and it still doesn't address how to get from picking supervisors for one interval to picking supervisors for another interval. For example, the solution omits mention of the fact that the algorithm has a main loop that continues picking supervisors until all the intervals are covered.

2. This is an example of *proof by intimidation*: asserting that something is obvious rather than explaining it. This is a relatively mild instance of proof by intimidation, since the assertion is really fairly obvious so it's debatable whether a justification is needed. But it's best to be safe and append a justification for the claim that the algorithm terminates, such as "...and the number of uncovered intervals strictly decreases in each iteration of the main loop." This only costs a half-sentence of effort, and makes the reasoning crystal clear.

   Textbooks can get away with proof by intimidation on some details because textbooks are intended to be consumed by students, who are supposed to fill in the gaps and ask questions when something seems really unclear. It's important to keep in mind that the majority of proofs you've read are written for a much different audience than that to which you (and most academics) write. In your homework, just like when submitting a paper to a journal you need to convince the grader (or referee) of the worth and validity of your statements. In this circumstance, you only stand to lose when you use proof by intimidation techniques indiscriminately.

3. It's true that you only need to use big-O notation to bound your runtimes, but unjustifiably high runtimes are still subject to point deduction. (The general rule of thumb is: if you give a tight analysis of the running time of your algorithm, and it runs in polynomial time, we won't deduct any points even if we know of a faster algorithm. But if you give an unjustifiably high upper bound on the running time of your own algorithm, e.g. claiming that a sorting step takes $O(n^2)$ time when there's no reason not to do it in $O(n \log n)$, then we reserve the right to deduct points.)

4. The word "clearly" in this sentence (more proof by intimidation?) obscures the fact that the claim being made in the sentence is justified in the following one. Additionally, the clumsy wording of the following sentence makes it hard to follow the justification. A better rewording of these two sentences would be: "First let us prove that the our algorithm returns a set that covers all the students. The proof is by contradiction: if some student i is uncovered at the end of the algorithm, then in the loop iteration that inspects that student's interval we would have discovered that it was uncovered and we would have covered it by taking the interval with the latest finish time that covers it."

5. This last paragraph is another common example of where proofs can go wrong. What is going on here is that you are just stating the thing you want to prove in slightly different words and arguing that because those two statements are equivalent your algorithm is correct. There are a few possible reasons you might be doing this, but there are good reasons not to do it:

   (a) Even if you are trying to save yourself points by hoping the grader won't notice, you're really just making the grader's job harder. Frankness is worth a thousand words.

   (b) In general, this class focuses around proof techniques for approaching various categories of algorithms. If you can't think of how your proof technique relates to something you learned in class, you should be suspicious about your approach. There are certainly cases where the most elegant way to prove something is not necessarily the approach provided in class, just make sure deviating from the recommended style (ie Greedy Stays Ahead, an Exchange Argument, proving a Recurrence, etc) adds to the clarity of your argument rather than diminishing it.

   As a concrete illustration of why the proof given in this paragraph is inadequate, consider the following critique: "For all I know, it's possible that covering this interval using a shift with an earlier finish time would allow me to make different choices later on, resulting in an overall decrease in the size of the supervising committee. You have to convince me that choosing the latest finish time in each step doesn't commit the algorithm to making suboptimal choices later on." The hypothetical scenario raised by the critique is, in fact, not possible. But it's not self-evidently absurd. The "proof" given in the solution above does nothing to explain why this hypothetical possibility is precluded.

1.2 Too much detail

1.2.1 Algorithm

```
/* init some things */ (1)
# WaitressInit (&app.waith);
# BarGetXdgConfigDir (PACKAGE "/ctl", ctlPath, sizeof (ctlPath));
# /* FIXME: why is r_+_ required? */
# app.ctlFd = fopen (ctlPath, "r+");
# if (app.ctlFd != NULL) {
# app.selectFds[1] = fileno (app.ctlFd);
# FD_SET(app.selectFds[1], &app.readSet);
# BarUiMsg (MSG_INFO, "Control fifo at %s opened\n", ctlPath);
# } else {
# app.selectFds[1] = -1;
# }

lolcats = new ArrayList<>();

for lol in lolcats:

  for lol2 in lolcats.instersects(lol): (2)
  {
    if ( lol.s < lol2.e && lol2.e > max_end) { (3)
      //take lol2
    } else if (lol.s < lol2.e && lol2.e < max_end) {
      //save it for later
    } else if (lol.s < lol2.e && lol2.s < lol.e) {
       //overlaps
    } else {
      hmm.....
    }
  }


//struct (4)
#typedef struct {
# PianoHandle_t ph;
# WaitressHandle_t waith;
# struct audioPlayer player;
# BarSettings_t settings;
# /* first item is current song */
# PianoSong_t *playlist;
# PianoSong_t *songHistory;
# PianoStation_t *curStation;
# char doQuit;
# fd_set readSet;
# int maxFd;
# int selectFds[2];
# FILE *ctlFd;
#} BarApp_t;
#
#endif
```

(5)

1.2.2 Runtime

We start with the simplest step the body of our nested loops. Here there are four possible cases, and three boolean statements. The first statement has two inequalities that need to be evaluated, the second also has two, and the third also has two. The last block is an else so there is no boolean statement there. So in total this is O(6). (6)

Our datastructure, lolcats, enumerates the power set of all the intervals and hashes all of the subsets of the intervals which overlap with a given interval. This means we have constant time retrieval of the set of intervals that intersect with an interval, and there are O(n) intervals. (7)

Finally, there are n intervals that need to be covered, so the outer loop requires O(n) iterations to complete. So in total we have $O(n * n * 6) = O(n^2)$ steps in our algorithm, so our runtime is polynomial in n. (8)

1.2.3 Correctness

We will show correctness using the Greedy Stays Ahead method. When talking about any set of intervals, assume that it is actually a sequence of intervals sorted in order of earliest finish time first. We know we can do this because we can use Merge Sort as explained in lecture, and the run time of merge sort is less than the runtime of our algorithm anyways. (9)

Let T be some sequence of intervals. Then we define $p(T, j)$ to be the finish time of the j^{th} interval in that sequence. If T has fewer than j intervals, define $p(T, j) = \infty$. Let LFT denote our algorithm's solution and let OPT denote the optimal solution. We argue by induction on j, that

$$p(LFT, j) \geq p(OPT, j). \tag{1}$$

We note that after proving this we have proved the correctness of our algorithm and we are done. This is true because if $p(LFT, j) \geq p(OPT, j)$ for all j, then consider the k such that $p(OPT, k+1) = \infty$. Notice that $k = |OPT|$ because the k^{th} interval of OPT is the last one for which a finish time is defined, meaning that after the k^{th} there are no more intervals. If there were more intervals after, then they would have start time infinity, which would be preposterous. Because we are doing induction and we assumed an induction hypothesis, and this induction hypothesis is:

$$p(LFT, k+1) \geq p(OPT, k+1)$$

we know that $p(LFT, k+1) \geq p(OPT, k+1)$ so $p(LFT, k+1) = \infty$. (10) By the same logic that $k = |OPT|$, k is also the cardinality of LFT. This means our algorithm uses just as many intervals as OPT and as the problem is to minimize the number of intervals used, our algorithm is returning an optimal solution.

Before we get into our induction proof, note that we will use $(opt)_j$ and $(lft)_j$ to denote the j^{th} intervals in the set of intervals chosen by OPT and LFT respectively. Now for our proof of (1).

For $j = 1$, let s_1 be the interval with the earliest finish time in the input set S. We know that $(opt)_1$ must overlap with this interval because if some other interval with later finish time in OPT overlapped with it then we wouldn't need $(opt)_1$. We know s_1 was the first interval LFT considered and it picked the interval $(lft)_1$ because it was the interval with the latest finish time that overlapped with s_1. So we know $p(LFT, 1) \geq p(OPT, 1)$ because both $(opt)_1$ and $(lft)_1$ overlap with s_1 and by definition, $(lft)_1$ is such an interval with the latest possible finish time.

Now assume $(*)$ holds for $j-1$ and prove it holds for j. Well by our hypothesis we know $p(LFT, j-1) \geq p(OPT, j-1)$. Compare the set of intervals that remain to be covered by LFT and OPT:

We observe that the set in S which remains to be covered by LFT is a subset of that which remains to be covered by OPT. We arrive at this conclusion because for both OPT and LFT there are no intervals before $(opt)_{j-1}$ and $(lft)_{j-1}$ which still need to be covered. This holds because if some interval with finish time before $(opt)_{j-1}$ was covered by some interval with finish time after $(opt)_{j-1}$ then $(opt)_{j-1}$ would be completely covered by that interval with later finish time, and it would be unnecessary–contradicting that OPT is an optimal solution! We can assume the same for LFT because it covers intervals in order of increasing finish time. With this in mind, all the intervals left to be covered by either OPT or LFT have finishing time after $(opt)_{j-1}$ and $(lft)_{j-1}$, respectively. We also know that neither OPT nor LFT can have covered anything with start time after their $(j-1)^{\text{th}}$ intervals and that $p(LFT, j-1) \geq p(OPT, j-1)$. This means that the set of intervals that remain to be covered by LFT must have cardinality smaller than or equal to that of the set to be covered by OPT, this is true because consider the sets $((lft)_{j-1}, \infty), ((opt)_{j-1}, \infty) \in \mathbb{R}$. $((opt)_{j-1}, \infty) \subseteq ((lft)_{j-1}, \infty)$ because by our inductive hypothesis $p(LFT, j-1) \geq p(OPT, j-1)$. (10)

Not only is the set to be covered by LFT a subset, but the finishing time of the next interval to be covered by LFT is at least as late as, if not later than that of OPT. Now when LFT goes to pick its j^{th} interval, it will pick the interval which intersects with s_j that has the latest finish time. There is no way $(opt)_j$ can end later than $(lft)_j$ because the s_j to be covered by LFT ends later than (possibly as late as) the s_j to be covered by OPT. This allows LFT to pick from a set of intervals to cover its s_j which end at least as late as the set of intervals available to OPT. This ensures $p(LFT, j) \geq p(OPT, j)$, as desired.

1.3 Critique

1. Ah! Random commented out code? When writing real code you may be in the habit of leaving code around in comments in case it might be useful later. Don't do that here! When the reader sees extra code, they will assume that it is meaningful, and if it isn't they may spend a lot of time trying to parse meaning for it before becoming frustrated and giving up.

2. So far the pseudo code has switched between the syntax of three different languages. In general, it is bad practice to present pseudocode that is actual code. If you are trying to define a class or struct in your pseudocode, you are probably doing something wrong. Even when you need to explain data structures, it is fine to explain them in plain English.

3. The whole idea of this algorithm is supposed to be to select an interval, look at everything that intersects with it, and add the latest ending intersecting interval to the committee. Having a bunch of inequalities in terms of the start and end times does not make your explanation more believable, it makes the reader create truth tables in their head to try to reconstruct what the English would be to express your idea.

4. More commented out code. Structs?!

5. It is at times instructive to include a pseudo block in your description of your algorithm, but it is never sufficient. You need to include a plain English description as well, and in the majority of cases, your English description should be enough to convey the meaning of your Algorithm.

6. For Big-O notation, $O(f(n))$ you only need that cf where c is some constant be an upper bound on what you are analysing as the variable goes to infinity. $O(1)$ is just as good as $O(6)$. Also, it really isn't necessary to count up how many constant factors there are in a subroutine of your algorithm. This paragraph could be dispensed with one sentence.

7. In this paragraph you are assuming the ability to construct a non-trivial data structure. (Hash tables are actually a real pain in the butt to implement if you want to make collisions impossible!) If you're using a non-trivial data structure in your algorithm, you're responsible for explaining how the data structure accomplishes its operations within the stated running time, or (if using a data structure from the book) cite the section of the book where the data structure is analyzed.

8. We finally reach the end of the runtime section after three paragraphs. The whole analysis of running time could have been dispensed with in a couple of concise sentences.

9. The biggest danger with giving too much detail in your proofs is that you will make the reading of your proof disconnected so that by the time your reader finishes your explanation of why something is true, they have already forgotten why you needed to show it is true. One way to help counteract this problem is breaking your correctness proof off into lemmas as appropriate. In general you should consider articulating a claim or lemma any time that you're asserting something whose proof requires more than one or sentences.

10. Really the same problem as above. In addition, to keep in mind that one page front and back is a good length to shoot for in a problem that only asks for one algorithm. This isn't a hard and fast

requirement, but on average it is a good amount of detail for one problem where you are asked to give an algorithm. (Of course, if you manage to write a complete solution using even less space, that's even better!)

1.4 Just the right amount of detail

1.4.1 Algorithm

We work to fill up a set C which makes up the committee. We use a set S to hold "uncovered" students for bookkeeping.

1. Initialize S to be the set of all intervals.
2. Sort the intervals by finish time with earliest finish time first.
3. Take the interval i in S with earliest finish time.
4. Construct set $O = \{s \in S \mid s \text{ intersects } i\}$
5. Take $o \in O$ with the latest finish time and add o to C.
6. Find all the intervals in S that intersect with o and delete them from S.
7. Repeat 3-6 until S is empty.
8. Output C.

1.4.2 Runtime

Sorting the intervals takes $O(n \log n)$ time. There will be $O(n)$ iterations of the main loop (steps 3-6) because S initially has n elements and at least one element (namely o) is deleted in each iteration. There could be as many as $O(n)$ intervals intersecting i or o, so each loop iteration takes $O(n)$ time. That puts our total running time at $O(n^2 + n \log n)$ or just simply $O(n^2)$.

1.4.3 Correctness

We will show correctness using an exchange argument. Let us call a committee *valid* if every student's shift overlaps with the shift of a student on the committee. Let C denote the committee constructed by our algorithm, and let $c_1, c_2, \ldots, c_m$ denote its elements, listed in order of increasing finish time.

Lemma 1. *Let D be any valid committee that does not contain C as a subset, and let $d_1, \ldots, d_\ell$ be its elements listed in order of increasing finish time. If k is the least index such that $c_k \notin D$, then there is another valid committee D' such that $|D'| \le |D|$ and $\{c_1, \ldots, c_k\} \subseteq D'$.*

The proof of the lemma is presented below. First, let us assume the lemma and prove that C is optimal. Among all optimal committees, let D be the one containing the longest initial segment of the sequence $c_1, \ldots, c_m$. If D doesn't contain all of C, then the lemma ensures the existence of another optimal committee D' containing a strictly longer initial segment of $c_1, \ldots, c_m$, contradicting our choice of D. On the other hand, if D contains C then it must equal C (since D has minimum cardinality among valid committees) and C itself is therefore optimal.

Proof of Lemma 1. Let i_k denote the value of the variable i in the loop iteration of our algorithm that added c_k to the set C. As D is a valid committee, it must contain an interval d_j that covers i_k. Note that d_j does not belong to $\{c_1, \ldots, c_{k-1}\}$ since i_k was uncovered at the start of the loop iteration in which c_k was picked, whereas $c_1, \ldots, c_{k-1}$ had already been picked at that time.

Let D' be defined by removing d_j from D and inserting c_k in its place. By construction $|D'| = |D|$ and $\{c_1, \ldots, c_k\} \subseteq D'$. So to complete the proof of the lemma we only need to show that D' is valid, i.e. that it covers every interval. Assume, by way of contradiction, that some interval x is not covered by D'. In the committee chosen by our greedy algorithm, x is not covered by any of $c_1, \ldots, c_k$ (as all of these intervals belong to D') so it was still uncovered at the end of the loop iteration in which c_k was selected. This implies that it finishes later than i_k, the interval with the earliest finish time of all uncovered intervals at the start of that iteration. We now have the following chain of inequalities:

$$\text{finish}(d_j) \le \text{finish}(c_k) < \text{start}(x),$$

where the first inequality holds because c_k has the *latest* finish time of all intervals that overlap i_k, and the second inequality holds because x has no overlap with c_k and it cannot finish before c_k starts, since it finishes after the end of i_k.

From the chain of inequalities above, we may conclude that x does not overlap d_j. Thus, some other interval $d' \in D$ overlaps x. But d_j is the only interval in D that does not belong to D'. In particular, d' belongs to D' and hence D' covers x, contradicting our hypothesis that x was uncovered by D'. $\square$