Section 3: Solutions

In this section, we're going to walk step-by-step through good problem-solving strategies applied to one algorithm design question. There are also a couple of problems at the end we won't have time to cover.

1. Line covering

Your new towing company wants to be prepared to help along the highway during the next snowstorm. You have a list of integers $t_1, t_2, \ldots, t_n$ in increasing order, representing mile markers on the highway where you think it is likely someone will need a tow (entrances/exits, merges, rest stops, etc.). To ensure you can help quickly, you want to place your tow trucks so that from every marker, at least one truck is at most 3 miles away. Find a minimum length list of sites where you can place tow trucks to satisfy the requirement, encoded as a list of integers $a_1, a_2, \ldots, a_m$ in increasing order. Note that the sites that you pick need not be a subset of the marked locations.

1.1. Getting started

When reading a long word problem, it is useful to summarize it. A common way is:

Input: …
Expected output: …

- mathematical definitions of any special words used above

(a) Write a summary of the above problem.

Solution:

Input: A list of increasing integers $t_1, t_2, \ldots, t_n$
Expected output: A shortest list of increasing integers $a_1, a_2, \ldots, a_m$ covering the input

- **cover**: for all $i \in \{1, \ldots, n\}$, there exists $j \in \{1, \ldots, m\}$ such that $|t_i - a_j| \leq 3$.

After summarizing the problem, you'll want to consider which big category of techniques you want to use.

- Does the problem remind you of an algorithm you've seen in class, like stable matching, graph algorithms, etc.? If so, you can try a reduction or modifying the known algorithm, like you've done on the past few homework sets.
- If not, you'll want to try techniques for developing algorithms from scratch. **Greedy algorithms** are the first category we're learning about. In the coming weeks, we'll also learn the techniques of divide and conquer and dynamic programming.

For today, since this problem doesn't sound like anything we've seen in class so far, we'll try to use a greedy technique!

1.2. Generating ideas

When developing an algorithm from scratch, it can be difficult to come up with ideas to get started, but there is a common method.

- First, **solve many examples** by hand. In the beginning, don't worry about the general strategy. But as you start to try larger examples, keep the techniques you've learned like greedy methods (and later dynamic programming, etc.) in mind. You will start to see **patterns** after enough examples to give you an idea.
- Your first idea will probably be wrong. Whenever you have an idea, you should **ask yourself**, "Can I break my strategy with a nasty example? Does my strategy ever waste time? Can I optimize it?" If it turns out that your strategy is bad, go back to examples to see how to modify it or to see what other ideas might work.

- After several cycles of this process, you will likely have an idea that you believe to be correct and efficient. At this point, you can exit the idea generating process and begin writing.

For greedy algorithms in particular, keep in mind that your strategies should meet the following criteria:

- Follows a rule to keep picking something
- Doesn't consider the future
- Doesn't go back to fix things

(Of course, considering the future or backtracking to fix mistakes are important things that are sometimes necessary, but the point of greedy algorithms is that very often, those more advanced ideas aren't needed. Check whether or not easy greedy rules first, before trying other techniques that we'll learn later in class.)

Coming up with many greedy ideas should be easy. But finding the correct greedy idea will usually require trial and error or insight, so don't be discouraged.

(b) We will practice the idea generating process.

(i) Solve these examples by hand. Don't worry too much about greedy strategies yet.

1, 2, 4, 10, 12

0, 1, 3, 5, 7, 8, 13, 14

Solution:

For the first: 2 trucks. Many solutions, for example at 2 and 11.

For the second: 3 trucks. Many solutions, for example at 0, 7, and 13.

(ii) Suppose you came up with the greedy idea, "Put a truck on the first uncovered marker." Check that this idea works on the above examples. Then, try to break this idea by coming up with an example where it doesn't work.

Solution:

0, 6 can be covered by one truck at 3, this method gives two trucks. There are many other examples.

(iii) Come up with a new greedy idea that solves your new example. Does the idea work? If not, continue the process until you have a working idea.

Solution:

Place a truck at the farthest location that still covers the next uncovered marker. That is, if t_i is the next uncovered marker, place a truck at $t_i + 3$.

1.3. Writing up your idea

Once you have an efficient, working idea, you should:

- Translate the idea into **pseudocode**. Recall that pseudocode is of form of writing more precise than English, but easier to understand than code. There are no hard rules, but see the handout from last week for common styles.
- **Prove** the pseudocode correct.
- Finally, write up the **running time** analysis.

(c) Write the pseudocode for the solution.

Solution:

1: $i_1 \leftarrow 1$ and $j \leftarrow 1$
2: **while** $i_j \leq n$ **do**
3: $\quad a_j \leftarrow t_{i_j} + 3$
4: $\quad$ Let $i_{j+1} \leftarrow i$, then repeatedly increment i_{j+1} until $t_{i_{j+1}} > a_j + 3$ (or $i_{j+1} > n$).
5: $\quad j \leftarrow j + 1$
6: **return** the list of all a_j

As a refresher, for the proof, always show **validity**, **termination**, and **correctness**.

- Correctness always means, "My algorithm's output matches the problem summary's expected output."
- For greedy algorithms, the expected output is always something about an optimal solution. So, there are two things to prove:
  - "My output is a valid solution.": In today's case, that means "The list $a_1, \ldots, a_m$ is in increasing order and covers all markers."
  - "My output is optimal.": In today's case, that means "All other valid solutions use at least m trucks."

Lastly, the optimality proofs of greedy algorithms also tend to have a consistent structure that you can use to help you. There are a few types:

- "Greedy stays ahead": For all other solutions, show by induction that at every step, your solution is at least as good.
- "Exchange argument": For all other solutions that differ from yours, show how to change a part of the other solution, so that the quality improves or stays the same (but never decreases).
- "Structural argument": (less common) Find a "hard subset" of the input that immediately implies why other solutions must also be as bad as yours (or worse).

(d) Write a proof that your pseudocode is correct.

(i) **Validity**:

Solution:

No lines require validity justification.

(ii) **Termination**:

Solution:

We have $t_{i_j} = s_j - 3 < s_j + 3 < t_{i_{j+1}}$, thus $i_j \neq i_{j+1}$, so i increases every iteration, and there are at most n iterations. Line 4's inline "repeat" ends in at most n iterations as well, since we stop if $i_{j+1} > n$.

(iii) **"The output is in increasing order."**:

Solution:

Again, $t_{i_j} < t_{i_{j+1}}$, thus we conclude that $a_j = t_{i_j} + 3 < t_{i_{j+1}} + 3 = a_{j+1}$.

(iv) **"The output covers all markers."**:

Solution:

We increment i_{j+1} to $i_{j+1} + 1$ if and only if $t_{i_{j+1}}$ is covered by a_j. Since we exit the loop when $i_j > n$,

every marker is covered.

(v) **"All other valid solutions use at least m trucks."**:

Solution:

We show that all other solutions use at least m trucks using "greedy stays ahead".

Let $o_1, \ldots, o_M$ be any other valid solution. It suffices to show by induction on j:

$$P(j) = \text{"Sites } a_1, \ldots, a_j \text{ cover all } t_i \text{ that are covered by } o_1, \ldots, o_j \text{ (and possibly more)."}$$

Base case: We will show $P(1)$. Recall that we set $a_1 = t_1 + 3$. If $o_1 > a_1$, then $o_1 > t_1 + 3$ and does not cover t_1, and neither do $o_2, \ldots, o_M$ because they are larger than o_1 (valid solutions are increasing). This is a contradiction, so this case is impossible.

If $o_1 \leq a_1$, since t_1 is the smallest marker, a_1 covers everything that o_1 covers, which is the claim.

Inductive hypothesis: Suppose that $P(j)$ holds for $j \leq k$.

Inductive step: We will show $P(k+1)$. Note that t_{i_j} is the smallest marker not covered by $a_1, \ldots, a_{j-1}$. (This is a loop invariant, formally prove it by induction.)

So when $j = k + 1$, $t_{i_{k+1}}$ is not covered by $a_1, \ldots, a_k$, and thus nor it is covered by $o_1, \ldots, o_k$, by the IH. We set $a_{k+1} = t_{i_{k+1}} + 3$. If $o_{k+1} > a_{k+1}$, then like in the base case, neither $o_1, \ldots, o_k$ nor $o_{k+1}, \ldots, o_M$ cover a_i, a contradiction.

If $o_{k+1} \leq a_{k+1}$, like in the base case, since $t_{i_{k+1}}$ is the smallest uncovered marker, a_{k+1} covers everything that o_{k+1} newly covers. Combined with the IH, that $a_1, \ldots, a_k$ cover everything that $o_1, \ldots, o_k$ cover, we get $P(k+1)$.

Note: It's also acceptable to use $P(0)$ as the base case (where both lists are empty). It's slightly slicker, but thinking through $P(1)$ will often also give you hints for the inductive step, as it did here.

(e) Analyze and prove the running time with big-O in a few sentences.

Solution:

Line 4's inline repeat occurs n times acorss all iterations of the outer loop, and the rest of the outer loop takes constant time per iteration, for up to n iterations. Hence, the running time is $O(n)$.

The following problems will not be covered in section, but may be useful to think about.
We recommend trying them by yourself first. Solutions will be posted in the evening.

2. Minimizing covers again

You have a set, $\mathcal{X}$, of (possibly overlapping) closed intervals of $\mathbb{R}$. (The closed intervals of $\mathbb{R}$ are the sets commonly denoted $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$.) You wish to choose a subset $\mathcal{Y}$ of the intervals to cover the full set. Here, cover means the union of all intervals in $\mathcal{X}$ is equal to the union of all intervals in $\mathcal{Y}$. Describe (and prove correct) an algorithm which gives you a cover with the fewest intervals.

Solution:

Key idea: Consider all intervals that cover the next point, and among all such intervals take one that goes the farthest right.

Algorithm:

Input: A set $\mathcal{X}$ of subsets of $\mathbb{R}$.
Expected output: A smallest set $\mathcal{Y} \subseteq \mathcal{X}$ such that $\bigcup_{X \in \mathcal{X}} X = \bigcup_{Y \in \mathcal{Y}} Y$.

1: Sort $\mathcal{X}$ so that it is increasing by start point (any order is fine for ties).
2: $i \leftarrow 1$
3: **while** $\mathcal{X} \neq \varnothing$ **do**
4: $\quad y_i \leftarrow \max(b_{i-1}, s)$, where $[s, t]$ is the interval with smallest remaining start point in $\mathcal{X}$.
5: $\quad$ Among all intervals $[a, b] \in \mathcal{X}$ satisfying $y_i \in [a, b]$, let $[a_i, b_i]$ be the one with largest end point.
6: $\quad$ Delete all elements of $\mathcal{X}$ with end point b_i or earlier.
7: $\quad i \leftarrow i + 1$
8: **return** the set of all $[a_i, b_i]$

Each line is valid: The only line that needs a justification is line 4, for which we need to show there exists an interval $[a, b] \in \mathcal{X}$ such that $y \in [a, b]$. This is because either $y_i \leftarrow s$ in line 4, in which case $y_i \in [s, t]$, or $y \leftarrow b_{i-1}$, in which case still $y_i \in [s, t]$, but now because $s < y$ and $t > b_{i-1}$, otherwise we would have deleted $[s, t]$ in the previous iteration.

(By convention, let $\max(b_{i-1}, s)$ return s when b_{i-1} is undefined.)

Termination: We delete at least $[a_i, b_i]$ in each iteration, so the size of $\mathcal{X}$ decreases every iteration, so we terminate.

Correctness/Covering: First, observe that the b_i are increasing, since $b_{i-1} \leq y_i$ (by line 4) and $y_i \leq b_i$ (by line 5), but $b_{i-1} \neq b_i$, because if $b_{i-1} = b_i$, the interval $[a_i, b_i]$ would have been deleted in the $(i-1)$th iteration.

With this observation, for covering, it suffices to notice that if we delete an element $[a, b] \in \mathcal{X}$ in iteration i, then $[a, b]$ is covered by $[a_1, b_1] \cup \cdots \cup [a_i, b_i]$. To show the forward direction, suppose for contradiction that $x \in [a, b]$ is not covered. It is not possible that $x > b_i$, otherwise $[a, b]$ would not have been deleted. Thus, $x \leq b_i$. By the above observation, there is some j for which $b_{j-1} < x \leq b_j$.

For x to be uncovered, in iteration j, we must have had $x < y_j$ because $[y_j, b_j] \subseteq [a_j, b_j]$. Then $b_{j-1} < y_j$, so from line 4, $y_j = s$ where $[s, t]$ was the interval with the smallest remaining start point in $\mathcal{X}$. But $x < y_j = s$ was still a point in $\mathcal{X}$ at this time, contradiction.

Correctness/Optimality: Let $\mathsf{ALG} = [a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]$ be the list of intervals found by the algorithm, and let $\mathsf{OTH} = [o_1, p_1], \ldots, [o_j, p_j]$ be the list of intervals in any other cover.

Note that the a_i are increasing. To show this, recall that the b_i are increasing, so suppose for contradiction $a_{i-1} \geq a_i$, while we know that $b_{i-1} < b_i$. Then $[a_{i-1}, b_{i-1}] \subseteq [a_i, b_i]$. Then $[a_i, b_i]$ was a interval containing $y_{i-1} \in [a_{i-1}, b_{i-1}]$ with larger end point than b_{i-1}, which is a contradiction.

We may assume for this proof that the o_i are increasing, too. (If they are not, there is no harm in renaming the indices.) After that, we may also assume that the p_i are increasing, since like above, an inverted pair would

result in some $[o_i, p_i] \subseteq [o_{i-1}, p_{i-1}]$, and hence $[o_i, p_i]$ can be removed from this solution to make it better. It now suffices to show that:

Claim 1. *For all i, we have $b_i \geq p_i$.*

Once we have this, by noting that for every $i < k$, there are uncovered points larger than b_i after iteration i (hence also larger than p_i), and that the p_i are increasing, we conclude that OTH must also have at least k intervals, as was to be shown.

Proof of Claim 1.

BC: Both ALG and OTH must cover y_1. Furthermore, $[o_1, p_1]$ must cover y_1, because the o_i are increasing, and y_1 is the smallest start point of any interval $\mathcal{X}$. Then, b_1 was chosen to be the largest end point of the all intervals covering y_1, so $b_1 \geq p_1$.

IH: Suppose $b_j \geq p_j$.

IS: Let ϵ be the half the smallest distance between any two numbers that appear in $\mathcal{X}$. We know that $y_{j+1} + \epsilon$ is not covered by $[a_1, b_1], \ldots, [a_j, b_j]$, since the b_i are increasing and $y_{j+1} \geq b_j$ (by line 4), thus $y_{j+1} + \epsilon > b_j$. In fact, there are no uncovered start/end points smaller than $y_{j+1} + \epsilon$. This is a loop invariant that can be proven similar to "Correctness/Covering".

Furthermore, $y_{j+1}+\epsilon$ must be eventually covered, because either $y_{j+1} = s$ for some $[s, t] \in \mathcal{X}$ (and by construction of ϵ, this means $y_{j+1} + \epsilon \in [s, t]$), or $y_{j+1} = b_j > s$, but we must have $t > b_j$ (hence $t \geq b_j + 2\epsilon$) because $[s, t]$ was not deleted in iteration j, so again $y_{j+1} + \epsilon \in [s, t]$.

By IH, $[o_1, p_1], \ldots, [o_j, p_j]$ also does not cover $y_{j+1} \geq b_j \geq p_j$. Since all start/end points smaller than $y_{j+1} + \epsilon$ are covered, and OTH is a valid cover and sorted with no intervals contained in each other, $[o_{j+1}, p_{j+1}]$ must cover something new, which must include $y_{j+1} + \epsilon$. Now, consider the execution of the algorithm: it looked at all intervals containing y_{j+1} and chose the one with the latest end time. Thus, $b_i \geq p_i$. □

Running Time: Sorting takes $\mathcal{O}(n \log n)$ time. Note that afterwards, the entire algorithm including the deletion step can be implemented $\mathcal{O}(n)$ time: Line 4 starts at the first non deleted interval in $\mathcal{X}$, then Line 5 continues stepping forward until the start time exceeds y_i. For line 6, instead of searching through $\mathcal{X}$ immediately, we can just remember to delete all elements of $\mathcal{X}$ with end point b_i or earlier when we encounter them in the next iteration. This way, every interval is seen at most twice, since all intervals seen in lines 4 and 5 get deleted in the next iteration (possibly more). In total, this takes $\mathcal{O}(n \log n)$ time.

3. Art commissions

You've just started a new one-person art company. You've convinced n of your friends to each put $\$c$ of their current money into a bank account, which will eventually be withdrawn when they commission you to make art, supporting your dreams. It takes you one month to finish a commissioned piece (you are only working in your limited free-time). At the beginning of every month, one of your friends will withdraw the entire value in their account to pay you to make their artwork.

The bank accounts all earn small (and varying) rates of interest. Friend i earns interest at the rate of r_i, compounding monthly. That is, the amount in their bank account is r_i times what it was at the start of the last month (until they withdraw their money, and $r_i > 1$). To reiterate, your friends decide to pay you both the original $\$c$ and all of the interest earned at the time you start their commission.

Describe an ordering to take the commissions that will maximize the amount you are paid (you may assume you know the r_i for each of your friends).

Solution:

Note: This solution is only a sketch. It includes the only main idea and the core optimality proof. The solutions you submit on homework should have more detail than this.

Key idea: You should take on the commissions in increasing order of r_i (starting with the smallest). Intuitively, you give the fastest growing account the longest to grow.

Correctness/Optimality: By exchange argument. Let OTH be any other solution, and ALG be the solution we have. Suppose that OTH and ALG are different from each other.

Since ALG keeps elements in sorted order, it must be that interest rates in OTH are not in increasing sorted order. Rename the friends so that OTH takes them in order $1, \ldots, n$. Then there will be a consecutive pair of elements, call them $j, j+1$, where $r_j > r_{j+1}$. We will show that swapping these friends increases our earnings.

In OTH, we get $cr_j^j + cr_{j+1}^{j+1}$ from j and $j+1$. Now suppose we swap just j and $j+1$. We will instead earn $cr_{j+1}^j + cr_j^{j+1}$. Observe that with this swap, we have not affected the amount earned for any other commission. Thus if we manage to show that $cr_{j+1}^j + cr_j^{j+1} > cr_j^j + cr_{j+1}^{j+1}$, we will have proven that OTH is not optimal.

To show the desired inequality, we begin with the observation that:

$$r_j^j(r_j - 1) > r_{j+1}^j(r_{j+1} - 1)$$

which follows from $r_j > r_{j+1}$ (that this pair is not in sorted order) and that all terms in the expression are positive (since the rates themselves are greater than 1).

Distributing, we have

$$r_j^{j+1} - r_j^j > r_{j+1}^{j+1} - r_{j+1}^j$$

Rearranging, so everything is positive we have

$$r_j^{j+1} + r_{j+1}^j > r_{j+1}^{j+1} + r_j^j$$

Multiplying by c and reordering the terms, we have

$$cr_{j+1}^j + cr_j^{j+1} > cr_j^j + cr_{j+1}^{j+1}$$

as desired. Thus, OTH is not optimal.

Solution:

Remarks:

- The algebra in the proof above is much more easily discovered backwards (starting from the desired conclusion), but remember to put proofs in proper logical order.
- This proof relies on the r_i being greater than 1. The problem is different if that isn't the case!
- You could also write a "greedy stays ahead" proof! Though you'd have to keep track of both the rates themselves and the amounts in the accounts to write the proof.