CSC236 Midterm Test Solutions

A correlation between the midterm test and exam has been confirmed. Please use this resource to study well!

Written by AlexTheMango, with pleasure `[^_-]/`

Prepared for December 19, 2024

Question #1

Let $\mathcal{F}$ be the collection of all functions with domain $\mathbb{N}$ and co-domain $\mathbb{R}$.
Given $A, B \in \mathcal{P}(\mathcal{F})$, define addition on $\mathcal{P}(\mathcal{F})$ by

$$A + B \coloneqq \{f + g : f \in A, g \in B\}.$$

Recall that $f + g$ is the function with domain $\mathbb{N}$ and co-domain $\mathbb{R}$ such that

$$(f + g)(n) = f(n) + g(n).$$

Also recall that, if $h \in \mathcal{F}$, then

$$O(h) = \{q \in \mathcal{F} : (\exists n_0, c \in \mathbb{N})(\forall n \geq n_0)[|q(n)| \leq c|h(n)|]\}.$$

Claim: For arbitrary nonnegative $u, v \in \mathcal{F}$, it follows that $O(u) + O(v) = O(u + v)$.

Proof.
This proof demonstrates a double-subset inclusion to show equality.

Forward Inclusion — $O(u) + O(v) \subseteq O(u + v)$:
Let $h \in [O(u) + O(v)]$. Then, $h = f + g$, where $f \in O(u)$ and $g \in O(v)$.

By definition, there exists $c_1, c_2, n_1, n_2 > 0$ such that

- $|f(n)| \leq c_1|u(n)|$ for all $n \geq n_1$;
- $|g(n)| \leq c_2|v(n)|$ for all $n \geq n_2$.

Choose $n_0 = max(n_1, n_2)$ and $c = max(c_1, c_2)$.
Using the definition, triangle inequality, and assumption that u, v are nonnegative functions,

it follows that

$$\begin{aligned}|h(n)| = |f(n)+g(n)| \leq |f(n)| + |g(n)| \leq c_1|u(n)| + c_2|v(n)| \\ &\leq c|u(n)| + c|v(n)| \\ &\leq c(|u(n)| + |v(n)|) \\ &= c(u(n)+v(n)) = c(|u(n)+v(n)|) = c(|(u+v)(n)|).\end{aligned}$$

Thus, $|h(n)| \leq c|(u+v)(n)|$.

By definition, $h \in O(u+v)$. Therefore, $O(u) + O(v) \subseteq O(u+v)$.

Backward Inclusion — $O(u) + O(v) \supseteq O(u+v)$:
Let $h \in O(u+v)$.

By definition, there exists $c, n_0 > 0$ such that $|h(n)| \leq c|(u+v)(n)|$ for all $n \geq n_0$.

It follows that $|h(n)| \leq c|u(n)+v(n)| = c(u(n)+v(n))$, as u, v are nonnegative functions.

Let $w(n) = h(n) - cu(n)$. Consider the following cases for $w(n)$.

Case — $w(n) > 0$:
Notice that $w(n) > 0 \implies h(n) - cu(n) > 0 \implies h(n) > cu(n)$.
Since u is a nonnegative function, then $h(n)$ must be positive.

Recall $|h(n)| \leq c|(u+v)(n)| = c|u(n)+v(n)|$, and both u, v are nonnegative functions.

It follows that

$$\begin{aligned}|w(n)| &= |h(n) - cu(n)| = h(n) - cu(n) \\ &= |h(n)| - cu(n) \leq |cu(n) + cv(n)| - cu(n) \\ &= \cancel{cu(n)} + cv(n) - \cancel{cu(n)} = cv(n) = c|v(n)|.\end{aligned}$$

Thus, $|w(n)| \leq c|v(n)|$. This means $w(n) \in O(v)$.

Write $h(n) = cu(n) + w(n)$. It is obvious that $cu(n) \in O(u)$, and recall that $w(n) \in O(v)$.

Thus, $h(n) \in O(u + v)$.

<u>Case — $w(n) \leq 0$:</u>
Notice that $w(n) \leq 0 \implies h(n) - cu(n) \leq 0 \implies h(n) \leq cu(n)$. So, choose $h(n) = cu(n)$.
Then, $w(n) = h(n) - cu(n) = \cancel{cu(n)} - \cancel{cu(n)} = 0$.

Notice that $|w(n)| = |0| = 0 \leq c|v(n)|$, in fact, for any $c > 0$.
Clearly, $w(n) \in O(v)$.

Write $h(n) = cu(n) + w(n)$. It is obvious that $cu(n) \in O(u)$, and recall that $w(n) \in O(v)$.

Thus, $h(n) \in O(u + v)$.

<u>Conclusion of Cases:</u>
In all cases, $h(n) \in O(u + v)$ has been demonstrated.

Therefore, $O(u) + O(v) \subseteq O(u + v)$.

<u>Conclusion:</u>
Since both inclusions hold, $O(u) + O(v) = O(u + v)$.

$\square$

Question #2

Let $\mathcal{F}$ be as in *Question #1.* Let $\mathcal{G}$ be the collection of all functions with domain $\mathcal{N} \times \mathcal{N}$ and co-domain $\mathcal{R}$. Let $V \in \mathcal{G}$.
For every $i \in \mathbb{N}$, let $g_i(n) = \sum_{j=1}^{i} V(j, n)$, and let $f_i(n) = V(i, n)$.

(a)

Claim: For all $i \in \mathbb{N}$, it follows that $O(g_i) = \sum_{j=0}^{i} O(f_j)$.

Proof.
Denote the predicate:

$$P(i) \coloneqq O(g_i) = \sum_{j=0}^{i} O(f_j)$$

Proceed using the principle of simple induction over $P(i)$ for all $i \in \mathbb{N}$.

Base Case:
Let $i = 0$.

Then,

$$\begin{aligned} O(g_i) &= O(g_0) \\ &= O(\sum_{j=0}^{0} V(j, n)) \\ &= O(V(0, n)) \\ &= O(f_0) \\ &= \sum_{j=0}^{0} O(f_j) \\ &= \sum_{j=0}^{i} O(f_j). \end{aligned}$$

Thus, $P(0)$.

Induction Hypothesis:

Assume for some $k \in \mathbb{N}$, $P(k)$.

This means $O(g_k) = \sum_{j=0}^{k} O(f_j)$.

Induction Step:
Notice that

$$\begin{aligned}
O(g_{k+1}) &= O(\sum_{j=0}^{k+1} V(j,n)) \\
&= O(\sum_{j=0}^{k+1} f_j) \\
&= O(\sum_{j=0}^{k} f_j + f_{k+1}) \\
&= O(\sum_{j=0}^{k} f_j) + O(f_{k+1}), \text{ by } \textit{Question \#1} \\
&= \sum_{j=0}^{k} O(f_j) + O(f_{k+1}), \text{ by the Induction Hypothesis} \\
&= \sum_{j=0}^{k+1} O(f_j).
\end{aligned}$$

Thus, $P(k) \implies P(k+1)$.

Conclusion:
Therefore, by the principle of simple induction, $P(i)$ holds for all $i \in \mathbb{N}$.

□

(b)

Claim: If $g(n) = g_n(n)$, then $O(g) = \sum_{j=0}^{n} O(f_j)$ does **not** necessarily hold.

(b)

Claim: If $g(n) = g_n(n)$, then $O(g) = \sum_{j=0}^{n} O(f_j)$ does **not** necessarily hold.

Proof.
To show that the equivalence in the claim does not necessarily hold, consider a counterexample.

Fix n_0. Define $f_j(n)$ as follows:

$$f_j(n) = \begin{cases} n^2 & \text{if } j = n, \\ 1 & \text{if } j \neq n. \end{cases}$$

Consider the function $g_n(n) = \sum_{j=0}^{n} f_j(n)$.
For $n > n_0$, compute $g_n(n)$ as follows:

$$g_n(n) = \sum_{j=0}^{n} f_j(n) = \sum_{j=0}^{n-1} f_j(n) + f_n(n).$$

Substituting the definition of $f_j(n)$, this leads to:

$$\sum_{j=0}^{n-1} f_j(n) = \sum_{j=0}^{n-1} 1 = n.$$

Since $f_n(n) = n^2$, it follows that:

$$g_n(n) = n + n^2.$$

Therefore, $O(g_n) = O(n + n^2) = O(n^2)$. Let this be the left-hand side (LHS).

On the other hand, consider $\sum_{j=0}^{n} O(f_j)$:

$$f_j(n) = 1 \text{ for all } j \neq n.$$

Hence, $O(f_j) = O(1)$. There are n terms where $f_j(n) = 1$, so:

$$\sum_{j=0}^{n} O(f_j) = \sum_{j=0}^{n} O(1) = (n+1)O(1).$$

This simplifies to $O(n+1) = O(n)$. Let this be the right-hand side (RHS).

Clearly, $LHS = O(n^2) \neq O(n) = RHS$.

Note that this analysis holds for $n > n_0$, as n_0 is fixed and n can grow arbitrarily large. Fixing n_0 ensures a concrete starting point, while allowing $n > n_0$ provides generality for the counterexample. The counterexample demonstrates that $O(g) = \sum_{j=0}^{n} O(f_j)$ does not necessarily hold in general.

Thus, the equivalence in the claim is disproved.

$\square$

Question #3

Claim: $f(n) = \lceil\sqrt{n}\rceil - \lfloor\sqrt{n} - 4\rfloor$ is asymptotically constant (i.e. $f(n) \in \Theta(1)$).

Proof.
By definition, if x and y are arbitrary real numbers, then

$$(x \leq \lceil x\rceil < x + 1)$$

and

$$(y - 1 < \lfloor y\rfloor \leq y).$$

Rewrite the second inequality as $-y \leq -\lfloor y\rfloor < -(y - 1)$.

By adding the two inequalities, it follows that $x - y \leq \lceil x\rceil - \lfloor y\rfloor < x + 1 - (y - 1) = x - y + 2$.

Let $x = \sqrt{n}$ and $y = \sqrt{n} - 4$, for arbitrary natural n.
Then, $\lceil x\rceil - \lfloor y\rfloor = \lceil\sqrt{n}\rceil - \lfloor\sqrt{n} - 4\rfloor = f(n)$. As well, $x - y = \cancel{\sqrt{n}} - (\cancel{\sqrt{n}} - 4) = 4$.

This means $x - y \leq \lceil x\rceil - \lfloor y\rfloor < x - y + 2 \implies 4 \leq f(n) < 4 + 2 \implies 4 \leq f(n) < 6$.

Let $n_0 = 0, c = 4, d = 6$. Let $g(n) = 1$.
Notice that $4 \leq f(n) < 6 \implies cg(n) \leq f(n) \leq dg(n)$, for all $n \geq n_0 = 0$ with $c = 4, d = 6$.

Therefore, $f(n) \in \Theta(g(n)) \implies f(n) \in \Theta(1)$. Indeed, $f(n)$ is asymptotically constant.

$\square$

Question #4

Claim: The recurrence, $T(n) = 3T(\frac{n}{3}) + n^2 - n$, can be solved using the master theorem, and there exists a function $g(n)$ such that $T \in \Theta(g(n))$.

Proof.
The recurrence $T(n) = 3T(\frac{n}{3}) + n^2 - n$ has the form $T(n) = aT(\frac{n}{b}) + f(n)$, where $a = 3$, $b = 3$, and $f(n) = n^2 - n$. Since $f(n) = n^2 - n$ asymptotically behaves like n^2, it follows that $f(n) \in \Theta(n^2)$, implying $k = 2$.

Master theorem applies to recurrences of this form, provided $a > 0$, $b > 1$, and $f(n)$ is non-negative for sufficiently large n. Here, $a = 3$, $b = 3$, and $f(n) = n^2 - n$ satisfies all these conditions since n^2 dominates n as $n \to \infty$.

Next, compute $\log_b a$:

$$\log_b a = \log_3 3 = 1.$$

Compare $\log_b a$ with k:

$$k = 2 > \log_3 3 = 1.$$

By the master theorem, when $k > \log_b a$, this leads to $T(n) \in \Theta(n^k)$.
Thus:

$$T(n) \in \Theta(n^2).$$

Therefore, there exists a function $g(n) = n^2$ such that $T(n) \in \Theta(g(n))$.

□

Question #5

<u>Claim:</u> Every regex without the Kleene star $*$ represents a finite language.

Proof.
Let r be a regular expression without the Kleene star $*$.
Define the predicate:

$$P(r) \coloneqq \mathcal{L}(r) \text{ is a finite language.}$$

Proceed using the principle of structural induction over $P(r)$ for all regular expressions r without the Kleene star $*$.

<u>Base Case:</u>
By the definition of regular expressions,

- $\mathcal{L}(\varnothing) = \varnothing$
- $\mathcal{L}(\epsilon) = \{\epsilon\}$
- $\mathcal{L}(a) = \{a\}$, where $a \in \Sigma$ is an arbitrary symbol

Clearly, all three languages as denoted above are finite.

Thus, $P(\varnothing), P(\epsilon), P(a)$ all hold.

<u>Induction Hypothesis:</u>
Assume that for some regular expressions r_1, r_2 without the Kleene star $*$, $P(r_1), P(r_2)$ hold.

This means the languages $\mathcal{L}(r_1), \mathcal{L}(r_2)$ are finite.

<u>Induction Step:</u>
Consider that every language without the Kleene star $*$ can be obtained by the union or concatenation of languages.

By the Induction Hypothesis, $\mathcal{L}(r_1)$ and $\mathcal{L}(r_2)$ are finite languages. Recall that languages are sets of elements, where the elements are symbols of some alphabet Σ.

By definition, $\mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$, which is finite as the union of finite sets (languages) is a finite set.
By definition, $\mathcal{L}(r_1 r_2) = \mathcal{L}(r_1)\frown\mathcal{L}(r_2)$, which is finite as the concatenation of two finite languages remains finite.

Conclusion:

By the principle of structural induction, every regular expression without the Kleene star $*$ represents a finite language.

$\square$

Question #6

Claim: The collection of regular languages is closed under complementation (i.e. if L is a regular langauge on an alphabet Σ, then $\Sigma^* \setminus L$ is also a regular language).

Proof.
Suppose L is a regular langauge over an arbitrary alphabet Σ.
Then, there exists a DFA $\mathcal{D} = (\Sigma, Q, \delta, s, F)$ that accepts L.

Consider the DFA $\mathcal{D}' = (\Sigma, Q, \delta, s, Q \setminus F)$, and proceed to show that D' accepts $\Sigma^* \setminus L$.

Let $w \in \Sigma^* \setminus L$.
Then, $\mathcal{D}$ rejects w, and $\delta(s, w) \notin F \implies \delta(s, w) \in Q \setminus F$. Clearly, $\mathcal{D}'$ accepts w.

Conversely, let $x \in \Sigma^*$ such that $\mathcal{D}'$ accepts x.
Then, $\delta(s, x) \in Q \setminus F \implies \delta(s, x) \notin F$. This means $\mathcal{D}$ rejects x, so $x \notin L$ but $x \in \Sigma^* \setminus L$.

Therefore, the DFA $\mathcal{D}'$ accepts $\Sigma^* \setminus L$, demonstrating that $\Sigma^* \setminus L$ is a regular language.

□

Question #7

Consider a program that takes an array of intervals `intervals` where `intervals[i]` $=$ $[\texttt{start}_i, \texttt{end}_i]$ and returns an optimal schedule:

```
def optimalschedule(intervals):
    sort intervals by the end times
    S = []
    f = -infty
    for i in [1, ..., n]:
        if start_i >= f:
            S.append([start_i, end_i])
            f = end_i
    return S
```

Definitions, Notes, and Examples:

- An **optimal schedule** is a subarray of `intervals` in which all the intervals are non-overlapping, and the subarray has the maximum possible size.
- $[1, 2]$ and $[2, 3]$ are non-overlapping.
- There may be multiple optimal schedules for an arbitrary array of intervals.
- All optimal schedules have the same size.
- In general, `intervals` $= [[\texttt{start}_1, \texttt{end}_1], \ldots, [\texttt{start}_n, \texttt{end}_n]]$ for some $n \in \mathbb{N}^+$ and $\texttt{start}_i, \texttt{end}_i \in \mathbb{R}^+$.
- The length of `intervals` is at least 1 (`intervals` is non-empty).
- If S is the subarray (in the program) at the j^{th} iteration and there exists some optimal schedule Opt such that $[\texttt{start}_i, \texttt{end}_i] \in Opt \iff [\texttt{start}_i, \texttt{end}_i] \in S$, then S is **looking good**.
- Let S be the subarray on the j^{th} iteration of the program. Define the predicate, $P(S) : S$ is **looking good**.

(a)

Claim: The `optimalschedule()` program terminates.

Proof.
Consider the loop variant $Var = n - i$ in the i^{th} iteration. Denote $\widetilde{Var}$ as the loop variant in the subsequent $((i+1)^{\text{th}})$ iteration.

Then, notice that $\widetilde{Var} = n - (i+1) < n - i = Var$, where $n, i \in \mathbb{N}$ and $i \leq n$; $(n - i) \in \mathbb{N}$.

Therefore, the loop variant decreases in every subsequent iteration. With n iterations and a step to return the result, the program terminates after $n + 1$ iterations.

□

(b)

Claim: S is **looking good** at the beginning of the first iteration.

Proof.
At the start of the first iteration, $S =$ `[]`, which is trivially a subset of every optimal schedule *Opt*. Indeed, S satisfies the definition of **looking good**.

Namely, there are no positive integers $i < j = 1$, making $P(n)$ trivially hold.

□

(c)

Claim: If S is **looking good** at the beginning of the first iteration, then the first iteration executes, and S is looking **looking good** at the beginning of the second iteration.

Proof.
Assume S is **looking good** at the beginning of the first iteration.

Since $i = 1$, the first iteration of the loop executes.
As well, the if-statement on *Line 6* of the program evaluates to `true` as `f` $=$ `-infty`.

Then, *Line 7* appends $[\texttt{start}_1, \texttt{end}_1]$ to S and *Line 8* updates $\texttt{f}$ to become $\texttt{end}_i$.

This completes the first iteration.

For the second iteration of the loop, consider an arbitrary optimal schedule Opt. Construct a new schedule Opt' (of the same size) obtained by replacing the first interval with $[\texttt{start}_1, \texttt{end}_1]$.

The `intervals` list is sorted, so Opt' is a schedule with no overlaps. Namely, $\texttt{end}_1 \leq \texttt{end}_a$, where $\texttt{end}_a$ is the first endpoint of Opt.

Opt' agrees with S on the first $j - 1 = 1$ interval. Thus, Opt' must be an optimal schedule.

Therefore, S is **looking good** at the start of the second iteration.

□

(d)

Claim: If S is **looking good** at the beginning of every iteration, including the iteration after the last that fails to be executed, then S is an optimal schedule.

Proof.
Assume S is **looking good** at the beginning of every iteration, including the iteration after the last that fails to be executed. Then, there exists an optimal schedule Opt such that for all $i < n + 1$ (n is the last iteration number),

$$[\texttt{start}_i, \texttt{end}_i] \in Opt \iff [\texttt{start}_i, \texttt{end}_i] \in S.$$

By definition, Opt is a maximal size subarray of intervals that are non-overlapping. Notice that S is constructed to be in the same way, through greedily selecting intervals based on the nearest start time (*Line 6* of the program). Thus, Opt and S are subarrays of the same size.

Since all intervals in *Opt* and S are in common, it follows that S is equivalent to *Opt*. This makes S an optimal schedule.

□

(e)

Claim: If S is **looking good** at the beginning of the j^{th} iteration implies S is **looking good** at the beginning of the $(j+1)^{\text{th}}$ iteration, then `optimalschedule()` is correct.

Proof.
Denote the loop invariant:

$$Q(k) : P(s) \text{ holds at the beginning of the } k^{\text{th}} \text{ iteration.}$$

To show that `optimalschedule()` is correct—that is, `optimalschedule()` returns an optimal schedule, show that $Q(k)$ is true for all $k \in \mathbb{N}$.

Proceed using the principle of simple induction on $Q(k)$ over $k \in \mathbb{N}$.

Base Case:
Let $i = 0$.
The claim that S is **looking good** at the beginning of the first iteration has been proved above.

Induction Hypothesis:
Assume for some $k \in \mathbb{N}$, $Q(k)$ is holds.

This means S is **looking good** at the beginning of the k^{th} iteration.

Induction Step:
By the induction hypothesis and the assumption, S is also **looking good** at the beginning of the $(j+1)^{\text{th}}$ iteration.

Induction Conclusion:

Therefore, $Q(k)$ holds for all $k \in \mathbb{N}$.

Next, the claim that the `optimalschedule()` program terminates has also already been proved. Namely, the program's loop terminates at the beginning of the $(n+1)^{\text{th}}$ iteration.

The last claim proved guarantees that if S is **looking good** at the beginning of every iteration, including the iteration after the last that fails to be executed, then S is an optimal schedule. This means S is constructed to be an optimal schedule after the program's loop terminates.

By finally returning S, the program satisfies its postcondition of returning an optimal schedule. Therefore, the program is correct.

$\square$

♢ Thank You! ♢

You've reached the end!

Thx for reading through, and I hope these solutions helped!

\(^_^)/

Wishing you lots of success and happiness in your studies!