CSC236 Exam Review

Notes from CSC236 Lecture 12

Alexander He Meng

Typed on November 27, 2024

Question #1

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**.

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.

$\square$

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.

$\square$

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 `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.

□

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.

□

<u>**Claim:**</u> 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 $k \in \mathbb{N}$.

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

<u>Induction Hypothesis:</u>
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.

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

<u>Induction Conclusion:</u>
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.

$\square$

Question #2

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

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.

$\square$

Steps to show that a DFA does not accept a language

1. Show that there exists $x, y \in \Sigma^*$ such that $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$.
2. Show that there exists $z \in \Sigma^*$ such that $xz \in L \iff yz \notin L$.
3. Clarify the contradiction that $\hat{\delta}(q_0, xz) = \hat{\delta}(q_0, yz) \implies \hat{\delta}(q_0, xz) \in L$.

Question #3

Prove that $L = \{a^{n^2} \mid n \in \mathbb{N}\}$ is **not** a regular language.

Proof.
Assume, for contradiction, that $L = \{a^{n^2} \mid n \in \mathbb{N}\}$ is regular. Then there exists a deterministic finite automata (DFA) $\mathcal{D} = \{Q, \Sigma, \delta, s, F\}$ that accepts L.
Let $|Q| = k$, where k is the number of states in $\mathcal{D}$.

Since L contains strings of the form a^{n^2}, choose $w = a^{j^2}$, where j is large enough such that $j^2 > k$. Clearly $w \in L$.

As the DFA processes w, which has $j^2 > k$ symbols, it must visit more states then there are in Q. By the Pigeonhole Principle, at least one state must repeat.
Namely, while processing w, there exist integers α and β such that:

$$\hat{\delta}(s, a^{\alpha}) = \hat{\delta}(s, a^{\alpha+\beta}),$$

where $\beta \geq 1$.
This means that after reading the first α symbols, the DFA enters some state q, and reading β additional symbols loops back to q.

Because $\mathcal{D}$ accepts $w = a^{j^2}$, it follows that:

$$\hat{\delta}(s, a^{j^2}) \in L.$$

Now consider the strings $a^{j^2+\beta}$, $a^{j^2+2\beta}$, and so on. Since the DFA loops at state q, adding multiples of β symbols to w does not change the final state.
Therefore:

$$\hat{\delta}(s, a^{j^2+\beta}) \in L \quad \text{and} \quad \hat{\delta}(s, a^{j^2+2\beta}) \in L$$

Thus, the DFA also accepts these strings.

Recall that with k states processing the first $k+1$ symbols of w must cause a state to repeat (the Pigeonhole Principle).

- Let α be the number of symbols leading up to the first occurrence of a repeated state q.
- Let β be the number of states causing the DFA to loop back to q.
- Let γ account for any remaining symbols to reach the end of $w = a^{j^2}$.

Thus, $j^2 = \alpha + \beta + \gamma$. Note that $\hat{\delta}(s, a^\alpha) = \hat{\delta}(q, a^\beta) = q$; denote this as *Corollary 1*.

It is now possible to show explicitly the strings which the DFA accepts.
Consider that:

$$\begin{aligned}
\hat{\delta}(s, a^{j^2+\beta}) &= \hat{\delta}(s, a^{(\alpha+\beta+\gamma)+\beta}) \\
&= \hat{\delta}(\hat{\delta}(s, a^\alpha), a^{\beta+\beta+\gamma}) \\
&= \hat{\delta}(q, a^{\beta+\beta+\gamma}), \text{ by } \textit{Corollary 1} \\
&= \hat{\delta}(\hat{\delta}(q, a^\beta), a^{\beta+\gamma}) \\
&= \hat{\delta}(q, a^{\beta+\gamma}), \text{ by } \textit{Corollary 1} \\
&= \hat{\delta}(\hat{\delta}(s, a^\alpha), a^{\beta+\gamma}), \text{ by } \textit{Corollary 1} \\
&= \hat{\delta}(s, a^{\alpha+\beta+\gamma}) \\
&= \hat{\delta}(s, a^{j^2})
\end{aligned}$$

This equivalence shows that $\mathcal{D}$ accepts $w = a^{j^2+\beta}$. By continually applying *Corollary 1* in the same argument, $\mathcal{D}$ also accepts $a^{j^2+2\beta}, a^{j^2+3\beta}$.

However, notice that one of $a^{j^2+2\beta}, a^{j^2+3\beta}$ is not a square and, thus, not a member of L. Yet, $\mathcal{D}$ accepts both strings. This is a contradiction.

Therefore, $L = \{a^{n^2} \mid n \in \mathbb{N}\}$ must not be regular.

$\square$

Here's an alternative proof using the ***Pumping Lemma****.*

Proof.
This proof demonstrates that $L = \{a^{n^2} \mid n \in \mathbb{N}\}$ is not a regular language using the pumping lemma for regular languages.

The pumping lemma states that if L is a regular language, then there exists a pumping length $p \geq 1$ such that for all $w \in L$ where $|w| \geq p$, w can be written as $w = xyz \mid_{x,y,z \in \Sigma^*}$ satisfying:

$$|xy| \leq p, \quad |y| \geq 1, \quad \text{and} \quad xy^i z \in L, \text{ for all } i \in \mathbb{N}.$$

Assume for contradiction that L is regular. Let $p \geq 1$ be the pumping length given by the pumping lemma.
Choose $w = a^{p^2} \in L$. Notice that $|w| = p^2 \geq p$, so the conditions of the pumping lemma hold.

By the pumping lemma, w can be split into $w = xyz$ such that:

- $|xy| \leq p$,
- $|y| \geq 1$,
- $xy^i z \in L$, for all $i \in \mathbb{N}$.

Since $|xy| \leq p$, the string xy consists of at most p a's. Still, y consists entirely of a's, so write $y = a^k$ for some $k \geq 1$.

Now, consider $i = 2$. The pumped string xy^2z is:

$$xy^2z = xa^{2k}z.$$

The length of xy^2z is:

$$|xy^2z| = |x| + 2|y| + |z| = (|x| + |y| + |z|) + |y| = p^2 + k.$$

To remain in L, the length $p^2 + k$ must be a perfect square. However, there are specific values leading to a contradiction. Let $p = 2$, so $p^2 = 4$. Then:

$$w = a^4 \quad \text{and} \quad y = a^1 \,(\text{since } |y| \geq 1).$$

Pumping y with $i = 2$, it follows that:

$$xy^2z = a^{4+1} = a^5.$$

The string a^5 is not in L, because 5 is not a perfect square.

This contradicts the pumping lemma, which requires $xy^iz \in L$ for all $i \geq 0$.

Therefore, L is not a regular language.

$\square$

Question #4

Prove that $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ is **not** a regular language.

Proof.
Seeking a contradiction, assume that L is a regular language. Then, by the definition of regular languages, there exists a deterministic finite automata (DFA) M with p states that accepts L.

Let $n \in \mathbb{N}$ such that $n > p$. Choose $w = 0^{n+300}1^{n+300}$.
Clearly, $w \in L$, so M accepts w. By the Pigeonhole Principle, since M has p states and processes w, some state in M must be repeated while reading the first $n + 300$ zeroes of w.

Let $x, y, z \in \Sigma^*$ be strings such that $w = xyz$, where:

- xy corresponds to the prefix of w up to the repeated state,
- $y \neq \varepsilon$ (i.e., y is the part of w causing the repetition),
- z is the remainder of w.

Thus, $w = 0^{n+300}1^{n+300}$, and $x = 0^a$, $y = 0^b$, $z = 0^c 1^{n+300}$, where $a+b+c = n+300$ and $b > 0$.

Now, consider the string $w' = xy^2z$, which is obtained by repeating y once. Then:

$$w' = 0^a 0^{2b} 0^c 1^{n+300} = 0^{n+300+b} 1^{n+300}.$$

Clearly, $w' \notin L$ because the number of zeroes exceeds the number of ones ($n + 300 + b > n + 300$). This contradicts the assumption that M accepts L, as M would also accept w', which is not in L.

Hence, L is not a regular language.

$\square$