These five problems are due Wednesday at the start of class. Hand in Problems A-D on paper; hand in Problem E electronically, as described below.

Problems A and B test whether you remember how to write basic proofs, as well as your understanding of basic notation. Depending on your background, these problems may or may not be easy. Let Σ be an alphabet of $m > 0$ symbols.

A. Using induction, prove for all $n \geq 0$ that $|\Sigma^n| = m^n$.

B. Let $\Sigma^{\leq n} = \bigcup_{k=0}^{n} \Sigma^k$. Find a closed formula for $\left|\Sigma^{\leq n}\right|$.

In the textbook and in class we showed that regular languages are closed under intersection; we then used that result to show that regular languages are closed under union. Now I want to go the other way.

C. Prove that regular languages are closed under union, without using the fact that they are closed under intersection. Then use closure under union to prove closure under intersection.

Problem D is about checking the validity of an HTML file. You don't need to know HTML. For simplicity, we allow only 12 HTML tags, and we ignore all of the text in the file other than the tags. The HTML file is regarded as a string over the alphabet

$$\Sigma = \{\texttt{html}, \texttt{/html}, \texttt{head}, \texttt{/head}, \texttt{title}, \texttt{/title}, \texttt{body}, \texttt{/body}, \texttt{ul}, \texttt{/ul}, \texttt{li}, \texttt{a}\}.$$

The file is valid if and only if it meets the following six criteria. (These criteria may not exactly match the real HTML standards, but they're good enough for our purposes.)

- The file must begin with `html` and end with `/html`. These symbols cannot occur anywhere else in the file.
- Immediately after `html` there must be a *header*, which begins with `head` and ends with `/head`. These two tags cannot occur elsewhere.
- Immediately after the header there must be a *body*, which begins with `body` and ends with `/body`. These two tags cannot occur elsewhere.
- The header must either contain no tags at all, or must contain a single *title*. A title begins with `title`, ends with `/title`, and contains no other tags. These two tags cannot occur elsewhere.
- The body may contain any number (0 or more) of *unordered lists*. An unordered list begins with `ul` and ends with `/ul`. Inside the unordered list there may be any number of `li` tags

and `a` tags (and nothing else), but if there are any tags inside an unordered list then the first tag must be `li`. The `li` tag cannot occur outside an unordered list.

- The body may also contain any number of `a` tags outside any unordered lists.

D. Draw a DFA that accepts valid HTML and rejects invalid HTML, as just defined. To keep your drawing clear, please adopt the following convention. If a state q has no outgoing transition arrow for a symbol a, then it is understood that the machine rejects its input when it is in state q and sees symbol a.

For our final problem we need to agree on a uniform way of describing DFAs in Python. Let us adopt the convention that the states of a DFA are numbered $q_0, q_1, \ldots, q_{n-1}$, where n is the number of states. Similarly, let's agree that the symbols are numbered $a_0, a_1, \ldots, a_{m-1}$. Then one can specify a DFA in Python by the following data.

1. A list `delta` of lists, such that each list in delta has the same length. Let n be the length of `delta` and m the length of each list in `delta`; these are the number of states and the number of symbols, respectively. This `delta` is the table for the DFA's transition function δ. Explicitly, $\delta(q_i, a_j)$ is the state corresponding to the integer `delta[i][j]`.
2. A number `s` belonging to the set $\{0, 1, \ldots, n-1\}$ to indicate the start state.
3. A list `F` of numbers from $\{0, 1, \ldots, n-1\}$, with no repeats, to indicate the final states.

We have not specified the set Q of states or the alphabet Σ, but we know how big each is from the structure of `delta`, and we know how to compute with them because the states and symbols are uniquely identified as numbers in $\{0, 1, \ldots, n-1\}$ and $\{0, 1, \ldots, m-1\}$, respectively. Thus the data `delta`, `s`, and `F` encode everything essential to the DFA.

The input string to a DFA on m symbols shall be represented in Python as a list of numbers from the set $\{0, 1, \ldots, m-1\}$. For example, the string $a_3a_3a_1a_0a_5$ shall be represented as `[3,3,1,0,5]`. Finally, our Python DFA will output `True` or `False` rather than `Yes` or `No`, because the former are more Pythonic.

E. In a file `dfa.py`, write a Python function `dfa` that simulates a given DFA on a given input. That is, `dfa` takes in four arguments — `delta`, `s`, `F`, and `input` — and outputs either `True` or `False`, according to whether the DFA described by `delta`, `s`, `F` would output `True` or `False` when given the input `input`. Include in your file at least two test cases. I suggest that you use examples from class, such as detecting whether the number of `0`s in a bitstring is divisible by 3 and detecting whether an integer given in base-2 is divisible by 5. Hand in your file on the COURSES file server, either manually (navigate to this course, find your hand-in folder, and drop it there) or via the `hsp` program.