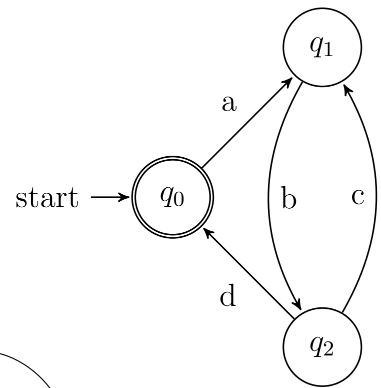
Question 1. Give a regular expression for the language accepted by the following NFA N:



Thoughts:
Best method is probably to look at strings that are in L(N) and hypothesize the answer.

We are not expected to memorize long algorithms like nfa-to-reg conversion.

Answer:
We explore what strings are accepted by N to deduce the language $L(N)$.

$$\begin{aligned}
\lambda &\in L(N) \\
abd &\in L(N) \\
abcbd &\in L(N) \\
abcbcbcbd &\in L(N) \\
abcbcbdabcbd &\in L(N)
\end{aligned}$$

- There is an obvious kleene star cycle, that begins with a and ends with d.
- There must be a b after the first a and if there is a c, it must be followed by b.

The regular expression $r_N = (ab(cb)^*d)^*$ satisfies $L(r_N) = L(N)$. An alternate answer can be $r_N = (a(bc)^*bd)^*$.

Question 2. Give a regular expression for the language $L = \{w \in \Sigma^* | \ |w| \text{ is even OR } w \text{ starts with an } a\}$.

Thoughts:
One way to start is to address "$|w|$ is even" and "w starts with an a" seperately. In this case the **OR** makes a merger of the two sub regular expressions easy.

Answer:

Part 1:

$|w|$ is even : $(aa + bb + ba + bb)^* = ((a + b)(a + b))^*$

Part 2:

w starts with a : $a(a + b)^*$

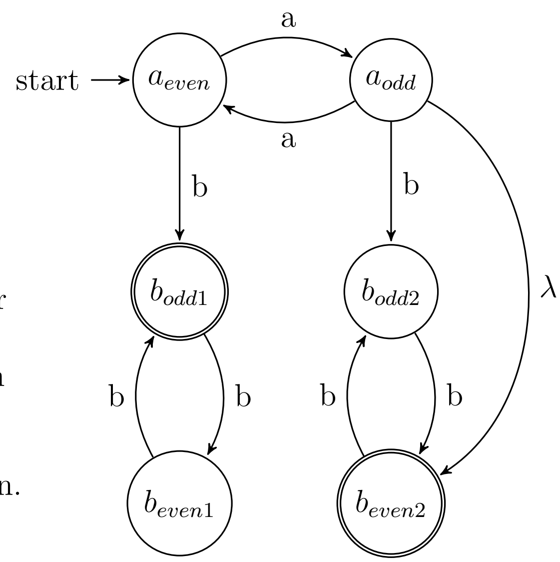
So $r_L = ((a + b)(a + b))^* + a(a + b)^*$ is a regular expression for the language L because **OR** can be represented by $+$.

Alternatively $r_L = (aa + ab + ba + bb)^* + a(a + b)^*$ is also a regular expression for L.

Question 3. Give a NFA accepting the regular language $L = \{a^i b^j \mid i, j \geq 0 \text{ and } |i - j| \text{ is odd}\}$.

Thoughts:
"$|i - j|$ is odd" feels similar to "$|w| \equiv k \mod t$" type of languages. In fact "$|i - j|$ is odd" is the same as $|i - j| \equiv 1 \mod 2$. We can try a similar approach. A good idea with every question is to see if there are similarities or patterns relative to other problems you have solved .

NFA N**:**



Answer:
To keep track of the difference between the number of a's and b's being odd ($|\#a - \#b| \equiv 1 \mod 2$) we first keep track of the parity of the a's and then accept on the opposite parity for the b's.

$|$ even a's $-$ even b's $|$ is even.
$|$ odd a's $-$ odd b's $|$ is even.
$|$ even a's $-$ odd b's $|$ is odd.✓
$|$ odd a's $-$ even b's $|$ is odd.✓

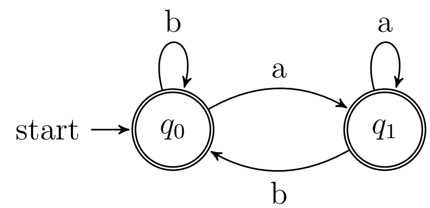
We have to be able to accept on an odd number of a's and zero b's so there is an extra lambda transition added. The NFA $N = \{Q, \Sigma, \delta, a_{even}, F\}$ where $\Sigma = \{a, b\}$ described by the diagram above accepts the language L.

Question 5. Give a minimal DFA that has exactly two final states.

Thoughts:
Probably looking for knowledge of:
-minimal
-DFA and
-final states .

Natural First Try

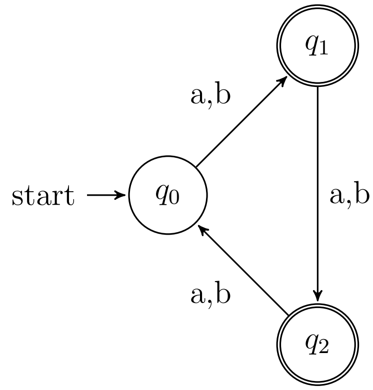
M:



M is a legal DFA over the alphabet $\Sigma = \{a, b\}$, there are no λ transitions, and every state has a path for all possible inputs. Now is minimality addressed? Well this DFA accepts all strings over Σ. The minimal DFA for the language $L((a + b)^*)$ has one state and thus M has one indistinguishable state and is not minimal. Now you might think what if we made the alphabet $\Sigma = \{a, b, c\}$? To keep M a legal DFA, c would have to be accounted for as input when the automaton is in state q_0 or q_1. Again we create a DFA that accepts all strings since every state is an accepting state, and therefor not minimal.

Answer:
The next natural try is a DFA with three states, keeping it simple we will define it over $\Sigma = \{a, b\}$. There are several that work, but to avoid the risk of the DFA not being minimal, we can create a DFA that considers strings whose lengths are multiples of three. Since we need two accepting states, we can create a DFA for the language $L = \{w \in \Sigma \mid |w| \not\equiv 0 \mod 3\}$.

D:



There are no indistinguishable states, each state keeps track of the remainder of the length of the string when divided by three. Thus D is a proper minimal DFA, and has two final states.