Definition

To run on this simulator a nondeterministic finite automata must be defined as a json object with the following keys (left) and values (right):

1. Q: Array os states,
2. sigma: Array of characters composing the alphabet,
3. delta: A json object, representing the transition function as follows:
    - Every key in delta represents an state S in Q. Every value is a json object representing the possible transitions from the state S, as follows:
        - Every key in delta[S] represents a character "char" that must be read from input, or "epsilon" meaning the empty string;
        - The value is an Array of the states that are reached being on state S and reading "char" or "epsilon"
4. q0: String representing the start state
5. F: Array of Strings, representing the accept states

Computation

A **formal definition** of computation on this simulator is very similar to the formal definition of computation on a nondeterministic finite automata, being:

Let N be an NFA defined as described above and w be a string over the alphabet "sigma". We say that N accepts w if we can write w as $w = y1y2y3...ym$ where each yi is a member of "sigma" or "epsilon" and a sequence of states $r0,r1,r2...rm$ exists in Q with three conditions:

1. $r0 = q0$
2. $ri+1 \in$ "delta"["ri"]["$yi+1$"], for $i=0,...,m-1$, and
3. $rm \in$ "F"

An **not-so-formal** definition would be:

- The computation start with the call of the `compute` method, with the given string.
- The `compute` method validates the string, to check if it is valid according to the alphabet on the definition, and then call `accept` passing the start state defined in "q0" and the string as arguments.
- `accept` is a recursive method that receives an state and a string as arguments and:
    - Returns **true** if the string is empty and the state is in the set of final states;
    - Grabs the first char of the string and then the possible next states, according to the definition.

        - The possible next states are the states in "delta"["state"]["char"] appended to "delta"["next state"]["epsilon"] for every next state in "delta"["state"]["char"].
    - Call `itself` again with each possible next state and the string from the second char to the end as parameters.
    - Returns **true** if any of these calls also return **true**.
    - Return **false** otherwise.
- The value returned by `accept` is the result of the computation.