How to Simulate a Finite Automaton

Objective. The goal is:

- to ask the user what automaton they want to simulate, and then
- to simulate how this automaton will check whether a given word is accepted or not.

What we want to extract from the user. From the user, we want to extract all the information about the automaton:

- the number of states $N$; the corresponding states will be denoted $q_0, \ldots, q_{N-1}$, so that $q_0$ is the starting state; in programming terms, this number is clearly an integer;
- the number of symbols $M$; the corresponding symbols will be denoted $s_0, \ldots, s_{M-1}$; in programming terms, this number is also clearly an integer;
- we need an information about where we should go if in state $q_n$ we see a symbol $s_m$; in the formal description of the automaton, this information is described by a 2-D table, so its natural representation is a 2-D array; we will call it $\text{state}[n][m]$;
- we also need to describe which of $N$ states are final and which are not; a natural way to represent this information is by a boolean array $\text{final}[n]$.

Comment. For example, if the alphabet consists of $M = 3$ letters $a$, $b$, and $c$, then we can, e.g., denote $a$ as $s_0$, $b$ as $s_1$, and $c$ as $s_2$.

How can we ask the user for this information. First, we ask for the values $N$ and $M$. If you have designed a reader called $\text{reader}$, we can do it the usual way:

```java
int N;
System.out.println("Please enter the number of states.");
N = reader.nextInt();
```

Then, we similarly ask for $M$. Now, we can declare the two desired arrays:

```java
int[][] state = new int[N][M];
```
Similarly, we can declare the boolean array \( \text{final}[n] \).

Once the arrays are defined, we can ask the user to provide the needed information. For example, to fill in the array \( \text{state}[n][m] \), we can use the double loop:

```java
for(int n = 0; n < N; n++)
    for(int m = 0; m < M; m++)
    {
        System.out.println("What state do you want to move to " +
        "if you are in state " + n +
        "and you read the symbol " + m + ")
        state[n][m] = reader.nextInt();
    }
```

Similarly, we can fill in the array \( \text{final}[n] \). This way, we will have all the needed information about the automaton.

**How to represent the input to the resulting automaton.** Since each symbol is represented by an integer, a word – i.e., a sequence of symbols – is simply a sequence of integers. A natural way to represent this sequence is by an array \( \text{word}[i] \).

**Example.** For example, in the alphabet of three letters \( a, b, \) and \( c \) in which we denoted \( s_0 = a, s_1 = b, \) and \( s_2 = c \), the word \( cab \) will be represented as an array in which \( \text{word}[0] = 2, \text{word}[1] = 0, \) and \( \text{word}[2] = 1 \). In general, \( \text{word}[i] = j \) means that on the \( i \)-th place in this word, we have a symbol \( s_j \). In the above example:

- \( \text{word}[0] = 2 \) means that on the 0-th place in the word is the symbol \( s_2 = c \);
- \( \text{word}[1] = 0 \) means that on the 1-st place in the word is the symbol \( s_0 = a \);
- \( \text{word}[2] = 1 \) means that on the 2-nd place in the word is the symbol \( s_1 = b \).

Together, this array represents the word \( cab \).

**How to represent the input to the resulting automaton (cont-d).** We should ask the user for this array. The same automaton should work for several words, so we want to have a loop. We do not know how many words the user will input, so this should be a while loop, controlled, e.g., by a boolean variable \( \text{done} \).

```java
boolean done = false;
while(!done){
    <ask for the word, process the word>
    System.out.println("Do you want to continue: Y/N");
    <read the answer>
    if(answer == 'N'){done = true;}
}
```
How do we simulate the automaton? We need a variable `currentState` whose initial value is 0 – which corresponds to the starting state. As we read the `i`-th symbol `word[i]`, then from the state `currentState` we move to a state determined by the array `state`. Once we are done, if we are in the final state, we accept the word, else we reject it:

```java
int currentState = 0;
for(int i = 0; i < word.length; i++){
    currentState = state[currentState][word[i]];
    if(final[currentState])
        System.out.println("The word is accepted.");
    else
        System.out.println("The word is rejected."))
```

General comment. These comments are just to help. Feel free to simulate automata differently.