https://github.com/Matei-Daria/LFTC

A `FiniteAutomata` class is defined to solve this problem.

The class has the following fields, denoting the elements of the 5-tuple definition of FA: `states`, `alphabet`, `initialState`, `finalStates`, `transitionFunctions`.

Additionally, there are fields `filePath`, used to read from file, and `isDFA`, a boolean that indicates if the FA represented by the current instance of the class is deterministic.

From all fields, of note is the fact that `transitionFunctions` is represented as a `Map` with key of type `TransitionFunction` and value of type `Set<String>`.

- `TransitionFunction` is another class defined in this program, which is a simple record with parameters (`String state, String alphabetSymbol`). This represents the transition function's input, meaning the state and the alphabet symbol used in the transition.
- `Set<String>` is used as a representation for the values of the `Map`, meaning that each `TransitionFunction` can have, as function value, a set of values. A transition function can have multiple values, in which case the FA is non-deterministic.

The following functions are defined in this class:

- `private void readFromFile()`
    - this function reads from the file with the name `filePath` (in this implementation `FA.in`) the FA; this file is structured in the following way: comma-separated states, newline, comma-separated alphabet symbols, newline, initial state, newline, comma-separated final states, newline, an unknown number of 3-tuples, comma-separated transition function parameters (state, alphabet symbol, state), each on a new line (see EBNF form at the end of the documentation)
    - each of the read elements are assigned to the respective class
    - for the `transitionFunctions`, it is checked if the newly read function's input parameters, those that are contained in the class `TransitionFunction` which represents the map keys, are already present in the map in order to add them to the map accordingly
    - this function is called in the constructor of the class
- `private boolean verifyDFA()`
    - this function verifies if the FA is deterministic
    - it uses functional programming to check that all `transitionFunctions` elements have the size of the set, representing the values of the functions, smaller or equal to 1
    - this function is called in the constructor of the class and its value assigned to `isDFA`
- `public boolean verifySequence(String sequence)`
    - this function verifies if a sequence is accepted by the FA, only for a DFA
    - the sequence is a user-input concatenation of only letters that represent alphabet symbols
    - the sequence is split into a list of `String`, each list element representing an alphabet symbol, and the list is parsed

    - starting from the initial state, we try to reach one of the final states, using the alphabet symbols from the list; if at some point we cannot continue or if we finish parsing through the list and have not reached one of the final states, we return false; if we finish parsing the list and are at one of the final states, we return true
- getters and a function which prints the elements of the `transitionFunctions` nicely

In `Main`, for the FA, we have a simple menu with the following options:

```
Get the following information about the FA:
1. States
2. Alphabet
3. Transitions
4. Initial state
5. Final states
6. Verify if it is DFA
7. Enter a sequence and see if it's valid
8. Close the application
```

EBNF form of FA.in

letter = 'a'|'b'|...|'z'|'A'|'B'|...|'Z';

digit = '0'|'1'|...|'9';

alphabetSymbol = letter | digit;

alphabet = alphabetSymbol,{alphabetSymbol,};

state = letter;

states = state,{state,};

initialState = state;

finalStates = state,{state,};

transition = state,alphabetSymbol,state;

transitions = transition'\n'{transition'\n'};

FA = states '\n' alphabet '\n' initialState '\n' finalStates '\n' transitions;