CSE 422 - THEORY OF COMPUTATION
GEBZE TECHNICAL UNIVERSITY

String Pattern Matching with Finite Automata

Esra Eryılmaz - 171044046
Tuğçe Karagöz - 171044099
Ayşe Gül Demirbilek - 1801042088

18 May 2022

1 WHAT IS FINITE AUTOMATA?

Finite automata is an abstract computing device. It is a mathematical model of a system with discrete inputs, outputs, states and a set of transitions from state to state.

Finite State Automata or Finite State Machine is the simplest model used in Automata. Finite state automata accepts regular language. In this, the term finite means it has a limited number of possible states, and number of alphabets in the strings are finite.

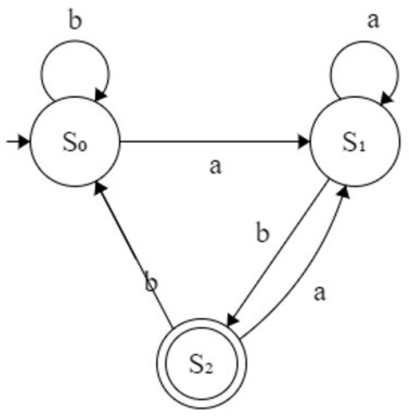


Figure 1.1: Finite Automata Example

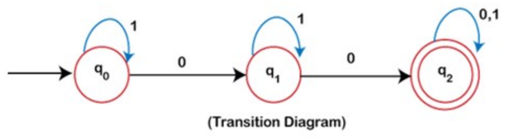
1.1 Tuples of Finite Automata

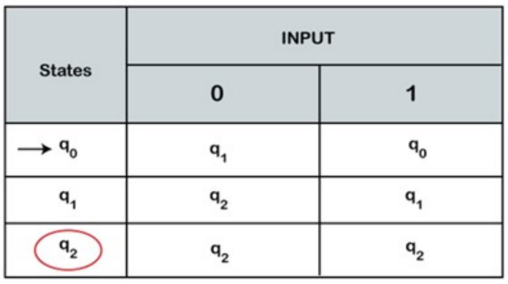
A finite automaton M is a 5-tuple (Q, q_0, A, Σ, δ) where

- Q is a finite set of states,
- $q_0 \in Q$ is the start state (initial state),
- $A \subseteq Q$ is a notable set of accepting states,
- Σ is a finite input alphabet,
- δ is the transition function that gives the next state for a given current state and input.

1.2 Representation of Finite Automata

We can represent Finite automata in two ways:
1) Transition Diagram : The transition diagram is also called a transition graph; it is represented by a diagraph.
2) Transition Table : It is the tabular representation of the behavior of the transition function that takes two arguments, the first is a state, and the other is input, and it returns a value, which is the new state of the automata.

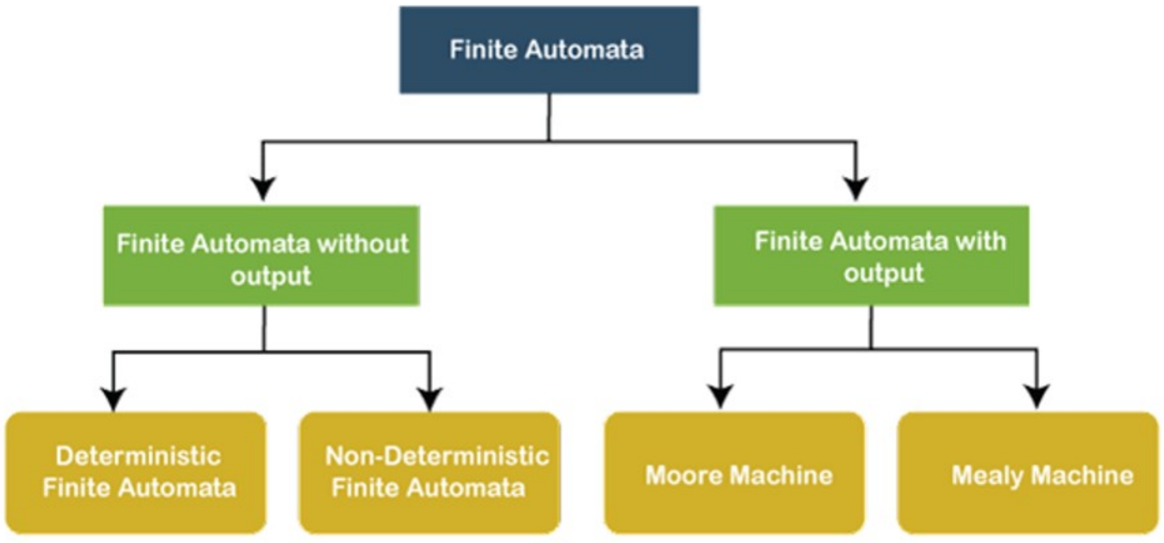




States	INPUT	
	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_1
q_2	q_2	q_2

Figure 1.2: Transition Diagram & Transition Table

1.3 Types of Finite Automata



Finite Automata
Finite Automata without output
Finite Automata with output
Deterministic Finite Automata
Non-Deterministic Finite Automata
Moore Machine
Mealy Machine

Figure 1.3: Types of Finite Automata

2 What is Pattern Matching?

Pattern : A collection of strings described in some formal language.
Pattern Matching : In computer science, pattern matching is the act of checking a given sequence of tokens for the presence of the constituents of some pattern.

2.1 Pattern Matching Algorithms

1. Naive Pattern Searching
2. KMP Algorithm
3. Rabin-Karp Algorithm
4. **Finite Automata**
5. Boyer Moore Algorithm
6. Aho-Corasick Algorithm
7. Suffix Array
8. Kasai's Algorithm
9. Z algorithm (Linear time pattern searching Algorithm)
10. Manacher's Algorithm
11. Ukkonen's Suffix Tree Construction

2.2 Applications of Pattern Matching

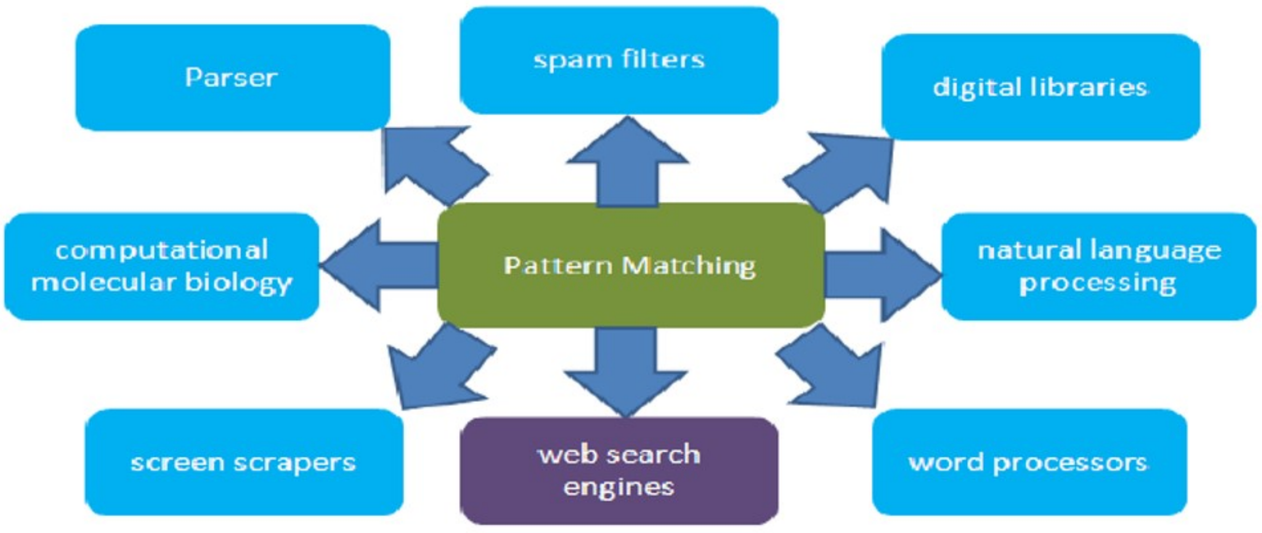


Figure 2.1: Applications of Pattern Matching

3 How to match string using finite automata? (with example)

A finite automaton accepts strings in a specific language. It begins in state S_0 and reads characters one at a time from the input string. It makes transitions based on these characters, and if when it reaches the end of the tape it is in one of the accept states, that string is accepted by the language.

Example :
Suppose we want to "grep nano". Rather than just starting to write states down, let's think about what we want them to mean. At each step, we want to store in the current state the information we need about the string seen so far. Say the string seen so far is "...stuvwxy", then we need to know two things:

1) Have we already matched the string we're looking for ("nano")?
2) If not, could we possibly be in the middle of a match?

If we're in the middle of a match, we need to know how much of "nano" we've already seen. Also, depending on the characters we haven't seen yet, there may be more than one match that we could be in the middle of - - for instance if we've just seen "...nan", then we have different matches if the next characters are "o..." or if they're "ano...". But let's be optimistic, and only remember the longest partial match.

So we want our states to be partial matches to the pattern. The possible partial matches to "nano" are "", "n", "na", "nan", or (the complete match) "nano" itself. In other words, they're just the prefixes of the string. **In general, if the pattern has m characters, we need m+1 states**; here m=4 and there are five states.

The start and accept states are obvious: they are just the 0- and m-character prefixes. So the only thing we need to decide is what the transition table should look like. If we've just seen "...nan", and see another character "x", what state should we go to? Clearly, if x is the next character in the match (here "o"), we should go to the next longer prefix (here "nano"). And clearly, once we've seen a complete match, we just stay in that state. But suppose we see a different character, such as "a"? That means that the string so far looks like "...nana". The longest partial match we could be in is just "na". So from state "nan", we should draw an arrow labeled "a" to state "na". Note that "na" is a prefix of "nano" (so it's a state) and a suffix of "nana" (so it's a partial match consistent with what we've just seen).

In general the transition from state+character to state is the longest string that's simultanously a prefix of the original pattern and a suffix of the state+character we've just seen. This is enough to tell us what all the transitions should be. If we're looking for pattern "nano", the transition table would be :

States	n	a	n	o
S0	S1	S0	S0	S0
S1	S1	S2	S0	S0
S2	S3	S0	S0	S0
S3	S1	S2	S4	S0
S4	S0	S0	S0	S0

Figure 3.1: Transition Table

For instance the entry in row "nan" and column n says that the largest string that's simultaneously a prefix of "nano" and a suffix of "nan"+n="nann" is simply "n". We can also represent this as a state diagram :

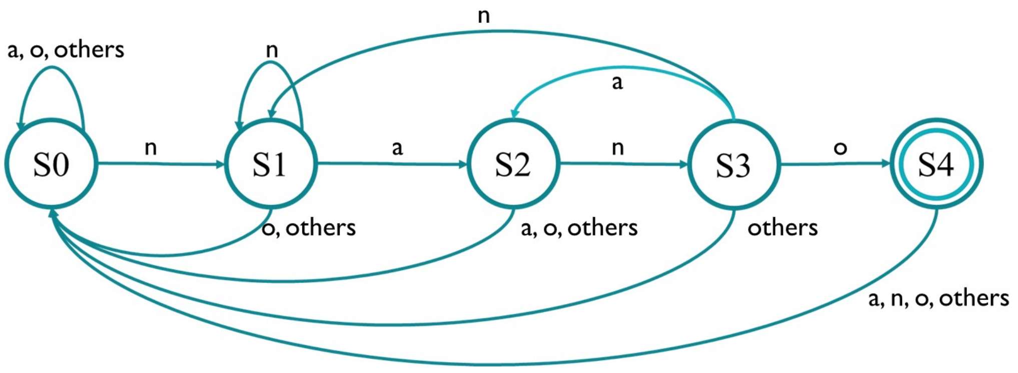


Figure 3.2: Finite Automata

4 User Interface

All implementation is done with Python.

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. The interface was created using tkinter (Tk). This interface allows to entering string and pattern. With pressing FIND button, it gives pattern index position.

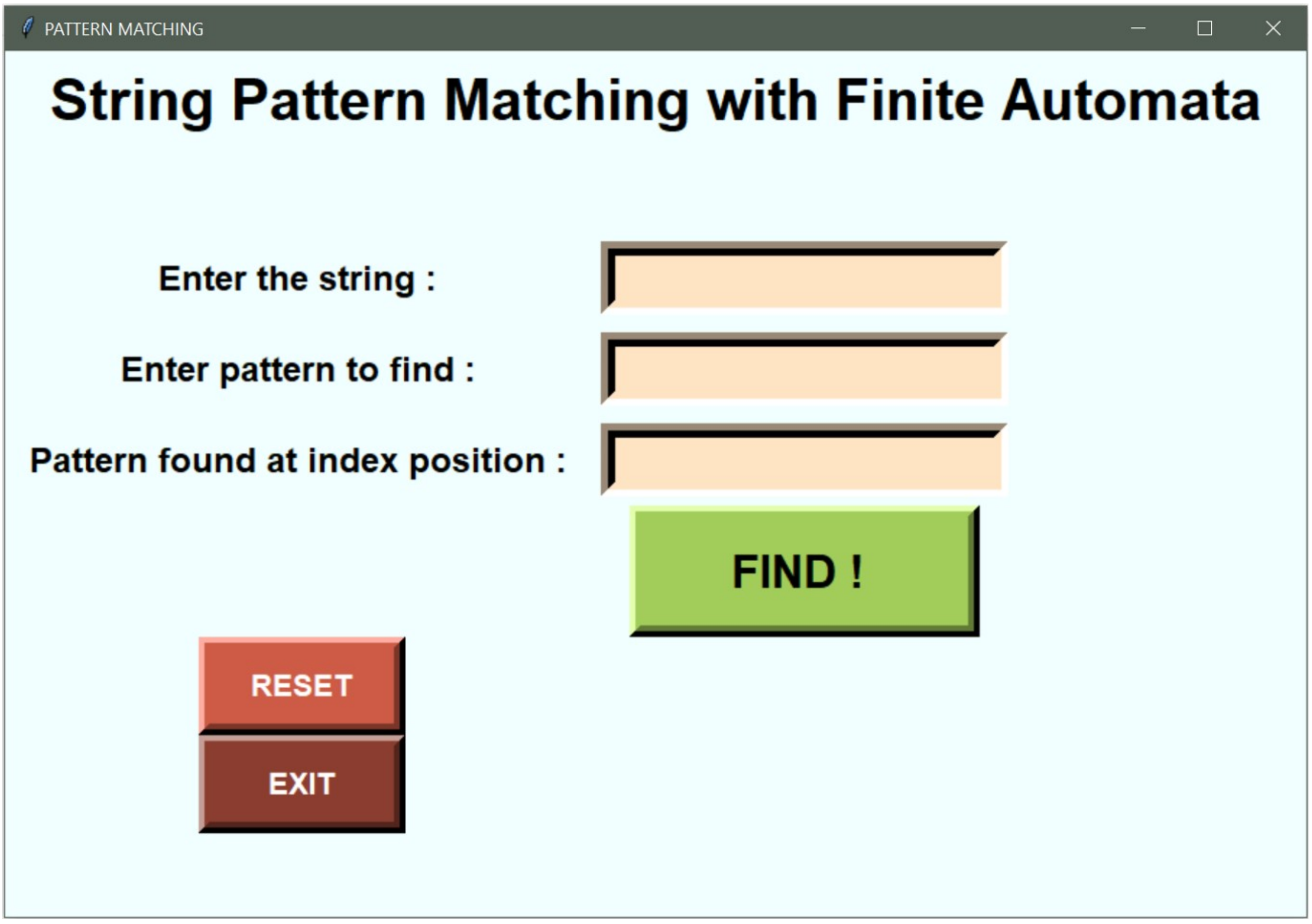
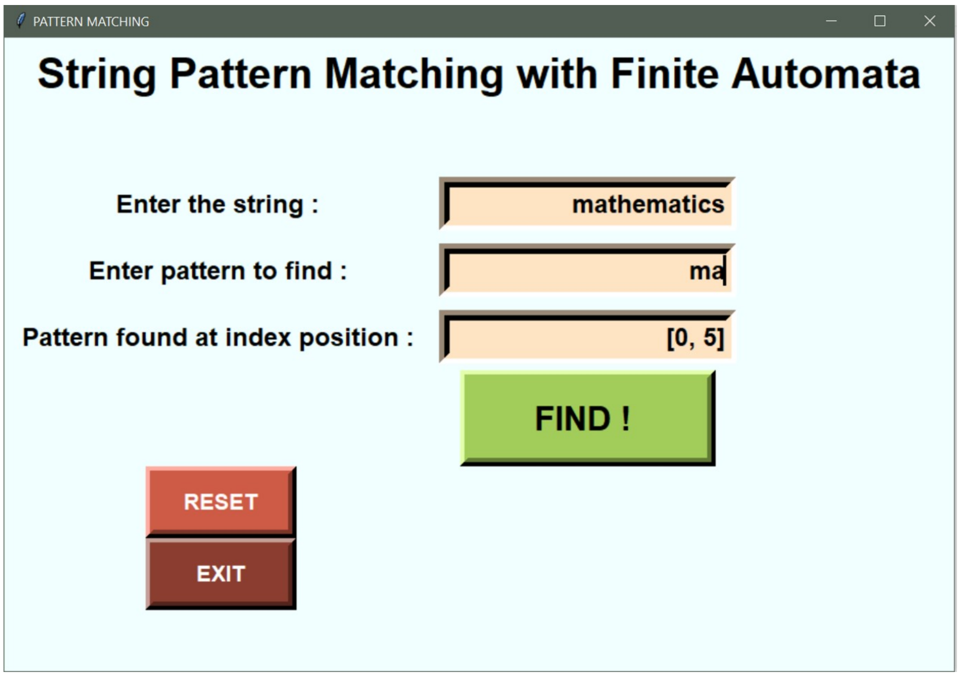

PATTERN MATCHING
String Pattern Matching with Finite Automata
Enter the string :
Enter pattern to find :
Pattern found at index position :
FIND !
RESET
EXIT

Figure 4.1: User Interface

5 Output Examples


PATTERN MATCHING

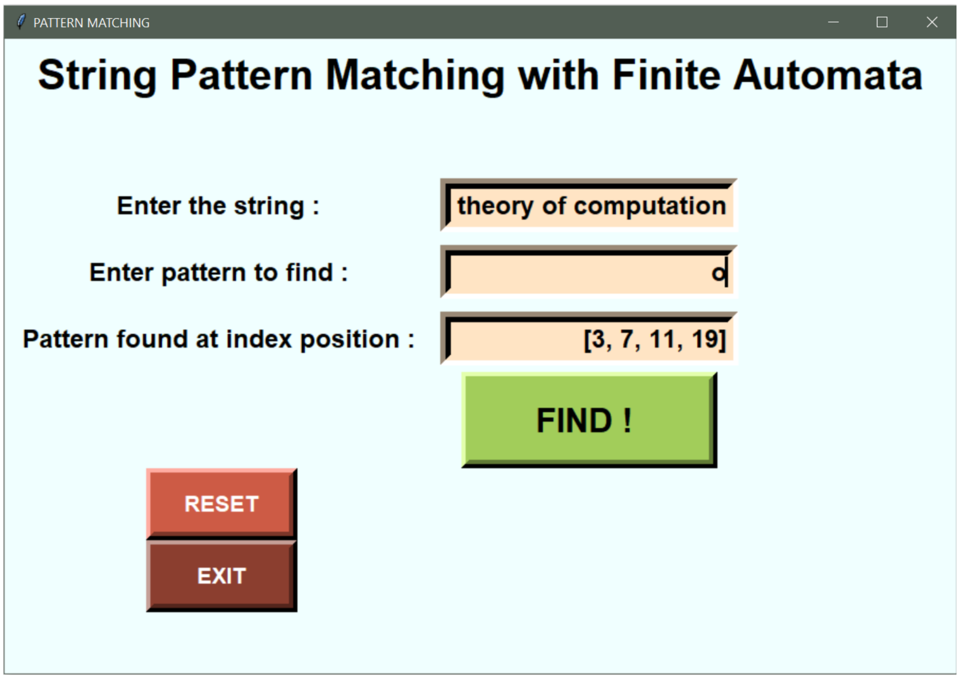
String Pattern Matching with Finite Automata

Enter the string : mathematics

Enter pattern to find : ma

Pattern found at index position : [0, 5]

Figure 5.1: Output Example 1


PATTERN MATCHING

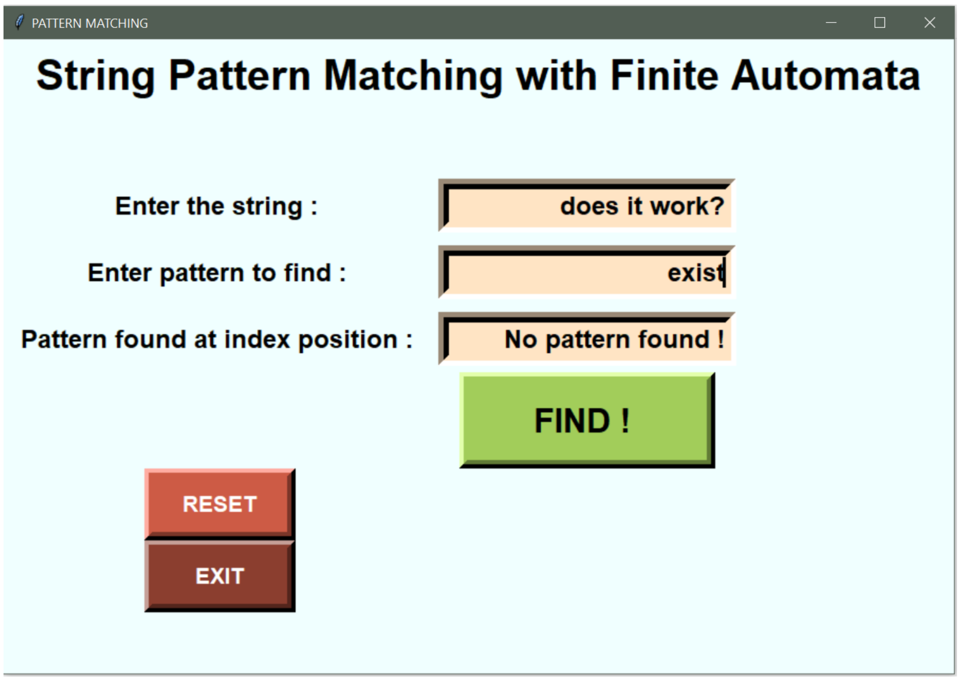
String Pattern Matching with Finite Automata

Enter the string : theory of computation

Enter pattern to find : o

Pattern found at index position : [3, 7, 11, 19]

Figure 5.2: Output Example 2


PATTERN MATCHING

String Pattern Matching with Finite Automata

Enter the string : does it work?

Enter pattern to find : exist

Pattern found at index position : No pattern found !

Figure 5.3: Output Example 3

References

[1] Introduction to Theory of Computation Michael Sipser, Thomson Course Technology, 2006.

[2] Javatpoint : https://www.javatpoint.com/automata-tutorial

[3] Donald Bren School of Information and Computer Sciences (ICS) : https://www.ics.uci.edu/~eppstein/161/960222.html

[4] https://www.youtube.com/watch?v=XgldL2gVLIo

[5] GeeksforGeeks : https://www.geeksforgeeks.org/algorithms-gq/pattern-searching/

[6] Wikipedia : String-searching algorithm

[7] http://homepages.math.uic.edu/~leon/cs-mcs401-r07/handouts/finite-automata.pdf