**AA\_LAB\_05\_Assignment**

**CE\_054**

**Aim :-** String matching using Finite Automata Algorithm.

1. Implement a String matching Algorithm using Finite Automata algorithm.

Code :-

# -\*- coding: utf-8 -\*-

"""

Created on Fri Aug 14 09:28:52 2020

@author: DHRUV

"""

import numpy as np

def unique(pattern):

    x = np.array(list(pattern))

    return (np.unique(x))

string = input("Enter the string :- ")

pattern = input("Enter the pattern :- ")

smallest\_char, len\_pattern = min(pattern), len(pattern)

states, acceptstate = len\_pattern + 1, len\_pattern

distinct\_ele = len(unique(pattern))

table = []

input\_, \_\_, curr\_state = [], [0, 1, 2, 3, 4], 0

for \_ in range(states):     # Take input Table!

    input\_ = []

    print(f"Current state : {\_}\tInput : {\_\_}\tNext state :")

    input\_ = list(map(int, input().split()))

    table.append(input\_)

print("\n")

for row in table:           # print Automata table!

    print(\*row)

if (len(string) == 0):

    print("string not valid!")

else:

    for \_\_\_ in range(len(string)):

        dist = ord(string[\_\_\_]) - ord(smallest\_char)

        curr\_state = table[curr\_state][dist]

        if (curr\_state == acceptstate):

            print(f"pattern found at index of : {\_\_\_ - len\_pattern + 1}")

Output :-

1. Output :-

String :- WWXYXYYWXYXYZYZWYXWZ

Pattern :- WXYXYZYZ

Automata table for 1st output :

W X Y Z dummy

0 1 0 0 0 0

W 1 2 0 0 0

X 1 0 3 0 0

Y 1 4 0 0 0

X 1 0 5 0 0

Y 1 0 0 6 0

Z 1 0 7 0 0

Y 1 0 0 8 0

Z 1 0 0 0 0

1. Output :-

String :- ACBCBCBAABCABCBCCBAABC

Pattern :- ABCBCCBA

Automata table for 2nd output :

A B C dummy

0 1 0 0 0

A 1 2 0 0

B 1 0 3 0

C 1 4 0 0

B 1 0 5 0

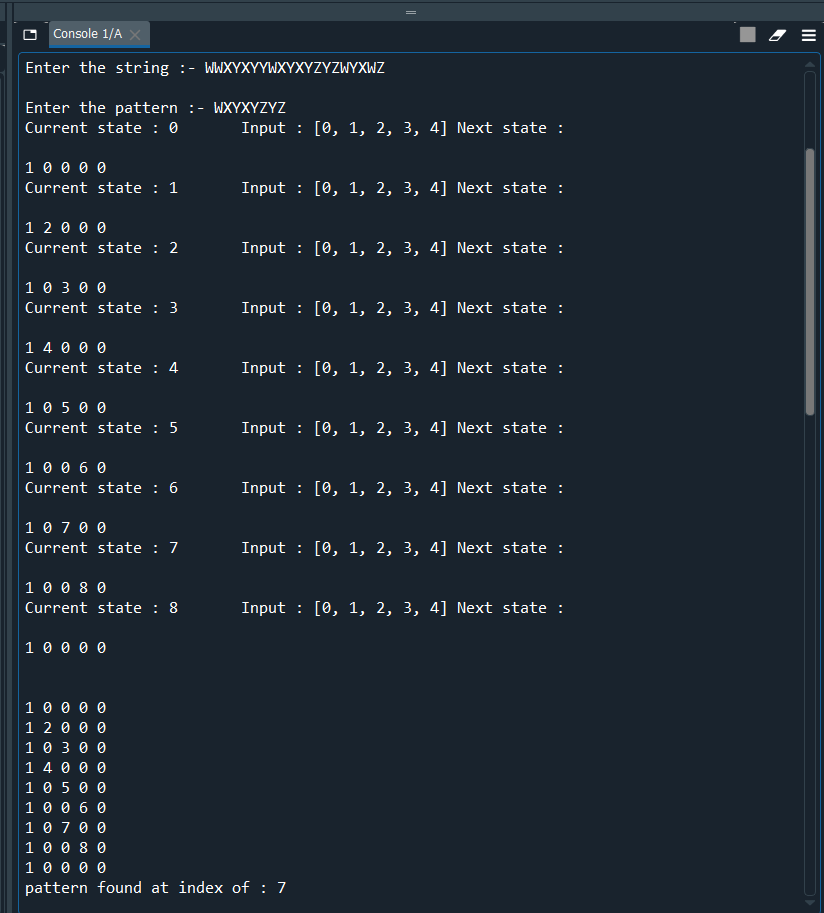
C 1 0 6 0

C 1 7 0 0

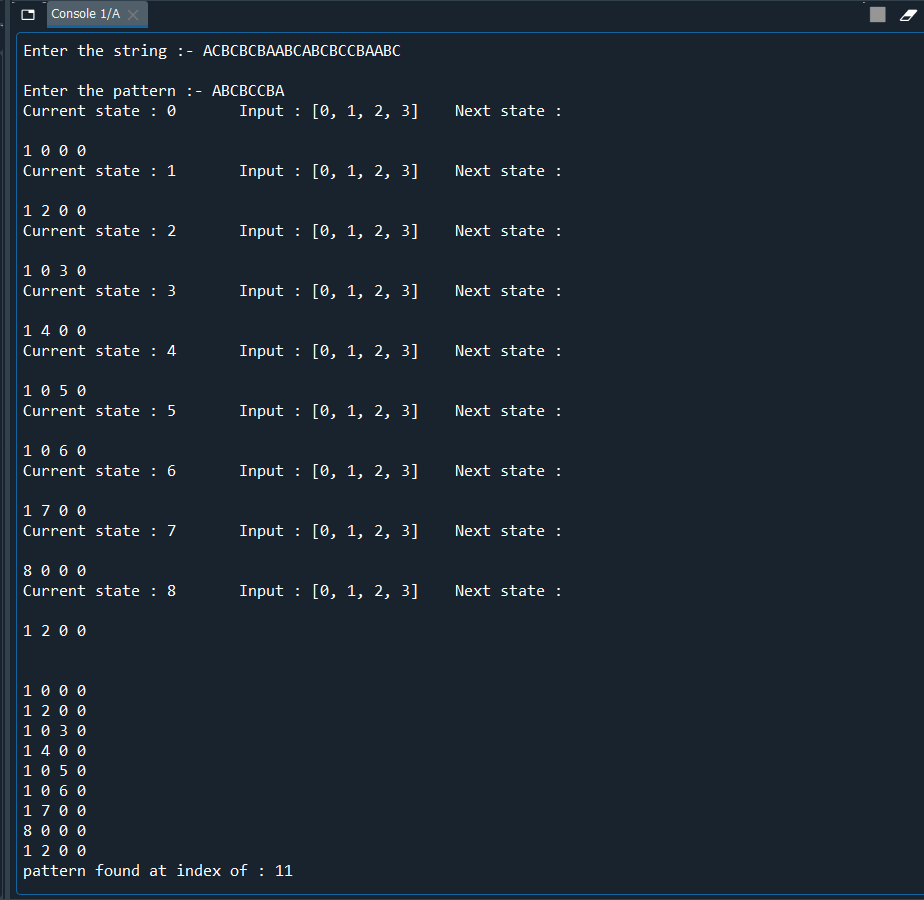
B 8 0 0 0

A 1 2 0 0

* 1st Output :-



* 2nd Output :-



* Comparison between All String Matching Algorithm.
* We Implemented string pattern matching algorithm by directly giving the finite automaton table as input. Hence, Time taken to take input will be Big O((length of pattern) \* (no of distinct chars in string)). To checking the occurrence of the pattern will take time of Big O(length of string). So the time complexity of the implemented algorithm will be Big O((length of pattern) \* (number of distinct chars in string) + (length of string) ). If we Don’t give Finite Automata table directly as a input and is that Mechanism build by Algorithm itself then time complexity of this algorithm will be increase.
* Time complexity Comparison for All string matching Algorithm :-
* M = Length of String
* N = Length of Pattern
* D = Number of Distinct char in string.

|  |  |
| --- | --- |
| **Algorithm** | **Time Complexity in Big O function** |
| Naïve | M \* N |
| Boyer – Moore | (N + D) \* M |
| Rabin Karp | M + N |
| Knuth-Morris-Pratt | M + N |
| Finite Automata | (N \* D) + M |