AA_LAB_05_Assignment

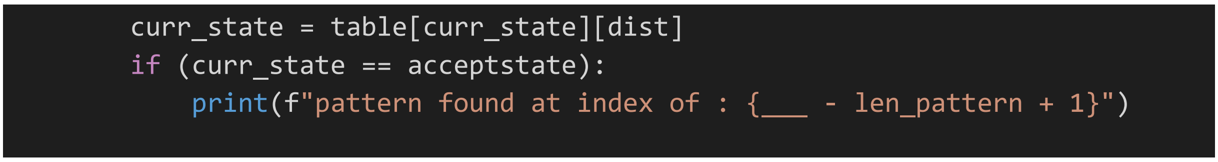
CE_054

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

2. Output :-

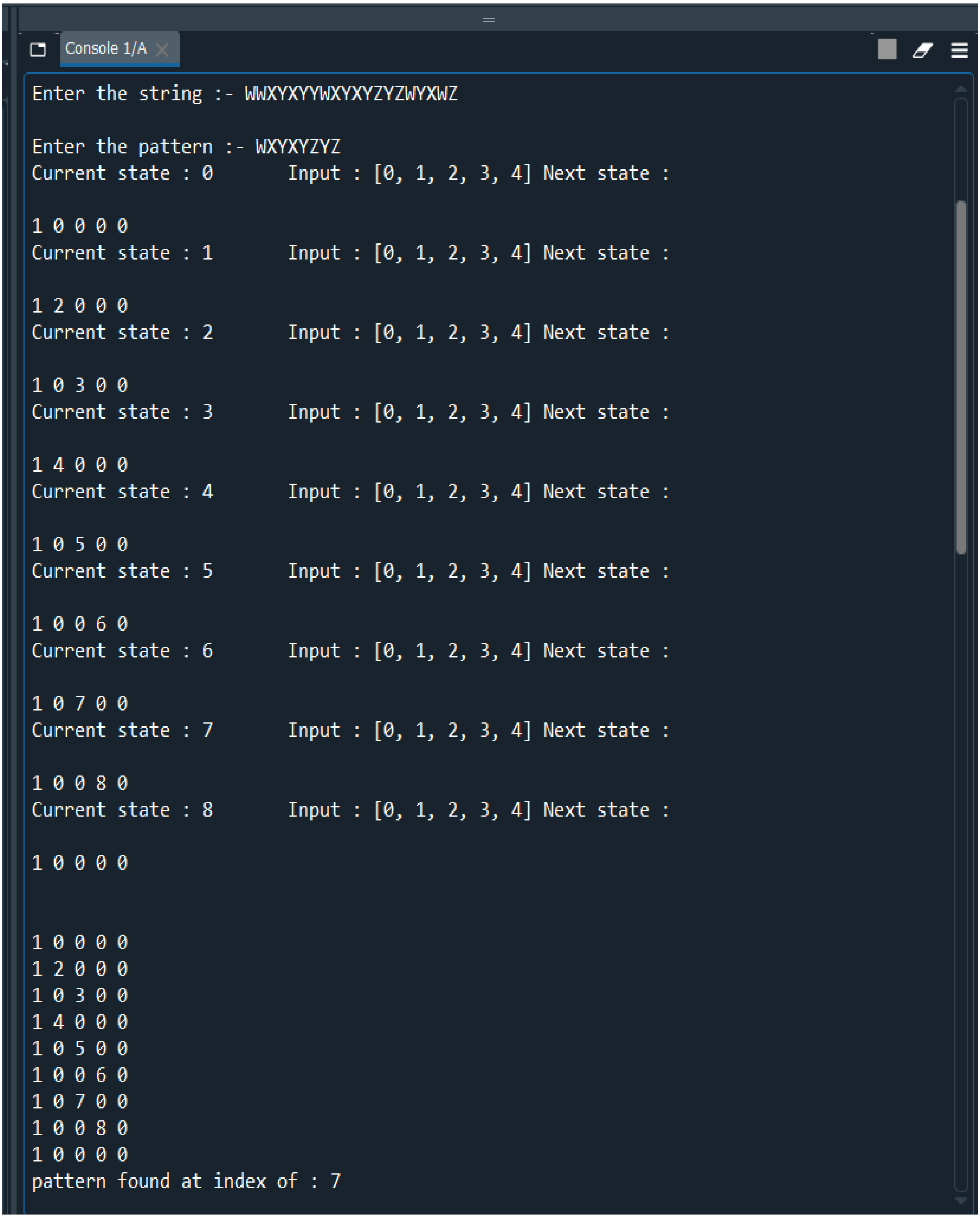
String :- ACBCBCBAABCABCBCCBAABC

Pattern :- ABCBCCBA

Automata table for 2nd output :

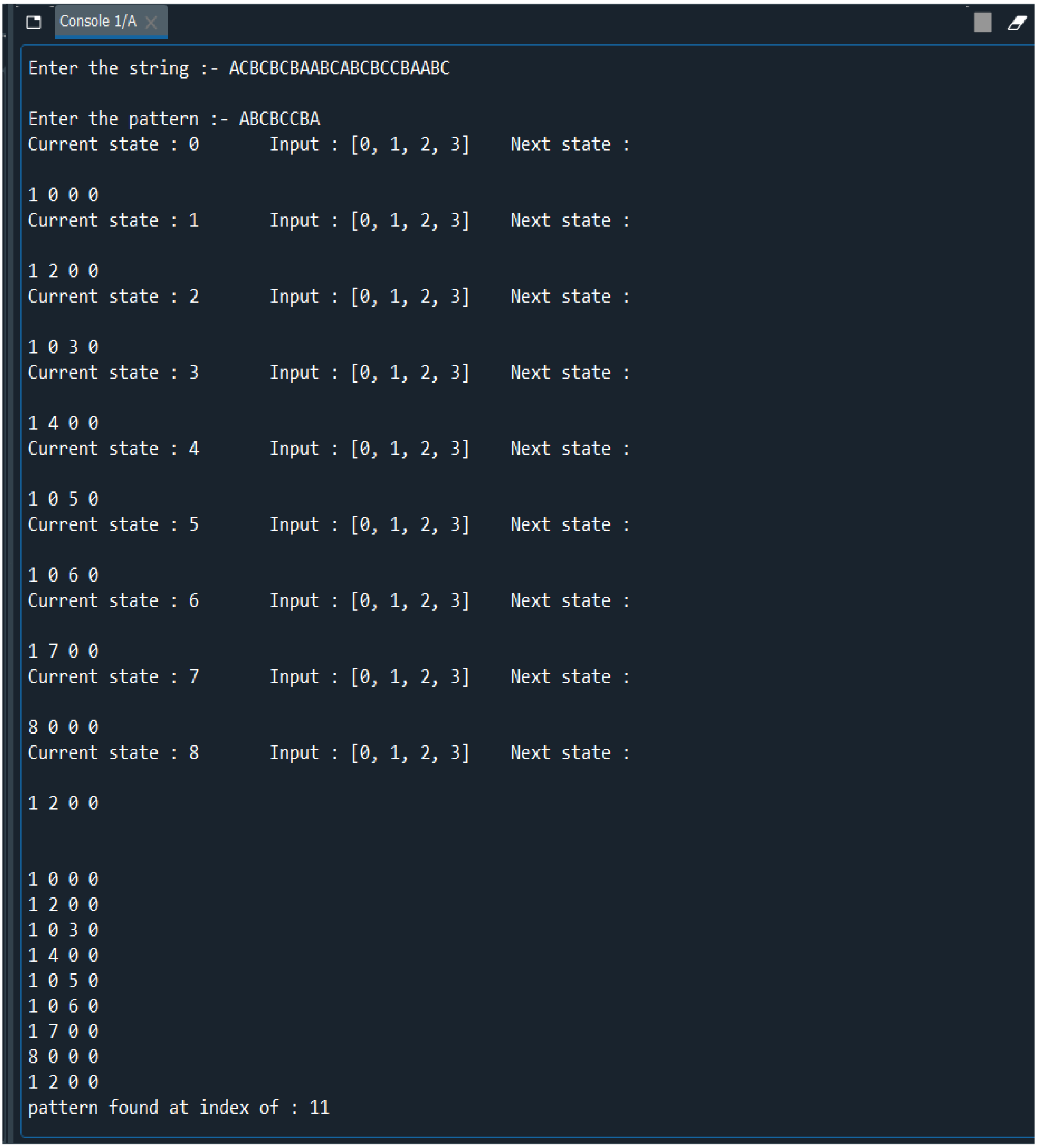
	A	B	C	dummy
0	1	0	0	0
A	1	2	0	0
B	1	0	3	0
C	1	4	0	0
B	1	0	5	0
C	1	0	6	0
C	1	7	0	0
B	8	0	0	0
A	1	2	0	0

- 1st Output :-



```
Enter the string :- WWXYXYYWXYXYZYZWYXWZ

Enter the pattern :- WXYXYZYZ
Current state : 0       Input : [0, 1, 2, 3, 4] Next state :

1 0 0 0 0
Current state : 1       Input : [0, 1, 2, 3, 4] Next state :

1 2 0 0 0
Current state : 2       Input : [0, 1, 2, 3, 4] Next state :

1 0 3 0 0
Current state : 3       Input : [0, 1, 2, 3, 4] Next state :

1 4 0 0 0
Current state : 4       Input : [0, 1, 2, 3, 4] Next state :

1 0 5 0 0
Current state : 5       Input : [0, 1, 2, 3, 4] Next state :

1 0 0 6 0
Current state : 6       Input : [0, 1, 2, 3, 4] Next state :

1 0 7 0 0
Current state : 7       Input : [0, 1, 2, 3, 4] Next state :

1 0 0 8 0
Current state : 8       Input : [0, 1, 2, 3, 4] Next state :

1 0 0 0 0


1 0 0 0 0
1 2 0 0 0
1 0 3 0 0
1 4 0 0 0
1 0 5 0 0
1 0 0 6 0
1 0 7 0 0
1 0 0 8 0
1 0 0 0 0
pattern found at index of : 7
```

- 2nd Output :-



```
Enter the string :- ACBCBCBAABCABCBCCBAABC

Enter the pattern :- ABCBCCBA
Current state : 0       Input : [0, 1, 2, 3]    Next state :

1 0 0 0
Current state : 1       Input : [0, 1, 2, 3]    Next state :

1 2 0 0
Current state : 2       Input : [0, 1, 2, 3]    Next state :

1 0 3 0
Current state : 3       Input : [0, 1, 2, 3]    Next state :

1 4 0 0
Current state : 4       Input : [0, 1, 2, 3]    Next state :

1 0 5 0
Current state : 5       Input : [0, 1, 2, 3]    Next state :

1 0 6 0
Current state : 6       Input : [0, 1, 2, 3]    Next state :

1 7 0 0
Current state : 7       Input : [0, 1, 2, 3]    Next state :

8 0 0 0
Current state : 8       Input : [0, 1, 2, 3]    Next state :

1 2 0 0


1 0 0 0
1 2 0 0
1 0 3 0
1 4 0 0
1 0 5 0
1 0 6 0
1 7 0 0
8 0 0 0
1 2 0 0
pattern found at index of : 11
```

- 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