horizontal line

**Assignment 05 | Advance Algorithms**

**CE-092**

Assignment submission for Advance Algorithms subject week 5.

nevilparmar24@gmail.com

**─**

**Task 1:**

Implement finite automata based string matching algorithm.

Code:

1. **CPP Implementation**

/\*

*\* @Author: nevil*

*\* @Date: 2020-08-07 15:51:28*

*\* @Last Modified by: nevil*

*\* @Last Modified time: 2020-08-19 02:25:36*

\*/

/\*

*finite automata based string matching algorithm.*

*It creates the table and checks for the matching string in the text on its own.*

\*/

#*include* <bits/stdc++.h>

using namespace std;

#*define* *NO\_OF\_CHARS* 256

int *getNextState*(string pat, int M, int state, int x)

{

*if* (state < M && x == pat*[*state*]*)

*return* state + 1;

int ns, i;

*for* (ns = state; ns > 0; ns--)

{

*if* (pat*[*ns - 1*]* == x)

{

*for* (i = 0; i < ns - 1; i++)

*if* (pat*[*i*]* != pat*[*state - ns + 1 + i*]*)

*break*;

*if* (i == ns - 1)

*return* ns;

}

}

*return* 0;

}

void *computeTF*(string pat, int M, int TF[][*NO\_OF\_CHARS*])

{

int state, x;

*for* (state = 0; state <= M; ++state)

*for* (x = 0; x < NO\_OF\_CHARS; ++x)

TF[state][x] = *getNextState*(pat, M, state, x);

}

void *search*(string pat, string txt)

{

int M = pat.*size*();

int N = txt.*size*();

int TF[M + 1][*NO\_OF\_CHARS*];

*computeTF*(pat, M, TF);

bool found = false;

int i, state = 0;

*for* (i = 0; i < N; i++)

{

state = TF[state][txt*[*i*]*];

*if* (state == M)

{

found = true;

cout *<<* " Pattern found at index " *<<* i - M + 1 *<<* *endl*;

}

}

*if*(!found)

cout *<<* "Pattern Not Found" *<<* *endl*;

}

int *main*()

{

string txt, pat;

cout *<<* "Enter Text : ";

cin *>>* txt;

cout *<<* "Enter your pattern to be searched :";

cin *>>* pat;

*search*(pat, txt);

*return* 0;

}

1. **Python 3 Implementation**

# *-\*- coding: utf-8 -\*-*

# *@Author: nevil*

# *@Date: 2020-08-07 15:49:25*

# *@Last Modified by: nevil*

# *@Last Modified time: 2020-08-07 15:49:40*

# *Python program for Finite Automata*

# *Pattern searching Algorithm*

numChars = 256

def *nextForm*(PATTERN, M, state, x):

*if* state < M and x == ord(PATTERN[state]):

*return* state+1

i=0

*for* nosaj *in* range(state,0,-1):

*if* ord(PATTERN[nosaj-1]) == x:

*while*(i<nosaj-1):

*if* PATTERN[i] != PATTERN[state-nosaj+1+i]:

*break*

i+=1

*if* i == nosaj-1:

*return* nosaj

*return* 0

def *MakeAutomata*(PATTERN, M):

*global* numChars

TF = [[0 *for* i *in* range(numChars)]\

*for* \_ *in* range(M+1)]

*for* state *in* range(M+1):

*for* x *in* range(numChars):

z = nextForm(PATTERN, M, state, x)

TF[state][x] = z

*return* TF

def *search*(PATTERN, TEXT):

strout = ""

*global* numChars

M = len(PATTERN)

N = len(TEXT)

TF = MakeAutomata(PATTERN, M)

state=0

*for* i *in* range(N):

state = TF[state][ord(TEXT[i])]

*if* state == M:

strout += str(i-M+1)

strout += " "

*return* strout

def *main*():

*with* open("input.txt", "r") *as* ins:

array = []

*for* line *in* ins:

array.append(line.strip('\n'))

file = open("output.txt","w")

TEXT = array[1]

PATTERN = array[0]

reverseTXT = TEXT[::-1]

file.write(str(search(PATTERN, TEXT)))

file.write(str(search(PATTERN, reverseTXT)))

file.write('\n')

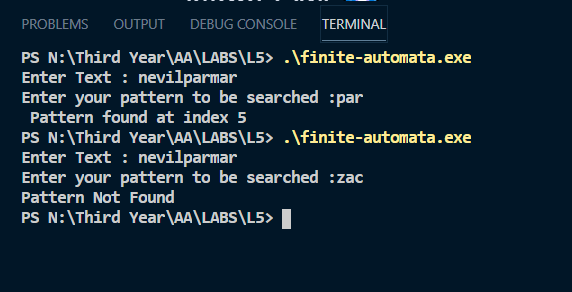
ins.close()

file.close()

*if* \_\_name\_\_ == '\_\_main\_\_':

main()

Output:



**Complexity :**

In a Finite Automata based string matching algorithm, the pattern need to be pre-processed and a state table (2-D array) is to be generated for the finite automata and then each character in the text is mapped with the corresponding state in the state table and the next state is determined. Finally, if we reach the accepting state in the state table, it means we have found the required pattern.

Hence the searching time complexity for the average case will be O(n).

**Comparison :**

|  |  |
| --- | --- |
| **Algorithms** | **Time complexity (Average case)** |
| Naive string matching | O(m\*n) |
| Horsepool’s algorithm | O(n) |
| Rabin Karp’s algorithm | O(m+n) |
| Finite Automata based algorithm | O(n) |

Nevil Parmar

CE-092

https://nevilparmar.me