Searching for Patterns | Set 5 (Finite Automata)

Given a text *txt[0..n-1]* and a pattern *pat[0..m-1]*, write a function *search(char pat[], char txt[])* that prints all occurrences of *pat[]* in *txt[]*. You may assume that n > m.

Examples:
1) Input:

```
  txt[] =  "THIS IS A TEST TEXT"
  pat[] = "TEST"
```

Output:

Pattern found at index 10

2) Input:

```
txt[] =  "AABAACAADAABAAABAA"
pat[] = "AABA"
```

Output:

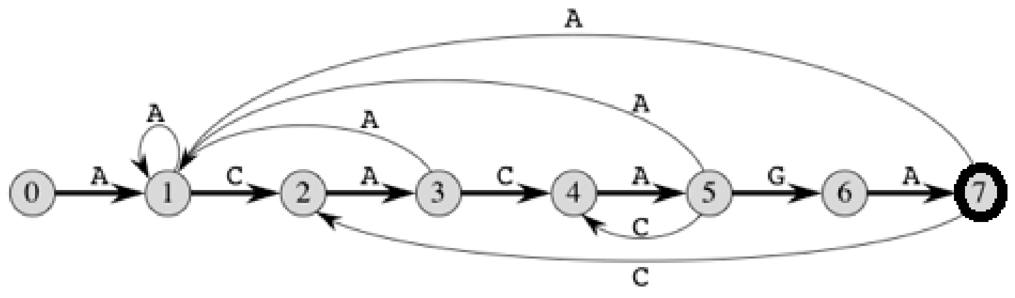
```
 Pattern found at index 0
 Pattern found at index 9
 Pattern found at index 13
```

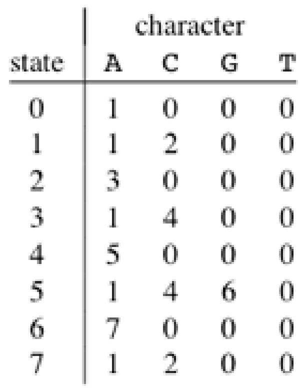
Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

We have discussed the following algorithms in the previous posts:

Naive Algorithm
KMP Algorithm
Rabin Karp Algorithm

In this post, we will discuss Finite Automata (FA) based pattern searching algorithm. In FA based algorithm, we preprocess the pattern and build a 2D array that represents a Finite Automata. Construction of the FA is the main tricky part of this algorithm. Once the FA is built, the searching is simple. In search, we simply need to start from the first state of the automata and first character of the text. At every step, we consider next character of text, look for the next state in the built FA and move to new state. If we reach final state, then pattern is found in text. Time complexity of the search prcess is O(n).
Before we discuss FA construction, let us take a look at the following FA for pattern ACACAGA.





state	A	C	G	T
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	7	0	0	0
7	1	2	0	0

The abvoe diagrams represent graphical and tabular representations of pattern ACACAGA.

Number of states in FA will be M+1 where M is length of the pattern. The main thing to construct FA is to get the next state from the current state for every possible character. Given a character x and a state k, we can get the next state by considering the string “pat[0..k-1]x” which is basically concatenation of pattern characters pat[0], pat[1] … pat[k-1] and the character x. The idea is to get length of the longest prefix of the given pattern such that the prefix is also suffix of “pat[0..k-1]x”. The value of length gives us the next state. For example, let us see how to get the next state from current state 5 and character ‘C’ in the above diagram. We need to consider the string, “pat[0..5]C” which is “ACACAC”. The lenght of the longest prefix of the pattern such that the prefix is suffix of “ACACAC”is 4 (“ACAC”). So the next state (from state 5) is 4 for character ‘C’.

In the following code, computeTF() constructs the FA. The time complexity of the computeTF() is O(m^3*NO_OF_CHARS) where m is length of the pattern and NO_OF_CHARS is size of alphabet (total number of possible characters in pattern and text). The implementation tries all possible prefixes starting from the longest possible that can be a suffix of “pat[0..k-1]x”. There are better implementations to construct FA in O(m*NO_OF_CHARS) (Hint: we can use something like lps array construction in KMP algorithm). We have covered the better implementation in our next post on pattern searching.

```
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

int getNextState(char *pat, int M, int state, int x)
{
    // If the character c is same as next character in pattern,
    // then simply increment state
    if (state < M && x == pat[state])
        return state+1;

    int ns, i;  // ns stores the result which is next state

    // ns finally contains the longest prefix which is also suffix
    // in "pat[0..state-1]c"

    // Start from the largest possible value and stop when you find
    // a prefix which is also suffix
    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;
}

/* This function builds the TF table which represents Finite Automata for a
```

```
   given pattern  */
void computeTF(char *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);
}

/* Prints all occurrences of pat in txt */
void search(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    int TF[M+1][NO_OF_CHARS];

    computeTF(pat, M, TF);

    // Process txt over FA.
    int i, state=0;
    for (i = 0; i < N; i++)
    {
       state = TF[state][txt[i]];
       if (state == M)
       {
         printf ("\n patterb found at index %d", i-M+1);
       }
    }
}

// Driver program to test above function
int main()
{
   char *txt = "AABAACAADAABAAABAA";
   char *pat = "AABA";
   search(pat, txt);
   return 0;
}
```

Output:

```
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
```

References:
Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Recursively print all sentences that can be formed from list of word lists
- Check if a given sequence of moves for a robot is circular or not
- Find the longest substring with k unique characters in a given string
- Function to find Number of customers who could not get a computer
- Find maximum depth of nested parenthesis in a string
- Find all distinct palindromic sub-strings of a given string
- Find if a given string can be represented from a substring by iterating the substring "n" times
- Suffix Tree Application 6 – Longest Palindromic Substring

Tags: Pattern Searching

Tweet 1 g+1 1

Writing code in comment? Please use **ideone.com** and share the link here.

23 Comments **GeeksforGeeks** 1 **Login**

Recommend Share Sort by Newest

Join the discussion…

lord_vader • 17 days ago

Here's a more concise code:

```
int string_match_auto(char* T,char* P)

{

int m ,n ,i ,j;

char ch;

m = strlen(T);

n = strlen(P);

j = -1;

for(i = 0;i < m;i++)

{

ch = *(T + i);
```

see more

Reply • Share ›

David Lu • 4 months ago

Help me, I cannot figure out this expression: pat[i] != pat[state-ns+1+i]. Why (state-ns+1+i)?

• Reply • Share ›

Anurag Singh David Lu • 4 months ago

Here we are looking of there is a prefix which is also a proper suffix in pattern. You need to visualize it how prefix and suffix match will happen.

e.g. In pattern ACACAC, longest prefix which is a proper suffix is ACAC. Here length of pattern is 6, so longest possible proper suffix length would be 5. So we start from that length and match for prefix and suffix.
For length 5,
1st (from prefix) character is matched with 2nd (from suffix) character.
2nd (from prefix) character is matched with 3rd (from suffix) character.
....
....
5th (from prefix) character is matched with 6th (from suffix) character.

If all characters match, we got longest prefix length as 5. If it fails, we check if there is a prefix of length 4 (reduce by 1) which is a proper suffix. Here
1st (from prefix) character is matched with 3rd (from suffix) character.
2nd (from prefix) character is matched with 4th (from suffix) character.
... and so on..

see more

• Reply • Share ›

Fazili • 5 months ago

void matcher(String txt,String pattern,int offset){

int count=0;

for(int i=0;i<pattern.length();i++) if="" (txt.charat(i)="=pattern.charAt(i))" {="" count++;="" if="" (count="=pattern.length()){" system.out.println(offset);="" if="" (txt.substring(pattern.length()).length()="">=pattern.length())

matcher(txt.substring(pattern.length()),pattern,offset+pattern.length());

return;

}

continue;

}

else

see more

• Reply • Share ›

Fazili • 5 months ago

JAVA
==============================

public class Test{

public static void main(String[] args){

MatcherClass a= new MatcherClass ();

a.matcher("geeksforgeeks","ks",0);
}

class MatcherClass{

void matcher(String txt,String pattern,int offset){

int count=0;

for(int i=0;i<pattern.length();i++) if="" (txt.charat(i)="=pattern.charAt(i))" {="" count++;="" if="" (count="=pattern.length()){" system.out.println(offset);="" if=""

see more

• Reply • Share ›

rahul • 10 months ago

```
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;
}
```

Please explain this piece of code

• Reply • Share ›

Guest • a year ago

Plz someone explain imp of line --> if(pat[ns-1]==x) in above code in function getNextstate()

```
// Start from the largest possible value and stop when you find
// a prefix which is also suffix
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;
}
```

• Reply • Share ›

alien • 2 years ago

Can someone please explain what is the difference between FA string matching and KMP String matching.

• Reply • Share ›

dag alien • 2 years ago

in case of KMP, there is no overhead of creating FA and storing FA, rather we have to construct lps which takes O(m) time only

• Reply • Share ›

Iqbal Hawre • 2 years ago

/* I think this is Simplest PROGRAM */

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int state=0;
int count=0;
char *s,*pattern;
int Machine(int state,char input)
{
switch(state)
```

{
case 0: if (input==pattern[0]){state=2;}
else{state=1;}
break;
case 1:
if (input==pattern[0]){state=2;}
else{state=1;}

see more

Reply • Share ›

sameer karjatkar Iqbal Hawre • 9 months ago

Can you explain the logic ? I did not understand as to why only four states ?

Reply • Share ›

Sanjay Agarwal • 2 years ago

This is one of the solutions (Written in C++)
Time Complexity: O(n*k)
(n = size of given string, k = size of the pattern)
Note: Solutions exist which have linear time complexity.

```
 #include<iostream>
#include<conio.h>
#include<string.h>
using namespace std;

void pattern_matching_naive(char *str, char *pattern, int n, int k);

int main()
{
    char str[100] = {'&#92&#48'}, pattern[100]= {'&#92&#48'};
    int n,k;

    cout<<"\nEnter the string: ";
```

see more

1 • Reply • Share ›

abhishek08aug • 2 years ago

Intelligent :D

Reply • Share ›

zeus • 3 years ago

my code looks like this

#include
using namespace std;

const int d=256;// here d represnts no. of types of letters used which is 4 here as A,B,C,D {Number of characters}

int getvalueforTF(char *b,int m,int i,int j) // i is value of state and j value corresponding to the character
{
int max=0;
char z1=j;
int flag=0;
for(int a1=0;a1<i+1;a1++)
{
flag=0;
if(z1==b[a1])

see more

• Reply • Share ›

zeus • 3 years ago

how can u equate x and pat[ns-1] ?
pls reply asap

• Reply • Share ›

Azim zeus • 2 years ago

Yes it is wrong.

The condition should be like this
if(state < patLength && pat[state] == (character + 65)) // for uppercase letters

• Reply • Share ›

Steven Bi • 3 years ago

What is the advantage of using this algorithm?

• Reply • Share ›

zeus Steven Bi • 3 years ago

this execute in big theta of n though it is a complex algo
but it is far far better than other algos

• Reply • Share ›

Azim zeus • 2 years ago

Yes searching takes O(n) but to create transition table takes more than O(n^2) complexity.

^ | ⌄ • Reply • Share ›

Avi • 3 years ago

```
 #include<stdio.h>
#include<conio.h>
#include<string.h>
void patern_Search(char *,char *);
int main()
{
    char input_String[] = "THIS IS A TEST TEXT";
    char patern[] ="TEST";
    int i = 0;

    patern_Search(input_String,patern);
    getch();
}
void patern_Search(char *a,char *b)
{
     int index,i,j,match,flag;
     i = 0;
     index = 0;
```

see more

^ | ⌄ • Reply • Share ›

zeus ➦ Avi • 3 years ago

ur code is a bit better than naive algo as it use the knowledge of preprocesed data

^ | ⌄ • Reply • Share ›

zeus ➦ Avi • 3 years ago

your code is a bit better than naive string search algo
as it uses the knowledge of previous processed data...

```
/* Paste your code here (You may delete these lines if not writing code) */
```

^ | ⌄ • Reply • Share ›

kartik ➦ Avi • 3 years ago

@Avi: Your code looks like implementation of Naive String Matching algorithm. The time complexity of Naive is O((n-m+1)*m) in worst case. Worst case time complexity of FA based and KMP algorithms is linear in worst case.

^ | ⌄ • Reply • Share ›

-
-
-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms
-

- ## Popular Posts
    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding “extern” keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.
    - Check if a binary tree is BST or not
    - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments
    - lt_k

i need help for coding this function in java...

Java Programming Language · 2 hours ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 2 hours ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago