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.

Google™ Custom Search

GeeksforGeeks

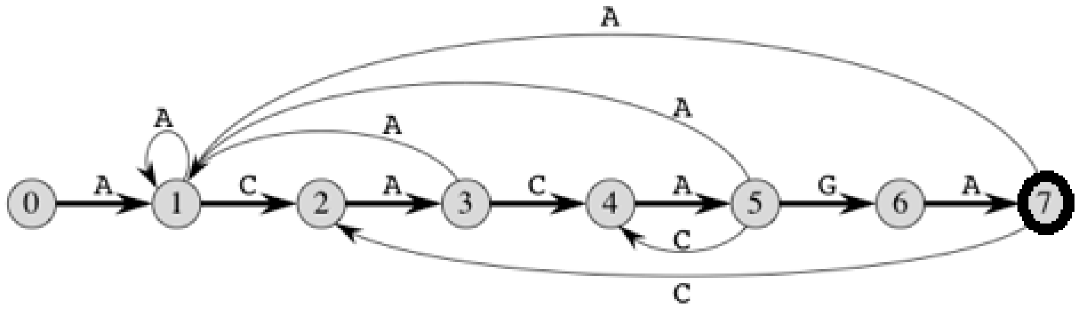
52,731 people like GeeksforGeeks.

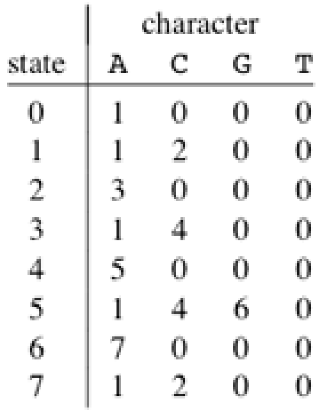
Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

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.





	character			
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.

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

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
   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.

Better Than Hadoop.
HPCC Systems is Big Data Processing and Analytics
Open Source. Proven. Trusted.
LexisNexis
Learn More

Related Tpoics:

- Printing Longest Common Subsequence
- Suffix Array | Set 2 (nLogn Algorithm)
- Rearrange a string so that all same characters become d distance away
- Recursively remove all adjacent duplicates
- Find the first non-repeating character from a stream of characters
- Dynamic Programming | Set 33 (Find if a string is interleaved of two other strings)
- Remove “b” and “ac” from a given string
- Dynamic Programming | Set 29 (Longest Common Substring)

1 Tweet 0 0

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

15 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

alien • 8 months ago

Can someone please explain what is the difference between FA string matchi

Reply • Share ›

dag alien • 7 months ago

in case of KMP, there is no overhead of creating FA and storing FA, ra
takes O(m) time only

Reply • Share ›

Iqbal Hawre • 8 months 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 ›

Sanjay Agarwal · 10 months 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;
```

see more

1 ^ | ˅ • Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

^ | ˅ • Reply • Share ›

zeus • 2 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

int getvalueforTF(char *b,int m,int i,int j) // i is value of state and j value corresp
{
int max=0;
char z1=j;
int flag=0;
for(int a1=0;a1<i+1;a1++)
{
flag=0;
if(z1==b[a1])
{
for(int a2=0;a2<a1;a2++)
{

see more

Reply • Share ›

zeus • 2 years ago

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

Reply • Share ›

Azim ➦ zeus • 8 months ago

Yes it is wrong.

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

Reply • Share ›

Steven Bi • 2 years ago

What is the advantage of using this algorithm?

Reply • Share ›

zeus ➦ Steven Bi • 2 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 • 8 months ago

Yes searching takes O(n) but to create transition table takes m

Reply • Share ›

Avi • 2 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;
```

see more

^ | ⌄ • Reply • Share ›

zeus ➦ Avi • 2 years ago

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

^ | ⌄ • Reply • Share ›

zeus ➦ Avi • 2 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 wri
```

^ | ⌄ • Reply • Share ›

kartik Avi • 2 years ago

@Avi: Your code looks like implementation of Naive String Matching alg
is O((n-m+1)*m) in worst case. Worst case time complexity of FA bas
worst case.

• Reply • Share ›

Subscribe Add Disqus to your site