Searching for Patterns | Set 2 (KMP Algorithm)

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
```

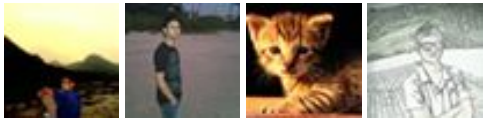
2) Input:

```
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.



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

We have discussed Naive pattern searching algorithm in the previous post. The worst case complexity of Naive algorithm is O(m(n-m+1)). Time complexity of KMP algorithm is O(n) in worst case.

KMP (Knuth Morris Pratt) Pattern Searching

The Naive pattern searching algorithm doesn't work well in cases where we see many matching characters followed by a mismatching character. Following are some examples.

```
txt[] = "AAAAAAAAAAAAAAAAAB"
pat[] = "AAAAB"

txt[] = "ABABABCABABABCABABABC"
pat[] =  "ABABAC" (not a worst case, but a bad case for Naive)
```

The KMP matching algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to O(n). The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text (since they matched the pattern characters prior to the mismatch). We take advantage of this information to avoid matching the characters that we know will anyway match.

KMP algorithm does some preprocessing over the pattern pat[] and constructs an auxiliary array lps[] of size m (same as size of pattern). Here **name lps indicates longest proper prefix which is also suffix.**. For each sub-pattern pat[0…i] where i = 0 to m-1, lps[i] stores length of the maximum matching proper prefix which is also a suffix of the sub-pattern pat[0..i].

```
lps[i] = the longest proper prefix of pat[0..i]
           which is also a suffix of pat[0..i].
```

Examples:

For the pattern "AABAACAABAA", lps[] is [0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5]

For the pattern "ABCDE", lps[] is [0, 0, 0, 0, 0]

For the pattern "AAAAA", lps[] is [0, 1, 2, 3, 4]

For the pattern "AAABAAA", lps[] is [0, 1, 2, 0, 1, 2, 3]

For the pattern "AAACAAAAAC", lps[] is [0, 1, 2, 0, 1, 2, 3, 3, 3, 4]

Searching Algorithm:

Unlike the Naive algo where we slide the pattern by one, we use a value from lps[] to decide the

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

next sliding position. Let us see how we do that. When we compare pat[j] with txt[i] and see a mismatch, we know that characters pat[0..j-1] match with txt[i-j+1...i-1], and we also know that lps[j-1] characters of pat[0...j-1] are both proper prefix and suffix which means we do not need to match these lps[j-1] characters with txt[i-j...i-1] because we know that these characters will anyway match. See KMPSearch() in the below code for details.

Preprocessing Algorithm:
In the preprocessing part, we calculate values in lps[]. To do that, we keep track of the length of the longest prefix suffix value (we use len variable for this purpose) for the previous index. We initialize lps[0] and len as 0. If pat[len] and pat[i] match, we increment len by 1 and assign the incremented value to lps[i]. If pat[i] and pat[len] do not match and len is not 0, we update len to lps[len-1]. See computeLPSArray () in the below code for details.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void computeLPSArray(char *pat, int M, int *lps);

void KMPSearch(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    // create lps[] that will hold the longest prefix suffix values fo
    int *lps = (int *)malloc(sizeof(int)*M);
    int j  = 0;  // index for pat[]

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);

    int i = 0;  // index for txt[]
    while(i < N)
    {
      if(pat[j] == txt[i])
      {
        j++;
        i++;
      }

      if (j == M)
      {
        printf("Found pattern at index %d \n", i-j);
```

```
            j = lps[j-1];
        }

        // mismatch after j matches
        else if(pat[j] != txt[i])
        {
          // Do not match lps[0..lps[j-1]] characters,
          // they will match anyway
          if(j != 0)
           j = lps[j-1];
          else
           i = i+1;
        }
    }
    free(lps); // to avoid memory leak
}

void computeLPSArray(char *pat, int M, int *lps)
{
    int len = 0;  // lenght of the previous longest prefix suffix
    int i;

    lps[0] = 0; // lps[0] is always 0
    i = 1;

    // the loop calculates lps[i] for i = 1 to M-1
    while(i < M)
    {
       if(pat[i] == pat[len])
       {
         len++;
         lps[i] = len;
         i++;
       }
       else // (pat[i] != pat[len])
       {
         if( len != 0 )
         {
           // This is tricky. Consider the example AAACAAAA and i = 7.
           len = lps[len-1];

           // Also, note that we do not increment i here
         }
         else // if (len == 0)
         {
           lps[i] = 0;
           i++;
```


Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 18 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 38 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 38 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 2 hours ago

```
            }
        }
    }
}

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

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

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)

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

60 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

Vinay Dsouza • 17 days ago

@Rajesh M D

when the suffix of the Pattern does not matches prefix. ie. pat[i] != pat[len] and if len!=0 , then len = lpx[len-1] , which basically means if the prefix and suffix char dont match, then len = second last array element from lps array.

This is done so that we check again for the prefix and suffix, and the len has tc

Check the link below for a detailed explanation.

https://www.youtube.com/watch?...

Reply • Share ›

Rajesh M D • 20 days ago

can anyone explain me why this part is implemented.

---

if(len != 0)

{

// This is tricky. Consider the example AAACAAAA and i= 7.

len = lps[len-1];

// Also, note that we do not increment i here

}

---

we could have assign len = 0 directly right.

Reply • Share ›

Zheng Luo • a month ago

Good implementation, thanks for sharing.

Reply • Share ›

Gourab Mitra • 3 months ago

Consult http://jakeboxer.com/blog/2009... for step by step preparation of the lps

11 Reply • Share ›

gaurav jindal Gourab Mitra • a month ago

Thanks a lot buddy. Your explanation helped a lot, and put an end to my

Reply • Share ›

shashi jey • 4 months ago

//following is short and easy code of kmp algorithm and its easy to understand

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void KMPSearch(char *pat, char *txt)

{

```
int M = strlen(pat);

int N = strlen(txt);

// create lps[] that will hold the longest prefix suffix values for pattern

int j = 0; // index for pat[]

// Preprocess the pattern (calculate lps[] array)
```

see more

1 ▲ | ▼ • Reply • Share ›

groomnestle • 5 months ago

Should lps[i] indicates the longest common prefix/suffix for [0..i-1] ?

▲ | ▼ • Reply • Share ›

rahul ➦ groomnestle • 5 months ago

hmmm....

▲ | ▼ • Reply • Share ›

patrick • 8 months ago

Does anyone have an idea about implementation of KMP with pattern having w

▲ | ▼ • Reply • Share ›

karan • 9 months ago

@geeksforgeeks:When we compare pat[j] with txt[i] and see a mismatch, we with "txt[i-j+1...i-1]".I think it's a bit wrong. It should be "txt[i-j...i-1]".

It's because the two lengths don't match.

pat[0...j-1] has length of (j-1)-0+1=j.

8 ▲ | ▼ • Reply • Share ›

Muthukumar • 9 months ago

@geeksforgeeks

If we have a substring as ABABABABBA : the array should be [0,0,1,2,3,4,5,6,

I have a problem with the BBA part. the algo will give an output [0,0,1,2,3,4,5,6,

Correct me if i am wrong.

▲ | ▼ • Reply • Share ›

Muthukumar ➦ Muthukumar • 9 months ago

Sorry, the algo does give the correct answer. A better explanation to hc

2 ▲ | ▼ • Reply • Share ›

Karthick • 9 months ago

Can we use "len--" instead of "len=lps[len-1]" ? If not,can u give a test case for

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

2 ▲ | ▼ • Reply • Share ›

its_dark ➦ Karthick • 5 months ago

0 1 2 3 4 5 6 7 8 9

if we take pat="A B A B C A B A B A",

lps array : 0 0 1 2 0 1 2 3 4 3

then, when j=8, len=4 (ABAB has been matched).

Now, pat[9] != pat[4],

we know that pat[4] also has some lps number, in this case it is 2.That

also, there is a prefix ("AB") of size 2, that is also a suffix.

now, if index 8 has lps number 4, this means "ABAB" is a prefix as wel

Now, at index 4, we have "AB" matched (at index : 0-1) , therefore at in
(at index : 0-1).

therefore, the main point is if pat[9] doesn't match with the pat[4], then
lps[8]=4 anymore.
BUT, we know that whatever is the lps of pat[3], pat[8] will match with t
pat[0] and pat[1])

see more

6 ^ | v • Reply • Share ›

anjaneya2 • 10 months ago

in your code mismatch after j matches i.e
else if(pat[j] != txt[i])
{
// Do not match lps[0..lps[j-1]] characters,
// they will match anyway
if(j != 0)
j = lps[j-1];
else
i = i+1;
}

i think j = lps[j-1] should be lps[j]. Correct me if wrong

^ | v • Reply • Share ›

anjaneya2 • 10 months ago

why you are taking
if(j != 0)
j = lps[j-1];

else
i = i+1;
}

3 ^ | v • Reply • Share ›

anjaneya2 • 10 months ago

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

^ | v • Reply • Share ›

Saisundar Raghavan • 10 months ago

Harbhanu Sahai

^ | v • Reply • Share ›

Vishnu Vasanth R • 11 months ago

This is the implementation based of CLRS book.

[sourcecode language="C++"]
/* Paste your code here (You may delete these lines if not writing code) */
[/#include<iostream>
#include<string>

using namespace std;
void computeLongestPrefixSuffix(string &P,int lps[])

void KMPMatcher(string &T, string &P){
int n = T.size();
int m = P.size();

int *lps = new int[P.size()]; // similar to int lps[P.size()];

computeLongestPrefixSuffix(P,lps);

// also we ll not access index -1 in function or matcher

see more

• Reply • Share ›

rakshify • 11 months ago

@GeeksForGeeks:- Can you please explain how worst case complexity of KM
Looking at this piece:-

```
while(i < N)
   {
     if(pat[j] == txt[i])
     {
       j++;
       i++;
     }

     if (j == M)
     {
       printf("Found pattern at index %d \n", i-j);
       j = lps[j-1];
     }

     // mismatch after j matches
```

see more

• Reply • Share ›

kartik ➚ rakshify • 11 months ago

The loop actually runs at-most 2n times. Therefore, the time complexit

Like Naive string matching, we slide the pattern over and match them a

move to next character in text. So total iterations of loop is 2n.

Reply • Share ›

rakshify ➚ kartik • 11 months ago

Oh, that was so stupid to miss that.
Thanks Kartik.

Reply • Share ›

rakshify • 11 months ago

@GeeksForGeeks:- Can you please explain how worst case complexity of KMP is O(n)?
Looking at this piece:-

```
while(i < N)
   {
     if(pat[j] == txt[i])
     {
       j++;
       i++;
     }

     if (j == M)
     {
       printf("Found pattern at index %d \n", i-j);
       j = lps[j-1];
     }
```

see more

Reply • Share ›

pritybhudolia • 11 months ago

@GeeksForGeeks
Hi,A very simple approach in O(n) complexity. Can someone tell me that why s
other algo. I am really confused as it works for all cases according to me.

/
#include<stdio.h>
#include<string.h>
void search(char *pat, char *str)
{
int M = strlen(pat);
int N = strlen(str);
int index=0,i,j,flag=0;
for(i=0,j=0;i<=N;i++)
{

if(str[i]==pat[j] && ((str[i+1]==pat[j+1])||(j==M-1)))
{

i++;

see more

• Reply • Share ›

GeeksforGeeks ➚ pritybhudolia • 11 months ago
Could you please post the code again in sourcecode tags. Also, please
algorithm.

• Reply • Share ›

pritybhudolia ➚ GeeksforGeeks • 11 months ago
@GeeksforGeeks Yes ofcourse, actually we start with the first
through the entire string.everytime while traversing we compare
PAT and only if it matches we increment both(i.e index of STR
there is a matching pattern,else we increment index of STR alc

flag is 1 and pattern is traversed completely once, we print the
zero to iterate again and search for another pattern if exists.

```
#include<stdio.h>
#include<string.h>
void search(char *pat, char *str)
{
    int M = strlen(pat);
    int N = strlen(str);
    int index=0,i,j,flag=0;
    for(i=0,j=0;i<=N;i++)
    {
    if(str[i]==pat[j] && ((str[i+1]==pat[j+1])||(j==M-1)
```

see more

• Reply • Share ›

Pandian ➚ pritybhudolia • 9 months ago

Your code fails for the following case :
text : AAAAAAAAAAAAAAAAAAB
pattern : AAAAAAAAAAAB

• Reply • Share ›

TheRock ➚ Pandian • 5 months ago

Dude, it works for this test case..

• Reply • Share ›

prity • 11 months ago

@GeeksForGeeks
Hi,A very simple approach in O(n) complexity. Can someone tell me that why
other algo. I am really confused as it works for all cases according to me.

```
 /
#include<stdio.h>
#include<string.h>
void search(char *pat, char *str)
{
    int M = strlen(pat);
    int N = strlen(str);
    int index=0,i,j,flag=0;
    for(i=0,j=0;i<=N;i++)
    {

                                    if(str[i]==pat[j] && ((str[i+1]==pat
                                    {
```

see more

• Reply • Share ›

Gagan • a year ago

For a much elaborate and clear explanation of this algorithm please refer to "L Algorithms by Prof.SunderVishwanathan, Department of Computer Science E mentioned link:
http://www.youtube.com/watch?v...

• Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

• Reply • Share ›

Rama Krishna Linga • a year ago

Following is the Java version and does not have the issues listed by Ramesh.

```
 // Takes a pattern and returns a new array containing count of
// longest proper prefix of pat[i] which is also suffix of pat[i]
private static int [] buildLPS(char []pat)
{
    int [] lps = new int[pat.length];

    for (int len=0, i=1; i < pat.length; i++)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
```

see more

 • Reply • Share ›

Ramesh.Mxian • a year ago

I think the code given in the post for 2 method will not work for the following inp

Text : ABCAAAABBBABCBCA
Pattern: ABC

It will cause segmentation fault in the following line
// mismatch after j matches
else if(pat[j] != txt[i])

Because last character in the text 'A' will match the 1st character 'A' in the pat
Now 'i' will became the length of the Text given, so Text[i] will give segmentatic

nikhil • a year ago

```
 void KMPSearch(char *pat, char *txt)
{
    int m = strlen(pat);
    int n = strlen(txt);
    int i=0, len=0;
    computeLPSArray(pat, m, lps);
    while (i<n)
    {
        while (len!=0 && txt[i]!=pat[len]) len=b[len]; //backtrack
        if(pat[len] == txt[i]) { len++;} //if pattern matches , incr
        i++; //to match next pattern
        if (len==m)
        {
            //print pattern found at i;
            len=lps[len]; //backtrack to last match position
        }
    }
}
```

see more

Vibhu Tiwari • a year ago

This is the source code for pattern searching in much less effort with the time for various strings by passing the lengths of the two strings to be matched.The the number of times that substring occurs in the string.

```
#include<stdio.h>
#include<conio.h>
void patternsearch(char *a,char *b,int n,int m)
```

```
{ int k,count=0,j=0,i=0,c=0;
while(i!=n)
{ if(j==m)
{j=0;
c=c+1;count=0;
i=c;}
k=a[i]-b[j];
if(k==0){
count++;}
if(count==m)
{printf("Pattern Match found\n");}
i=i+1;
```

see more

• Reply • Share ›

rana_leaner • a year ago

For pattern "AABAACAABAA " lps[] is

Def of lps[i] = the longest proper preefix of pat[0..i] which is also a suffix of pat|
Steps:
lps[0]--> pat[0] = A -->0 (represents length of match prefix,suffix)
lps[1]--> pat[0..2] = A/*A*/ -->1 (Proper prefix =A ,Suffix = A)
lps[2]-->pat[0..3] = AAB -->0 (No any equal prefix,suffix)
lps[3]-->pat[0..4] = /*A*/AB/*A*/-->1 (prefix = A ,sufficx =A)
lps[4]-->pat[0..5] = /*AA*/B/*AA*/ -->2 (prefix = AA ,sufficx =AA)
lps[5]-->pat[0..6] = AABAAC -->0
lps[6]-->pat[0..7] =/*A*/ABAAC/*A*/ -->1

.... so on
lps[] = [0,1,0,1,2,0,1,2,3,4,5]

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

• Reply • Share ›

anonymus • 2 years ago

I was trying to understand this algorithm form back two months,
Now I finally go it with the help of geeksforgeeks,
THANKS GEEKSFORGEEKS

• Reply • Share ›

Yogesh Batra anonymus • 2 years ago

Thanks Geeksforgeeks! :)

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

• Reply • Share ›

deep • 2 years ago

great code

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

• Reply • Share ›

sparco • 2 years ago

The below code is more readable and understandable.
Logic is same as the notes.
Just worth sharing!

```
 void KMPSearch(char *pat, char *txt)
{
    int m = strlen(pat);
    int n = strlen(txt);
```

```
int i=0, len=0;
computeLPSArray(pat, m, lps);
while (i<n)
{
    while (len!=0 && txt[i]!=pat[len]) len=b[len]; //backtrack
            if(pat[len] == txt[i]) { len++;} //if pattern matches
            i++; //to match next pattern
    if (len==m)
    {
        //print pattern found at i;
```

see more

• Reply • Share ›

anonymous sparco • a year ago

@sparco

In your code of computeLPSArray you wrote j instead of len.

• Reply • Share ›

samesh • 2 years ago

Hi,could anyone put some light on this example.
According to me itz a wrong example??Help me out...

txt[] = "ABABABCABABABCABABABC"
pat[] = "ABABAC" (not a worst case, but a bad case for Naive)

• Reply • Share ›

suresh kumar • 2 years ago

```
 Hi,could anyone put some light on this example.
According to me itz a wrong example??Help me out...
txt[] = "ABABABCABABABCABABABC"
pat[] =  "ABABAC" (not a worst case, but a bad case for Naive)
```

▮

^ | ⌄ • Reply • Share ›

Franky • 3 years ago

// This is tricky. Consider the example AAACAAAA and i = 7.
len = lps[len-1];

Can you explain why we need to set len equal to lps[len-1] in the function?

^ | ⌄ • Reply • Share ›

sharat • 3 years ago

Hi Algorist,

Read CLR book and then come back here.....

^ | ⌄ • Reply • Share ›

Arpit Gupta • 3 years ago

In this article,the complexity of naive method has been wrongly mentione as (r

^ | ⌄ • Reply • Share ›

Sandeep ➦ Arpit Gupta • 3 years ago

@Arpit Gupta: Thanks for pointing this out. We have corrected the typc

^ | ⌄ • Reply • Share ›

sharat04 • 3 years ago

Hi Geeks,

Thanks for coming up with this post. I am still struggling to understand the cor

Basically I am looking for two things here.

1) A technical definition of "proper prefix" and " proper Suffix"
2) A detailed run down of any of the examples in your listing. explaining how th

From the listing above, For the pattern “AAACAAAAAC”, lps[] is [0, 1, 2, 0, 1, 2

In the above mentioned example, why is the lps[3](element C in the pattern) "0
"AAA" is before C and after C in the pattern??

Please help me understand the algorithm here.

Thanks..

• Reply • Share ›

sharat04 sharat04 • 3 years ago

I think I figured it out.. I looked at the wiki http://en.wikipedia.org/wiki/Sub

In any case, I would request you to add more detailed description and a
mentioned.

Thanks

• Reply • Share ›

Cracker • 3 years ago

Code For KMP

```
 // precomputation time: O(m) where m is length of string to be matche
// net time: O(n+m) where n = length of string to which another string

#include<stdio.h>

void kmp(char[],char[]);

int main()
{
        char a[100], b[100];
```

```
        gets(a);
        gets(b);

        kmp(a,b);
```

see more

Reply • Share ›

Load more comments

Subscribe Add Disqus to your site