Anagram Substring Search (Or Search for all permutations)

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[] and its permutations (or anagrams) in txt[]. You may assume that n > m. Expected time complexity is O(n)

Examples:

```
1) Input:  txt[] = "BACDGABCDA"  pat[] = "ABCD"
   Output:   Found at Index 0
             Found at Index 5
             Found at Index 6
2) Input: txt[] =  "AAABABAA" pat[] = "AABA"
   Output:   Found at Index 0
             Found at Index 1
```

```
Found at Index 4
```

This problem is slightly different from standard pattern searching problem, here we need to search for anagrams as well. Therefore, we cannot directly apply standard pattern searching algorithms like KMP, Rabin Karp, Boyer Moore, etc.

A simple idea is to modify Rabin Karp Algorithm. For example we can keep the hash value as sum of ASCII values of all characters under modulo of a big prime number. For every character of text, we can add the current character to hash value and subtract the first character of previous window. This solution looks good, but like standard Rabin Karp, the worst case time complexity of this solution is O(mn). The worst case occurs when all hash values match and we one by one match all characters.

We can achieve O(n) time complexity under the assumption that alphabet size is fixed which is typically true as we have maximum 256 possible characters in ASCII. The idea is to use two count arrays:

1) The first count array store frequencies of characters in pattern.
2) The second count array stores frequencies of characters in current window of text.

The important thing to note is, time complexity to compare two count arrays is O(1) as the number of elements in them are fixed (independent of pattern and text sizes). Following are steps of this algorithm.
1) Store counts of frequencies of pattern in first count array *countP[]*. Also store counts of frequencies of characters in first window of text in array *countTW[]*.

2) Now run a loop from i = M to N-1. Do following in loop.
…..a) If the two count arrays are identical, we found an occurrence.
…..b) Increment count of current character of text in countTW[]
…..c) Decrement count of first character in previous window in countWT[]

3) The last window is not checked by above loop, so explicitly check it.

Following is C++ implementation of above algorithm.

```
// C++ program to search all anagrams of a pattern in a text
#include<iostream>
#include<cstring>
#define MAX 256
using namespace std;

// This function returns true if contents of arr1[] and arr2[]
// are same, otherwise false.
bool compare(char arr1[], char arr2[])
{
    for (int i=0; i<MAX; i++)
        if (arr1[i] != arr2[i])
            return false;
    return true;
}

// This function search for all permutations of pat[] in txt[]
void search(char *pat, char *txt)
{
    int M = strlen(pat), N = strlen(txt);
```

```
    // countP[]:  Store count of all characters of pattern
    // countTW[]: Store count of current window of text
    char countP[MAX] = {0}, countTW[MAX] = {0};
    for (int i = 0; i < M; i++)
    {
        (countP[pat[i]])++;
        (countTW[txt[i]])++;
    }

    // Traverse through remaining characters of pattern
    for (int i = M; i < N; i++)
    {
        // Compare counts of current window of text with
        // counts of pattern[]
        if (compare(countP, countTW))
            cout << "Found at Index " << (i - M) << endl;

        // Add current character to current window
        (countTW[txt[i]])++;

        // Remove the first character of previous window
        countTW[txt[i-M]]--;
    }

    // Check for the last window in text
    if (compare(countP, countTW))
        cout << "Found at Index " << (N - M) << endl;
}

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

Output:

```
Found at Index 0
Found at Index 5
Found at Index 6
```

This article is contributed by **Piyush Gupta**. 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

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

Kaiyu • 2 months ago

Is there any way to improve the performance of the compare function? For example, comparing "zxx" to "xxz" is so inefficient;

Atul Yadav • 5 months ago

public class SubstringAllPermutation {

/**
* @param args
*/

public char[] sort(char x[])
{
int n=x.length;
for(int i=0;i<n;i++) {="" for(int="" j="i+1;j<n;j++)" {="" if(x[i]<x[j])="" {="" char="" t="x[i];" x[i]="x[j];" x[j]="t;" }="" }="" }="" return="" x;="" }="" public="" void="" printpositionallpermutation(string="" s,string="" x)="" {="" int="" l="x.length();" char="" t[]="new" char[l];="" char="" pat[]="x.toCharArray();" pat="sort(pat);" int="" count="0;" for(int="" i="0;i<=s.length()-l;i++)" {="" t="s.substring(i,i+l).toCharArray();" t="sort(t);" if(isequal(t,pat))="" {="" system.out.println(i);="" }="" }="" }="" public="" boolean="" isequal(char="" x[],char="" y[])="" {="" for(int="" i="0;i<x.length;i++)" {="" if(x[i]!="y[i])" return="" false;="" }="" return="" true;="" }="" public="" static="" void="" main(string[]="" args)="" {="" todo="" auto-generated="" method="" stub="" new="" substringallpermutation().printpositionallpermutation("bacdgabcda","abcd");="" new="" substringallpermutation().printpositionallpermutation("aaababaa","aaba");="" }="" }="">

Sumit Kesarwani ➦ Atul Yadav • 5 months ago

if u want share ur code then please use this editor.. its neat and clean...
http://www.codeshare.io

• Reply • Share ›

Ashish • 6 months ago

In line,
char countP[MAX] = {0}, countTW[MAX] = {0};
Shouldn't the type be int instead of char?

5 • Reply • Share ›

Santhosh • 6 months ago

Solution in Java :

```
package com.santhosh.test;

import java.util.HashMap;

import edu.emory.mathcs.backport.java.util.Arrays;

public class AnagramSearch {

public static void main(String args[]) {

HashMap<character,integer> patMap = new HashMap<character,integer>();

char pat[] = "ABCD".toCharArray();

char txt[] = "BACDGABCDA".toCharArray();

buildMapFromChar(patMap,pat);

for(int i = 0 ; i < txt.length ; i++) {
```

see more

• Reply • Share ›

Vãîbhåv Joshî • 8 months ago

java code
http://ideone.com/c36gWy

• Reply • Share ›

Александр Яковлев • 8 months ago

O(n) solution with lowest constant, i believe.
http://codeshare.io/lLytY

2 • Reply • Share ›

Глеб Степанов • 8 months ago

http://codeshare.io/bHsCw

• Reply • Share ›

Глеб Степанов • 8 months ago

{{{
def get_substr(t,p):
d = {}

for c in p:
if c in d:
d[c] += 1
else:
d[c] = 1

cnt = len(p)
d2 = {}
j = 0

for i in range(len(t)):
if t[i] not in d:
j = i + 1
cnt = len(d)
d2 = {}

see more

• Reply • Share ›

Guest • 8 months ago

def get_substr(t,p):
d = {}

for c in p:
if c in d:
d[c] += 1
else:
d[c] = 1

cnt = len(p)
d2 = {}
j = 0

for i in range(len(t)):
if t[i] not in d:
i = i + 1

```
cnt = len(d)
d2 = {}
continue
```

see more

• Reply • Share ›

setu • 9 months ago

This is a implementation with time complexity O(n) without iterating over array.

https://ideone.com/lfyVAs

• Reply • Share ›

Bala • 9 months ago

Count arrays shld be int type right?

1 • Reply • Share ›

Guest80 • 9 months ago

O(mn) solution with constant space using XOR is here.

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

void Check(char str1[],char str2[],int n)
{
int a=str1[0],i,j,k,b;
int m=strlen(str2);
for(i=1;i<n;i++) a^="str1[i];" for(i="0;i&amp;lt;m;i++)" {="" j="i;" if(j+n<="m)" {="" k="j+n;" b="str2[j++];"
while(j<k)="" b^="str2[j++];" if(!(a^b))="" printf("found="" at="" %d="" \n",k-n);="" }="" else="
break;=" }="" }="" int="" main()="" {="" char="" s1[]="ABCD" ,s2[]="BACDGABCDA" ;="
check(s1,s2,strlen(s1));="" return="" 0;="" }="">
```

• Reply • Share ›

jack • 9 months ago

why comparing two count arrays is O(1)? you need to scan the whole two arrays

• Reply • Share ›

jack ➚ jack • 9 months ago

ok, I got what you mean. You mean the array size is fixed as 256, so it is a constant.

1 • Reply • Share ›

dexter • 9 months ago

my solution based on some suggestions from others

https://github.com/kishore3385...

• Reply • Share ›

anand kiran • 9 months ago

Posting solution with Complexity O(n)

```
public class anagram {
public static void anagramCheck(String str,String pattern)
{
int sum = 0,sum1 = 0;
for(int i = 0; i < pattern.length(); i ++)
{
sum = sum + (Character.getNumericValue(pattern.charAt(i)));
}
int i=0 , j= 0 , k = pattern.length(), z = 0;
for( i = z ,j = k; j <= str.length(); i ++,j++)
{
if(k == pattern.length())
{
for(int x = i ; x < j ; x ++)
{
sum1 = sum1 + (Character.getNumericValue(str.charAt(x)));
```

see more

• Reply • Share ›

ad • 9 months ago

Hi All, here is my algorithm in Python
1. Make a dict of the pattern to be matched
2. a window would slide over the string s , make a dict of that part and compare with the dict of the pattern (in step 1)
3. If there is a match, i.e if the two dicts are equal, then return the index.
However, i dont know the time complexity of this sol. Can someone help me with this?

My thought : the time complexity of making a string into dict in 0(N), again moving the sliding window over the string is o(N). so should it be o(N^2)?

• Reply • Share ›

Idéophage ad • 9 months ago

Can you define what is "N" in your complexity? Is it the length of the pattern or the length of the text to search in. I think you've made a confusion, but you almost have to the correct complexity.

• Reply • Share ›

Ad ad • 9 months ago

http://ideone.com/WujKOC

• Reply • Share ›

anand kiran • 9 months ago

Another simple solution using ascii values

```
public class anagram {

public static void anagramCheck(String str,String pattern)

{

int sum = 0;

int sum1 = 0;

for(int i = 0; i < pattern.length(); i ++)

{

sum = sum + (Character.getNumericValue(pattern.charAt(i)));

}

int i=0, j= 0, k = pattern.length(), z = 0;
```

see more

• Reply • Share ›

dexter • 9 months ago

Hi,
why cant we just find the Ascii sum of the pattern. For example asci (A +B+C+D) = lets say 200.. in the example and then search the text string linearly?. Is it because of the time complexity or some other reason. Please let me know. Thanks

• Reply • Share ›

Idéophage ➚ dexter • 9 months ago

The sum does not represent faithfully the multi-set of the characters of the pattern. For instance, 'B'+'C' = 'A'+'D'.
To represent faithfully the pattern, you can associate to the characters different weights than their ascii values. For instance, let's say the pattern is 2 characters long and you have 2 characters in your alphabet. The idea is to reserve one digit for each possible character. Let x be our base of numeration. Then, the weight of the first character is x^0 and the weight of the second one is x^1. The base have to be strictly greater than the size of the pattern, otherwise if we have too many occurrences of a given character, this will affect the next digit. Hence, if m is the length of the pattern, the base of numeration will be m+1 and the ith character will be associated to (m+1)^i. We can actually take a base of m, since the first character can be associated to a weight of 0 without making our map unfaithful

without making our map unfaithful.
Another approach with the same (weird) idea is to associate the ith character to the ith prime number and to multiply instead of add on. (numbers are multisets)

However, this is not a useful approach in practice, since the numbers will be too big. Another problem with this method is we cannot (easily) benefit from the fact that at most two digits change at each step (as Kyoo Sn M M have done in his code).

PS : A classical problem in combinatorics is to count the number of possible patterns modulo permutations of the characters given the size of the alphabet and the size of the pattern.

• Reply • Share ›

Idéophage ➚ Idéophage • 9 months ago

I didn't use the example I gave...

• Reply • Share ›

Idéophage ➚ Idéophage • 9 months ago

> We can actually take a base of m, since the first character can be associated to a weight of 0 without making our map unfaithful.

It is false, sorry. But you still can associate a weight of zero to one of the characters.

• Reply • Share ›

dexter ➚ Idéophage • 9 months ago

thanks for the reply. It helped :)

• Reply • Share ›

Idéophage ➚ dexter • 9 months ago

Ah, and I just learned that in the case of an unfaithful mapping (well chosen -- let's call it f), the algorithm is called the Rabin-Karp algorithm : if f(pattern) and f(current substring of the text) are not equals, you say "pattern not found at this position" and if they are equals, you also check character by character to eventually say "pattern found". It is said in the article (I didn't see, sorry).

PS: I said "However, this is not a useful approach in practice". Neither in theory. This is a really weird hack where you just compare numbers instead of comparing strings.

• Reply • Share ›

dexter ➚ Idéophage • 9 months ago

I implemented this program. Please have a look when you have time https://github.com/kishore3385...

• Reply • Share ›

Idéophage ➚ dexter • 9 months ago

(don't like code :))

1- Why do you write "&arr[0]"? If you write "arr", it is automatically transformed to a pointer to the first element (ie &arr[0]).
2- Why is this constant in your code: 65? It would be better if you write 'A'.
3- Your code is not well indented (and not well written). On ideone, the main function returns -1. Write "return 0;" (or EXIT_SUCCESS declared in stdlib.h).
4- Most importantly, when you find a substring with the same hash than the pattern, you immediately say "found here". I've already said that it is not always the case. Checking if the hashes are the equals only helps to eliminate an important number of cases where the pattern is not found, but it does NOT say that the pattern is effectively found. You also have to check if the substring is an anagram of the pattern. An example of fail is "BBB" vs "AAC".
5- Try to reduce you complexity : you can compute the hash quicker. I'll give you hints if you ask.

• Reply • Share ›

Idéophage ➚ Idéophage • 9 months ago

My 4th point is only if your hash is supposed to be unfaithful. Otherwise, if it's supposed to be faithful, it's not. But doing with a faithful map is not a good solution, as I've already explained.

• Reply • Share ›

dexter ➚ Idéophage • 9 months ago

Thanks a lot for your comments and feedback. Why i used &arr[0] was to remove the warning from compiler.
2) Agree

3)agree
4) yes, missed this one..i thought of using powers of 2 as weight..this doesnt work...using primes will be too complex...will think of something....but thanks!

• Reply • Share ›

Aditya • 9 months ago

This is another approach, This can be make more optimized :) with some checks...correct me if m wrong.

```
#include <stdio.h>

int search(char *text, char *pat, int text_len, int pat_len){

int i = 0, j = 0, temp = 0;

int weight = 0, count = 0;

for(; i < pat_len; i++){

weight += pat[i]-65;

}

i = 0;

for(; i < (text_len- (pat_len -1)); i++){
```

see more

• Reply • Share ›

Subham • 9 months ago

http://ideone.com/E56tmS

my o(n) solution..

• Reply • Share ›

Idéophage • 9 months ago

Hello,

The complexity can be O(n) even if we don't assume the alphabet is constant. You can keep track of the number of differences between the two frequency arrays without recalculating it entirely : you only have to check for four changes at each step.

• Reply • Share ›

Ideophage Idéophage • 9 months ago

It's two changes, not four, sorry.

• Reply • Share ›

Ankit Goyal • 9 months ago

Good Question. Good Solution.

• Reply • Share ›

Prashant • 10 months ago

Alternative: you can assign different prime numbers to all 26 characters. compute product of such values for the pattern. and start computing and comparing product of first N characters in string where N = strlen of pattern.

This approch has better time complexity as it reduces need for loop in above compare()

routune

• Reply • Share ›

Kyoo Sn M M • 10 months ago

I modify the algorithms with just 1 count array and reducing the time, the idea is just decreasing the past element in the count and increasing the new and verify if we have founded all the elements.

http://ideone.com/i6r5As

1 • Reply • Share ›

bani ➚ Kyoo Sn M M • 10 months ago

I cant understand what u mean ... perhaps it cant be done by a single count array .. we need to hash both the strings....

explain briefly and paste your code on ideone and provide link here

• Reply • Share ›

Kyoo Sn M M ➚ bani • 9 months ago

Here is my code and explication: http://ideone.com/i6r5As :)

1 • 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