# Check if a string is rotation of another WITHOUT concatenating

There are 2 strings , how can we check if one is a rotated version of another ?

```
For Example : hello --- lohel
```

One simple solution is by `concatenating` first string with itself and checking if the other one is a `substring` of the concatenated version.

Is there any other solution to it ?

I was wondering if we could use `circular linked list` maybe ? But I am not able to arrive at the solution.

string algorithm language-agnostic

edited Aug 19 '12 at 22:17

asked Aug 19 '12 at 17:18
h4ck3d
**2,158** 7 30 58

Checking ifm the second string is a substring of the first is not enough - the size may be different. When the second string is deleted from the doubled first string, then the remainder is the original first string (in case of rotation). – steenslag Aug 24 '12 at 20:52

## 12 Answers

> One simple solution is by concatenating them and checking if the other one is a substring of the concatenated version.

I assume you mean concatenate the first string with itself, then check if the other one is a substring of that concatenation.

That will work, and in fact can be done without any concatenation at all. Just use any string searching algorithm to search for the second string in the first, and when you reach the end, loop back to the beginning.

For instance, using Boyer-Moore the overall algorithm would be O(n).

answered Aug 19 '12 at 21:37
BlueRaja - Danny Pflughoeft
**47.4k** 21 123 208

perfect ! i coded it using naive string matching (n^2). – h4ck3d Aug 19 '12 at 22:18

I don't understand:"search for the second string in the first, and when you reach the end, loop back to the beginning". What can we get by searching "cdab" in "abcd" ? – kahofanfan Dec 28 '15 at 8:42

There's no need to concatenate at all.

First, check the lengths. If they're different then return false.

Second, use an index that increments from the first character to the last of the source. Check if the destination starts with all the letters from the index to the end, and ends with all the letters before the index. If at any time this is true, return true.

Otherwise, return false.

**EDIT:**

An implementation in Python:

```
def isrot(src, dest):
  # Make sure they have the same size
  if len(src) != len(dest):
    return False

  # Rotate through the letters in src
  for ix in range(len(src)):
    # Compare the end of src with the beginning of dest
    # and the beginning of src with the end of dest
    if dest.startswith(src[ix:]) and dest.endswith(src[:ix]):
      return True

  return False

print isrot('hello', 'lohel')
print isrot('hello', 'lohell')
print isrot('hello', 'hello')
print isrot('hello', 'lohe')
```

edited Aug 19 '12 at 21:27
BlueRaja - Danny Pflughoeft
47.4k 21 123 208

answered Aug 19 '12 at 17:28
Ignacio Vazquez-Abrams
470k 71 838 988

Could you dry run it on an example? I didn't really understand it completely. – h4ck3d Aug 19 '12 at 18:14

@sTEAK. That's pretty much what the code in this answer does. – Alexey Frunze Aug 19 '12 at 18:19

Can u rather put up a pseudo code? Don't know python. @AlexeyFrunze No they are different. – h4ck3d Aug 19 '12 at 18:33

1 Note that this algorithm is O(n^2), since each call to `startswith` and `endswith` is O(n). It is possible to construct an overall O(n) algorithm that is not much more complicated than this, however. – BlueRaja - Danny Pflughoeft Aug 19 '12 at 21:38

True, but `str.startswith()` and `str.endswith()` are implemented in C which means that their O(n) is much smaller than the cost of slicing (which can also be gotten around; this is actually a fairly naive algorithm). – Ignacio Vazquez-Abrams Aug 19 '12 at 21:41

You could compute the lexicographically minimal string rotation of each string and then test if they were equal.

Computing the minimal rotation is O(n).

This would be good if you had lots of strings to test as the minimal rotation could be applied as a preprocessing step and then you could use a standard hash table to store the rotated strings.

answered Aug 19 '12 at 17:32
Peter de Rivaz
21.5k 3 21 45

Could you explain how to compute the minimal rotation in O(n)? – R.. Aug 20 '12 at 2:43

Never mind, it's in the link. The fact that it's possible in O(n) is not entirely surprising, but also non-trivial. And the only solution that's also O(1) space is highly non-trivial. – R.. Aug 20 '12 at 2:45

Trivial O(min(n,m)^2) algorithm: (n - length of S1, m - length of S2)

isRotated(S1 , S2):

```
if (S1.length != S2.length)
    return false
for i : 0 to n-1
    res = true
    index = i
    for j : 0 to n-1
       if S1[j] != S2[index]
           res = false
           break
       index = (index+1)%n
    if res == true
        return true
return false
```

**EDIT:**

Explanation -

Two strings S1 and S2 of lengths m and n respectively are cyclic identical if and only if m == n and exist index 0 <= j <= n-1 such S1 = S[j]S[j+1]...S[n-1]S[0]...S[j-1].

So in the above algorithm we check if the length is equal and if exist such an index.

edited Aug 20 '12 at 7:43

answered Aug 19 '12 at 17:36
barak1412

**695** 4 14

1 Always explain your algorithm first rather than just coding it. – h4ck3d Aug 19 '12 at 18:11

1 @sTEAK. Read the code. It is trivial. – Alexey Frunze Aug 19 '12 at 18:18

1 @AlexeyFrunze: It's also wrong. See Ignacio's answer for a working example. – Michael Foukarakis Aug 19 '12 at 18:30

@MichaelFoukarakis True. – Alexey Frunze Aug 19 '12 at 18:33

@MichaelFoukarakis I had small index error. It is fine now. I wanted to give simple trivial pseudo-code instead of programming language dependent algorithm. – barak1412 Aug 19 '12 at 21:11

|

A very straightforward solution is to rotate one of the words n times, where n is the length of the word. For each of those rotations, check to see if the result is the same as the other word.

answered Aug 19 '12 at 17:27
David
**1,255** 6 7

2 Would't it be very inefficient? – h4ck3d Aug 19 '12 at 18:14

It would depend on your language, but if you're working in a language where you can rotate with O(1) work (by, for example, just moving the last character to the end and moving your pointer forward by one), then doing the rotations would just be O(n) with no horrendous constants. Each comparison would also be O(n), so you'd be looking at O(n^2), I guess. – David Aug 19 '12 at 18:20

Simple solution in Java. No need of iteration or concatenation.

```
private static boolean isSubString(String first, String second){
        int firstIndex = second.indexOf(first.charAt(0));
        if(first.length() == second.length() && firstIndex > -1){

            if(first.equalsIgnoreCase(second))
                return true;

            int finalPos = second.length() - firstIndex ;
            return second.charAt(0) == first.charAt(finalPos)
            && first.substring(finalPos).equals(second.subSequence(0, firstIndex));
        }
        return false;

    }
```

Test case:

```
String first = "bottle";
String second = "tlebot";
```

Logic:

Take the first string's first character, find the index in the second string. Subtract the length of the second with the index found, check if first character of the second at 0 is same as character at the difference of length of the second and index found and substrings between those 2 characters are the same.

edited Oct 23 '13 at 2:04

answered Oct 23 '13 at 0:31
Nikola Despotoski
**32.5k** 11 73 112

"Find the index in the second string" takes `O(n)` already (and also contributes to the number of comparisons) – justhalf Oct 23 '13 at 1:11

1 And this will fail for test case like: `bottle` and `tXYbZW` – justhalf Oct 23 '13 at 1:13

@justhalf I did not consider indexOf implementation time, bad. Thanks for your thoughts. – Nikola Despotoski Oct 23 '13 at 1:52

Solving the problem in O(n)

```
void isSubstring(string& s1, string& s2)
{
    if(s1.length() != s2.length())
        cout<<"Not rotation string"<<endl;
    else
    {
        int firstI=0, secondI=0;
        int len = s1.length();
        while( firstI < len )
        {
            if(s1[firstI%len] == s2[0] && s1[(firstI+1) %len] == s2[1])
                break;

            firstI = (firstI+1)%len;
        }

        int len2 = s2.length();
        int i=0;
        bool isSubString = true;
        while(i < len2)
        {
            if(s1[firstI%len] != s2[i])
            {
                isSubString = false;
                break;
            }
            i++;
        }

        if(isSubString)
            cout<<"Is Rotation String"<<endl;
        else
            cout<<"Is not a rotation string"<<endl;
    }
}
```

answered May 25 '16 at 17:06
preeti
1 1

---

Another python implementation (without concatenation) although not efficient but it's O(n), looking forward for comments if any.

Assume that there are two strings s1 and s2.

Obviously, if s1 and s2 are rotations, there exists two sub strings of s2 in s1, the sum of them will total to the length of the string.

The question is to find that partition for which I increment an index in s2 whenever a char of s2 matches with that of s1.

```
def is_rotation(s1, s2):
    if len(s1) != len(s2):
        return False
    n = len(s1)
    if n == 0: return True

    j = 0
    for i in range(n):
        if s2[j] == s1[i]:
            j += 1
    return (j > 0 and s1[:n - j] == s2[j:] and s1[n - j:] == s2[:j])
```

The second and condition is just to ensure that the counter incremented for s2 are a sub string match.

edited May 11 '14 at 8:06
answered May 11 '14 at 8:01
sysuser
404 4 14

---

input1= "hello" input2="llohe" input3="lohel"(input3 is special case)

if length's of input 1 & input2 are not same return 0.Let i and j be two indexes pointing to input1 and input2 respectively and initialize count to input1.length. Have a flag called isRotated which is set to false

while(count != 0){

When the character's of input1 matches input2

1. increment i & j
2. decrement count

If the character's donot match

1. if isRotated = true(it means even after rotation there's mismatch) so break;
2. else Reset j to 0 as there's a mismatch. Eg:

Please find the code below and let me know if it fails for some other combination I may not have considered.

```
    public boolean isRotation(String input1, String input2) {
    boolean isRotated = false;
    int i = 0, j = 0, count = input1.length();

    if (input1.length() != input2.length())
```

```
        return false;

    while (count != 0) {

        if (i == input1.length() && !isRotated) {
            isRotated = true;
            i = 0;
        }

        if (input1.charAt(i) == input2.charAt(j)) {
            i++;
            j++;
            count--;
        }

        else {
            if (isRotated) {
                break;
            }
            if (i == input1.length() - 1 && !isRotated) {
                isRotated = true;
            }
            if (i < input1.length()) {
                j = 0;
                count = input1.length();

            }
            /* To handle the duplicates. This is the special case.
             * This occurs when input1 contains two duplicate elements placed side-by-side
as "ll" in "hello" while
             * they may not be side-by-side in input2 such as "lohel" but are still valid
rotations.
             Eg: "hello" "lohel"
            */
            if (input1.charAt(i) == input2.charAt(j)) {
                i--;
            }
            i++;
        }
    }
    if (count == 0)
        return true;
    return false;
}

public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.println(new StringRotation().isRotation("harry potter",
            "terharry pot"));
    System.out.println(new StringRotation().isRotation("hello", "llohe"));
    System.out.println(new StringRotation().isRotation("hello", "lohell"));
    System.out.println(new StringRotation().isRotation("hello", "hello"));
    System.out.println(new StringRotation().isRotation("hello", "lohe"));
}
```

answered Sep 21 '14 at 22:43
user1556718
**27** 6

---

You can do it in `O(n)` time and `O(1)` space:

```
def is_rot(u, v):
    n, i, j = len(u), 0, 0
    if n != len(v):
        return False
    while i < n and j < n:
        k = 1
        while k <= n and u[(i + k) % n] == v[(j + k) % n]:
            k += 1
        if k > n:
            return True
        if u[(i + k) % n] > v[(j + k) % n]:
            i += k
        else:
            j += k
    return False
```

See my answer here for more details.

answered Dec 1 '15 at 16:07
Padraic Cunningham
**114k** 9 66 122

---

```
    String source = "avaraavar";
    String dest =   "ravaraava";

    System.out.println();
    if(source.length()!=dest.length())
        try {
            throw (new IOException());
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    int i = 0;
    int j = 0;
    int totalcount=0;
    while(true)
    {
        i=i%source.length();
        if(source.charAt(i)==dest.charAt(j))
        {
            System.out.println("i="+i+" , j = "+j);
            System.out.println(source.charAt(i)+"=="+dest.charAt(j));
            i++;
            j++;
            totalcount++;
        }
        else
        {
            System.out.println("i="+i+" , j = "+j);
            System.out.println(source.charAt(i)+"!="+dest.charAt(j));
            i++;
            totalcount++;
            j=0;
        }
        if(j==source.length())
        {
            System.out.println("Yes its a rotation");
            break;
        }
        if(totalcount >(2*source.length())-1)
        {
            System.out.println("No its a rotation");
            break;
        }
    }
```

answered Jun 2 '16 at 13:26
ukdaga
1

---

This can be done in O(1) time and O(n) space (C#). Please note the actual search is done in O(1) time.

- Preprocess the source string in n iterations, where n is the length of the source, and add the result as a key to a dictionary.
- Return true if the pattern exists in the dictionary and false otherwise.

Sample code:

```
if (string.IsNullOrWhiteSpace(source) && string.IsNullOrWhiteSpace(rotated)) return true;

Dictionary<string, int> s = new Dictionary<string, int>();
string t = source;
StringBuilder builder = new StringBuilder();
for (int i = 0; i < source.Length; i++)
{
    string a = t[t.Length - 1] + builder.Append(t, 0, t.Length - 1).ToString();
    if (!s.ContainsKey(a)) s.Add(a, 0);
    t = a;
    builder.Clear();
}

return s.ContainsKey(rotated);
```

edited Feb 2 at 6:44
Pang
5,564 14 48 79

answered Feb 2 at 6:13
yellowcountry
1

It's more like O(n) time, since key hashing and comparing algorithm will need to iterate `rotated` . And it's more like O(n*n) space since you're storing all circular rotations of a string. – default locale Feb 2 at 6:46