Caesar Cipher in javascript

Posted on February 9, 2019 | by Prashant Yadav

Posted in Algorithms, String | Tagged Easy

An algorithm to solve the Caesar Cipher problem.

Caesar Cipher: An earlier encryption technique which used to substitute the current alphabets with alphabet after a number of count.

Example

```
Input:
text = ABCD , Key = 13
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
13 shift to A is N
13 shift to B is O
13 shift to C is P
13 shift to D is Q

Output:
NOPQ
```

We will implement a simple algorithm with different approaches to implement Caesar cipher. Everything will be written in ES6.

First Approach

Implementation

- We will create an object with decoded letter for every alphabet.
- Then we will loop through the string and creat the deciphered string with the corresponding decoded letters.

Practically prepare for your JavaScript interview

JavaScript Revision
JavaScript-Concept Based Problems
Data Structures
Algorithms
Machine Coding
Web Fundamentals

```
let ceaserCipher = (str) => {
  //Deciphered reference letters
  let decoded = {
    a: 'n', b: 'o', c: 'p',
    d: 'q', e: 'r', f: 's',
    g: 't', h: 'u', i: 'v',
    j: 'w', k: 'x', l: 'y',
    m: 'z', n: 'a', o: 'b',
    p: 'c', q: 'd', r: 'e',
    s: 'f', t: 'g', u: 'h',
    v: 'i', w: 'j', x: 'k',
    y: 'l', z: 'm'
  }

  //convert the string to lowercase
  str = str.toLowerCase();

  //decipher the code
  let decipher = '';
  for(let i = 0 ; i < str.length; i++){
    decipher += decoded[str[i]];
  }

  //return the output
  return decipher;
}
```

```
Input:
console.log(ceaserCipher('attackatonce'));
console.log(ceaserCipher('prashantyadav'));

Output:
"nggnpxngbapr"
"cenfunaglnqni"
```

Time complexity: O(n).

Space complexity: O(1).

Time and Space complexity

- We are just deciphering each letter of the string, so Time complexity is O(n).
- We are using constant space, so Space complexity O(1).

This approach is good but there are few problems with this method.
1). The cipher is fixed for 13 letter substitution.
2). Also, we are just doing it for lowercase letters.

Second Approach

Implementation

- We will solve the above problem with different keys or dynamic keys and mathematical calculation.

- We will use string inbuilt methods and regular expressions.

```
let caesarCipher => (str, key) {
  return str.toUpperCase().replace(/[A-Z]/g, c =>
String.fromCharCode((c.charCodeAt(0)-65 + key ) % 26 + 65));
}
```

```
Input:
console.log(ceaserCipher('ATTACKATONCE', 13));
console.log(ceaserCipher('PRASHANTYADAV', 13));

Output:
"NGGNPXNGBAPR"
"CENFUNAGLNQNI"
```

Time complexity: O(n).

Space complexity: O(1).

Time and Space complexity

- We are just deciphering each letter of the string, so Time complexity is O(n).
- We are using constant space, so Space complexity O(1).

This approach is also good but there is still one problem with this method. It is still not handling case sensitive strings.

Handling Case Sensitive

Implementation

- We will loop through each letter of the string and check if its uppercase or lowercase.
- Then we will decipher it accordingly using mathematical computation.

```
//check if letter is uppercase
function isUpperCase(str) {
    return str === str.toUpperCase();
}

//decipher the string
let ceaserCipher = (str, key) => {
  let decipher = '';

  //decipher each letter
  for(let i = 0; i < str.length; i++){

    //if letter is uppercase then add uppercase letters
    if(isUpperCase(str[i])){
      decipher += String.fromCharCode((str.charCodeAt(i) + key - 65) % 26 + 65);
    }else{
      //else add lowercase letters
      decipher += String.fromCharCode((str.charCodeAt(i) + key - 97) % 26 + 97);
    }
  }


  return decipher;
}
```

```
Input:
console.log(ceaserCipher('ATTACKATONCE', 13));
console.log(ceaserCipher('prashantyadav', 13));

Output:
"NGGNPXNGBAPR"
"cenfunaglnqni"
```

Time complexity: O(n).

Space complexity: O(1).

Time and Space complexity

- We are just deciphering each letter of the string with mathematical computation, so Time complexity is O(n).
- We are using constant space, so Space complexity O(1).

Learn more about the String.fromCharCode() and charCodeAt().

Recommended Posts:

Count all substrings having character k.

Javascript program to find leap year

Median of two sorted arrays

Find square of a number without using *, / and pow()

Fractional knapsack problem
Find the biggest perfect square in an array
Program to reverse a queue
Best way to compare strings in javascript
Print right view of a binary tree
Absolute difference between diagonals of matrix

Prev Next

Comments

Gorreri Franco says:

November 25, 2021 At 9:23 Pm

Hi! Great post!

I have a question in the second approach. Why you have to subtract 65 before adding the k ey and then doing the mod 26 and finally adding 65?

String.fromCharCode((c.charCodeAt(0)-65 + key) % 26 + 65)

Why can’t i just do?
String.fromCharCode(c.charCodeAt(0) + key)

Reply

chars says:

January 11, 2021 At 11:55 Am

I r eally like your blog.. v e ry nice colors & theme.

Did you design this website yours e lf or did you hire someone to do it for yоu?

Plz reply as I’m looking to construct my
own blog and would like to find out where u got this from.

kudos

Reply

Prashant Yadav says:

January 11, 2021 At 12:22 Pm

I designed it myself

Reply

Bill says:

January 5, 2020 At 1:00 Pm

Hi,

This is great! Thanks for posting, I found it very useful.

What would you do to handle negative key values?

Reply

Prashant Yadav says:

January 5, 2020 At 1:04 Pm

We are deciphering alphabets not numbers so there are no negative numbers i gues s.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Start typing...

Name*

Name

Email*

Email

POST COMMENT

Handcrafted with ♥somewhere in **Mumbai**