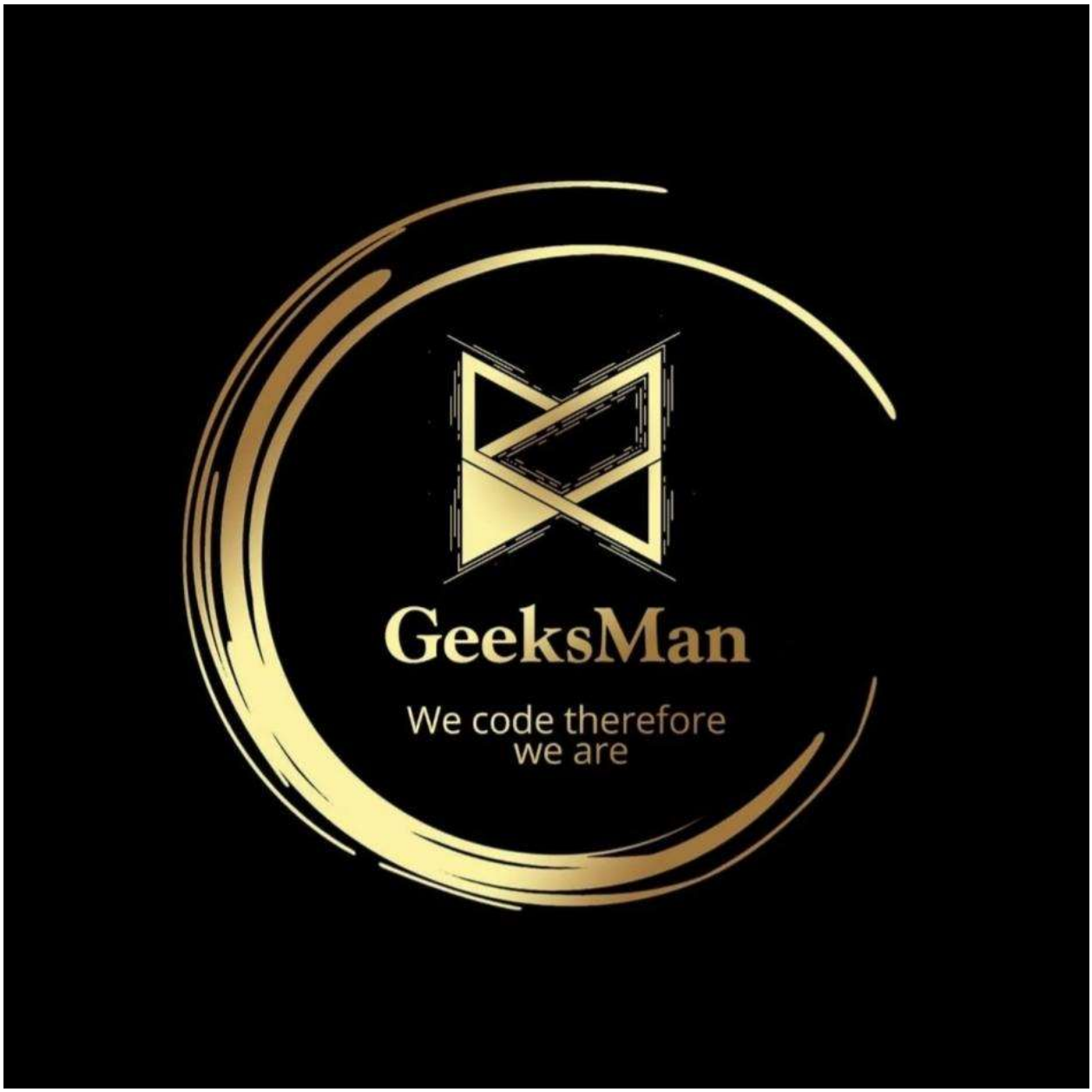
GeeksMan

Tries Lesson 1

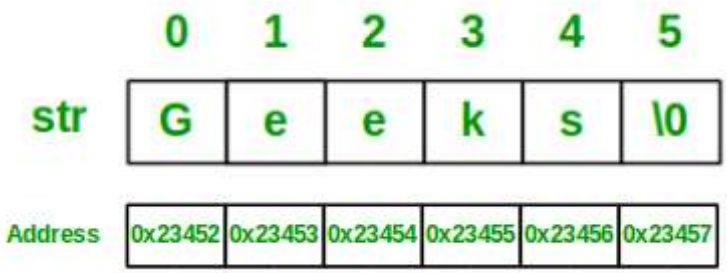


String data structure

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

Character array:

Declaration: char str_name[size];



String :String class stores the characters as a sequence of bytes with a functionality of allowing access to single byte character.

Character array vs strings:

- A character array is simply an array of characters can terminated by a null character. A string is a class which defines objects that be represented as stream of characters.
- Size of the character array has to allocated statically, more memory cannot be allocated at run time if required. Unused allocated memory is wasted in case of character array. In case of strings, memory is allocated dynamically. More memory can be allocated at run time on demand. As no memory is preallocated, no memory is wasted.

- There is a **threat of** array decay in case of character array. As strings are represented as objects, **no array decay** occurs.
- Character array **do not offer** much **inbuilt functions** to manipulate strings. String class defines **a number of functionalities** which allow manifold operations on strings.

STRING DECLARATION AND INITIALIZATION:

```
string str1 = "Hello";
```

OPERATIONS ON STRINGS:

1. **getline()** :- This function is used to **store a stream of characters** as entered by the user in the object memory.

2. **push_back()** :- This function is used to **input** a character at the **end** of the string.

3. **pop_back()** :- This function is used to **delete the last character** from the string.

4. **capacity()** :- This function **returns the capacity** allocated to the string, which can be **equal to or more than the size** of the string.

5. resize() :- This function **changes the size of string**, the size can be increased or decreased.

6.length():-This function **finds the length of the string.**

CHECK IF A STRING IS SUBSTRING OF ANOTHER STRING

Given two strings s1 and s2, find if s1 is a substring of s2. If yes, return the index of the first occurrence, else return -1.

EXAMPLE:

Input: s1 = "for", s2 = "geeksforgeeks"

Output: 5

Explanation:

String "for" is present as a substring of s2.

ALGORITHM:

Simple Approach: The idea is to run a loop from start to end and for every index in the given string check whether the sub-string can be formed from that index. This can be done by running a nested loop traversing the given string and in that loop run another loop checking for substring from every index.

TIME COMPLEXITY: O(m * n) where m and n are lengths of s1 and s2 respectively. A nested loop is used the outer loop runs from 0 to N-M and inner loop from 0 to M so the complexity is O(m*n).

SPACE COMPLEXITY: O(1).

CODE:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
int isSubstring(string s1,string s2)
{
    int m=s1.length();
    int n=s2.length();
    int i,j;
    for(i=0;i<=n-m;i++)
    {
        for(j=0;j<m;j++)
        {
            if(s2[i+j]!=s1[j])
                break;
        }
        if(j==m)
            return i;
    }
```

```
19.   return -1;
20.   }
21. int main()
22.   {
23.       string s1 = "man";
24.       string s2 = "geeksman";
25.       int res = isSubstring(s1, s2);
26.       if (res == -1)
27.           cout << "Not present";
28.       else
29.           cout << "Present at index " << res;
30.       return 0;
31.}
```

Sapphire link: https://sapphireengine.com/@/fogreb

HOW TO SOLVE THIS QUESTION IN O(N) ?

Using KMP algorithm(searching algorithm).

KMP ALGORITHM(for finding whether a string is a substring of another or not in O(N) time complexity):

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:

Input: txt[] = "THIS IS A TEST TEXT"
pat[] = "TEST"
Output: Pattern found at index 10.

ALGORITHM:

- KMP algorithm preprocesses pat[] and constructs an auxiliary lps[] of size m (same as size of pattern) which is used to skip characters while matching.
- name lps indicates longest proper prefix which is also suffix. A proper prefix is prefix with whole string not allowed. For example, prefixes of "ABC" are "", "A", "AB" and "ABC". Proper prefixes are "", "A" and "AB". Suffixes of the string are "", "C", "BC" and "ABC".
- We search for lps in sub-patterns. More clearly we focus on sub-strings of patterns that are either prefix and 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].

CODE:

```
#include <bits/stdc++.h>
void computeLPSArray(char* pat, int M, int* lps);
void KMPSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int lps[M];
    computeLPSArray(pat, M, lps);
    int i = 0;
    int j = 0;
    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];
            }
        else if (i < N && pat[j] != txt[i]) {
            if (j != 0)
                j = lps[j - 1];
```

```
24.             else
25.                 i = i + 1;
26.         }
27.     }
28.   }
29.   void computeLPSArray(char* pat, int M, int* lps)
30.   {
31.  int len = 0;
32.     lps[0] = 0;
33.     int i = 1;
34.     while (i < M) {
35.         if (pat[i] == pat[len]) {
36.             len++;
37.             lps[i] = len;
38.             i++;
39.         }
40.         else // (pat[i] != pat[len])
41.     {
42.             if (len != 0) {
43.                 len = lps[len - 1];
44.             }
45.             else // if (len == 0)
46.             {
47.                 lps[i] = 0;
48.                 i++;
49.             }
```

```
        }
    }
    }
    int main()
    {
        char txt[] = "ABABDABACDABABCABAB";
        char pat[] = "ABA";
        KMPSearch(pat, txt);
        return 0;
    }
```

CONSTRUCTION OF LPS:

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].

pat[] = "***AAACAAAA***"

len = 0, i = 0.
lps[0] is always 0, we move
to i = 1

len = 0, i = 1.
Since pat[len] and pat[i] match, do len++,

store it in lps[i] and do i++.
len = 1, **lps[1] = 1**, i = 2

len = 1, i = 2.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 2, **lps[2] = 2**, i = 3

len = 2, i = 3.
Since pat[len] and pat[i] do not match, and len > 0,
set len = lps[len-1] = lps[1] = 1

len = 1, i = 3.
Since pat[len] and pat[i] do not match and len > 0,
len = lps[len-1] = lps[0] = 0

len = 0, i = 3.
Since pat[len] and pat[i] do not match and len = 0,
Set **lps[3] = 0** and i = 4.
We know that characters pat
len = 0, i = 4.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 1, **lps[4] = 1**, i = 5

len = 1, i = 5.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 2, **lps[5] = 2**, i = 6

len = 2, i = 6.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.

len = 3, **lps[6] = 3**, i = 7

len = 3, i = 7.
Since pat[len] and pat[i] do not match and len > 0,
set len = lps[len-1] = lps[2] = 2

len = 2, i = 7.
Since pat[len] and pat[i] match, do len++,
store it in lps[i] and do i++.
len = 3, **lps[7] = 3**, i = 8

We stop here as we have constructed the whole lps[].

LONGEST PREFIX SUFFIX:

Given a string of characters, find the length of the longest proper prefix which is also a proper suffix.

Example 1:

Input: s = "abab"
Output: 2
Explanation: "ab" is the longest proper
prefix and suffix.

Ques link: https://practice.geeksforgeeks.org/problems/longest-prefix-suffix2527/1

CODE:

```
1. #include <bits/stdc++.h>
2. using namespace std;  //User function template for C++
3. class Solution{
```

```
4.  public:
5.      int lps(string s)
6.        {
7.         int n=s.length();
8.         int lps[n];
9.         lps[0]=0;
10.        int len=0;
11.        int i=1;
12.        while(i<n)
13.        {
14.         if(s[i]==s[len])
15.         {
16.         len++;
17.         lps[i]=len;
18.         i++;
19.         }
20.        else{
21.        if(len!=0)
22.        {
23.        len=lps[len-1];
24.        }
25.        else{
26.         // len=0;
27.        lps[i]=0;
28.        i++;
```

```
        }
        }
        }
        int res=lps[n-1];
        return res;
        }
};

// { Driver Code Starts.

int main()
{
        ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    cout.tie(NULL);
        int t;
        cin >> t;
        while(t--)
        {
                string str;
                cin >> str;
                Solution ob;
                cout << ob.lps(str) << "\n";
        }
    return 0;}
```