

28/02/2024

Project Plan	
Objectives	

Day 2 OF DSA

* Tasks:

check # string
Box

- Basic operations : Concatenation, Substring search.

- Pattern Matching, Brute Force, Naive pattern matching

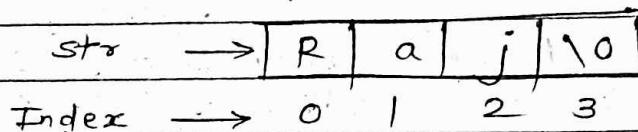
- Implement string operations : Concatenation, substring search

- Solve problems related to pattern matching

* String :

- It is a sequence of characters used to represent text.

string str = "Raj"



* String Operations:-

- ① Concatenation - Combining two strings
- ② Length - Determining no. of characters in str
- ③ Access - Accessing individual characters in string by Index
- ④ Substring - Extracting portion of a string
- ⑤ Comparison - Comparing two strings to check equality
- ⑥ Search - Finding position of a specific substring
- ⑦ Modification - Changing or replacing characters within a string

* String Declaration:-

string str1 = "Hello World!";

string str2 ("Hello Buddy!");

Program	
Output	

* General operations on string

- (1) Concatenation of strings - Combining more than one string.

"Hello" + "World" = "Hello World"

↑ ↑ ↑
str1 str2 result

* Methods -

(1) Append() - str1.append(str2);

(2) '+' operator - result = str1 + str2;

(3) strcat() - strcat(str1, str2);

(4) Using For Loop -
 For(0 → '\0') { result += str1[i]; }
 For(0 → '\0') { result += str2[i]; }

(2) find in string - Find something in given string

str = "Hello World";

str1 = "Hello";

size_t result = str.find(str1);

- Find occurrence of a character -

str = "Hello"; char c = "H";

• size_t result = str.find(c);

result = str.find(c, result + 1);

- Search for partial string.

String str = "Hello World";

size_t found = str.find("Hello", 0, 5);

(3) *

Substring :-

From to



string result = s1.substr(0, 3);

↑ . ↑

NEW str. find a substring in s

- Print all substring in string:-

str. str.length.

void substring (string & s, int n) {

 for (i = 0 → n) {

 for (j = i → n - i) {

 s.substr(i, len)

}

}

*

Some Functions :- to Note

stoi (str) / string to integer.

(4) *

Finding length of string:-

① str.size()

② str.length()

③ Using while loop

④ - IT - For loop

⑤ strlen (str.c_str());

Praya Mo.		
Dera		

* Check if a string is substring of another -

S1 = "Hello"

S2 = "World Hello"

OUTPUT : 6

① Functions -

int M = S1.length(); "Raj" - S1

int N = S2.length(); "Kashid Raj" - S2

For(0 → N-M) {

 int j;

 For(j = 0 → M) {

 if (S2[i+j] == S1[i]) K R X

 break;

 R - R ✓

 A - A ✓

 J - J

 if (j == M) return j; Mean we get our

 substring

 return -1;

Time complexity = O(M*N)

Space complexity = O(1)

② Using Built-in Method:

```
if ( S2.Find(S1) != string::npos )
    return S2.Find(S1);
return -1;
```

Time complexity : O(N)

Space complexity : O(1)

(5) Trim a string

if remove spaces bet' the string

`str = " Raj kashid"`

`result = "Rajkashid";`

With simple I -

`string result;` In which spaces
For(char ch; str) { are their

case IF(ch == ' ') {

`result += ch;`

y

`return result;`

Time complexity = $O(N)$

Space complexity = $O(1)$

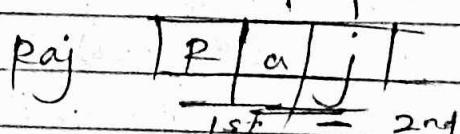
(6) Reverse & Rotation of a string

(1) Rotation

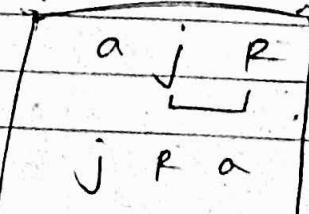
Take Temp variable

1st copy 2nd part \rightarrow temp

2nd copy 1st part \rightarrow temp



`temp = j R a`



```

int len = strlen(str);
char temp[len];
for (0 → len)
{
    int j = i;
    int k = 0;
    while (str[j] != '\0')
    {
        temp[k] = str[j];
        k++;
        j++;
    }
}

```

R A J

$i \rightarrow 2$

temp = AJ

$j = 0;$

while ($j < i$)

$\downarrow \quad 2$

temp[k] = str[j]; temp = AJR

$j++;$

$k++;$

\downarrow

printf("cout << temp ;

Time complexity : $O(N^2)$

Space complexity : $O(N)$

* Reverse in strings:-

Way 1 - `for(int i=0; i < n/2; i++)`

\downarrow
`swap(str[i], str[n-i-1]);`

Way 2 - `for(int i = str.length() - 1; i >= 0; i--)`
 \downarrow
`cout << str[i];`
 \downarrow

Way 3 -

`reverse(str.begin(), str.end());`

Way 4 - `for(i=0, j=n-1; i < j; i++, j--) {`
`swap(str[i], str[j]);`

Way 5 -

`stack<char> st;`

`for(0 → str.length()) {`
`st.push(str[i]);`
 \downarrow

`for(0 → str.length())`
`str[i] = st.top();`
`st.pop();`
 \downarrow

* KMP Algorithm -

String - a b a b c a b c a b a b a b d
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern = j	0	1	2	3	4	5	LPS Index
	a	b	a	b	d		
	0	0	1	2	0		

1) we find out LPS of pattern

2) Suppose $i=1$ & $j=0$

3) we compare $i == j+1$. till we don't find mismatch.

& move $i++$ & $j++$

4) where we don't find match then

At j th place what is the LPS index
 check it & we move our j to

the array's index using LPS index

5) Repeating this 3rd & 4th step

6) After completing traversing if we find we have cover pattern
 then we can say that our pattern is in string

* Rabin-Karp Algorithm:-

- 1) 1st calculating hash value for substring & checking is it equal to our pattern hash code.
- 2) IF they match then we check is it they are the original once means the checking character one by one if they Not then it is called Spurious hit
- 3) Otherwise your pattern is present in string.

(Just pseudocode explainat.)

Time Complexity :- $O(N)$
Space Complexity : $O(1)$

PAGE NO.	/ /
DATE	/ /

- 1) Check whether two strings are anagrams of each other by storing all characters in Hash Map:

Approach:-

1) We count the frequency of characters in the string 1 using Hash Map

2) And after that Using that frequency count we traversed the map & check for string 2's freq characters
if character is present then we decrease the count by -1

3) And Finally we traversed in map, if we get all values to be 0
then it is Anagram else not

1) if ($a.length() \neq b.length()$) return false
 $abc \neq abc$

2) unordered_map<char, int>Map;

Freq. Count 3) For ($0 \rightarrow a.length()$) { $Map[a[i]]++$; }

4) For ($0 \rightarrow b.length()$) {

if ($Map.find(b[i]) \neq Map.end()$) {

$Map[b[i]] -= 1$;

} else { return False; }

5) For (auto items : Map) {

if (items.second != 0) { return false; }

else :

return True

6) return true

2) Find the first repeated character in

Input str = "hello"

Output = 1

1 is the 1st element that repeats.

Approach:-

1) Declaring two variables

ans, & index = INT_MAX;

2) Then we loop the string.

& store that character in temp
by i+n iteration

3) After we again loop another loop which $j = i + 1$
is present inside the main loop.

& check if that is equal to temp.
If it is then it will store in
index & that character
store in ans

4) At the end we return ans

$\text{str} = \text{aabcb}$
 $i \quad j$
 $0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5$

(1) $i = 0$

$\text{temp} = a$

$j = 1$

$\text{str}[1] == \text{temp}$

$a == a$

(1st Index)

$\text{Index} = 1$

$\text{any} = a$

(2) $i = 1$

$\text{temp} = a$

$j = 2$

$b \text{ str}[2] == \text{temp}$

$b == a \quad X$

loop break;

(3) $i = 2$

$\text{temp} = b$

$\text{str}[j] = 3$

$c == b \quad X$

loop break;

(4) $i = 3$

$\text{temp} = c$

$j = 4$

$b == c \quad X$

loop break;

(5) $i = 4$

$\text{temp} = b$

$j = 5$

$X \quad \text{loop break}$

3) First non-repeating character
of given string

Approach:

1) simply iterate through string check
each character's count

2) If count == 1 then that is our
non-repeating character

```
for (auto i : string) {
    if (count(string.begin(), string.end(),
              i) == 1) {
```

fnc = i;

break;

} else {

index++ = 1;

} if (index == string.size() - 1) {

All are repeating

} else {

First non-repeating is: fnc

}



lexicographic rank of a string:-

Approach:-

1) First we building two function

1) Factorial

2) Find Rank

2) factorial Function is recursive funct'.

3) In Find Rank we check the rank of string
we use 2 loops.

1st loop for iterating all string characters

2nd loop for counting the character
calculating

Count who is greater than other character

3) After inside loop run

we calculate rank by

$$\text{rank} += \text{count} * \text{Fact}(n - i - 1)$$

$i = 0$	$n = 5$	Rough
1	4	
2	3	
3	1	
5	0	

4) Return rank at the end.

Time complexity : $O(N^2)$

Space complexity : $O(1)$

5) Palindrome check:-

* Approach:-

1) Take low = 0
high = n - 1;

2) Take while loop
and check if str[low] != str[high]
return False;

3) Else move low ++ & high --;

Time complexity = $O(N)$
space complexity = $O(1)$