

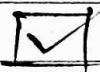
3/3/2024

PAGE NO.	
DATE	11

Day 6 OF DSA

Tasks

checkbox strings



string Reversal & Palindrome Detection:
Basic Approaches.



Regular Expressions : Basic Syntax, simple
pattern Matching.



Implement string reversal & palindrome
detection Algorithm.



Solve problems related to regular
Expressions.

* Different Methods to Reverse a string in C++.

① Custom Function:

```
void reversestring ( string & str)
{
    int n = str.length();
    for( int i=0 ; i<n/2 ; i++)
        swap( str[i] , str[n-i-1]);
}
```

..... abc

0 1 2

① i=0 : a - b - c - d - e - f

swap(a,c) : c - b - a -

② i=i+1 (i<n/2) : 3/2 = 1 < 1 X

Time Complexity : O(N)

Space Complexity : O(1)

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

PAGE NO.	11
DATE	

- ② Using recursion function with two pointer approach.

a b c d e F g h

i ++ -- n

 |

Swapping.

length - 1) 0

* void reverse (string & str, int n, int i)
if (n <= i) { return ; }

swap (arr [i], arr [n]);

reverse (str, n - 1, i + 1);

)

① | a | b | c | d | e | F | g | h |

i ++ -- n

② | h | b | c | d | e | F | g | a |

i ++

-- n

③ | h | g | c | d | e | F | g | a |

i ++ -- n

i

④ | h | g | F | d | e | c | b | a |

⑤ | h | g | F | e | d | c | b | a |

✓

✓

✓

Page No.	
Date	/ /

(3) Inbuilt reverse Function:

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

$$\begin{array}{|c|c|} \hline T+C & O(N) \\ S-C & O(1) \\ \hline \end{array}$$

(4) Reverse a string Using the constructor:

```
string str = "RAT";
```

```
string rev = string(str.begin(), str.end());
```

(5) Using a Temporary string L:

```
string str = "Raj";
string rev;
for(int i=n-1; i>=0; i--)
{
    rev.push_back(str[i]);
}
```

* Palindrome Check Approaches :-

NAMAN → original
 reversed → NAMAN } same

that's called palindrome.

① Using In-built reverse Function

- ① Take a string variable as $p = \text{original str}$
- ② reversed it using $\text{reverse}(p.begin(), p.end())$
- ③ Compare it with original,

IF ($\text{str} == p$) . . . Yes
 else . . . No

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

PAGE NO.	
DATE	11

(2)

By Traversing the string:

Just we traverse half of the string

& every time we check

1st & last character

• If it is ~~compar~~ at half of the
 until string

then return "Yes"

else

return "No"

Time Complexity: $O(N)$

Space complexity: $O(1)$

* Regex (Regular Expression) :

① regex-match()

string a = "Raj kashid";

regex b ("(Raj)(. *)");

Class obj AFTer Raj

"kuch bhi ho satta
hai"

if(regex_match(a,b))

cout << "String 'a' matches

Regular Expression 'b'"

if(regex_match(a.begin(), a.end(), b))

Count if "String 'a' matches with regex exp. 'b'
in the range from 0 to end

Page No.	
Date	11

(2)

regex_search() :- Used to search for a pattern matching the regular expression.

string s = "RajStarKid";

regex::smatch m;

Checks: Alphanumeric Uppercase & lowercase Characters from string

Match m storing the result

regex_search(s, m, r);

Main str. to find result

(3)

regex_replace()

Used to replace the pattern matching to the regular expression with a string

```
String s = "Rajskukkuy";
```

```
regex r ("Raj([a-zA-Z]+)");
```

```
cout << std::regex_replace (s, r, "raj");
```

```
string result;
```

```
regex_replace (back_inserter(result),
```

```
s.begin(), s.end(), r,
```

```
"raj");
```

```
cout << result;
```

* Pattern Searching : Linear Search

We are using Find Function.

`int Found = txt. Find (pat);`

↓↓↓↓↓
main string pattern.

return index where
pattern is.

`while (Found != string::npos) {`

Jab tak string has..

↑
tak tak ..

cout << "pattern found at index " <<
found << endl;

`Found = txt. Find (pat, found + 1);`

har bar hum aage check karte
jaise kisi ifahang mein

y,

Time Complexity : $O(NM^2)$

space complexity : $O(N + M)$

txt.length() pat.length()

①

Reverse a string!

Just Noticed me:

ek left hai jo 0 pe ruka hai.

Aur ek right hai jo n-1 pe ruka hai
to unha migya hai.

So auto hum kya karenge

while (left < right) {

swap(s[left], s[right]);

left++,

right--;

y

Time Complexity : O(n)

Space Complexity : O(1)

(2) Anagram Detection:

Approach:-

- 1) we take a map to count the frequency of characters in the original string.
- 2) & then for checking we reduce the frequency of count by taking string, and when
- 3) If all strings' count become 0 then we return true.

Time Complexity: $O(n)$

Space Complexity: $O(n)$

③

Palindrome Detection

Approach

1) First we lowercase all string

all non-alphanumeric characters

2) Then left = 0

right = $m - 1$

3) Moving loop while ($left < right$)

4) If we find any other from
non-alphanumeric then
we skip it.

$left \rightarrow$ $\leftarrow right$

5) Otherwise we compare
left right element
& move left -- &
right ++

If not return false.

Time complexity is $O(n)$

Space complexity is $O(1)$

(+)

String Compression

"wwwaaaccc"

"w3a3c3"

String compression.

Approach:

① Initialize string compressed & int count = 1;

② Traverse whole string.

③ if (i + 1 < s.length && s[i] == s[i + 1])
then count ++;

④ else

we put that character & count
in compressed &
& reset count = 1;

⑤ print compressed

Time Complexity: $O(n)$
Space Complexity: $O(n)$



(5)

Largest substring without repeating characters

Approach 1

① We initialized unordered-set
to store freq count of characters
& left = 0, right = 0, max-length = 0.

② we traverse the string.

& check every character
if it is present or not

If not then we insert in set
& find out max length
& right++

If yes then erase it

& move left++

③ At the end return maxlength

Time Complexity : O(n)

Space Complexity : O(min (n, m))

PAGE NO.	
DATE	/ /

(6)

Valid parenthesis - .

Approach:-

- ① Initialize stack `stack stk;`
- ② traverse the string.
- ③ If character is '`'C'`' or '`'S'`' or '`'[`' push into stack.
- ④ else
 - if `c = ')'` or `stk.top is not 'C'`
 - $c = '}' - \text{if } c = '}' - \text{if } c = '}'$
 - then again return False
- ⑤ `stk.pop()`
- ⑥ otherwise return `stk.empty()`

Time Complexity : $O(n)$

Space Complexity : $O(n)$

7)

strings to integer

$$'42' = 42$$

check

① First remove whitespace & signs + or -
 int result = 0;

② traverse strings

store int digit = str(i) - '0'

int - int

③ result = result * 10 + digit;

for (i++)

④ return result + sign;

Time Complexity : O(n)

Space Complexity : O(1)

Page No.	
Date	/ /

⑧

Implement strstr();

Q12 :-

haystack = "Hello" needle = "ll"

Output = 2

Approach:-

① if (needle is empty then return 0)

② m = haystack length

n = needle length

③ traverse in 1st loop from i=0 → m-n

in that declare int j;

& another loop from j=0 → n

& check (haystack[i+j] == needle[j])
break)

④ & AFTER inner loop, if i+j == n then
return i.

⑤ else return -1;

Time complexity: $O(m-n+1)$

Space complexity: $O(n)$

Time complexity: $O(2^n)$

Space complexity: $O(2^n)$

PAGE NO.

DATE

11

①

Count & say.

It is like

Fibonacci but in

- 1) 1
- 2) 11
- 3) 21 \leftarrow input = 3
- 4) 1211
- 5) 111221 \leftarrow -11- = 5

Count = value

Approach -

① if $n = 1$, return 1

Recursive function

② string prev = CountAndSay(n-1), ..

③ string result = "";

④ char curr = prev[0], ..

⑤ int count = 1, ..

⑥ traverse prev string.

if ($prev[i] == curr$) count++

else {

result += to_string(count) + curr;

curr = prev[i]

count = 1, ..

y:

⑦ result += string(count) + curr;

return result;

(10)

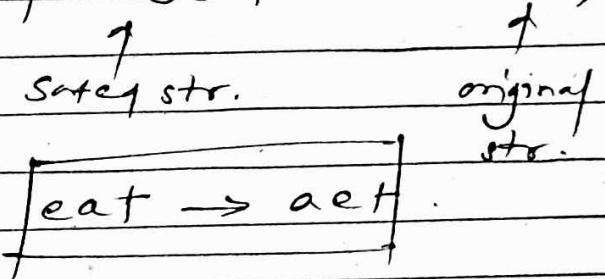
Group Anagrams! -

Approach:-

(1) Initialized unordered_map groups.

(2) traversed vectors of string
then take a new string & put
every str in that.
sort that string.

& In groups[key].push_back(str);



(3) Take another string 2D vector result.

& store all group of string in it.

(4) & return result.

Time complexity :- $O(n \times k \log k)$
space complexity :- $O(n \times k)$