3/3/2024

PAGE No.
DATE / /

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.

PAGE No.
DATE / /

* 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]);
    }
}
```

a b c
0 1 2

(1) i = 0
swap (a, c) c b a

(2) i = i (i < n/2) 3/2 = 1 < 1 ✗

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

Time Complexity : O(N)
space Complexity : O(N)

PAGE NO.
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

(2) h b c d e F g a

(3) h g c d e F b a

(4) h g F d e c b a

(5) h g F e d c b a

(3) Inbuilt reverse Function:-

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

T.C = O(N)
S-C = O(1)

(4) Reverse a string Using the constructor:-

string str = "RAJ";

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

(5) Using a Temporary string :-

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

PAGE No.
DATE / /

* 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 = original str

② reversed it using reverse(P.begin(), P.end())

③ Compare it with original,

IF (str == P) → Yes
else → No

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

PAGE No.
DATE / /

② By Traversing the string :

Jwtwe traverse half of the string & every time we check 1st & last character. if it is compare half of the until string then return "yes" else return "No"

Time Complexity: O(N)
Space complexity: O(1)

PAGE No.
DATE / /

Regex (Regular Expression):

① regex_match() :-

```
string a = "Rajkashid";
regex b ("(Raj)(.*)");
```

Class obj — AFter Raj "kuch bhi ho sakta hai"

```
if (regex_match (a,b))
{
cout << " String 'a' matches
        Regular Expression 'b' ";
}

if( regex_match(a.begin(), a.end(), b)
{
    Cout << " String 'a' matches with regex exp. 'b'
            in the range from 0 to end
```

PAGE No.
DATE / /

(2) regex_search() :- Used to search for a pattern matching the regular Expression

String s = "RajSrashid";

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

Checks Alphanumeric Uppercase & Lowercase Characters from string

Smatch m; Storing the result

regex_search(s, m, r);

main str | which we have to find result

PAGE No.
DATE / /

(3) : regex_replace() :-

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

```
string s = "Rajskajkid";

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;
```

PAGE No.
DATE 11

Pattern searching :-

We are using Find function :-

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

main string — pattern.

Returns index where pattern is.

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

Jab tak string pai

tab tak

```
Cout<< "pattern found at index "<<
    Found <<endl;
```

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

har baar hum aage check karte

rahenge

```
}
```

Time Complexity : $O(NM^2)$

Space complexity : $O(N+M)$

txt.length() pat.length()

① Reverse a string:

Jst Noticed me:

ek left hai jo 0 pe ruka hai.

Aur ek right hai jo n-1 pe ruka hai

to Unha Mijna hai.

to hum kya karenge

```
while ( left < right)
    swap( s[left] , s[right]);
    left++;
    right--;
}
```

Time Complexity ; O(n)
Space Complexity ; O(1)

PAGE No.
DATE / /

(2) Anagram Detection:-

Approach:-

1) We take a map to count the Frequency of characters in the original string.

2) & then for checking we reduced the Frequency of count by 2nd string.

3) If all strings count become 0 then we return true.

Time complexity : O(n)
Space complexity : O(n)

✓

PAGE No.
DATE / /

③ Palindrome Detection:-

Approach:-

1) First we lowercase all string characters.

2) Then left = 0
right = n-1;

3) Moving loop while (left < right)

4) if we find any other from isalnum then we skip it.

left → ++ ← right --

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

IF Not return false.

Time complexity : O(n)
space complexity : O(1)

PAGE No.
DATE / /

(4) string "Compression"

"WWWaaaccc"

String Compression.

"W3a3c3"

Approach:-

(1) Initialized 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)

✓

PAGE No.
DATE 1 1

(5) Longest substring without repeating characters

Approach :-

(1) We initialized unordered_set to store the count of characters & left = 0, right = 0, max_length = 0.

(2) we traverse the string.
& checks 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) Initialized stack stk or stk;

(2) traverse the string.

(3) if (character is '(' or '{' or '[')
push into stack.

(4) else
stack is empty. return false
if c = ')' or stk top is not '('
c = '}' — " — '{'
c = ']' — " — '['
then again return False

stak.pop();

(5) otherwise return stk.empty();

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

PAGE No.
DATE / /

7) strings to integer

'+2' = +2

check

① First remove whitspaces & sign + or -
int result = 0;

② traverse strings.

store int digit = str[i] - '0'

int - int

③ result = result * 10 + digit;
i++

④ return result * sign;

| Time Complexity : O(n)
| Space Complexity : O(1)

PAGE No.
DATE / /

(8) Implement strStr();

012
haystack = "hello" needle = "ll"
11

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 j == n then return i.

(5) else return -1 j

Time Complexity : O(m-n+1)
Space complexity : O(n)

Time complexity : $O(2^n)$
Space complexity : $O(2^n)$

PAGE No.
DATE / /

(9) Count & Say. It is like Fibonacci but in pattern not in calculation

1) 1
2) 11
3) 21 ← input = 3
4) 1211
5) 111221 ← —11— = 5

Count = value

Approach :-

(1) if n == 1, return 1

Recursive funct°

(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;
}
}
(7) result += to string (count) + curr;
return result;

PAGE No.
DATE / /

(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);

key → sorted str.
str → original str.

eat → aet

(3) Take another string 2D vector result.

& store all group of string in it.

(4) & return result.

Time complexity :- O(n * klogk)
space complexity :- O(n * k)