28/02/2024

Day 2 of DSA

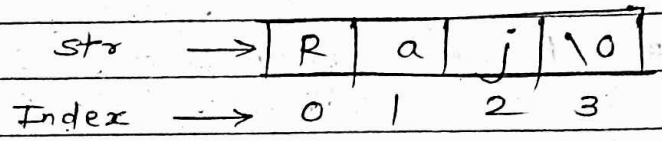
* Tasks:

Check Box	# String
☑	- 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!");
```

* 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 occurence 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 ( 0 -> n) {
    For ( 1 -> 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. -II- For loop
5. strlen (str.c_str());

* 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(); "kaWid 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; Means 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 :-

Remove spaces bet' the string

str = " raj kashid "
result = "rajkashid";

Just simple :-

```
string result;
For( char ch : str) {
    if( ch != ' ') {
        result += ch;
    }
}
return result.
```

En which spaces are there

Time complexity	= O(N)
Space complexity	= O(1)

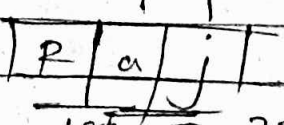
⑥ Reverse & Rotation of a string :

① Rotation

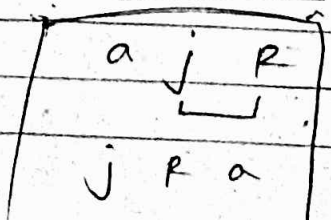
Take Temp variable

1st copy 2nd part → temp
2nd copy 1st part → temp

raj



~~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++;
   }

   j = 0;
   while( j < i)
   {
      temp[k] = str[j];
      j++;
      k++;
   }

   printf( 'cout << temp;
```

R A J

1 → 2

temp = A J

0 → 1 R

2

temp = A J R

Time complexity : O(N²)
Space complexity : O(N)

* Reverse in strings :-

Way 1 -
```
for( int i=0; i<n/2; i++)
{
    swap( str[i], str[n-i-1]);
```

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

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

* 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

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

LPS Index

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'th index using LPS Index

5) Repeating this 3rd & 4th steps

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

PAGE No.
DATE 1 1

* Rabin-Karp Algorithm:-

1) 1st calculating hash value for substring & checking is it equall to our pattern hash code.

2) IF they match then we check is it they are the original once means the checking character one bigone if they Not then it is called Spurious hit

3) Otherwise your pattern is present in string.

(Just pseudocode explaination.

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 HashMap:-

→

Approach :-

1) We count the frequency of characters in the string1 using HashMap

2) And after that using that frequency count we traversed the map & check for string2's ~~freque~~ 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() != b.length()) return false
abc ≠ abca

2) unordered_map<char, int> Map;

Freq. count 3) For(0 → a.length) { Map[a[i]]++; }

4) For(0 → b.length) {
if (Map.Find(b[i]) != Map.end()) {
Map[b[i]] --= 1;
} else { return False; }

decreasing Count

5) For(auto items : Map) {
if (item.second != 0) { return false; }

Checking 0's

else; return True

6) return true

PAGE No.
DATE / /

2) Find the First repeated character in

Input str = "hello".
Output = l
l 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 ith iterat^n

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

PAGE No.
DATE / /

str = aabcb
0 1 2 3 4

① i = 0

temp = a

j = 1

str[1] == temp

a == a

(1 ≤ Index)

a

Index = 1

ans = a

② i = 1

temp = a

j = 2

b str[2] == temp

b == a. ✗

loop break;

③ i = 2

temp = b

~~str~~ j = 3

c == b ✗

loop break;

④ i = 3

temp = c

j = 4

b == c ✗

loop break;

⑤ i = 4

temp = b

j = 5 ✗ loop breaks.

PAGE No.
DATE / /

3) First non-repeating character of given string:-

Approach:-

1) simply iterate through string check each characters 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
}
```

PAGE No.
DATE 1 1

* lexicographic rank of a string:-

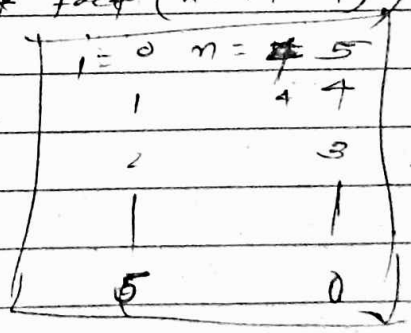
Apprach:-

1) First we building two function
   1) Factorial
   2) FindRank

2) Factorial Function is recursive funct^.

3) In FindRank 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
   rank += count * fact(n-i-1);



4) Return Rank At the end.

Time complexity : O(N²)
Space complexity : O(1)

PAGE No.
DATE / /

5) Palindrome check!-

* Approach!-

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

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

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

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