Strings must do question for placement

String	Reverse a String
String	Check whether a String is Palindrome or not
String	Find Duplicate characters in a string
String	Why strings are immutable in Java?
String	Write a Code to check whether one string is a rotation of another
String	Write a Program to check whether a string is a valid shuffle of two strings or not
String	Count and Say problem
String	Write a program to find the longest Palindrome in a string.[Longest palindromic Substring]
String	Find Longest Recurring Subsequence in String
String	Print all Subsequences of a string.
String	Print all the permutations of the given string
String	Split the Binary string into two substring with equal 0's and 1's
String	Word Wrap Problem [VERY IMP].
String	EDIT Distance [Very Imp]
String	Find next greater number with same set of digits. [Very Very IMP]
String	Balanced Parenthesis problem.[Imp]
String	Word break Problem[Very Imp]
String	Rabin Karp Algo

String	KMP Algo
String	Convert a Sentence into its equivalent mobile numeric keypad sequence.
String	Minimum number of bracket reversals needed to make an expression balanced.
String	Count All Palindromic Subsequence in a given String.
String	Count of number of given string in 2D character array
String	Search a Word in a 2D Grid of characters.
String	Boyer Moore Algorithm for Pattern Searching.
String	Converting Roman Numerals to Decimal
String	Longest Common Prefix
String	Number of flips to make binary string alternate
String	Find the first repeated word in string.
String	Minimum number of swaps for bracket balancing.
String	Find the longest common subsequence between two strings.
String	Program to generate all possible valid IP addresses from given string.
String	Write a program tofind the smallest window that contains all characters of string itself.
String	Rearrange characters in a string such that no two adjacent are same
String	Minimum characters to be added at front to make string palindrome
String	Given a sequence of words, print all anagrams together
String	Find the smallest window in a string containing all characters of another string

String	Recursively remove all adjacent duplicates
String	String matching where one string contains wildcard characters
String	Function to find Number of customers who could not get a computer
String	Transform One String to Another using Minimum Number of Given Operation
String	Check if two given strings are isomorphic to each other
String	Recursively print all sentences that can be formed from list of word lists

(STRINGS)

Page No.:
Date: 3/2/2022
YOUVA

Ques → Reverse a string :-

Given a string, return its reverse

example - Harsh → hasrH.

["H", "a", 'r', 's', 'h']
 0 1   2   3   4 → ['h', 's', 'r', 'a', 'H']

Intution :-
What we are doing here is to simply changin [0] → [n-1], first & last index i++ & j--;

Code :-

```
reverseString (vector<char> &s).
{
    int n = s.size();

for(int i=0; i<n; i++)
    {
        Swap( s[i], s[n-i-1]);
// OR reverse(s.begin(), s.end());
}   cout<< s;
```

Page No.:
Date:

Ques Palindrome String

Given a string S, check for palindrome.

Palindrome : aba => aba ; gog => gog.
=> gfg -> gfg.

~~Intution~~ Intution :-
low & high => if low = high l++ & h--
else false

Algo :-
→ given string → str.
→ Initialize low = [0] & high = [n-1];
→ For loop → ~~if (low[0] == high~~
if (str[low] = str[high])
low++, high--
→ if str[low] ≠ str[high]
return false

Code :-

```
while(h > l) {
    if (str[l++] != str[h--])
{       cout << "false"; }

    cout << "true"
}
```

Page No.:
Date:
YOUVA

Que- Printing all the duplicate in input string

Print all the duplicate and their count in given input string.

Str[] = "test string"
O/P => s, count = 2 ; t, count = 3.

Intution :-
we use map, in which we store each character & its count value.
→ print char, whose count > 0.

algo :- [TC- O(N) (length of str) SC- O(1)]

→ map<char, int> count;
→ for (i=0 → i<size).
co. put string element inside count & increment.
→ another for loop to print character and its count.

Code :-
unordered_map<char, int> count;

for (int i=0; i<str.size(); i++)
count[str[i]]++;

for (auto it : count) // Iterate through map
if (it.second > 1) // If count of char is greater than 1, print
cout << it.first << " " << it.second << "\n";

Page No.:
Date:
YOUVA

Ques- A string are rotation of each other or not ?

2 string s1 & s2, check whether s2 is rotation of s1.

example: s1 = abcd s2 = bcda.
← left by 1.

or → abcd → Right by (3)
bcda ←

Intution.

left = x & right = (n-x)
if left = right → it is rotation
where, x = no. of left rotation
n = no. of char in str.

Method 1: st1 = mypencil st2 = encilmyp

↓ → we'll fix 1st char 'm' which will create a partition
TC- O(n x n) → move m to the right side & add it => ypencilm
=> O(n²) → compare new string with st2, using 'strcmp()' funcⁿ, if equal returns 0.
→ do this for all char, if nothing match return false. →
Case 1) → length of both should equal; (case-2) → match

Method 2 - using KMP also pattern
s1 - mypencil s2 - encilmyp
→ take text string → text = s1+s1 => mypencilmypencil
we'll doing this because if s2 is sub string of text the that means they are matching
eg - mypencilmypencil
3 left after 3 left rotation we found our string == st2;

Page No.:
Date:
YOUVA

Ques Check string is shuffled string of another string.

2 string s1 & s2, find s1 is shuffled form of s2 Yes or no.

Example: st1 = "onetwofour" st2 =
st2 = "Hellofourtwoone world" O/P = Yes
st1 is shuffled insid st2

Intution :-

1) $\underset{s+1}{n} > \underset{s+2}{m}$, then st1 can never be substring of st2.

(2) Traverse st2, put all char of st2 in new str.
→ sort str & compare (str = st1) if.
- if str = st1, then "YES"
- else, repeat till index of st2 as (i + n-1 > m).
(Bcoz after this index lengt of remaining string st2 will be less than st1).
- if str is not equal to st1, then "NO".

TC- (m*n*log(n)) → SC - O(N)

Method 2 - (using 2 count array) + sliding window
- 1st array store freq of char in a pattern.
- 2nd count array store freq of char in current window.
- Store count of freq of patten in countP[] & count of freq of char in 1st window in countTW[].
TC of 2 count array is O(1) bcoz no. of element is fix.
* Run Loop (i = M to N-1)
(a) if countP[] = countTW[], fount the occurrence
(b) Use count of curr char of text in countTW[].
(c) Use count of 1st char. in prev window in countTW[].
(3) Last window is not checkable, do it explicitly

(worst)

Ques- Count and say problem.

number → 3 3 2 2 2 5 1

O/P- 2 ③ 3 ② 1 5 1 1

* Sequence : 1, 11, 21, 1211, 111221, ---

Intution :- else delimeter to check last value just

if 3 3 2 2 2 5 1 $

count, temp;

if (str[j] = str[j-1])
count ++;
else { temp = temp+c;
/* to store that character */ t = t + s[j-1];
count = 1;

- Bcoz, whatever the data there, occurs atleast once

Algo :-

```
countAndSay(int n) {
if (n==1) return "1";
if (n==2) return "11";
 string s = "11";
for (i=3 → i<=n){
 s = s+'&';
to check last value of s
 for (int j=1 → j<s.size(),
  if (s[j] != s[j-1])
   temp += s to_string(c);
   t = t+s[j-1];
  c=1
  else c++;
 s = t; }. return s;
 } ;
```

temp var string
string temp = "";
int c=1; // count
change int to str

Page No.:
Date:
YOUVA

Ques- Longest Palindrome in string

Given string S, find longest palindrome substring.

sub-string - S → S[i--j] => $0 \le i \le j < s.size()$

palindrome : read same from both side.

example :- S = "aaaabbaa"

O/P = aabbaa → Longest pal.

S = "abc",

"a" & 'b' & 'c' → Ans 'a' bcoz it is the one with least index.

Intution :-

2 cases → for even string & odd string

even

caabbabd (2 pointers l & h)

↑ ↑ →
i j

→ caabbabd → cabbabd => abba

↑ ↑ Ans -
i j

(left - 1) ← i → right till end

odd cbabd => cbabd => bab

i = j i = j Ans.

TC - $O(n^2)$

Algo :-

```
while[ l >= 0 && h < s.size() && s[l] == s[h]
{ l = i-1; h = i; // odd  l = i-1 & h = i+1;
if (h-l+1 > end)                    /* code remain
{ start = l; end = h-l+1; }          same for
l--; h++;                            oddpart */.
```

even Part

pointers.

(dp Question)

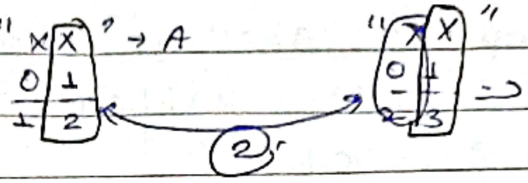
Page No.:
Date:
YOUVA

Ques: Longest Repeating Subsequence

Given, longest repeating subsequence :
2 subsq. don't've same str char at same pos.

example: str = "axxxy" → O/P = 2.

a x x x y ⇒ "xx" → A, "xx" → B



Intution :- (using dp)

S = "aabb" → pass same string 2 times

a a b b ⇒ if(same element found at diff index count++) s[i]==[j] & i ≠ j;

a a b b ⇒ (i = j) ⇒ prev. of i & prev. of j return max(from i/j)

a a b b

TC - (N^2) SC - (N^2)

Algo :-

```
int dp[n+1][n+1];
for(i=0; i<n) {
    for(j=0 → j<n) {
if (i==0 or j==0) dp[i][j] = 0;
else if (s[i-1]==s[j-1] && i!=j)
        dp[i][j] = dp[i-1][j-1]+1;

else
    dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
}
} cout<< dp[n][m];
```

Page No.:
Date:
YOUVA

Ans- Print all subsequence of string :-

Given find all subsequence.
Subsequence is generated by deleting some char from string.

example : 'abc' → a, b, c, ab, bc, ca, abc.

Method-1 :
Iterate entire string, from end to generate diff substr, & add it to the list.
then drop k^{th} char from substr attain. from generating diff. subseq.
If the subseq. is not in list return recurrance.
TC- $O(N^2)$.

→ Method-2
→ 1 by 1 fix char, & recursively generate all subsets. starting from them.
After every recursive call, remove last char so that next permutation can generate.
TC- $O(N)$

Code :-

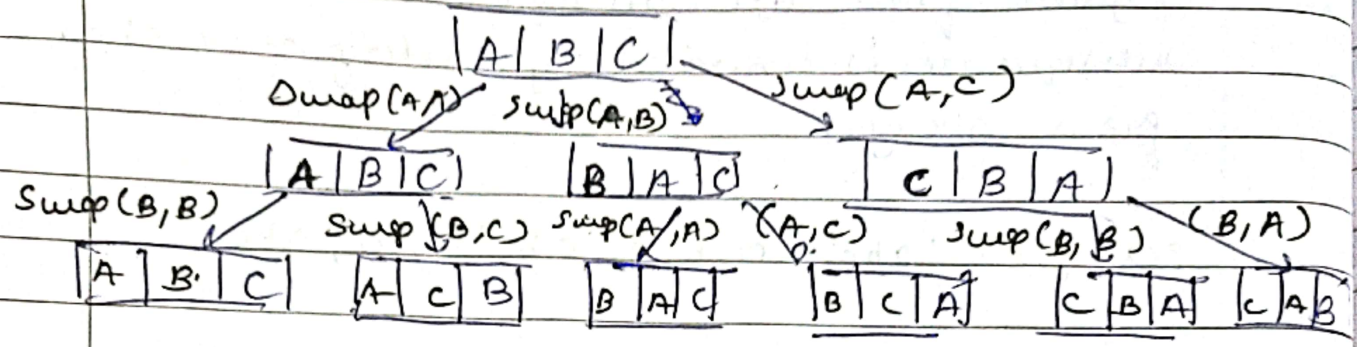
```
if (index == n) return;
if (!curr.empty) return curr;
for (i = index+1; i < n) {
    curr += str[i];
    printSubSeq (str, n, i, curr);
    curr = curr.erase (curr.size()-1);
}
return;
}
```

Backtracking

Page No.:
Date:
YOUVA

Ques- Print all the permutation of given String

S = "ABC"



Intution :-

→ if (i == length == end) cout << str;
else for (i = length → i <= end)
Swap (str[length], str[end]);
permutate (str, length+1, end)
Swap (str[length], str[end]);

3 3 ↓ Back tracking

TC - O(n × n!)

n! = n! permutation require in O(n) space

above soln will print the duplicate permutations also.

Page No.:
Date:

Ques→ Paranthesis Checker

String of expression 'x',
func" should return true for "[(])] {} {[()()]()}"
And false for exp = "[(])"

Intution :-

* Declare a char stack S.
→ Now traverse in expressing string.
° If current character is starting with {, (or { or [
Push into stack
° If current char is ending with) or] or }
pop from stack.
If popped char is matching with starting bracket
then fine else return False

Algo :-

→ Traverse in exp→ for(i=0→ i<length)
{ Push element in stack).
→ If current char is not opening then it must be closing, so stack cant be empty.
if (s.empty()) return false;

→ If (x=='{' || x=='[')
return false
(x=="(" || x=='[') false
(x=='(". || x=='{') false

Page No.:
Date:
YOUVA

Que- Word Break :-

given string A, and dictionary B, find A can be segmented into space seperated sequence of dict. word.

A = 'ilike' B = ["i", "sam", "like", "fun", "samsung", "go"]

Yes, A present in space siperated sequence 1 time.

Intution :- (using DP) & recursion

B=["a", "ab", "bcd", "cd", "d", "de"] A = "abcd"

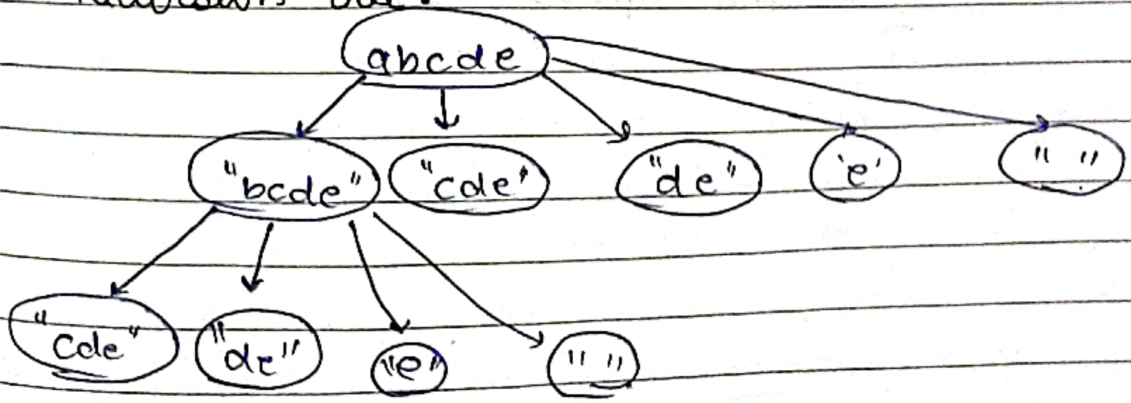
→ pick 'a' → a present, seach for bcd if present return, true.

→ if a not present go for ab, and if ab is present we send cd in recursion, if cd is present then true.

→ if ab not present, go abc, and if abc is present send d in B[], return true

→ if abc not present, search for 'abcd' if present true, else false

→ Riarsion Tree :-



Page No.:
Date:

Pseudo code

```
unordered_map<string, int> dp;
int solve(string s, vector<string>& b){
    if(ssz==0)
        int sz = s.length();
    if (sz==0) return 1;
    if (dp[s]!=0) return dp[s];

for (int i=1; i<=sz; i++){
    int f=0;
    string ss = s.substr(0,i);
    for (int j=0; j<b.size(); j++)
    {   if (ss.compare(b[j])==0){
            f=1; break
        }
    }
    if (f==1 and solve(s.substr(i, sz-i), b)==1)
        return dp[s]=1;
    }

    return dp[s] = -1;
}
```

Page No.:
Date: 6 | 2 | 2022
YOUVA

Ques → Rabin Karp Algorithm :-

given, text[] = [0...n-1] and pattern pat[0...m-1].
write func^n that print all occurance of pat[] in text[] assume n>m
example :-
text[] = "THIS IS A TEST TEXT"
(indices: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18)
Pat[] = "TEST" → Ans - found at ind 10.

* Naive Approch :-
Slide pattern one by one, check each character at current shift if all char matches Then print match.

* Rabin Karp approch
→ It matches hash value of patter with hash val of curr substr. of text. if it matches only then it start matching individual pattern
→ calc. hash value for : (1) Pattern itself (2) Substrings of text

→ Rehashig is done using formulas :-
hash(txt[s+1...s+m] = (d(hash(txt[s...s+m-1]) - txt[s]*h) + txt[s+m]) mod q.

hash(txt[s...s+m-1]) : hash value at shift s
hash(txt[s+1...s+m]) : hash value at next shift (shift+1)
where,
d -> no. of char in alphabet
q → a prime no.
h : d^(m-1)

Page No.:
Date:

Ques → Longest Prefix Suffix (KMP Algo) :-

String, find length of longest prefix & suffix.

example :-

S- a b c d e

Pprefix (lastchar(n-1) not include) a
ab, abc, abcd,

psufix (first char [0] not include)
e, de, cde, bcde

example -: aaa bdaaa

PP
a, aa, aaa,

Ps
a, aa, aaa => 3

Naive approach

add every char. in prefix and check with suffix.

It'll take TC- O(n²) → n for iteration another 'n' for P += "b";

Better Approach :- (KMP algo)

(1) Bypricess → (2) apply algo.

work on given pattern :-

Task → find longest prefix Suffix which are equal :-

LPS 1st

Prefix → a a b c d a a ← Suffix

if → not match we'll go for 2nd LPS i.e (p-1)

→ [ ] [ ] p-1 => a a b c d a a LPS 2

→ a a b d a a => a a b d a a → LSP-4
LSP3

→ a a b c d a a => aa = aa → count (1)
LSP1
Ans

Page No.:
Date:
YOUVA

Pseudo Code :-

→ we take int lps[n] ; lps=0.
→ len = 0 to for prefix & suffix lengths
→ loop to calculate lps[i]
  for i = 1 to n-1 ; while (i<n).
→ if (s[i] == s[len])
  { len++;
    lps[i] = len;
    i++; }
else { (pattern[i] != pattern[len].
  if (leng != 0).
  { len = lps[len-1] ;
    → Here we don't use 'i' }
  else → if(len == 0)
    lps[i] = 0; i++;
}

→ an integer result → res = lps[n-1]
as we are looking for no overlapping part

return (res > n/2) ? res/2 : res;
↓ using ternary operator

Page No.:
Date:
YOUVA

Ques- Convert a sentance into its equivalent mobile numeric keypad sequence.

Given sentence in uppercase convert into equivalent keypad sequence

1	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUV	9 WXYZ
*	0	#

Example → 'GFG'

Ans → 4 333 0 4
G F G

Intution :-

→ For each char, store the sequence which should be obtain at its respective pos. in an array, i.e- for Z → 9999, F-33 like this.

→ For each char. sub. ASCII value of 'A' & obtain position in the array pointed by that array

→ Add. sequence stored in that array to string.

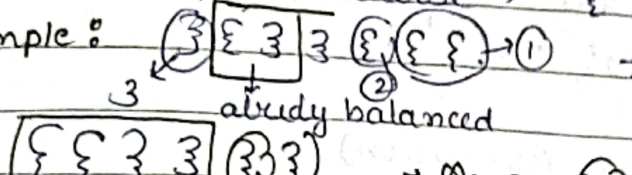
→ if char space, store 0, print overall sequence.

Code :-

```
int n = length();  String op = "";
for(i = 0; i < n)
{  if (input[i] == ' ')
       op = op + '0';
   else {
       int pos = input[i] - 'A';
       op = op + arr[pos];
   }
}
```

Ques- Count The ruursal :-

Given S contain '{' & '}', find out min no. ruursal req. required to convert string into balanced exp. ruurse → '{' → '}'

example : 

→ Ans → 3

→ Ans → 1

" }{{}}{{{ "

Intution :-

→ If Str.length() = odd can't be balanced → -1
∴ if (s.length() & 1) cout << "-1";

→ Do not concider balaced braces, conccider only unbalanced brackets. → Rev → Rev → Rev.

↳ " }{{}{{ → }{{}} → }{ "

→ take 2 counter c & c2,
if (ch == '{') push → c2++;
el if (ch == '}' & ! st.empty() and st.top() == '{'
St.pop(); c2--;
else c++;

→ after the above step, seal value of both the counter, and diuide it by 2.

if (c & 1) c = (c/2) + 1;
else & c = c/2;
if (c & 1) c2 = (c2/2) + 1;
else c2 = c2/2;
cout << c + c2 << endl;

Page No.:
Date: 7/2/2022 YOUVA

Ques → Count palindrome subsequences

Given a string str of N size, find no. of palindrome subsequence.

Example :- abcd → 'a', 'b', 'c', 'd' => 4 Ans
'aab' → 'a', 'a', 'b', 'aa',

★ Intution :- (using Recursion)

Init val : i=0; j=n-1;

in countPS(i,j) → every element is palindrome subsequence.

if (i==j) return 1 => palind. of length 1.

if 1st & last character are same, then consider it as palindrome & check for rest.

(i+1, j), (i, j-1)

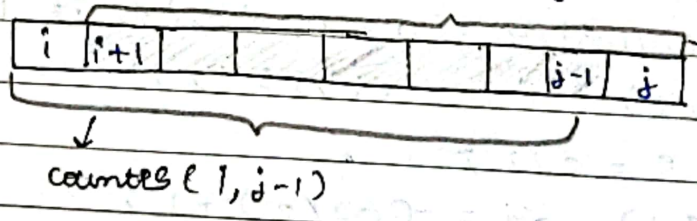
→ else if (str[i] == str[j])

return countPS(i+1, j) + countPS(i, j-1) + 1;

else,

check for rest subseq. & remove common palind. subsequence as they are counted twice when we do : "countPS(i+1, j) + countPS(i, j-1)"

return → countPS(i+1, j) + countPS(i, j-1) - countPS(i+1, j-1)



Page No.:
Date:
YOUVA

Ques → Count no. of given string in 2D character array (word)

Given a 2D array of string, find given string in 2D array, such that any char can be present L to R, T to B, or R to L, B to T.

Example :-

a = { { D, D, G, D }
{ G, E, E, K }
{ B, D, E, A }
{ B, B, K, F } }

Str = 'GEEK'

O/P => 1 + 1 = 2

Intution :-

Brute force method, traverse each char of matrix, & take each char as a start of string & search in all direct^n.

When word found increase count, after traversing whole matrix, return count.

Algo :-

(1) → for each char find a string in all four direction.
(2) if found, ↑se count
(3) when done with 1 char as start, repeat same for next char.
(4) calc. sum of count for each char.
(5) Return count.

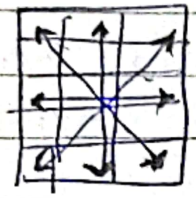
TC - $O(R * C \times 2^{2 * length})$

SC - O(1)

Page No.:
Date:

Ques- Search a word in 2D grid of char.

2D grid of char given, find all occourance of word in grid, word can be matched in all 8 direction at any point.

8 direct^n → 

grid → {"GEEKI=OR GEEKS
GEEGQUZGEEKS
IDEAQPRZTUWX"}

Text = "GEEK" => 4 ans

Intution :- (Brute force)

We check every cell.

If cell has 1st char, then we one by one Check all 8 direction.

We use 2 array x[] & y[] to find next move in all 8 direct^n.

Implemetation :-

```
int x[] = {-1,-1,-1,0,0,1,1,1}
    y[] = {-1,0,1,-1,1,-1,0,1}
func( ) {
```

→ if 1st char of word doesn't match with given starting point in grid

→ 3 condition :-

out of bound, matched, not math.

→ Nested for loop :- i => Search word in all 8 direction.

→ j => chick for above 3 condition

→ if all char match then value of k must be equal to length.

```
}
    return false;
}
```

Page No.:
Date: 8/2/22

Ques- Roman No. to Integer

Given string s is roman no., convert it into its integer.

Intution :-
Add all the roman values & its corres. int in map.
Storing the integral value of rightmost symbol/char of given string 's' using get method of HashMap.
Iterate string from R to L.
If our s < no. → res -= map val else res += .

Algo/Code :-

```
int res = map.get(s.charAt(s.length()-1));

for(i = s.length()-2; i>=0; i--)
{
  if(map.get(s.charAt(i)) < (map.get(s.charAt(i-1))
    res = res - map.get(s.charAt(i));

  else  res = res + map.get(s.charAt(i));
}
  return res;
};
```

Page No.:
Date:

Ques → Longest Common prefix

Find longest common prefise in all the given string.

example :-

str = ["Flower", "Flow", "Flight"]

O/P => Fl

Intution :-

Sort the string so that it arrange alphabatically.
initialize 2 ptrs as start & end of the string.
Iterate from 0 to size, if both start and end are equal add that a[i] in the 'ans'. else stop.

Code :-

```
int n = s.size();
starting ans;
sort (s.begin(), s.end());
starting a = s[0]; starting b = s[n-1];
for( i=0 ; i<size(); i++)
  { if (a[i] == b[i])
        ans += a[i];
    else
        break;
  }
return ans;
}
```

Ques→ Number of flips to make binary no. alternate

Given binary string, we need to make a seq. of alternate char by flipping some bits.
min. no. of flipps.

Example:- S = "001" → 0 0 1 ⇒ 0 1 0 ⇒ 1 flip

Intution:-
∵ task is minimum no. of flips,
we'll first find if string stores with 0 or 1, if start with '0' then initialize bit='0'; then check if (s[i]==0) and 2 count var. i.e zero_c=0; & one_c;
if (s[i] != bit) zero_c++; ⇒ ①
if if bit==0 ⇒ bit = '1';
else bit = 0;
Same goes if start with 1, replace 0 with 1 and use one_c to use count.

Code:-

For 0

```
char bit='0';
for(i=0; i<s.size())
if(s[i] != bit) zero_c++;
if (bit=='0')
    bit='1';
else
    bit='0';
```

For 1

```
bit='1';
for(i=0 → i<size())
if(s[i] != bit)
    one_c++;
if (bit=='0')
    bit='1';
else
    bit='0';
```

Page No.:
Date:
youva

Ques:- Find first repeated char in a string :-

Given a string s, task is to find repeated char in it.

We need to find char that occour more than once & whose index of 2nd occourance is smallest than the

Example :- S = "0 1 2 3 4 5 6 7 8 9
g e e k s g e e k s"

O/P → e

'e' repeat at third pos.

here → g & e are both repeating but e 2nd e occour early

Intution :- (use map)

Create an empty hash, scan each char of input string and insert value to each keys in hash.

When any char appear more than once, hash value use key 1, & return the char.

Ques → Second most repeating string in sequence :-

Given a seq. of string, find 2nd most repeated string in given sequence.

Example :-

arr[] = {aaa, bbb, ccc, bbb, aaa, aaa}

aaa = 3 , bbb = 2 → ans = bbb.

Intution :- (using Map)

We use map<str, int> to store all word with its occourance

Then we find 2nd largest occourance word.

as → if (it → second > first_max)

else if (it → second > sec_max && it → sec ! = first_max)

sec_max = it → second;

Return string with occourance equal to second max.

Code :-

unordered_map<string, int> occ;

for(int i = 0; i < sq.size(); i++) → occ[seq[i]]++;

{ int first_max = INT_MIN, sec_max = INT_MIN;

for(auto it = occ.begin(); it != occ.end(); it++)

if (it → second > first_max)

sec_max = first_max;

first_max = it → second;

else if (it → second > sec_max && it → second != first_max)

sec_max = it → second; }

for(auto it = occ.begin(); it != occ.end(); it++)

{ if (it → second == sec_max)

return it → first; }

(STRINGS)

Date 9/2/2022

Ques → Minimum swap for bracket balancing

Given a string of 2N char, 1N contain '[' & another N contain ']', calculate min. no. of swap necessary to make string balanced.

Example : [ ] [[ ]]] O/P - 2.
0 1 2 3 4 5

swap 3 and 4 to balance → [ ] [ ] [ ]

Intution :- (TC - O(N))

Go through the string and store the position of '[' in a vector say 'pos'. Take 'p' = 0 & use 'p' to traverse the vector 'pos'.

We maintain count for '[' if '[' occour use count++ vice versa if ever count goes -ve, then we need to swap. the ele pos[p] shows index of next '['. We sum by pos[p] - i, where 'i' is current index

Code :-

```
vector<int> pos;
for( i = 0 → i < size() ) if ( s[i] == '[' ) pos.push_back(i);
int count = 0, p = 0, sum = 0;
for ( i = 0 → i < size() ) {
  if ( s[i] == '[' )      ++count; ++p;
  else if ( s[i] == ']' )  --count;
    if ( count < 0 ) {      sum += pos[p] - i;
      swap ( s[i], s[pos[p]] ); ++p;
      count = 1
    }
}
return sum;
```

Ques → Program to generate all possible valid IP address from given string.

str of digit, restore it by returning all possible valid IP address combination. (0-255)

Example: 255255111135 → O/P = 2.

[255.255.111.35] or [255.255.11.135].

Ex- 25505011535 → [] no output generate in this string.

Intuition: (TC- O(4*n*3) = O(12n) = O(n))

We know that, 4 part of string of IP. strat from the end & go to the start of string

create a dp of 2D array of size (4×N)

There can be 2 values in DP array i.e 1 (true) or 0 (false). dp[0][i] tells if we can create, I part of IP from our str starting from i to end of str.

Same, dp[1][i] tells if we can create 2 part of IP from the substring starting from i to end of str.

After creating dp array, start create valid IP address.

We start from bottom left corner of the 2D dp array.

Max we can iterate 12 time (worst case).

Longest Common Subsequence (gfg)

Code :-

```
int helper(string s1, string s2, int a, int b,
        vector<vector<int>>& v) {
    if (a==0 || b==0)  return 0;

    if(v[a][b]!=-1)  return v[a][b];
if(s1[a-1]==s2[b-1]) return v[a][b]=1+help(s1,s2,a-1,b-1,v);
  else {
return v[a][b]=max(helper(s1,s2,a-1,b,v),helper(s1,s2,a,b-1,v));
    }
                    }
int lcs(int x, int y, string s1, string s2)
{
    vector<vector<int>>& v;
for(int i=0; i<x+1; i++)
 {
    vector<int> temp;
   for(int j=0; j<y+1; j++)
    {
      temp.push_back(-1);
    }
    v.push_back(temp);
  }
  return helper(s1, s2, x, y, v);
 }
};
```

Ques- Smallest distinct window

given a string 's', task to find smallest window length that contains all the char of given string at least one time.

Example: AABBBCBBAC O/P = 3 ;

sub string → 'BAC'

Input - aaab → O/P → ab

possible substring → {aaab, aab, ab}.

→ ab is the smallest sub str. contain all char of strs.

Intution :-

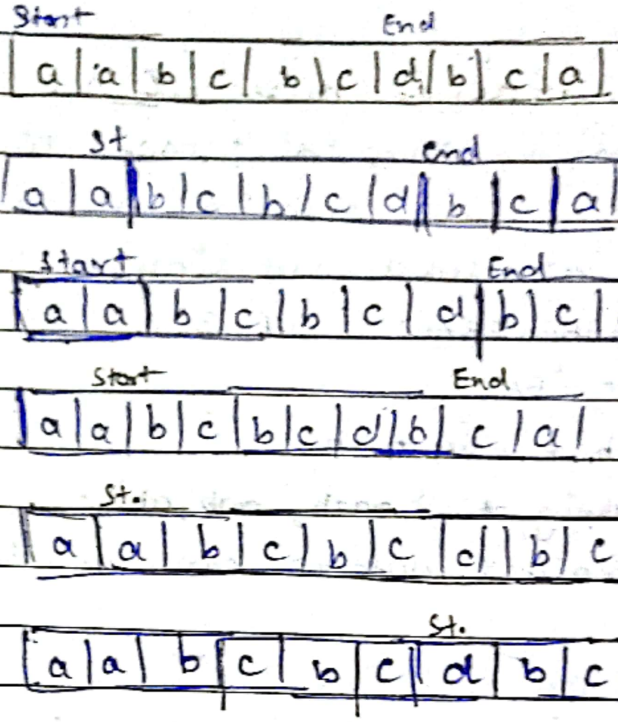
We use sliding window to arrive at the sol^n. (this tech. shows how nested for loop in few problem can be converted to single for loop & hence reduc TC).

→ Window of char is maintained by using 2 pointers 'start' & 'end', start & end used to shrink & use the size of the window.

of given string window shrink from left side whether whenever window contain all char to remove extra characters & then its length compared with smallest window found so far.

If in curr win. no more char can be deleted, then start use the size of window using end until all distinct chars present in str, are also there in window,

finally, find minsize of each window,


Start
End
a a b c b c d b c a
St
End
a a b c b c d b c a
Start
End
a a b c b c d b c a
Start
End
a a b c b c d b c a
St.
End
a a b c b c c b c a
St.
End
a a b c b c d b c a

10/2/22

Ques→ Rearrange character in a string such that no two adjacents are same.

Given string of repeated char, task is to rearrange chars. so that no 2 ston adjacent char are same

Example :-

I/P → aaabc => o/p → abaca.

I/P → aaabb → ababa

I/P → aa → Not Possible

Intution :- (priority-queue)

Idea is put highest freq. element first, use (max heap) priority queue. & put all char. & ordered by their frequencies.

One by one take highest freq. char from the heap & add it to the result. after add, use the freq of the char & temporarily move this char out of priority queue, so that not pick next time.

Algo :-

(1) Build priority queue or max heap pq.

(2) create temp. key, that will be used as previously visited element { char = '#', freq = '-1' }

(3) while pq is not empty

→ pop an element & add to the result

→ use freq. of poped element by '1'.

→ Push previous element back into priority queue.

if freq. > '0' & Make curr as prev. ele. for next iteration

(4) if str != res, NOT Possible

Date: / / Page no:

Ques → Min. char to be added at front to make string palindrome

(Hard)

Given string, o you tell min. char. to be added at front to make string palindrome.

Example:

'ABC' → BCABC → 2 (add B & C in front)

Intution :- (use LPS KMP algo.)

First concat given string, a special char, and reverse of given string, then we will add get LPS array (longest proper prefix also suffix).

For string ⇒ AACECAAA → New_Str = AACECAAAA$AAAACECAA

LPS array = {0, 1, 0, 0, 0, 1, 2, 2, 2, 0, 1, 2, 2, 2, 3, 4, 5, 6, 7}

Here Intention, Last value of lps array bcoz it show largest suffix of reversed string that matches prefix of original suffix

i.e - these many character already satisfy the palindrome palindrome property.

at the end, answere is length of input string minus last entry of our lps array.

Ques → Print Anagrams Together

Given array of string strs, group the anagrams together return ans in any order.

Anagram: word/phrase rearranging letters of a diff word.

Example:- strs: ["eat", "tea", "tea", "ate", "nat", "bat"]

O/P- [["bat"], ["nat", "tea"], ["ate", "eat", "tea"]].

Example: strs: [""] → O/P [[""]].

Intution:- (using hashmap).

In previous approach we though of sorting but it TLE. We'll take hashmap inside another hash-map to maintain frequency of character which will generate same hash func^n for diff. string having same freq. of char.

Approch:-

hash_map<HashMap, ArrayList>, inner hashmap will count the frequency of char. of each string ant the outer HashMap will check whether the hashmap is present or not.

if present then it will add that string to corresponding list.

Ques. Smallest window in a string containing all the char of another string

2 string S and P. Find smallest window in string S consisting of all the chars. of the string P. Else return -1.

Example :-

S - "time to practice" P : "toc"

=> toprac.

Intution :- (sliding window)

1) check if the length of string is less than pattern then no solution.
2) store the occourance of given pattern of given pattern in hash_pat[].
3) using 2 pointer start matching characters of pattern with the char. of string.
   If count if char. matches
4) check if (count == len(pattern)) means window found
5) if such window found try to minimize it by removing extra space from start of window.
6) delete 1st character from first & again find this delete key at right, once found apply step (4).
7) update min length.
8) print window.

11/02/2022

Ques → Remove all consecutive duplicate from string

Given string S, remove all consecutive duplicate.
Input :- aaaaabbbbbb, O/P - ab
Input geeksforgeeks → gekforgeks.

Intution :- (we can use recursion)
above problem can be solved using recursion.
1) if string is empty, return.
2) else compare adjacent chars of string, if they are same, then shift chars one by one left.
(call recursion on string S).
3) if not same then call recursion from S+1 string.

Ricursion tree would be :-

aabcca	S = aabcca
abcca	S = abcca
bcca	S = abcca
cca	S = abcca
ca	S = abca
a	S = abca
empty	S = abca (output)

Ques→ Transform one string to another using mini. no. of given operation.

2 string A & B, convert A to B if possible. The only operation allowed to put any char. from A and insert it at front.

Example: A = 'ABD', B = 'BAD' → O/P = 1.

Example: A = EBACBD, B = EABCD → B, A → A → E to front

Intution :-

1) Checking whether string can be transform, we need to check whether both are same in size.

2) start matching last char of both string
If match, reduce our task to n-1 char.,
If don't, then find post^n of B's mismatch char in A. The difference indicate that these many char of A must be moved before char of A.

Code :-

```
minOps (string &A, string &B) {
int m = A.length(), n = B.length();
if (n != m) // possible or not
  int count[256];
memset(count, 0, sizeof(count));
for(i=0 i=0; i<n) count[B[i]]++; // count A
for(i=0; i<n;) count[A[i]]--; // sub count for every B
for(i=0; i<256) if(count[i]) return -1;
int res=0;
for(i=n-1; j=n-1; j=n-1; i>=0) {
  i--; res++; i--; j--;
```

Date: 12/2/22 Page no:

Ques → Isomorphic String

given two string str1 & str2, check if these two string are isomorphic to each other.

Isomorphic : 2 string str1 and str2 ~~o~~, have one to one mapping possible for each char of str1 to ~~str~~ every char of str2 while preserving order.

Input : 1) aab 2) xxy ⇒ True (1)

'a' is mapped to 'x' & 'b' is mapped to 'y'.

Input : (1) aab (2) xyz ⇒ False (0)

1 occourance of 'a' in str1 has 'x' & 'a' has other occourance of 'y'.

Intution :-

We will count no. of occourance of a particular char in both string using 2 array,

compare 2 array if at ~~most~~ any moment with the loop the count of the current char in both string become diff. we return false(0), else after the loop ends we return true(1)

Code :- (E)

```
if (n != m) return false;
int count[MAX_CHARS] = {0};
int count[MAX_CHARS] = {0};
for( i=0 → i<n) {
count[str1[i] - 'a']++; dcount[str2[i] - 'a']++;
if (count[str1[i] - 'a'] != dcount[str2[i] - 'a'])
        return false;
}
return true;
```

(3)

Ques→ Recursively Print all sentense that can be formed from list of word list.

Given a list of word list, how to print all possible sintencis, taking 1 word from a list at a time.

Input : { { "You", "we" },
{ "have", "are" },
{ "sleep", "eat", "drink" } }.

You have sleep, You have eat, You have drink,
You are sleep, You are eat, You are drink,
we have sleep, we have eat, we have drink,
we are sleep, we are eat, we are drink.

Intution :- (DFS)
we use simple dfs, start with every word of 1st list. as 1st word of output sintense, then recurse from remaining list.

Algo :-

```
Func ( ) {}
// add current word to o/p array
output[m] = arr[m][n];
// if last word of curr_output sintence, then print output sintence.
if (m = R-1) {
    for (int i=0; i<R; i++)
        cout << output[i] << ".";
    cout << endl;
    return
}
```

Date: / / Page no:

// Recurr from the next row
for (i=0; i<c)
if (arr[m+1][i] != " ")
printUtil (arr, m+1, i, output);

3

// A wrapper over printUntil()
void print (String arr[r][c])

2

// create an array to store sentence
String output[R];

// consider all word for first row as starting point
// and print all sentence

for (i=0 → i<c) 1
if (arr[0][i] != " ")
printUntil (arr, 0, i, output);

3

3