Part-1

String Common Problem

DSA Problem for Placement

amazon Microsoft

Coding!

String:

- Valid Palindrome
- Reverse String
- Valid Anagram
- Longest Common Prefix
- Longest Substring Without Repeating Charcter
- Valid parentheses
- Longest Palindrome substring
- Print all the subsequences of a string
- Isomorphic Strings
- Group Anagram

1. Valid Palindrome

125. Valid Palindrome

Easy 4319 5656 Add to List Share

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string `s`, return `true` *if it is a* ***palindrome****, or* `false` *otherwise.*

Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

Example 2:

```
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
```

Example 3:

```
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.
```

Approach:- 1

1. In this we will take two variables i and j, where i equal to 0 and j equal to size of string
2. Now by using while loop, we will check the character from the beginning whether the character are alphanumeric or not, if it is not alphanumeric then skip that condition and increment i by 1.

3. If the above condition is fail then we will check whether the last character is alphanumeric or not, if is not alphanumeric then skip that condition also and decrement j by 1

4. Then converting it into lower case character and simultaneously check both i and j position character are equal or not if it is not equal then return false.

5. otherwise , Increment i by 1 and decrement j by 1

6 . At the end we return true.

Code :-

```cpp
class Solution {
public:
    bool isPalindrome(string s) {
        int i=0;
        int j= s.size()-1;
        while(i<j)
        {
           if(!isalnum(s[i]))
           {
               i++;
           }
            else if(!isalnum(s[j]))
            {
                 j--;
            }
            else if(tolower(s[i])!=tolower(s[j]))
            {
                return false;
            }
            else
            {
                i++;
                j--;
            }
        }
        return true;

    }
};
```

Time complexity :-

➡ O(n)

Dry Run :-

S = "A man, a plan, a canal : Panama"

A	-	m	a	n	,	-	a	-	p	l	a	n	,	-	a	-	c	a	n	a	l	:	-	p	a	n	a	m
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

a
29

```
int i = 0
int j = 29
While (i<j)
{   if(!ipalnum(S[0]))
      {
          Condⁿ false
      }
  else if(!ipalnum(S[29]))
      {
          Condⁿ false
      }
  else if(tolowor(S[0]) != tolowor(S[29]))
      {
          Condⁿ false
      }
  else
      { i++;
      } j--;
```

Now, i = 1, j = 8

```
{   if (!ipalnum(S[1]))
    {
        Condⁿ True
        i++;
    }
```

Now, i = 2, j = 28

```
While (2<28)
{
    if (!ipalnum(S[2]))
    {
        Condⁿ false
    }
    else if (!ipalnum(S[28]))
    {
        Condⁿ false
    }
    else if (tolower(S[2] != tolower(S[28]))
    {
        Condⁿ false
    }
    else
    {   i++;
        j--;
    }
```

** Same step as above until i<j.

2. Reverse String

344. Reverse String

Easy 5604 977 Add to List Share

Write a function that reverses a string. The input string is given as an array of characters `s`.

You must do this by modifying the input array in-place with `O(1)` extra memory.

Example 1:

```
Input: s = ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
```

Example 2:

```
Input: s = ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]
```

Constraints:

- `1 <= s.length <= 10^5`
- `s[i]` is a printable ascii character.

Approach:-1

1. Take i=0 and j = size of string , with the help of while loop swap character of ith position with the jth position character , after that increment i and decrement j by 1.

Code :-

```
class Solution {
public:
    void reverseString(vector<char>& s) {
        int i=0;
        int j = s.size()-1;

        while(i<j)
        {
            swap(s[i++],s[j--]);

        }

    }
};
```

Dry Run :-

S = ["h", "e", "l", "l", "o"]
0 1 2 3 4

i = 0
j = 4

While (0 < 4)
{
swap(s[0], s[4]);
}

S = ["o", "e", "l", "l", "h"]
0 1 2 3 4

i = 1
j = 3

While (1 < 3)
{ swap(s[1], s[3]);
}

S = ["o", "l", "l", "e", "h"]
0 1 2 3 4

i = 2
j = 2

While (2 < 2)
{
Cond^n false
}

S = ["o", "l", "l", "e", "h"]

Approach:-2

1. Push all the element from the end in vector with the help of for loop which is iterate from end positiion

Code :-

```
class Solution {
public:
    void reverseString(vector<char>& s) {

        vector<char>v;

         int n = s.size()-1;

         for(int i =n;i>=0;i--)
         {
             v.push_back(s[i]);
         }
          s=v;
    }
};
```

3. Valid Anagram

242. Valid Anagram

Easy 6374 234 Add to List Share

Given two strings `s` and `t`, return `true` *if* `t` *is an anagram of* `s`, *and* `false` *otherwise.*

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: s = "anagram", t = "nagaram"
Output: true
```

Example 2:

```
Input: s = "rat", t = "car"
Output: false
```

Constraints:

- `1 <= s.length, t.length <= 5 * 10^4`
- `s` and `t` consist of lowercase English letters.

Approach :-1

1. Sort both the string by using sort function.

2. check if both the string are equal then return true otherwise false

```cpp
class Solution {
public:
    bool isAnagram(string s, string t) {

            if(s.size()!= t.size())
              return false;

        sort(s.begin(),s.end());
        sort(t.begin(),t.end());
        return (s==t);
    }
};
```

Time Complexity :- O(n*logn)

Approach :-2

1. Firstly if the size is not equal of both the string then return false.

2. Now count the number of character occur in both the strings and also occurance of character in both the string are same then return true.

Code :-

```
class Solution {
public:
    bool isAnagram(string s, string t) {

        if (s.size() != t.size())
        {
            return false;
        }

        unordered_map<char, int> ump1,ump2;

        for (char c: s)
        {
         ump1[c]++;
        }
            for (char c: t)
            {
                ump2[c]++;
            }

        for (int i = 0; i < s.size(); i++)
        {
             if (ump1[s[i]] != ump2[s[i]])
             {
                 return false;
             }
        }
        return true;
    }
};
```

Time Complexity :- O(n)

4. Longest Common Prefix

14. Longest Common Prefix

Easy 9343 3229 Add to List Share

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string `""`.

Example 1:

```
Input: strs = ["flower","flow","flight"]
Output: "fl"
```

Example 2:

```
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

Constraints:

- `1 <= strs.length <= 200`
- `0 <= strs[i].length <= 200`
- `strs[i]` consists of only lowercase English letters.

Approach:-1

1. Take first position string of array and compare each character of that string with other position string charcater

2. If the character are not equal then mark same as false and break otherwise push character into ans variable

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {

      string ans = "";
        for(int i=0;i<strs[0].size();i++)
        {
            char ch = strs[0][i];

        bool same = true;

            for(int j=1;j<strs.size();j++)
         {

             if(strs[j].size()<i || ch != strs[j][i])

             {

                 same =  false;

                 break;
             }
         }

        if(same==false)
        {
            break;
        }
        else
        {
            ans.push_back(ch);
        }

        }
        return ans;
    }
};
```

Time Complexity :- O(n*m)

5. Longest Substring without repeating character

3. Longest Substring Without Repeating Characters

Medium 27115 1174 Add to List Share

Given a string `s`, find the length of the **longest substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

Constraints:

- `0 <= s.length <= 5 * 10^4`
- `s` consists of English letters, digits, symbols and spaces.

Approach:-1

1. Take a vector of int type & l and r equal to zero . By using while loop - till the last index

2. Check if the character on the right index exist or not as:
if (v [s [r]] ! = - 1)

if it is exist then , l = max (v [s [r]] + 1 , l) ;

3. Update , v [s [r]] = r ;

4. We get the length of current string as :

len = max (len , r - l + 1) ; then r++ and return length

Code :-

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {

        int l=0;
        int r=0;
        int len =0;
        int n = s.size();
        vector<int>v(256,-1);

        while(r<n)
        {
            if(v[s[r]]!=-1)
            {
                l = max(v[s[r]]+1,l);
            }
                v[s[r]] = r;

            len = max(len,r-l+1);
            r++;
        }
        return len;
    }
};
```

Time Complexity :- O (n)

6. Valid Parentheses

20. Valid Parentheses

Easy 15189 753 Add to List Share

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of parentheses only `'()[]{}'`.

Approach:-1

1. Iterate over each character of string & check if the current element of string is opening bracket then we will push it into stack .

2. Otherwise , if it is closing bracket then we will find the pair of bracket and remove from the stack.

3. At last we will check the stack is empty or not , if it is empty then return true otherwise false.

Code :-

```
class Solution {
public:
    bool isValid(string s) {

       stack <char> st;

        for (int i = 0; i<s.size();i++)
        {
            if(s[i] == '(' || s[i] == '{' || s[i] == '[')

                st.push(s[i]);


            else if(s[i] == ')' || s[i] == '}' || s[i] == ']')
            {
                if(st.empty())

                    return false;

                else if(s[i] == ')' && st.top() != '(')

                    return false;

                else if(s[i] == '}' && st.top() != '{')

                    return false;

                else if(s[i] == ']' && st.top() != '[')

                    return false;

                else
                    st.pop();
            }
        }

        if(st.empty())

            return true;

        else

            return false;
    }
};
```

Time Complexity :- O (n)

7. Longest Palindromic substring

5. Longest Palindromic Substring

Medium 21023 1211 Add to List Share

Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

Constraints:

- `1 <= s.length <= 1000`
- `s` consist of only digits and English letters.

Accepted 2,060,753 | Submissions 6,375,983

Approach:-1

- Bruteforce approach:-

1. Using substr function for finding all substring and then check it is palindrome or not .

2. If it is palindrome then check which is largest and return that string

```
class Solution {
public:

    bool thisstring(string s)
    {
        int i =0 ;
        int j = s.size()-1;

        while(i<j)
        {
            if(s[i++]!=s[j--])
                return false;
        }
        return true;
    }

    string longestPalindrome(string s) {

     int n = s.size();
        if(n==0)
           return "";

        if(n==1)
            return s;

        string ans = "";

        for(int i=0;i<n-1;i++)
        {
            for(int j=1;j<=n-i;j++)
            {
                if(thisstring(s.substr(i,j)))
                {
                    if(ans.size() < j)
                        ans = s.substr(i,j);
                }
            }
        }
        return ans;
    }
};
```

Approach:-2

1. In this we will use DP, we will check string is palindrome or not, if it is palindrome then dp[i][j] is true otherwise it will be false .
2. After that we will check which is the longest palindromic substring

Code :-

```
class Solution {
public:
    string longestPalindrome(string s) {
        int n = s.size();

        if (n == 0)
        {
            return "";
        }

        bool dp[n][n];

        memset(dp, 0, sizeof(dp));

        for (int i = 0; i < n; i++)
        {
            dp[i][i] = true;
        }

        string ans = "";

        ans += s[0];

        for (int i = n - 1; i >= 0; i--)
        {
            for (int j = i + 1; j < n; j++)
            {
                if (s[i] == s[j])
                {
                    if (j - i == 1 || dp[i + 1][j - 1])
                    {
                        dp[i][j] = true;

                        if (ans.size() < j - i + 1)

                            ans = s.substr(i, j - i + 1);
                    }
                }
            }
        }
        return ans;
    }
};
```

Time complexity :- O (N ^ 2)

8. Print all subsequences of a string

Print all subsequences of a string

Difficulty Level : Medium • Last Updated : 20 Aug, 2022

Read Discuss

Given a string, we have to find out all subsequences of it. A String is a subsequence of a given String, that is generated by deleting some character of a given string without changing its order.

Examples:

```
Input : abc
Output : a, b, c, ab, bc, ac, abc

Input : aaa
Output : a, a, a, aa, aa, aa, aaa
```

Approach:-1

1. Iterate over the entire String

2. Iterate from the end of string in order to generate different substring add the substring to the list

3. Drop kth character from the substring obtained from above to generate different subsequence.

4. if the subsequence is not in the list then recur.

Code :-

```
void subsequence(string str)
{
    for (int i = 0; i < str.length(); i++)
    {
        for (int j = str.length(); j > i; j--)
        {
            string sub_str = str.substr(i, j);
            st.insert(sub_str);

            for (int k = 1; k < sub_str.length(); k++)
            {
                string sb = sub_str;
                sb.erase(sb.begin() + k);
                subsequence(sb);
            }
        }
    }
}
```

Approach-2

1. One by one fix characters and recursively generates all subsets starting from them. After every recursive call, we remove last character so that the next permutation can be generated.

Code :-

```
void printSubSeqRec(string str, int n, int index = -1,string curr = "")
{
    if (index == n)
        return;

    if (!curr.empty())
    {
        cout << curr <<endl;
    }

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

O(n * 2n)

Time complexity :- **O (N * 2 ^ N)**

9. Isomorphhic string

205. Isomorphic Strings

Easy 4732 857 Add to List Share

Given two strings `s` and `t`, *determine if they are isomorphic*.

Two strings `s` and `t` are isomorphic if the characters in `s` can be replaced to get `t`.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Example 1:

```
Input: s = "egg", t = "add"
Output: true
```

Example 2:

```
Input: s = "foo", t = "bar"
Output: false
```

Approach:-

1. In this we will use unordered map - mp1 and mp2
   mp1 is used to map s [i] to t [i]
   mp2 is used to map t [i] to s [i]

2. Now firstly check whether both string are equal or not.
   if it is not equal then return false

3. By using for loop check whether s [i] is in mp1 or not and also check whether t [i] is in mp2 or not if these are not then we will map s[i] to t [i] and t [i] to s [i]

4 . Otherwise we will check whether it is isomorphic or not , by satisfying this condition :-

mp1[s [i]] != t [i] || mp2 [t [i]] != s [i]

If this condition is satisfied then return false that means it is not isomorphic.

5 . if it is isomorphioc then return true

Code :-

```cpp
class Solution {
public:
    bool isIsomorphic(string s, string t) {

        int n = s.size();
        int m = t.size();

        if(n != m)
            return false;

        unordered_map<char,char>mp1,mp2;

        for(int i =0 ;i<n;i++)

        {

        if(!mp1[s[i]] && !mp2[t[i]])
        {
            mp1[s[i]] = t[i];
            mp2[t[i]] = s[i];

        }

        else if( mp1[s[i]] != t[i] || mp2[t[i]] != s[i])
        {
            return false;
        }

        }
        return true;

    }
};
```

10. Group Anagram

49. Group Anagrams

Medium 12K 371

Companies

Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

Approach:-1

1. In this we will use hashmap, Intialize a temp variable as a string and pick every given string in temp variable and sort them

2. mp [temp] . push_back (strs[i])

By this all the same charcter string as temp string are push to the same vector of hashmap

3 . Then push all the string vector from map to ans.

4. Finally, return ans as a output

Code :-

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {

        int n = strs.size();

        unordered_map<string, vector<string>> mp;

        vector<vector<string>>ans;

        int i=0;

        while(i<n)
        {
            string temp = strs[i];
            sort(temp.begin(), temp.end());
            mp[temp].push_back(strs[i]);
            i++;
        }

      for(auto i=mp.begin(); i!= mp.end(); i++)
      {
            ans.push_back(i->second);
      }
         return ans;

    }
};
```

Time complexity :-

➡ **O (N*M log (M)**

Part :- 2 soon.....