October 2022

S	M	T	W	T	F	S
				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	29	30	31

Strings.

Week-40 (272-093)

THURSDAY 29
2-0-2-2 SEPTEMBER

#Minimum Time Difference.

Syntax for stoi
stoi (str);

23:59	00:00	→ vector of strings.

Take first two characters of string using sub-string function → substr(0,2).
and store it in an integer variable using 'stoi' function.
Store first two characters as hours.
and last two characters as minutes.

→ Now the total time will be:- (hours * 60) + minutes

→ Save all 'total time' in an array of integers.

→ Now, sort that array

→ Now get the minimum time difference by iterating the array.

★ Important:- We've gotten the minimum time difference from 0 to 24 hours but what about 24 to 0 hours?

→ To perform this:- Compare the first and last element of array. (Add 24 hours * 60 minutes = 1440 to first element to achieve it).

" Ans = min(str[0] + 1440 - str[n-1] , ans) "

I will not say I've failed a 1000 times; I'll say I found a 1000 ways that can cause failure. - Thomas Edison

Longest Common Prefix.

Use a variable 'check' to store first element of the string of given array.

```
char check;
String ans;
```

→ Now traverse the ~~arra~~ String (first string)

```
for (int i = 0; i < strs[0].length(); i++)
{
    check = strs[0][i] // store element in var.

for (int i=0; i< strs.size(); i++)
{
    // Traverse the array and check if
    the nth element of each string is same.

    if (strs[i].empty() || strs.size()==0) return "";
      ↳ case.

    if (i >= strs[i].length() || strs[i][i] != check)
    return ans; // Return the ans if the
    elements are not matching anymore.
}
    ans.push_back(check);
}
return ans;
```

Everybody thinks of changing the world; but no one thinks of changing himself. – Leo Tolstoy

Re-organise the String (Not-optimal Approach)

Create an array of '26' size to store the frequency of alphabets occuring.

```
int hash[26] = {0};
for(int i=0; i<str.length(); i++)
  { hash[str[i] - 'a']++;
  };
```

↳ -a cause 'a' to 'z' ranges from 97 to 97+26

now find the most occuring character using loop and store it in variable 'max-char' and also store the frequency in 'max-freq'

Now place the most occuring character adjacently using: index = 0;

```
while( index < str.length() && max-freq > 0)
  {
    str[index] = max-char;
    index += 2;
    max-freq--; }
```

Sunday 02

Now if most occuring character can't be placed we return "" ; to do so,

```
if(max-freq != 0) return "";
```

hash[max-freq-char - 'a'] = 0;

Now place other elements and return.

Group - Anagrams. (Bruteforce)

Create a map with 'key' string and value 'vector of strings'.

now iterate through the vector and store corresponding values in the vector (after sorting).

```
map < string, vector < string > > mp;

for (auto it : strs)
{
    string s = it; // store a duplicate.
    sort (s.begin(), s.end()); // sort the Duplicate.

    mp[s].push_back(it);
}

Vector < vector < string >> ans; // create an ans vector

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

→ push back the vector in ans vector of vectors

```
return ans;
```

T.C → O(N K log K) S.C = O(n k)

Group - Anagrams (optimal Approach). Without sorting.

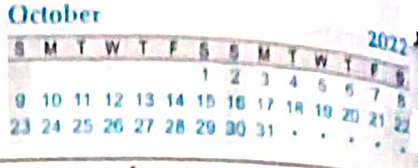
Rather than sorting the entire ~~arr~~ string we just create an array (hash array basically and will ~~put~~ store their frequency and will map it with the corresponding vector of string.

```
std:: array<int, 256> hash (string s)
{
    std:: array<int, 256> arr = {0};
    for(int i=0; i< str.length(); i++)
      { arr[s[i]]++; }
return arr;
};
```

Function to find the hash of a given string.

```
map<std:array<int, 256>, vector<string>> mp;
for ( auto str: strs)
{
    mp[hash(str)].push_back(str);
}
```

rest of the code will be same.



Longest Palindromic Substring (not optimal).

1. // find all substrings
2. // check which ones are palindrome.
3. // find the longest one in size.
4. // return the answer.

(O(n³)) solution.

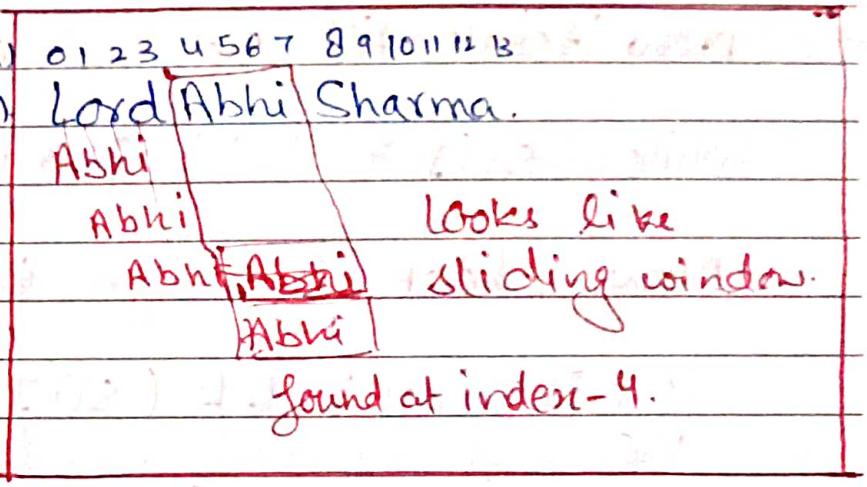
```
String ans = " ";
for (int i=0; i< s.size(); i++)
{
    for (j= i;  j< s.size(); j++)
    {
        if (isPalindrome (s, i, j))
        {
            string check = s.substr (i, j-i+1)
            ans = check.size() > ans.size() ? check:ans
        }
    }
}
return ans;
}
}
```

Find no. / groups of substrings

Index of first occurrence in a string.

Use sliding-window to get the output.

1.) Set m = needle.length()
2.) Set n = haystack.length()
3.) Now Iterate throughout the string (n-m times) as we've to return the first occurence.



Now if ~~str~~ ~~aa~~

~~haystack~~[i] needle[j] != haystack[i+j];
break the inner loop and increment i.

otherwise increment j and check if it matches.

```
for(int i=0; i<=n-m; i++)
{
    for(int j=0; j<m; j++)
    {
        if(needle[j] != haystack[i+j])
            break;

        if(j==m-1) return i;   O(m*n) t.c
    }
}.
```

Note:- It can be solved using Rabin Karp or Kmp algorithm.

07 FRIDAY
OCTOBER 2-0-2-2

Week-41 (280-085)

October 2022

S	M	T	W	T	F	S	S	M	T	W	T	F	S
							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	29	30	31				

Implementing Atoi() function.

```
int ans = 0, i = 0; sign = 1;   // assuming +ve as default.
```

Now iterate till no. '_' space is found.

```
while (s[i] == '_') i++;
```

Now Check for sign bit

```
if (i<s.size() && (s[i]=='-' || s[i]=='+'))
{
    sign = s[i] == '+' ? 1 : -1;
}
```

Now check for next digits and operate.

```
while (i<s.length() && isdigit(s[i]))
{
    if (ans > INT_MAX/10 || (ans == INT_MAX/10 && s[i] > '7'))
    { return sign==-1 ? INT_MIN : INT_MAX;
    }
    ans = ans*10 + (s[i] - '0')
    i++;
}
    return sign*ans;
}
```

I went into the business for the money, and the art grew out of it. – Charlie Chaplin

String Compression.

int ans = 0, i=0, index = 0, count = 1.

Char prev = s[0]. → initialize prev with first element

Initialise 3 pointers, i, prev and index.

```
for(int i=0; i< s.size(); i++)
{
    if(s[i] == prev) count++; // if elements are same increment the count.

    else
    {
        s[index++] = prev;
        if(count >1)
        {
            int start = index.
            while(count)
            {
                s[index++] = (count %10) + '0';
                count /= 10;
            } reverse(s.begin()+start(), s.begin()+index
        }
        count = 1; prev = s[i];
    }
    return index
```

Repeat it once more after loop.

Sunday 09

String Compression.

int ans = 0, i=0, index = 0, count = 1.

Char prev = s[0]. → initialize prev with first element

Initialise 3 pointers, i, prev and index.

```
for(int i=0; i< s.size(); i++)
{
    if(s[i] == prev) count++; // if elements are same increment the count.

else
{
  s[index++] = prev;
  if(count >1)
  {
     int start = index;
     while(count)
     {
        s[index++] = (count %10) + '0';
        count /=10;
     } reverse(s.begin()+start(), s.begin()+index).
  }
     count = 1; prev = s[i];
}
return index
```

Repeat it once more after loop.

Integer to Roman

Store Roman Integers and their corresponding values in an array.

```
String roman[] = {"M","CM","D","CD","C","XC",
                  "L","XL","X","IX","V",
                  "IV","I"};

Int values[]= {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9
               5, 4, 1};

String ans = "";

for(int i=0; i<13; i++)
{
    While (num >= values[i])
    {
        ans += roman[i];
        num -= values[i];
}   }

return ans;
}
```

g-zag Conversion.

a vector of strings to declare 'numRows' times strings.

ow use two pointers : int i=0, rows=0;
et :- bool direction = 1; // Default up to down direction.

If (numRows == 1) return s;

while (1)
{ if (i >= s.size()) break;

if (direction)
{ // fill all strings in up to down manner
~~zigzag[rows++].push_back(s[i++]);~~
while (rows < numRows && i < size())
{ zigzag[rows++].push_back(s[i++]);
}
row -= 2; // Set row for down-up direction

else {
while (rows >= 0 && i < s.size())
{ zigzag[rows--].push_back(s[i++]); }
row = 1; }
}
direction = !direction; (Change the direction).

Now Concat the strings and return the answer.

12 WEDNESDAY
OCTOBER 2-0-2-2

Week-42 (285-080)

October 2022

S	M	T	W	T	F	S
					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	29	30
31						

Largest Number.

first store all the no. in a string using a vector of strings.

```
vector<string> s;
string ans="";
```

```
for(int i=0; i<nums.size(); i++)
    s.push_back(to_string(nums[i]));
```

Now sort the string using comparater.

```
sort(s.begin(), s.end(), myComp);
```

Now store the answer in a string.

But for any comparater you have to do a thing to deal with.
that is.

for input (3, 30) the answer should be 330 but as it sorts in lexographical order it will be 303. to avoid that do.

```
myComp(string a, b)
{
    string t1 = a+b;
    string t2 = b+a;
    return t1>t2
}.
```