longest substring without repeating characters ( code done in java )

-

class Solution {

public int lengthOfLongestSubstring(String s) {

Set<Character> set = new HashSet<>();

Queue<Character> queue = new LinkedList<>();

int n = s.length();

int len = 0;

for (int i = 0; i < n; i++) { // O(n)

char c = s.charAt(i);

while (set.contains(c)) {

char head = queue.poll(); // O(1)

set.remove(head); // O(1)

}

set.add(c); // O(1)

queue.offer(c); // O(1)

len = Math.max(len, queue.size());

}

return len;

}

}

Longest palindromic substring ( code in python 3)

-

class Solution:

def longestPalindrome(self, s: str) -> str:

for x in reversed(range(1,len(s)+1)):

for y in range((len(s)+1)-x):

g=s[y:x+y]

if g[0]!=g[-1]:pass

elif g==g[::-1]:return g

Generate parenthesis ( pyton 3)

-

class Solution:

def generateParenthesis(self, n: int) -> List[str]:

res = []

def helper(s, m, d):

if len(s) == 2\*n:

res.append(s)

return

if d > m:

return

if m == d:

helper(s+'(', m+1, d)

elif m == n:

helper(s+')', m, d+1)

else:

helper(s+'(', m+1, d)

helper(s+')', m, d+1)

helper('', 0, 0)

return res

letter combinations of phone numbers ( c ++)

-

class Solution {

public:

vector<string> abc {"abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};

void dp(int i,string dig,string curr,vector<string>& ans)

{

if(i==dig.size())

ans.push\_back(curr);

else

{

int j=dig[i]-'0';

//going through all possible orientations for the specific digit

//j-2 as each numbers corresponding possible digits are stored at numbers-2 index

for(int a=0;a<abc[j-2].size();a++)

dp(i+1,dig,curr+abc[j-2][a],ans);

}

}

vector<string> letterCombinations(string digits) {

vector<string> ans;

if(digits.size()==0) return ans;

dp(0,digits,"",ans);

return ans;

}

};

Integer to roman ( javascript )

-

var intToRoman = function(num) {

let ones = ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X'];

let tens = ['', 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX', 'LXXX', 'XC', 'C' ];

let hundereds = ['', 'C', 'CC', 'CCC', 'CD', 'D', 'DC', 'DCC', 'DCCC', 'CM', 'M'];

let thousands = ['', 'M', 'MM', 'MMM']

let i = 1;

let romanNumeral = '';

while(num) {

let digit = num % 10;

if(i === 1) {

romanNumeral = ones[digit] + romanNumeral;

}

if( i === 2) {

romanNumeral = tens[digit] + romanNumeral;

}

if(i === 3) {

romanNumeral = hundereds[digit] + romanNumeral;

}

if(i === 4) {

romanNumeral = thousands[digit] + romanNumeral;

}

i++;

num = Math.floor(num / 10);

}

return romanNumeral;

};

// n % 10 -> returns right most digit

// Math.floor(n / 10) -> removes the right most digit