**Додаткове завдання №3**

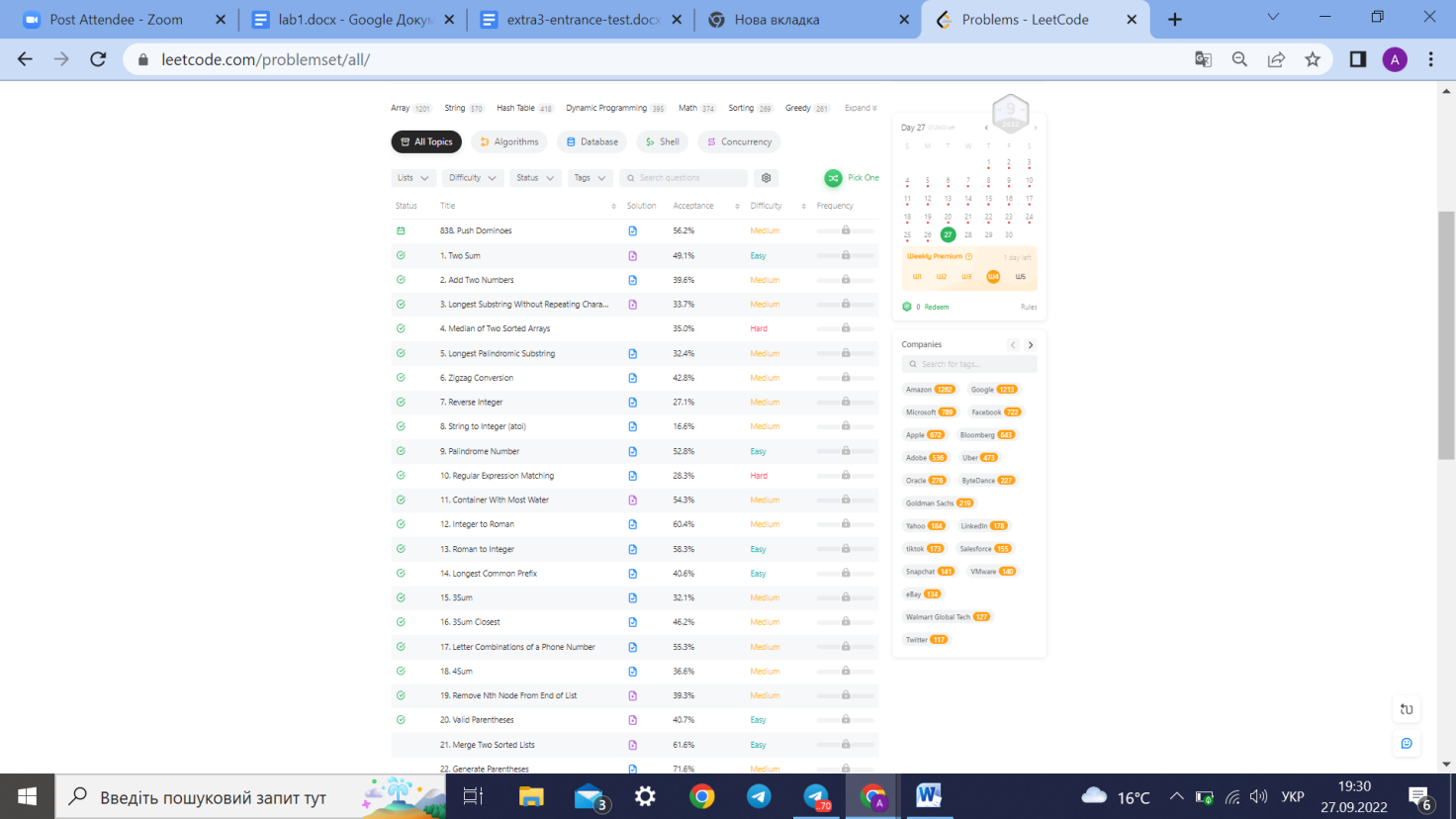
Виконала студентка

ІПС-23

Матвійчук Анастасія

Адреса профіля - <https://leetcode.com/matvchukk/>

Скріншот списку задач –



**Код виконаних завдань**

**1.**

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

std::vector<int> result;

for (size\_t i = 0; i < nums.size(); i++) {

for (size\_t j = i + 1; j < nums.size(); j++) {

if (nums[i] + nums[j] == target) {

result.push\_back(i);

result.push\_back(j);

return result;

}

}

}

return result;

}

};

**9.**

class Solution {

public:

bool isPalindrome(int x) {

bool result = false;

int temp = x;

int backward = 0;

if (x < 0) return result;

while (temp) {

backward = (backward \* 10) + (temp % 10);

temp = temp / 10;

}

if (x == backward) result = true;

return result;

}

};

**13**

class Solution {

public:

int romanToInt(std::string s) {

int result = 0;

std::vector<int> temp;

for (size\_t i = 0; i < s.length(); i++) {

switch (s[i]) {

case 'I':

temp.push\_back(1);

break;

case 'V':

temp.push\_back(5);

break;

case 'X':

temp.push\_back(10);

break;

case 'L':

temp.push\_back(50);

break;

case 'C':

temp.push\_back(100);

break;

case 'D':

temp.push\_back(500);

break;

case 'M':

temp.push\_back(1000);

break;

default:

std::cout << "default case" << std::endl;

return 0;

break;

}

}

for (size\_t i = 0; i < temp.size(); i++) {

if (i == temp.size() - 1 || temp[i] >= temp[i + 1]) result += temp[i];

else {

result -= temp[i];

}

}

return result;

}

};

**14**

class Solution {

public:

string longestCommonPrefix(vector<string>& strs) {

string result = strs[0];

string temp;

for (size\_t i = 1; i < strs.size(); i++) {

size\_t length = (result.length() < strs[i].length()) ? result.length() : strs[i].length();

for (size\_t j = 0; j < length; j++) {

if (result[j] != strs[i][j]) break;

temp.push\_back(result[j]);

}

result = temp;

temp.erase();

}

return result;

}

};

**20**

class Solution {

public:

bool isValid(string s) {

stack<char> stack;

for (size\_t i = 0; i < s.length(); i++) {

if (s[i] == '(' s[i] == '{' s[i] == '[') stack.push(s[i]);

else {

if (stack.empty() (stack.top() == '(' && s[i] != ')') (stack.top() == '{' && s[i] != '}') || (stack.top() == '[' && s[i] != ']')) return false;

stack.pop();

}

}

return stack.empty();

}

};

**2**

class Solution {

public:

ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {

ListNode \*result = new ListNode(0);

ListNode \*temp = result;

int step = 0;

while(l1 || l2){

int n = l1?l1->val:0;

int m = l2?l2->val:0;

int sum = m + n + step;

temp->next = new ListNode(sum % 10);

temp = temp->next;

step = sum / 10;

if(l1) l1 = l1->next;

if(l2) l2 = l2->next;

}

if(step) temp->next = new ListNode(1);

result = result->next;

return result;

}

};

**7**

class Solution {

public:

int reverse(int x) {

int result = 0;

while(x){

int lastDigit = x % 10;

if(result >= -(pow(2,31)/10) && result <= (pow(2,31) - 1)/10) {

result = result \* 10 + lastDigit;

x/=10;

}

else {

return 0;

}

}

return result;

}

};

**8**

class Solution {

public:

int myAtoi(string s) {

int i = s.find\_first\_not\_of(' ');

if(s.empty() i == -1)

return 0;

int result = 0;

int sign = 1;

int max = (pow(2, 31) - 1) / 10;

if (s[i] == '+' s[i] == '-')

sign = s[i++] == '+' ?: -1;

while (isdigit(s[i])) {

if (result > max || (result == max && s[i] - '0' > 7))

return sign > 0 ? pow(2, 31) - 1 : -pow(2, 31);

result = 10 \* result + (s[i++] - '0');

}

return sign \* result;

}

};

**11**

class Solution {

public:

int maxArea(vector<int>& height) {

int size = 0;

int i = 0;

int j = height.size() - 1;

while (i < j) {

int minHeight = min(height[i], height[j]);

size = max(size, (j - i) \* minHeight);

while (i < j && height[i] <= minHeight) i++;

while (i < j && height[j] <= minHeight) j--;

}

return size;

}

};

**12**

class Solution {

public:

string intToRoman(int num)

{

string result;

vector<string> romanNum = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};

vector<int> number = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};

for(size\_t i = 0; num != 0; i++)

{

while(num >= number[i])

{

result += romanNum[i];

num -= number[i];

}

}

return result;

}

};

**15**

class Solution {

public:

vector<vector<int>> threeSum(vector<int>& nums) {

vector<vector<int>> result;

std::sort(nums.begin(), nums.end());

for (size\_t i = 0; i < nums.size(); i++) {

int firstNum = nums[i];

size\_t second = i + 1;

size\_t third = nums.size() - 1;

while (second < third) {

if (nums[second] + nums[third] < -firstNum) second++;

else if (nums[second] + nums[third] > -firstNum) third--;

else {

vector<int> triplet = {nums[i], nums[second], nums[third]};

result.push\_back(triplet);

while (second < third && nums[second] == triplet[1]) second++;

while (second < third && nums[third] == triplet[2]) third--;

}

}

while (i + 1 < nums.size() && nums[i + 1] == nums[i]) {

i++;

}

}

return result;

}

};

**16**

class Solution {

public:

int threeSumClosest(vector<int>& nums, int target) {

vector<int> result;

vector<int> closeness;

int sum = 0;

std::sort(nums.begin(), nums.end());

for (size\_t i = 0; i < nums.size(); i++) {

int firstNum = nums[i];

size\_t second = i + 1;

size\_t third = nums.size() - 1;

while (second < third) {

sum = nums[second] + nums[third] + firstNum;

result.push\_back(sum);

closeness.push\_back(abs(sum - target));

if (sum < target) second++;

else if (sum > target) third--;

else {

return sum;

}

}

}

auto forMinDistance = minmax\_element(closeness.begin(), closeness.end());

return result[distance(closeness.begin(), forMinDistance.first)];

}

};

**17**

class Solution {

public:

vector<string> letterCombinations(string digits) {

if (digits.empty() || digits.length() > 4) return {};

vector<string> result;

result.push\_back("");

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

for(size\_t i = 0; i < digits.length(); i++) {

vector<string> temp;

string requiredSet = letterSets[(digits[i] - '0') - 2];

for(size\_t letter = 0; letter < requiredSet.length(); letter++) {

for(size\_t k = 0; k < result.size(); k++) {

temp.push\_back(result[k] + requiredSet[letter]);

}

}

result.swap(temp);

}

return result;

}

};

**18**

class Solution {

public:

vector<vector<int> > fourSum(vector<int> &nums, int target) {

vector<vector<int>> result;

if (nums.size() < 4) return result;

sort(nums.begin(), nums.end());

for (size\_t i = 0; i < nums.size(); i++) {

//long long target\_3 = (long long)target - (long long)nums[i];

long long target\_3 = target - nums[i];

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

long long target\_2 = target\_3 - (long long)nums[j];

int third = j + 1;

int fourth = nums.size() - 1;

while(third < fourth) {

int two\_sum = nums[third] + nums[fourth];

if (two\_sum < target\_2) third++;

else if (two\_sum > target\_2) fourth--;

else {

vector<int> quad = {nums[i], nums[j], nums[third], nums[fourth]};

result.push\_back(quad);

while (third < fourth && nums[third] == quad[2]) ++third;

while (third < fourth && nums[fourth] == quad[3]) --fourth;

}

}

while(j + 1 < nums.size() && nums[j + 1] == nums[j]) ++j;

}

while (i + 1 < nums.size() && nums[i + 1] == nums[i]) ++i;

}

return result;

}

};

**19**

class Solution {

public:

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

if (n < 1 || n > 30) return nullptr;

ListNode result(-1);

result.next = head;

ListNode \*current = &result;

ListNode \*currentWithStep = &result;

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

currentWithStep = currentWithStep->next;

while (currentWithStep->next)

{

currentWithStep = currentWithStep->next;

current = current->next;

}

ListNode \*remove = current->next;

current->next = current->next->next;

delete remove;

return result.next;

}

};

**10**

class Solution {

public:

bool isMatch(string s, string p) {

vector<vector<bool>> result(s.size() + 1, vector<bool>(p.size() + 1, false));

result[0][0] = true;

for (size\_t j = 2; j <= p.size(); ++j) {

if (p[j - 1] == '\*') {

result[0][j] = result[0][j - 2];

}

}

for (size\_t i = 1; i <= s.size(); ++i) {

for (size\_t j = 1; j <= p.size(); ++j) {

if (p[j - 1] != '\*') {

if ((p[j - 1] == '.' p[j - 1] == s[i - 1]) && result[i - 1][j - 1]) result[i][j] = true;

else result[i][j] = false;

}

else {

if (result[i][j - 2] (((p[j - 2] == '.') || (p[j - 2] == s[i - 1])) && result[i - 1][j])) result[i][j] = true;

else result[i][j] = false;

}

}

}

return result[s.size()][p.size()];

}

};

**3**

class Solution {

public:

int lengthOfLongestSubstring(string s)

{

if(s.empty()) return 0;

int length = 0;

map<char, int> charactersWithIndexes;

int temp = 0;

for (size\_t i = 0; i < s.length(); ++i)

{

if (charactersWithIndexes.find(s[i]) != charactersWithIndexes.end())

{

temp = (temp >= charactersWithIndexes.find(s[i])->second + 1) ? temp : charactersWithIndexes.find(s[i])->second + 1;

}

charactersWithIndexes[s[i]] = i;

length = (length >= i - temp + 1) ? length : i - temp + 1;

}

return length;

}

};

**5**

class Solution {

public:

string longestPalindrome(string s) {

if (s.size() <= 1) return s;

int leftPointer = 0;

int rightPointer= s.size() - 1;

int length = 1;

int result;

for (size\_t i = 0; i < s.size();) {

int left = i;

int right = i;

while (right < rightPointer && s[right + 1]==s[right]) right++;

i = right + 1;

while (right < rightPointer && left > 0 && s[right + 1] == s[left - 1]){

left--;

right++;

}

result = right - left + 1;

if (result > length) {

leftPointer = left;

length = result;

}

}

return s.substr(leftPointer, length);

}

};

**6**

class Solution {

public:

string convert(string s, int numRows) {

string result;

if(numRows == 0) return {};

if(numRows == 1) return s;

int step1;

int step2;

for(size\_t i = 0; i < numRows; ++i){

step1 = (numRows - i - 1) \* 2;

step2 = i \* 2;

int temp = i;

if(temp < s.size()) result += s[temp];

while(1) {

temp += step1; //t=6

if(temp >= s.size()) break;

if(step1) result += s[temp];

temp += step2;

if(temp >= s.size()) break;

if(step2) result += s[temp];

}

}

return result;

}

};

**4**

class Solution {

public:

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

for(size\_t i = 0; i < nums2.size(); i++) {

nums1.push\_back(nums2[i]);

}

sort(nums1.begin(), nums1.end());

if((nums1.size() % 2) != 0)

return nums1[nums1.size()/2];

else

return (nums1[nums1.size()/2] + nums1[nums1.size()/2-1])/2.00000;

}

};