export function getCompressedString(text){

if(!text.length)return ''

else if(text === '!')return '!'

else if(text === '.')return '.'

else if(text === ',')return ','

else if(text === '?')return '?'

const pattern = /[а-яё\w#]+|^[.,!?]+/gi;

// Извлечь по паттерну все слова из text

const words = text.toLowerCase().match(pattern)

// Сформировать словарик [[слово, кол-во]]

const map = new Map()

for(const word of words){

map.has(word) ?

map.set(word, map.get(word)+1) :

map.set(word, 1)

}

// Крафтим массив из словаря и сортируем массив по убыванию частоты встречаемости

const sorted = [...map].sort((a,b)=> b[1]-a[1])

map.clear()

// Перегон отсортированных слов в отдельный массив

const tmp = []

for(const word of sorted){

tmp.push(word[0])

}

sorted.length=0;

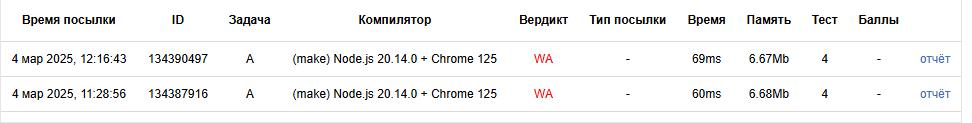
let result = text.toLowerCase();

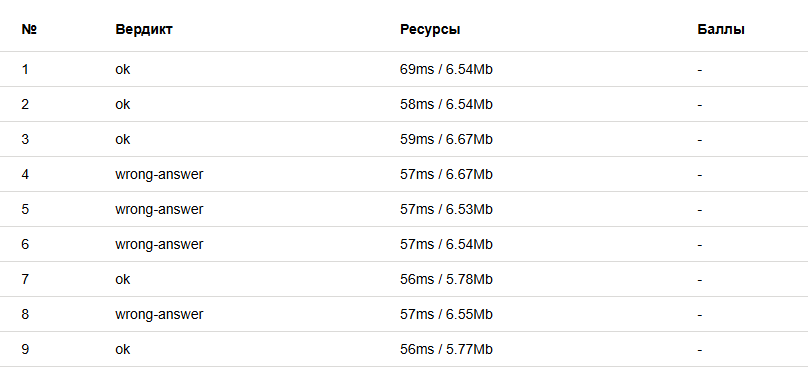
for(let i = 0; i<tmp.length; i++)

result = result.replace(new RegExp(tmp[i], "g"), `${i}`);

return result

}





function solution(input){

function sequenceIsTrue(sequence, rules){

for(const rule of rules){

const [a,b] = rule

if(!sequence.includes(a) || !sequence.includes(b))

continue

if(sequence.findIndex(item => item === a) > sequence.findIndex(item => item === b))

return false

}

return true

}

const [inputRules, inputSequences] = input.split('\n\n')

const rules = inputRules.split('\n').map(rule => rule.split('|'))

const sequences = inputSequences.split('\n').map(sequence => sequence.split(', '))

let result = 0;

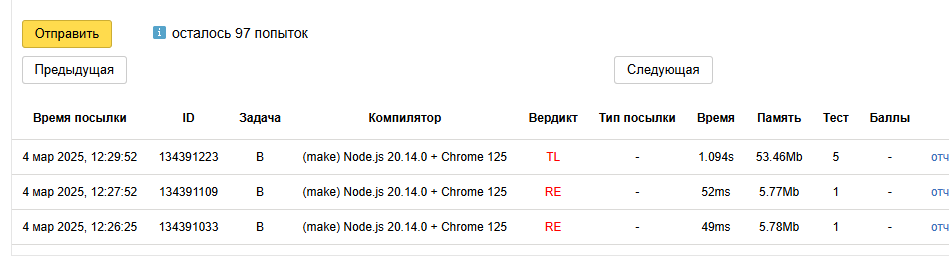
for(const sequence of sequences)

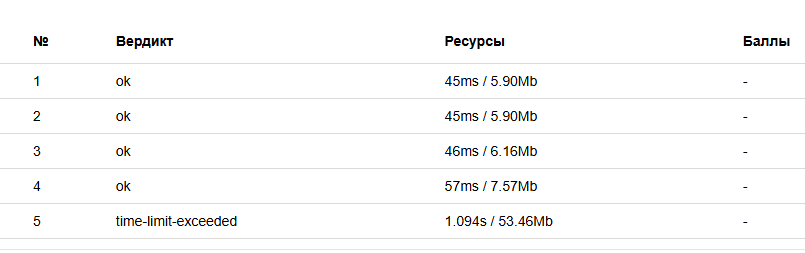
sequenceIsTrue(sequence, rules) && ++result

return result

}

module.exports = solution;





/\*

interface IRecord {

id: number

title: string

summary: string

details: string

}

function searchRecord(records: IRecord[], recordId: number): IRecord | null

\*/

function searchRecord(records, recordId){

if(!records.length)return null

for(const record of records){

if(record.id === recordId)return record

}

// if(!records.length)return null

// let from = 0;

// let to = records.length-1;

// do {

// let middle = Math.floor((from + to)/2)

// const current = records[middle].id

// if(recordId < current)

// to = middle - 1

// else if(recordId > current)

// from = middle + 1

// else

// return records[middle]

// } while(from <= to)

return null

}

//function recordsIsCorrect(records: IRecord[]): boolean

function recordsIsCorrect(records){

if(!Array.isArray(records))return false;

//for(const record of records){

//const countOfProperties = Object.keys(record).length

//if(countOfProperties !== 4)return false

//if(typeof(record.id) !== 'number')return false

//else if(typeof(record.title) !== 'string')return false

//else if(typeof(record.summary) !== 'string')return false

//else if(typeof(record.details) !== 'string')return false

//}

return true

}

/\*

interface IExtendedRecord extends IRecord {

getTitle: () => string

getSummary: () => string

getDetails: () => string

}

function getRecord(url: string, recordId: number): Promise<TRecord>

\*/

function getRecord(url, recordId){

return new Promise(async (resolve, reject) => {

const response = await fetch(url)

const records = await response.json()

const foundedRecord = searchRecord(records, recordId)

const resultRecord = {

...foundedRecord,

getTitle(){return this.title},

getSummary: () => this.summary,

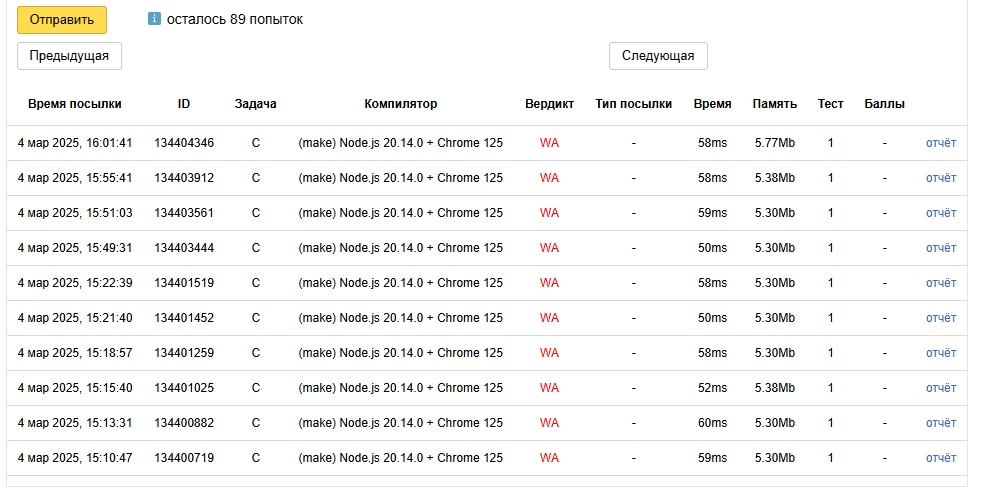
getDetails: () => this.details

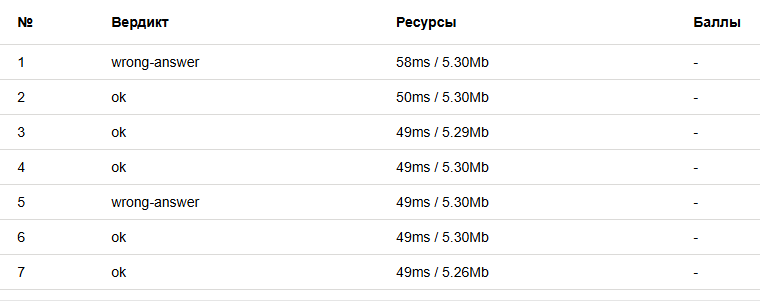
}

resolve(resultRecord)

})

}

module.exports = getRecord;



function isIntersect(a, b){

if(b.start === undefined || b.end === undefined) return true

else if(a.start === undefined || a.end === undefined) return true

if(a.start === a.end || b.start === b.end)return false

if(a.end === 0)

a.end = 24

if(b.end === 0)

b.end = 24

const min = a.start < b.start ? a : b

const max = (min === a ? b : a)

if(min.end === 0)

min.end = 24

if(max.start === 0)

max.start = 24

return ( min.end <= max.start )

}

module.exports = function createMeeting(config){

const {meetings, params} = config;

const mapMeetings = new Map();

for(const meeting of meetings){

mapMeetings.set(meeting.person, {from: meeting.from, to: meeting.to})

}

const {from, to, persons} = params

const reason = [];

for(const person of persons){

// console.log(`current person ${person}`)

// console.log(`a: {start: ${from}, end: ${to}; b: {start: ${mapMeetings.get(person).from}, end: ${mapMeetings.get(person).to}}`)

if(!isIntersect({start: from, end: to}, {start: mapMeetings.get(person)?.from, end: mapMeetings.get(person)?.to})){

// console.log('status: rejected')

reason.push(person)

}

// console.log('status: ok')

}

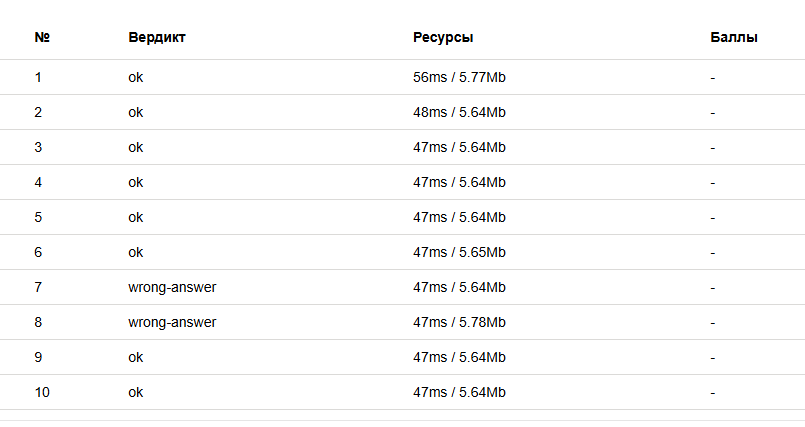
if(!reason.length)

return {status: 'CREATED', reason: null}

return {status: 'REJECTED', reason: reason.sort()}

}





// обработка сабмита

const form = document.getElementById("search");

form.onsubmit = (e) => {

clear()

const payload = e.target.value

const search = payload.value;

if(!search)return false

const list = document.getElementById("list");

const ul = list.children

for(const li of ul ){

domRangeHighlight(li, search)

}

return false;

}

// крафтим подсветку для искомых подстрок

function domRangeHighlight(node, search) {

const root = node.firstChild;

const content = root.nodeValue.toLowerCase();

const text = search === ' ' ? search : search.toLowerCase().trim();

const ranges = []

for(const index of getListIdx(content, text)){

const rng = document.createRange();

rng.setStart(root, index);

rng.setEnd(root, index + text.length);

ranges.push(rng)

}

for(let i = ranges.length-1; i >= 0; i--){

const highlightDiv = document.createElement('span');

highlightDiv.style.backgroundColor = 'yellow';

highlightDiv.className = 'highlight'

ranges[i].surroundContents(highlightDiv);

}

}

// получить массив индексов вхождений искомой подстроки

function getListIdx(str, substr) {

let listIdx = []

let lastIndex = -1

while ((lastIndex = str.indexOf(substr, lastIndex + 1)) !== -1) {

listIdx.push(lastIndex)

}

return listIdx

}

// удаление подсветки по предыдущему поиску

function clear(){

document.querySelectorAll('li').forEach(el => {

const spans = el.querySelectorAll('.highlight')

for(const span of spans){

el.innerText = el.getAttribute('data-value')

}

})

}

