Министерство цифрового развития, связи и массовых коммуникаций Государственное образовательного учреждение высшего образования

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Лабораторная работа № 2

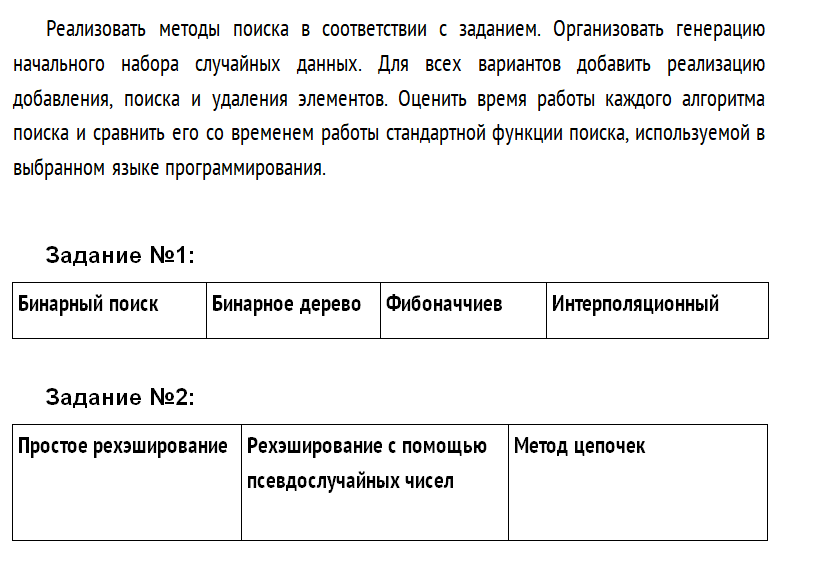
по дисциплине «Структура и алгоритмы обработки данных»

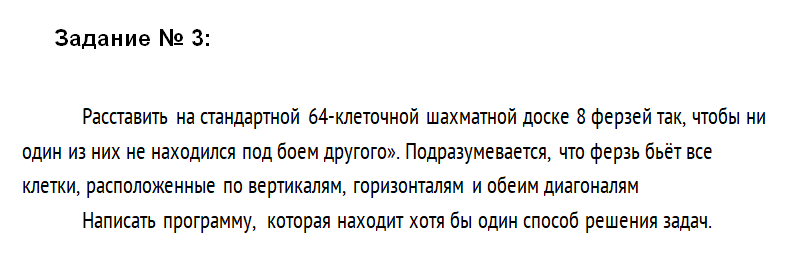
Выполнил студент группы БФИ-1901:

Бардюк Д. В.

Москва 2021

# Задание





Код программы

Задание 1

function generateArray(length) {

    let array = [length],

        minLimit = 0,

        maxLimit = 100;

    for (let i = 0; i < length; i++) {

        array[i] = minLimit + Math.floor(Math.random() \* (maxLimit - minLimit + 1));

    }

    return array;

}

function binarySearch(value, array) {

    let mass = array.sort((first, second) => first - second),

        first = mass[0],

        last = mass[mass.length - 1],

        position = -1,

        check = false,

        middle;

    while (check === false && first <= last) {

        middle = Math.floor((last + first) / 2);

        if (mass[middle] == value) {

            position = middle;

            check = true;

        } else if (mass[middle] > value) {

            last = mass[middle] - 1;

        } else {

            first = mass[middle] + 1;

        }

    }

    return position;

}

function InterpolationSearch(value, array) {

    let mass = array.sort((first, second) => first - second),

        low = 0,

        high = mass.length - 1,

        zond;

    while (mass[low] < value && mass[high] > value) {

        zond = low + Math.floor(((value - mass[low]) \* (high - low)) / (mass[high] - mass[low]));

        if (value < mass[zond]) {

            high = zond - 1;

        } else if (value > mass[zond]) {

            low = zond + 1;

        } else return zond;

    }

    if (mass[low] == value) return low;

    else if (mass[high] == value) return high;

    else return -1;

}

class Node {

    constructor(data) {

        this.data = data; // node value

        this.left = null; // left node child reference

        this.right = null; // right node child reference

    }

}

class BinarySearchTree {

    constructor() {

        this.root = null; // корень bst

    }

    insert(data) {

        let newNode = new Node(data);

        if (this.root === null) {

            this.root = newNode;

        } else {

            this.insertNode(this.root, newNode);

        }

    }

    insertNode(node, newNode) {

        if (newNode.data < node.data) {

            if (node.left === null) {

                node.left = newNode;

            } else {

                this.insertNode(node.left, newNode);

            }

        } else {

            if (node.right === null) {

                node.right = newNode;

            } else {

                this.insertNode(node.right, newNode);

            }

        }

    }

    search(node, data) {

        if (node === null) {

            return 'Sorry, element is undefinded';

        } else if (data < node.data) {

            return this.search(node.left, data);

        } else if (data > node.data) {

            return this.search(node.right, data);

        } else {

            return node;

        }

    }

    remove(data) {

        this.root = this.removeNode(this.root, data); // helper method below

    }

    removeNode(node, data) {

        if (node === null) {

            return null;

        // если данные, которые нужно удалить, меньше, чем данные корня, переходим к левому поддереву

        } else if (data < node.data) {

            node.left = this.removeNode(node.left, data);

            return node;

        // если данные, которые нужно удалить, больше, чем данные корня, переходим к правому поддереву

        } else if (data > node.data) {

            node.right = this.removeNode(node.right, data);

            return node;

        // если данные такие как данные корня, удаляем узел

        } else {

            // удаляем узел без потомков (листовой узел (leaf) или крайний)

            if (node.left === null && node.right === null) {

                node = null;

                return node;

            }

            // удаляем узел с одним потомком

            if (node.left === null) {

                node = node.right;

                return node;

            } else if(node.right === null) {

                node = node.left;

                return node;

            }

            // удаляем узел с двумя потомками

            // minNode правого поддерева хранится в новом узле

            let newNode = this.minNode(node.right);

            node.data = newNode.data;

            node.right = this.removeNode(node.right, newNode.data);

            return node;

        }

    }

}

function fibonachchi(value) {

    let f1 = 0,

        f2 = 1,

        cf = 1;

    for (let i = 1; i <= value; i++) {

        cf = f1 + f2;

        f1 = f2;

        f2 = cf;

    }

    return cf;

}

function fibonachchiSearch(value, start = 0, result = 0, array) {

    let mass = array.sort((first, second) => first - second),

    check = true,

    index = 0,

    f = 0;

    console.log(mass);

    while(check){

        f = fibonachchi(index);

        if(f > mass.length - 1){

            f = mass.length-1;

            if (mass[f] < value || mass.length == 0){return 'sorry'}

        }

        if(mass[f] == value){

            console.log('success');

            result+=f;

            return result;

        } else if (mass[f] > value){

            start = fibonachchi(index - 1);

            result+=start;

            check = false;

        } else { index++; }

    }

    if(check == false){

        mass = mass.splice(start,f-1);

        return fibonachchiSearch(value, start, result, mass);

    }

}

let array = generateArray(100);

let test = [1,2,3,4,5,6,8,19,20,22,23];

console.log(array);

const startNative = Date.now();

console.log(indexOf(array))

const endNative = Date.now();

console.log(`time is ${endNative - startNative}`)

// const start = Date.now();

// console.log(binarySearch(6, [4,6,5,1,2,3,11]));

// const end = Date.now();

// console.log(`time is ${end-start}'ms`);

// const start = Date.now();

// console.log(InterpolationSearch(101, array))

// const end = Date.now();

// console.log(`time is ${end-start}'ms`)

// let bTree = new BinarySearchTree()

// array.forEach(data => bTree.insert(data))

// const start = Date.now();

// console.log(bTree.search(bTree.root, 40))

// const end = Date.now();

// console.log(`time is ${end-start}'ms`)

// const startTime = Date.now();

// let start, index,result;

// console.log(fibonachchiSearch(101, start, result, array))

// const endTime = Date.now();

// console.log(`time is ${endTime-startTime}'ms`)

# Задание 2

const test = [8,19,14,12,10,5,7];

let hashTable = new Map(),

    hashRefTable = [];

function hashValue(value){

    return value%7;

}

function reHash(value){

    const hash = hashValue(value);

    for (let i = 0; i < 7; i++){

        if(hashTable.has((hash+i) % 7) == false){

            hashTable.set((hash+i) % 7, value);

            break;

        }

    }

}

function simpleReHash(num, array){

    array.forEach(value => reHash(value));

    const hash = hashValue(num);

    for (let [key, value] of hashTable){

        console.log(`в ячейке ${key} cодержится ${value} `);

    }

    for (let i = 0; i<7; i++){

        if(hashTable.get((hash + i)) == undefined){

            return `Элемент ${num} не найден` ;

        }

            else if( num == hashTable.get((hash+i)%7)) {

                return `Элемент ${num} найден в ячейке ${(hash+i)%7}`;

            }

    }

    return ` Элемент не найден `;

}

console.log(simpleReHash(10, test));

const hashTable = new Map(),

      test = [8,19,14,12,10,5,7];

class LinkedListNode {

    constructor(value, next = null) {

        this.value = value;

        this.next = next;

    }

    append(value) {

        if(this.next == null){

            this.next = new LinkedListNode(value);

        } else {

            this.next.append(value);

        }

    }

    find(value) {

        let currentNode = this;

        while (currentNode) {

            if (value !== undefined && currentNode.value === value) {

                return `Элемент ${value} найден в ячейке ${hashValue(value)}`;

            }

            currentNode = currentNode.next;

        }

        return null;

    }

}

function hashValue(value){

    return value%7;

}

function hashChain(array) {

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

        if(hashTable.has(hashValue(array[i])) == false){

            let node = new LinkedListNode(array[i], null);

            hashTable.set(hashValue(array[i]), node);

        } else {

            hashTable.get(hashValue(array[i])).append(array[i]);

        }

    }

    return hashTable;

}

function foundChain(num) {

    const hash = hashValue(num);

    if (hashTable.has(hash)){

        return hashTable.get(hash).find(num);

    }

    else {

        return ` Элемент не найден `;

    }

}

console.log(hashChain(test));

console.log(foundChain(7))

Задание 3

class chessBoard {

    constructor(a, b, c, d, e, f, g, h) {

        this.a = a;

        this.b = b;

        this.c = c;

        this.d = d;

        this.e = e;

        this.f = f;

        this.g = g;

        this.h = h;

    }

}

function chessQuins() {

    const board = {};

    let map = new Map(),

        diag1 = [],

        diag2 = [];

    for (let i = 0; i < 8; i++) {

        const xArr = ['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x'];

            for (let j = 0; j < 8; j++) {

                if (map.has(j) == 1) {

                    continue;

                }

                if (diag1.indexOf(i + j) == -1 && diag2.indexOf(Math.abs(i - j)) == -1) {

                    xArr[j] = 'Q';

                    map.set(j, 1);

                    diag1.push(i + j);

                    diag2.push(Math.abs(i - j));

                    board[(i + 1).toString()] = new chessBoard(...xArr);

                    break;

                }

            }

        board[(i + 1).toString()] = new chessBoard(...xArr);

    }

    return board;

}

console.table(chessQuins())

Результат работы

На рисунке 1 представлен результат работы программы задания 1

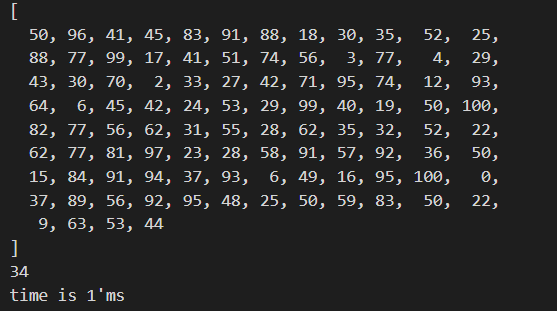


Рисунок - 1 Работа алгоритма поиска

На рисунке 1 представлен результат работы программы задания 2

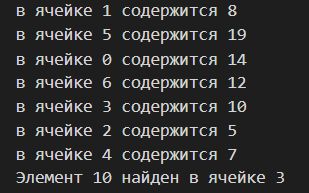


Рисунок - 2 Работа алгоритма хеширования

На рисунке 3 представлен результат работы программы задания 3

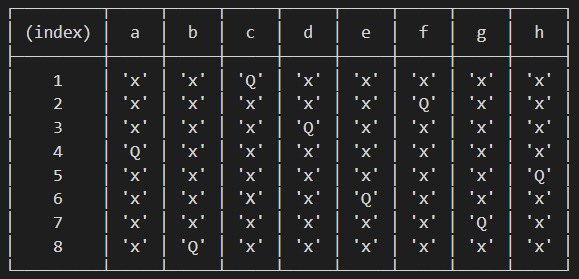


Рисунок - 3 Работа алгоритма расстановки ферзей

**Вывод**: в ходе выполнения данной работы я узнал об особенностях алгоритмов поиска элемента, написал каждый из них. Также узнал о работе алгоритмов хеширования и поиска элементов в хэш таблицах, разработал программу о расстановке 8 ферзей.