Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчёт по лабораторной работе № 2

«Методы поиска»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент группы БВТ1902

Холмовский Олег Васильевич

Москва

2021

Задание

Реализовать методы поиска в соответствии с заданием. Организовать генерацию начального набора случайных данных. Для всех вариантов добавить реализацию добавления, поиска и удаления элементов.

Задание №1:

|  |  |  |  |
| --- | --- | --- | --- |
| **Бинарный поиск** | **Бинарное дерево** | **Фибоначчиев** | **Интерполяционный** |

Задание №2:

|  |  |
| --- | --- |
| **Простое рехэширование** | **Метод цепочек** |

Задание №3:

Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого». Подразумевается, что ферзь бьёт все клетки, расположенные по вертикалям, горизонталям и обеим диагоналям. Написать программу, которая находит хотя бы один способ решения задач.

Код программы

Методы поиска:

const createRandomArray = (length = 10, min = 0, max = 10) => {

let array = [];

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

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

}

*return* array;

};

const sortedArray = createRandomArray().sort((a, b) => a - b);

console.log(`generated random sorted array: ${sortedArray}`);

let binarySearchArray = [...sortedArray];

*// O(logN)*

const binarySearch = (array, value) => {

let start = 0;

let end = array.length - 1;

*while* (start <= end) {

let middle = Math.floor((start + end) / 2);

*if* (array[middle] === value) {

*return* { status: true, index: middle };

} *else* *if* (array[middle] < value) {

start = middle + 1;

} *else* {

end = middle - 1;

}

}

*return* { status: false, index: end + 1 };

};

const binarySearchFind = (array, value) => {

binarySearch(array, value).status

? console.log(`binarySearchFind: value ${value} found`)

: console.log(`binarySearchFind: value ${value} not found`);

};

const binarySearchDelete = (array, value) => {

let { status, index } = binarySearch(array, value);

*if* (status) {

array.splice(index, 1);

console.log(

`binarySearchDelete: value ${value} deleted, edited arrray: ${array}`

);

} *else* console.log(`binarySearchDelete: value ${value} not found`);

};

const binarySearchAdd = (array, value) => {

let { status, index } = binarySearch(array, value);

array.splice(index, 0, value);

console.log(`binarySearchAdd: value ${value} added, edited arrray: ${array}`);

};

binarySearchFind(binarySearchArray, 11);

binarySearchDelete(binarySearchArray, 5);

binarySearchAdd(binarySearchArray, 3);

*// O(logN)*

*class* Node {

constructor(data, left = *null*, right = *null*) {

*this*.data = data;

*this*.left = left;

*this*.right = right;

}

}

*class* BST {

constructor() {

*this*.root = *null*;

}

add(data) {

const node = *this*.root;

*if* (node === *null*) {

*this*.root = *new* Node(data);

*return*;

} *else* {

const searchTree = *function* (node) {

*if* (data < node.data) {

*if* (node.left === *null*) {

node.left = *new* Node(data);

*return*;

} *else* *if* (node.left !== *null*) {

*return* searchTree(node.left);

}

} *else* *if* (data > node.data) {

*if* (node.right === *null*) {

node.right = *new* Node(data);

*return*;

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

*return* searchTree(node.right);

}

} *else* {

*return* *null*;

}

};

*return* searchTree(node);

}

}

find(data) {

let current = *this*.root;

*while* (current.data !== data) {

*if* (data < current.data) {

current = current.left;

} *else* {

current = current.right;

}

*if* (current === *null*) {

*return* *null*;

}

}

*return* current;

}

isPresent(data) {

let current = *this*.root;

*while* (current) {

*if* (data === current.data) {

*return* `bstFind: value ${data} found`;

}

*if* (data < current.data) {

current = current.left;

} *else* {

current = current.right;

}

}

*return* `bstFind: value ${data} not found`;

}

remove(data) {

const removeNode = *function* (node, data) {

*if* (node == *null*) {

*return* *null*;

}

*if* (data == node.data) {

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

*return* *null*;

}

*if* (node.left == *null*) {

*return* node.right;

}

*if* (node.right == *null*) {

*return* node.left;

}

var tempNode = node.right;

*while* (tempNode.left !== *null*) {

tempNode = tempNode.left;

}

node.data = tempNode.data;

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

*return* node;

} *else* *if* (data < node.data) {

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

*return* node;

} *else* {

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

*return* node;

}

};

*this*.root = removeNode(*this*.root, data);

}

inOrder() {

*if* (*this*.root == *null*) {

*return* *null*;

} *else* {

var result = *new* Array();

*function* traverseInOrder(node) {

node.left && traverseInOrder(node.left);

result.push(node.data);

node.right && traverseInOrder(node.right);

}

traverseInOrder(*this*.root);

*return* result;

}

}

}

const bst = *new* BST();

*for* (let num *of* sortedArray) {

bst.add(num);

}

const bstDelete = (value) => {

*if* (bst.find(value)) {

bst.remove(value);

console.log(

`bstDelete: value ${value} deleted, edited arrray: ${bst.inOrder()}`

);

} *else* console.log(`bstDelete: value ${value} not found`);

};

console.log(bst.isPresent(11));

bstDelete(5);

let fibMonaccianSearchArray = [...sortedArray];

*// O(logN)*

const fibMonaccianSearch = (arr, value) => {

let n = arr.length;

let fibMMm2 = 0;

let fibMMm1 = 1;

let fibM = fibMMm2 + fibMMm1;

*while* (fibM < n) {

fibMMm2 = fibMMm1;

fibMMm1 = fibM;

fibM = fibMMm2 + fibMMm1;

}

let offset = -1;

*while* (fibM > 1) {

let i = Math.min(offset + fibMMm2, n - 1);

*if* (arr[i] < value) {

fibM = fibMMm1;

fibMMm1 = fibMMm2;

fibMMm2 = fibM - fibMMm1;

offset = i;

} *else* *if* (arr[i] > value) {

fibM = fibMMm2;

fibMMm1 = fibMMm1 - fibMMm2;

fibMMm2 = fibM - fibMMm1;

} *else* *return* { status: true, index: i };

}

*if* (fibMMm1 && arr[n - 1] == value) {

*return* n - 1;

}

*return* { status: false, index: offset + 1 };

};

const fibMonaccianSearchFind = (array, value) => {

fibMonaccianSearch(array, value).status

? console.log(`fibMonaccianSearchFind: value ${value} found`)

: console.log(`fibMonaccianSearchFind: value ${value} not found`);

};

const fibMonaccianSearchDelete = (array, value) => {

let { status, index } = fibMonaccianSearch(array, value);

*if* (status) {

array.splice(index, 1);

console.log(

`fibMonaccianSearchDelete: value ${value} deleted, edited arrray: ${array}`

);

} *else* console.log(`fibMonaccianSearchDelete: value ${value} not found`);

};

const fibMonaccianSearchAdd = (array, value) => {

let { status, index } = fibMonaccianSearch(array, value);

array.splice(index, 0, value);

console.log(

`fibMonaccianSearchAdd: value ${value} added, edited arrray: ${array}`

);

};

fibMonaccianSearchFind(fibMonaccianSearchArray, 11);

fibMonaccianSearchDelete(fibMonaccianSearchArray, 5);

fibMonaccianSearchAdd(fibMonaccianSearchArray, 3);

let interpolationSearchArray = [...sortedArray];

*// O(loglogN)*

const interpolationSearch = (arr, target) => {

let left = 0;

let right = arr.length - 1;

*while* (left <= right) {

const rangeDelta = arr[right] - arr[left];

const indexDelta = right - left;

const valueDelta = target - arr[left];

*if* (valueDelta < 0) {

*return* { status: false, index: left };

}

*if* (!rangeDelta) {

*return* arr[left] === target

? { status: true, index: left }

: { status: false };

}

const middleIndex =

left + Math.floor((valueDelta \* indexDelta) / rangeDelta);

*if* (arr[middleIndex] === target) {

*return* { status: true, index: middleIndex };

}

*if* (arr[middleIndex] < target) {

left = middleIndex + 1;

} *else* {

right = middleIndex - 1;

}

}

*return* { status: false, index: left };

};

const interpolationSearchFind = (array, value) => {

interpolationSearch(array, value).status

? console.log(`interpolationSearchFind: value ${value} found`)

: console.log(`interpolationSearchFind: value ${value} not found`);

};

const interpolationSearchDelete = (array, value) => {

let { status, index } = interpolationSearch(array, value);

*if* (status) {

array.splice(index, 1);

console.log(

`interpolationSearchDelete: value ${value} deleted, edited arrray: ${array}`

);

} *else* console.log(`interpolationSearchDelete: value ${value} not found`);

};

const interpolationSearchAdd = (array, value) => {

let { status, index } = interpolationSearch(array, value);

array.splice(index, 0, value);

console.log(

`interpolationSearchAdd: value ${value} added, edited arrray: ${array}`

);

};

interpolationSearchFind(interpolationSearchArray, 11);

interpolationSearchDelete(interpolationSearchArray, 5);

interpolationSearchAdd(interpolationSearchArray, 3);

Простое рехэширование:

*class* HashTable {

constructor(size = 32) {

*this*.storage = *new* Array(size);

*this*.size = size;

*this*.i = 1;

}

print() {

console.log(*this*.storage);

}

hash(key) {

*return* key.toString().length % *this*.size;

}

set(key, value) {

let index = *this*.hash(key);

*if* (!*this*.storage[index]) {

*this*.storage[index] = [];

}

*if* (*this*.storage[index].length > 0) {

*this*.storage[(index + *this*.i \* index) % *this*.size] = [[key, value]];

*this*.i++;

} *else* *this*.storage[index].push([key, value]);

*return* index;

}

get(key) {

let index = *this*.hash(key);

*if* (*this*.storage[index][0][0] !== key) {

*return* *this*.storage[(index + 1 \* index) % *this*.size][0][1];

}

*return* *this*.storage[index][0][1];

}

remove(key) {

let index = *this*.hash(key);

*if* (*this*.storage[index].length === 1 && *this*.storage[index][0][0] === key) {

*delete* *this*.storage[index];

} *else* {

*for* (var i = 0; i < *this*.storage[index].length; i++) {

*if* (*this*.storage[index][i][0] === key) {

*delete* *this*.storage[index][i];

}

}

}

}

}

const ht = *new* HashTable(10);

ht.set('Josh', '100$');

ht.set('Max', '50$');

ht.set('Alex', '200$');

ht.set('Iliot', '10$');

ht.print();

console.log(ht.get('Josh'));

console.log(ht.get('Alex'));

ht.remove('Josh');

ht.print();

Метод цепочек:

*class* HashTable {

constructor(size = 32) {

*this*.storage = *new* Array(size);

*this*.size = size;

}

print() {

console.log(*this*.storage);

}

hash(key) {

*return* key.toString().length % *this*.size;

}

set(key, value) {

let index = *this*.hash(key);

*if* (!*this*.storage[index]) {

*this*.storage[index] = [];

}

*this*.storage[index].push([key, value]);

*return* index;

}

get(key) {

let index = *this*.hash(key);

*if* (!*this*.storage[index]) *return* *null*;

*for* (let bucket *of* *this*.storage[index]) {

*if* (bucket[0] === key) {

*return* bucket[1];

}

}

}

remove(key) {

let index = *this*.hash(key);

*if* (*this*.storage[index].length === 1 && *this*.storage[index][0][0] === key) {

*delete* *this*.storage[index];

} *else* {

*for* (var i = 0; i < *this*.storage[index].length; i++) {

*if* (*this*.storage[index][i][0] === key) {

*delete* *this*.storage[index][i];

}

}

}

}

}

const ht = *new* HashTable(10);

ht.set('Josh', '100$');

ht.set('Max', '50$');

ht.set('Alex', '200$');

ht.set('Iliot', '10$');

ht.print();

console.log(ht.get('Josh'));

console.log(ht.get('Alex'));

console.log(ht.get('Josh1'));

ht.remove('Alex');

ht.print();

Задача на расстановку ферзей:

const solveQueens = () => {

n = 8;

let result = [];

const queens = Array(n).fill(-1);

*function* makeBoard() {

const board = [];

*for* (let r = 0; r < n; ++r) {

let row = '';

*for* (let c = 0; c < n; ++c) {

*if* (queens[r] === c) {

row = row + 'Q';

} *else* {

row = row + '.';

}

}

board.push(row);

}

*return* board;

}

*function* isValid(row, col) {

*for* (let i = 0; i < row; ++i) {

*if* (queens[i] == col || Math.abs(i - row) == Math.abs(queens[i] - col))

*return* false;

}

*return* true;

}

*function* solve(row) {

*if* (row === n) {

const board = makeBoard();

result.push(board);

}

*for* (let col = 0; col < n; ++col) {

*if* (!isValid(row, col)) {

*continue*;

}

queens[row] = col;

solve(row + 1);

queens[row] = -1;

}

}

solve(0);

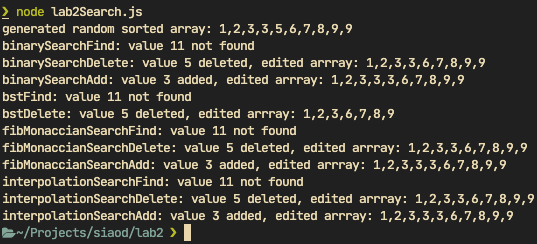
*return* result;

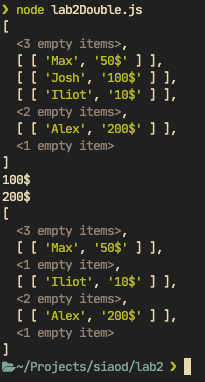
};

console.log(solveQueens());

console.log(solveQueens().length);

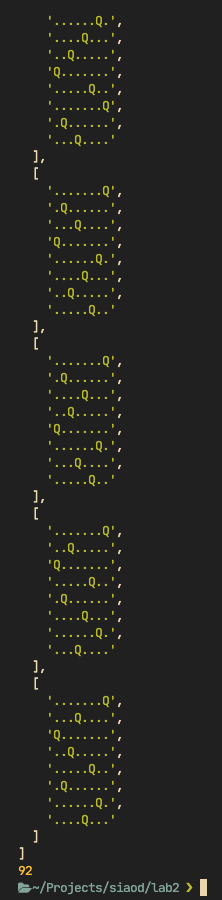
Cнимки экрана работы программы











Вывод

В ходе выполнения лабораторной работы, мной были получены навыки реализации различных алгоритмов поиска на примере числовых массивов, а так же реализации хэширования на примере рехэширования и метода цепочек.