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

«Методы поиска»

Выполнил студент

группы БВТ1902

Гинатуллин Айгиз

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 | Рехэширование с помощью  псевдослучайных чисел | 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.

# Выполнение

Листинг программы:

Lab2

import {createStore} from "redux";  
import {rootReducer} from "./redux/rootReducer.js";  
import {  
 binaryTreeSearch,  
 fibonacciSearch,  
 interpolationSearch,  
 binarySearch,  
 getPrimeArray,  
 getPrimarySought,  
 getHashSearch,  
 getHashArray,  
 getHashSearchNode,  
 getHashSearchRandom,  
 queenSetBoardSize,  
 queenSetQueensQuantity,  
 queenSearchAnswer,  
 queenGetAnswer,  
 hashDelete,  
 nudeHashDelete  
} from "./redux/actions.js";  
  
  
  
const store = createStore(rootReducer);  
  
store.dispatch(getPrimeArray())  
store.dispatch(getHashArray())  
  
store.dispatch(getPrimarySought())  
store.dispatch(binaryTreeSearch())  
store.dispatch(binarySearch())  
store.dispatch(fibonacciSearch())  
store.dispatch(interpolationSearch())  
  
store.dispatch(hashDelete())  
store.dispatch(nudeHashDelete())  
store.dispatch(getHashSearch())  
store.dispatch(getHashSearchNode())  
  
store.dispatch(queenSetBoardSize(8))  
store.dispatch(queenSetQueensQuantity(8))  
store.dispatch(queenSearchAnswer())  
store.dispatch(queenGetAnswer())

NextFib

export default (number) =>{  
 let array = []  
 array.push(1)  
 array.push(1)  
 while (array[array.length-1] <= number){  
 array.push(array[array.length-1]  
 +array[array.length-2])  
 }  
 return array  
}

HashTables

export class HashTable {  
 store = new ***Array***(32);  
  
 hash(key) {  
 let sum = 0;  
 for (let i = 0; i < key.length; i++) {  
 sum += key.charCodeAt(i);  
 }  
 return sum % this.store.length;  
 }  
  
 add(key) {  
 this.store[this.hash(key)] = key;  
 }  
  
 get(key) {  
 return [this.hash(key), this.store[this.hash(key)]]  
 }  
  
 delete(key) {  
 return this.store[this.hash(key)] = null  
 }  
}  
  
export class HashTableRandom {  
 store = new ***Array***(32);  
  
 hash(key) {  
 let sum = 0;  
 for (let i = 0; i < key.length; i++) {  
 sum += key.charCodeAt(i) \* i;  
 }  
 return sum % this.store.length;  
 }  
  
 // hash(a)  
 // {  
 // a = (a+0x7ed55d16) + (a<<12);  
 // a = (a^0xc761c23c) ^ (a>>19);  
 // a = (a+0x165667b1) + (a<<5);  
 // a = (a+0xd3a2646c) ^ (a<<9);  
 // a = (a+0xfd7046c5) + (a<<3);  
 // a = (a^0xb55a4f09) ^ (a>>16);  
 // if( a < 0 ) a = 0xffffffff + a;  
 // return a;  
 // }  
 add(key) {  
 this.store[this.hash(key)] = key;  
 }  
  
 get(key) {  
 return [this.hash(key), this.store[this.hash(key)]]  
 }  
  
 delete(key) {  
 this.store[this.hash(key)] = null  
 }  
}  
  
  
export class HashTableNode {  
 store = new ***Array***(30);  
  
 hash(key) {  
 let sum = 0;  
  
 for (let i = 0; i < key.length; i++) {  
 sum += key.charCodeAt(i);  
 }  
 return sum % this.store.length;  
 }  
  
 add(key) {  
 this.store[this.hash(key)] = this.store[this.hash(key)] || [];  
 this.store[this.hash(key)].push({  
 key: () => {  
 return this.hash(key)  
 },  
 value: key  
 }  
 );  
 }  
  
 get(key) {  
 const item = (this.store[this.hash(key)] || []).find((item) => item.value === key);  
 return item && [this.hash(key), item.value];  
 }  
  
 delete(key) {  
 return this.store[this.hash(key)] = null  
 }  
}

Print

let r=0;  
export const Print = (boardSize, board) => {  
 let abc =["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]  
 let ans = new ***Array***()  
 for (let i = 0; i < boardSize; i++) {  
 for (let j = 0; j < boardSize; j++){  
 if(board[i][j]===-1){  
 ans.push(abc[j]+(i+1))  
 }  
 }  
 }  
 r++  
 //console.log(r+ans)  
 return(r+" "+ans)  
}

queensQuantity

import {Solve} from "./solve.js";  
  
export const Queens = (boardSize, queensQuantity) => {  
 let board = new ***Array***()  
 for (let i = 0; i < boardSize; i++) {  
 board[i] = new ***Array***();  
 for (let j = 0; j < boardSize; j++) {  
 board[i][j] = 0  
 }  
 }  
 let answers = new ***Array***()  
 return Solve(0,boardSize,board,queensQuantity,answers)  
  
}

removeQueen

export const RemoveQueens = (i, j, boardSize, board) => {  
 for (let x = 0; x < boardSize; x++) {  
 board[x][j] -= 1  
 board[i][x] -= 1  
 if (0 <= i + j - x && i + j - x < boardSize) {  
 board[i + j - x][x]-=1  
 }  
 if (0 <= i - j + x && i - j + x < boardSize) {  
 board[i - j + x][x]-=1  
 }  
 }  
 board[i][j]= 0  
 return(board)  
}

setQueen

export const SetQueens = (i, j, boardSize, board) => {  
 for (let x = 0; x < boardSize; x++) {  
 board[x][j] += 1  
 board[i][x] += 1  
 if (0 <= i + j - x && i + j - x < boardSize) {  
 board[i + j - x][x]+=1  
 }  
 if (0 <= i - j + x && i - j + x < boardSize) {  
 board[i - j + x][x]+=1  
 }  
 }  
 board[i][j]= -1  
 return(board)  
}

solve

import {SetQueens} from "./setQueen.js";  
import {Print} from "./Print.js";  
import {RemoveQueens} from "./removeQueen.js";  
  
  
export const Solve = (i, boardSize, board,queensQuantity, answers) => {  
 for (let j = 0; j < boardSize; j++) {  
 if(board[i][j]===0){  
 board = SetQueens(i,j,boardSize,board)  
 if(i===queensQuantity-1) {  
 answers.push(Print(boardSize, board))  
 }  
 else  
 Solve(i+1,boardSize,board,queensQuantity,answers)  
 board = RemoveQueens(i,j,boardSize,board)  
 }  
 }  
 return answers || "None";  
}

binarySearch

export const binarySearch = (data,target,start=0, end=data.length,) => {  
  
 if(end < 1) return data[0].index;  
 const middle = Math.floor((start + (end - start)/2));  
  
 if (target === data[middle].value)  
 return data[middle].index;  
  
 if (end - 1 === start)  
 return Math.abs(data[start].value - target) > Math.abs(data[end].value - target)  
 ? data[end].index : data[start].index;  
  
 if (target > data[middle].value)  
 return binarySearch(data,target,middle,end);  
  
 if (target < data[middle].value)  
 return binarySearch(data,target,start,middle);  
}

binaryTree

export class BinaryTree {  
 constructor() {  
 this.root = null; // корень BT  
 }  
  
 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 null;  
 } 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.index;  
 }  
 }  
}  
  
class Node {  
 constructor(data) {  
 this.index = data.index  
 this.data = data.value  
 this.left = null  
 this.right = null  
 }  
}

fibonachiSearch

import nextFib from "../common/nexFib.js";  
  
export default (array,sought) =>{  
 let fibArray = nextFib(array.length+1)  
 let k = fibArray.length-1  
 let m = fibArray[fibArray.length-1] - array.length - 1  
 let i = fibArray[k] - m  
 let p = fibArray[k-1]  
 let q = fibArray[k-2]  
 return fibSearchLoop(array,sought,i,p,q)  
}  
const fibSearchLoop = (array,sought,i,p,q) => {  
 if(i < 0){  
 if(p===1){  
 return -1  
 }  
 i = i + q  
 p -= q  
 q -=p  
 return fibSearchLoop(array,sought,i,p,q)  
 }  
 if(i >= array.length){  
 if(q === 0){  
 return -1  
 }  
 i -=q  
 const t = p  
 p=q  
 q=t-q  
 return fibSearchLoop(array,sought,i,p,q)  
 }  
 if(sought < array[i].value){  
 if(q === 0){  
 return -1  
 }  
 i -=q  
 const t = p  
 p=q  
 q=t-q  
 return fibSearchLoop(array,sought,i,p,q)  
 }  
 if(sought > array[i].value){  
 if(p===1){  
 return -1  
 }  
 i = i + q  
 p -= q  
 q -=p  
 return fibSearchLoop(array,sought,i,p,q)  
 }  
 return array[i].index  
}

interpolationSearch

export const interpolationSearch = (array,sought,l,r) =>{  
  
 const indexSought =  
 Math.round((sought-array[l].value)  
 \*(l-r)/(array[l].value-array[r].value) + l)  
  
 if(array[indexSought].value === sought) return array[indexSought].index  
  
 if(array[indexSought].value > sought){  
 r = indexSought -1  
 return interpolationSearch(array,sought,l,r)  
 }  
  
 if(array[indexSought].value < sought){  
 l = indexSought +1  
 return interpolationSearch(array,sought,l,r)  
 }  
  
}

redux actions

import {  
 ***BINARY\_SEARCH***,  
 ***BINARY\_TREE\_SEARCH***,  
 ***FIBONACCI\_SEARCH***, ***GET\_HASH\_ARRAY***,  
 ***GET\_PRIME\_ARRAY***,  
 ***GET\_PRIME\_SOUGHT***, ***HASH\_DELETE***, ***HASH\_DELETE\_NODE***, ***HASH\_SEARCH***, ***HASH\_SEARCH\_NODE***, ***HASH\_SEARCH\_RANDOM***,  
 ***INTERPOLATION\_SEARCH***, ***QUEEN\_GET\_ANSWER***, ***QUEEN\_SEARCH\_ANSWER***, ***QUEEN\_SET\_BOARD\_SIZE***, ***QUEEN\_SET\_QUEENS\_QUANTITY***} from "./types.js";  
  
export function getPrimeArray() {  
 return {  
 type: ***GET\_PRIME\_ARRAY*** }  
}  
  
export function getHashArray() {  
 return {  
 type: ***GET\_HASH\_ARRAY***,  
 }  
}  
  
export function binaryTreeSearch() {  
 return {  
 type: ***BINARY\_TREE\_SEARCH*** }  
}  
  
export function binarySearch() {  
 return {  
 type: ***BINARY\_SEARCH*** }  
}  
  
export function fibonacciSearch() {  
 return {  
 type: ***FIBONACCI\_SEARCH*** }  
}  
  
export function interpolationSearch() {  
 return {  
 type: ***INTERPOLATION\_SEARCH***,  
 }  
}  
  
export function getPrimarySought() {  
 return {  
 type: ***GET\_PRIME\_SOUGHT***,  
 }  
}  
  
export function getHashSearch() {  
 return {  
 type: ***HASH\_SEARCH***,  
 }  
}  
  
export function getHashSearchNode() {  
 return {  
 type: ***HASH\_SEARCH\_NODE***,  
 }  
}  
  
export function getHashSearchRandom() {  
 return {  
 type: ***HASH\_SEARCH\_RANDOM***,  
 }  
}  
  
export function queenGetAnswer() {  
 return {  
 type: ***QUEEN\_GET\_ANSWER***,  
 }  
}export function hashDelete() {  
 return {  
 type: ***HASH\_DELETE***,  
 }  
}export function nudeHashDelete() {  
 return {  
 type: ***HASH\_DELETE\_NODE***,  
 }  
}  
export function queenSearchAnswer() {  
 return {  
 type: QUEEN\_SEARCH\_ANSWER,  
 }  
}export function queenSetBoardSize(boardSize = 8) {  
 return {  
 type: QUEEN\_SET\_BOARD\_SIZE,  
 data: {  
 boardSize  
 }  
 }  
}export function queenSetQueensQuantity(queensQuantity = 8) {  
 return {  
 type: QUEEN\_SET\_QUEENS\_QUANTITY,  
 data:{  
 queensQuantity  
 }  
 }  
}

rootReducer

import {combineReducers} from 'redux'  
import {  
 BINARY\_SEARCH, BINARY\_TREE\_SEARCH,  
 FIBONACCI\_SEARCH, INTERPOLATION\_SEARCH  
} from './types.js'  
import {BinaryTree} from "../modules/binaryTree.js";  
import {binarySearch} from "../modules/binarySearch.js";  
import fibSearch from "../modules/fibonachiSearch.js";  
import {interpolationSearch} from "../modules/interpolationSearch.js";  
import {  
 HashTable,  
 HashTableNode,  
 HashTableRandom  
} from "../modules/hash/hashTables.js";  
import {Queens} from "../modules/queens/queensQuantity.js";  
  
let n = 10, //search  
 minLimit = 0,  
 maxLimit = 10,  
 sought = 5  
  
let array = [];  
let array2= []  
for (let j = 0; j < n; j++) {  
 array[j] = {  
 value: Math.floor(Math.random() \* (maxLimit - minLimit) + minLimit),  
 index: j,  
 }  
 array2[j]=array[j].value  
}  
//console.log(array)  
console.log(array2)  
let stringSought = 'apple2' //hash search  
let stringArray = ["banana", "cat", "tac", "appel2", "dog", "potatoes", "cucumber", "apple2"];  
let nodeStringArray = ["apple", "anny", "tac", "cat", "apple2", "banana", "cat", "dog", "potatoes", "cucumber",];  
let deleteItem="apple"  
let boardSize = 8,  
 queensQuantity = 8  
let initialStatePrime = {  
 primeArray: array,  
 primeSought: sought,  
  
}  
let primeHashTaBle = new HashTable  
stringArray.map(item => {  
 primeHashTaBle.add(item)  
})  
let nodeHashTaBle = new HashTableNode  
nodeStringArray.map(item => {  
 nodeHashTaBle.add(item)  
})  
let initialStateHash = {  
 stringSought: stringSought,  
 deleteItem: deleteItem,  
 primeHashTaBle: primeHashTaBle,  
 nodeHashTaBle: nodeHashTaBle,  
}  
let initialStateQueens = {  
 boardSize: boardSize,  
 queensQuantity: queensQuantity,  
 answer: []  
}  
  
function primeArrayReducer(state = initialStatePrime, action) {  
 switch (action.type) {  
 case "BINARY\_TREE\_SEARCH": {  
 const BST = new BinaryTree()  
 state.primeArray.map((item) => {  
 BST.insert(item)  
 })  
 console.log("BINARY\_TREE\_SEARCH:", (BST.search(BST.root, state.primeSought)))  
 return state  
 }  
 case "BINARY\_SEARCH": {  
 console.log("BINARY\_SEARCH:", binarySearch(state.primeArray.sort(function compareNumbers(a, b) {  
 return a.value - b.value  
 }), state.primeSought))  
 return state  
 }  
 case "FIBONACCI\_SEARCH": {  
 console.log("FIBONACCI\_SEARCH:", fibSearch(state.primeArray.sort(function compareNumbers(a, b) {  
 return a.value - b.value  
 }), state.primeSought))  
 return state  
 }  
 case "INTERPOLATION\_SEARCH": {  
 console.log("INTERPOLATION\_SEARCH:", interpolationSearch(state.primeArray.sort(function compareNumbers(a, b) {  
 return a.value - b.value  
 }), state.primeSought, 0, state.primeArray.length - 1))  
 return state  
 }  
 case "GET\_PRIME\_ARRAY": {  
 //console.log(state.primeArray)  
 return state  
 }  
 case "GET\_PRIME\_SOUGHT": {  
 console.log("GET\_PRIME\_SOUGHT:", state.primeSought)  
 return state  
 }  
 default: {  
 return state  
 }  
 }  
}  
  
function HashReducer(state = initialStateHash, action) {  
 switch (action.type) {  
 case "HASH\_SEARCH": {  
 console.log("HASH\_SEARCH: ", primeHashTaBle.get(state.stringSought))  
 return state  
 }  
 case "HASH\_DELETE": {  
 console.log("HASH\_SEARCH\_DELETE: ", primeHashTaBle.delete(state.deleteItem))  
 return state  
 }  
 case "HASH\_SEARCH\_NODE": {  
 console.log("HASH\_SEARCH\_NODE: ", nodeHashTaBle.get(state.stringSought))  
 return state  
 }case "HASH\_DELETE\_NODE": {  
 console.log("HASH\_DELETE\_NODE: ", nodeHashTaBle.delete(state.deleteItem))  
 return state  
 }  
 default: {  
 return state  
 }  
 }  
}  
  
function queensOnTheBoard(state = initialStateQueens, action) {  
 switch (action.type) {  
 case "QUEEN/GET\_ANSWER": {  
 console.log("QUEEN/GET\_ANSWER: ", state.answer)  
 return state  
 }  
 case "QUEEN/SEARCH\_ANSWER": {  
 return {...state, answer: Queens(state.boardSize,state.queensQuantity)}  
 }  
 case "QUEEN/SET\_BOARD\_SIZE":{  
 return {...state, boardSize: action.data.boardSize}  
 }  
 case "QUEEN/SET\_QUEENS\_QUANTITY":{  
 return {...state, queensQuantity: action.data.queensQuantity}  
 }  
 default: {  
 return state  
 }  
 }  
}  
  
export const rootReducer = combineReducers({  
 primeArray: primeArrayReducer,  
 hashArray: HashReducer,  
 chess: queensOnTheBoard  
})

types

export const ***BINARY\_TREE\_SEARCH*** = 'BINARY\_TREE\_SEARCH'  
export const ***BINARY\_SEARCH*** = 'BINARY\_SEARCH'  
export const ***FIBONACCI\_SEARCH*** = 'FIBONACCI\_SEARCH'  
export const INTERPOLATION\_SEARCH = 'INTERPOLATION\_SEARCH'  
export const GET\_PRIME\_ARRAY = 'GET\_PRIME\_ARRAY'  
export const GET\_PRIME\_SOUGHT = 'GET\_PRIME\_SOUGHT'  
export const HASH\_SEARCH = 'HASH\_SEARCH'  
export const GET\_HASH\_ARRAY = 'GET\_HASH\_ARRAY'  
export const HASH\_SEARCH\_NODE = 'HASH\_SEARCH\_NODE'  
export const HASH\_SEARCH\_RANDOM = 'HASH\_SEARCH\_RANDOM'  
export const HASH\_DELETE = 'HASH\_DELETE'  
export const HASH\_DELETE\_NODE = 'HASH\_DELETE\_NODE'  
export const QUEEN\_GET\_ANSWER = 'QUEEN/GET\_ANSWER'  
export const QUEEN\_SEARCH\_ANSWER = 'QUEEN/SEARCH\_ANSWER'  
export const QUEEN\_SET\_BOARD\_SIZE = 'QUEEN/SET\_BOARD\_SIZE'  
export const QUEEN\_SET\_QUEENS\_QUANTITY = 'QUEEN/SET\_QUEENS\_QUANTITY'

**Вывод программы**

[  
 5, 2, 7, 0, 4,  
 2, 5, 8, 7, 0  
]  
GET\_PRIME\_SOUGHT: 5  
BINARY\_TREE\_SEARCH: 0  
BINARY\_SEARCH: 0  
FIBONACCI\_SEARCH: 6  
INTERPOLATION\_SEARCH: 6  
HASH\_SEARCH\_DELETE: null  
HASH\_DELETE\_NODE: null  
HASH\_SEARCH: [ 4, 'apple2' ]  
HASH\_SEARCH\_NODE: [ 10, 'apple2' ]  
QUEEN/GET\_ANSWER: [  
 '1 a1,e2,h3,f4,c5,g6,b7,d8',  
 '2 a1,f2,h3,c4,g5,d6,b7,e8',  
 '3 a1,g2,d3,f4,h5,b6,e7,c8',  
 '4 a1,g2,e3,h4,b5,d6,f7,c8',  
 '5 b1,d2,f3,h4,c5,a6,g7,e8',  
 '6 b1,e2,g3,a4,c5,h6,f7,d8',  
 '7 b1,e2,g3,d4,a5,h6,f7,c8',  
 '8 b1,f2,a3,g4,d5,h6,c7,e8',  
 '9 b1,f2,h3,c4,a5,d6,g7,e8',  
 '10 b1,g2,c3,f4,h5,e6,a7,d8',  
 '11 b1,g2,e3,h4,a5,d6,f7,c8',  
 '12 b1,h2,f3,a4,c5,e6,g7,d8',  
 '13 c1,a2,g3,e4,h5,b6,d7,f8',  
 '14 c1,e2,b3,h4,a5,g6,d7,f8',  
 '15 c1,e2,b3,h4,f5,d6,g7,a8',  
 '16 c1,e2,g3,a4,d5,b6,h7,f8',  
 '17 c1,e2,h3,d4,a5,g6,b7,f8',  
 '18 c1,f2,b3,e4,h5,a6,g7,d8',  
 '19 c1,f2,b3,g4,a5,d6,h7,e8',  
 '20 c1,f2,b3,g4,e5,a6,h7,d8',  
 '21 c1,f2,d3,a4,h5,e6,g7,b8',  
 '22 c1,f2,d3,b4,h5,e6,g7,a8',  
 '23 c1,f2,h3,a4,d5,g6,e7,b8',  
 '24 c1,f2,h3,a4,e5,g6,b7,d8',  
 '25 c1,f2,h3,b4,d5,a6,g7,e8',  
 '26 c1,g2,b3,h4,e5,a6,d7,f8',  
 '27 c1,g2,b3,h4,f5,d6,a7,e8',  
 '28 c1,h2,d3,g4,a5,f6,b7,e8',  
 '29 d1,a2,e3,h4,b5,g6,c7,f8',  
 '30 d1,a2,e3,h4,f5,c6,g7,b8',  
 '31 d1,b2,e3,h4,f5,a6,c7,g8',  
 '32 d1,b2,g3,c4,f5,h6,a7,e8',  
 '33 d1,b2,g3,c4,f5,h6,e7,a8',  
 '34 d1,b2,g3,e4,a5,h6,f7,c8',  
 '35 d1,b2,h3,e4,g5,a6,c7,f8',  
 '36 d1,b2,h3,f4,a5,c6,e7,g8',  
 '37 d1,f2,a3,e4,b5,h6,c7,g8',  
 '38 d1,f2,h3,b4,g5,a6,c7,e8',  
 '39 d1,f2,h3,c4,a5,g6,e7,b8',  
 '40 d1,g2,a3,h4,e5,b6,f7,c8',  
 '41 d1,g2,c3,h4,b5,e6,a7,f8',  
 '42 d1,g2,e3,b4,f5,a6,c7,h8',  
 '43 d1,g2,e3,c4,a5,f6,h7,b8',  
 '44 d1,h2,a3,c4,f5,b6,g7,e8',  
 '45 d1,h2,a3,e4,g5,b6,f7,c8',  
 '46 d1,h2,e3,c4,a5,g6,b7,f8',  
 '47 e1,a2,d3,f4,h5,b6,g7,c8',  
 '48 e1,a2,h3,d4,b5,g6,c7,f8',  
 '49 e1,a2,h3,f4,c5,g6,b7,d8',  
 '50 e1,b2,d3,f4,h5,c6,a7,g8',  
 '51 e1,b2,d3,g4,c5,h6,f7,a8',  
 '52 e1,b2,f3,a4,g5,d6,h7,c8',  
 '53 e1,b2,h3,a4,d5,g6,c7,f8',  
 '54 e1,c2,a3,f4,h5,b6,d7,g8',  
 '55 e1,c2,a3,g4,b5,h6,f7,d8',  
 '56 e1,c2,h3,d4,g5,a6,f7,b8',  
 '57 e1,g2,a3,c4,h5,f6,d7,b8',  
 '58 e1,g2,a3,d4,b5,h6,f7,c8',  
 '59 e1,g2,b3,d4,h5,a6,c7,f8',  
 '60 e1,g2,b3,f4,c5,a6,d7,h8',  
 '61 e1,g2,b3,f4,c5,a6,h7,d8',  
 '62 e1,g2,d3,a4,c5,h6,f7,b8',  
 '63 e1,h2,d3,a4,c5,f6,b7,g8',  
 '64 e1,h2,d3,a4,g5,b6,f7,c8',  
 '65 f1,a2,e3,b4,h5,c6,g7,d8',  
 '66 f1,b2,g3,a4,c5,e6,h7,d8',  
 '67 f1,b2,g3,a4,d5,h6,e7,c8',  
 '68 f1,c2,a3,g4,e5,h6,b7,d8',  
 '69 f1,c2,a3,h4,d5,b6,g7,e8',  
 '70 f1,c2,a3,h4,e5,b6,d7,g8',  
 '71 f1,c2,e3,g4,a5,d6,b7,h8',  
 '72 f1,c2,e3,h4,a5,d6,b7,g8',  
 '73 f1,c2,g3,b4,d5,h6,a7,e8',  
 '74 f1,c2,g3,b4,h5,e6,a7,d8',  
 '75 f1,c2,g3,d4,a5,h6,b7,e8',  
 '76 f1,d2,a3,e4,h5,b6,g7,c8',  
 '77 f1,d2,b3,h4,e5,g6,a7,c8',  
 '78 f1,d2,g3,a4,c5,e6,b7,h8',  
 '79 f1,d2,g3,a4,h5,b6,e7,c8',  
 '80 f1,h2,b3,d4,a5,g6,e7,c8',  
 '81 g1,a2,c3,h4,f5,d6,b7,e8',  
 '82 g1,b2,d3,a4,h5,e6,c7,f8',  
 '83 g1,b2,f3,c4,a5,d6,h7,e8',  
 '84 g1,c2,a3,f4,h5,e6,b7,d8',  
 '85 g1,c2,h3,b4,e5,a6,f7,d8',  
 '86 g1,d2,b3,e4,h5,a6,c7,f8',  
 '87 g1,d2,b3,h4,f5,a6,c7,e8',  
 '88 g1,e2,c3,a4,f5,h6,b7,d8',  
 '89 h1,b2,d3,a4,g5,e6,c7,f8',  
 '90 h1,b2,e3,c4,a5,g6,d7,f8',  
 '91 h1,c2,a3,f4,b5,e6,g7,d8',  
 '92 h1,d2,a3,c4,f5,b6,g7,e8'  
]

# Вывод

В ходе выполнения лабораторной работы, я реализовал разные методы поиска, рехэширования и решила задачу на шахматную доску.