Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное

бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и

информационных технологий

Лабораторная работа №2

на тему: «Методы поиска»

Выполнила:

Студентка группы БФИ1902

Струкова А.В.

Вариант 18

Проверил:

Москва, 2021 г.

Оглавление

[1. Цель работы 3](#_Toc72408691)

[2. Задание на лабораторную работу 3](#_Toc72408692)

[2.1. Задание №1: 3](#_Toc72408693)

[2.2. Задание №2: 3](#_Toc72408694)

[2.3. Задание № 3: 3](#_Toc72408695)

[3. Листинг программы 3](#_Toc72408696)

[4. Результат работы программы 9](#_Toc72408697)

[Список использованных источников 11](#_Toc72408698)

# **1. Цель работы**

Цель работы: рассмотреть и изучить основные методы поиска.

# **2. Задание на лабораторную работу**

Реализовать методы поиска в соответствии с заданием. Организовать генерацию начального набора случайных данных. Для всех вариантов добавить реализацию добавления, поиска и удаления элементов. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования.

## 2.1. Задание №1:

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

## 2.2. Задание №2:

|  |  |  |
| --- | --- | --- |
| Простое рехэширование | Рехэширование с помощью  псевдослучайных чисел | Метод цепочек |

## 2.3. 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.

# **3. Листинг программы**

*# -\*- coding: utf8 -\*-*

*#Задание 1*

*#Бинарный поиск*

def **BinarySearch**(lys, val):

    first = 0

    last = **len**(lys)-1

    index = -1

    while (first <= last) and (index == -1):

        mid = (first+last)//2

        if lys[mid] == val:

            index = mid

        else:

            if val<lys[mid]:

                last = mid -1

            else:

                first = mid +1

    return index

**print**(BinarySearch([10,20,30,40,50], 20))

*#Бинарное дерево*

from random import randint

*# Создание списка и вывод на экран*

arr = []

for i in **range**(15):

    arr.append(randint(1, 50)) *# append добавляет элемент в список*

**print**(arr)

*# искомое число*

value = int(**input**("Enter the number you are looking for: "))

D, L, R, I = 'data', 'left', 'right', 'index'

p = 0

def **BinaryTree**(tree, data, i):

    if tree is None:

        tree = {D: data, L: None, R: None, I: i}

    elif data <= tree[D]:

        tree[L] = BinaryTree(tree[L], data, i)

    else:

        tree[R] = BinaryTree(tree[R], data, i)

    return tree

tree = None

for i, el in **enumerate**(arr):

    tree = BinaryTree(tree, el, i)

def **BinaryTreeSearch**(tree):

    if value < tree[D] and tree[L] != None:

        BinaryTreeSearch(tree[L])

    elif value > tree[D] and tree[R] != None:

        BinaryTreeSearch(tree[R])

    elif value == tree[D]:

**print**(tree[I])

    else:

**print**("not found")

**print**("Index =", BinaryTreeSearch(tree))

*#Фиббоначиев*

def **FibonacciSearch**(lys, val):

    fibM\_minus\_2 = 0

    fibM\_minus\_1 = 1

    fibM = fibM\_minus\_1 + fibM\_minus\_2

    while (fibM < **len**(lys)):

        fibM\_minus\_2 = fibM\_minus\_1

        fibM\_minus\_1 = fibM

        fibM = fibM\_minus\_1 + fibM\_minus\_2

    index = -1;

    while (fibM > 1):

        i = **min**(index + fibM\_minus\_2, (**len**(lys)-1))

        if (lys[i] < val):

            fibM = fibM\_minus\_1

            fibM\_minus\_1 = fibM\_minus\_2

            fibM\_minus\_2 = fibM - fibM\_minus\_1

            index = i

        elif (lys[i] > val):

            fibM = fibM\_minus\_2

            fibM\_minus\_1 = fibM\_minus\_1 - fibM\_minus\_2

            fibM\_minus\_2 = fibM - fibM\_minus\_1

        else :

            return i

    if(fibM\_minus\_1 and index < (**len**(lys)-1) and lys[index+1] == val):

        return index+1;

    return -1

**print**(FibonacciSearch([1,2,3,4,5,6,7,8,9,10,11], 6))

*#Интерполяционный*

'''Интерполяционный поиск — это еще один алгоритм «разделяй и властвуй», аналогичный бинарному поиску. В отличие от бинарного поиска, он не всегда начинает поиск с середины. Интерполяционный поиск вычисляет вероятную позицию искомого элемента по формуле'''

def **InterpolationSearch**(lys, val): *#lys — входной массив, val — искомый элемент*

    low = 0 *#начальный индекс массива*

    high = (**len**(lys) - 1) *#последний индекс массива*

    while low <= high and val >= lys[low] and val <= lys[high]:

        index = low + int(((float(high - low) / ( lys[high] - lys[low])) \* ( val - lys[low])))

        if lys[index] == val:

            return index

        if lys[index] < val:

            low = index + 1;

        else:

            high = index - 1;

    return -1

**print**(InterpolationSearch([1,2,3,4,5,6,7,8], 6))

*#Задание 2*

*#Простое рехеширование*

class **prost\_rehash**:

*# Конструктор, создание словаря*

    def **\_\_init\_\_**(self):

        self.rhash=[None]\*256

    def **keys**(self, element): *# Формирование ключа*

        key=0

        for i in **range**(**len**(element)):

            key=key+**ord**(element[i])

        return int(key%256)

    def **add**(self, element): *# Добавление элемента*

        key=self.keys(element)

        while self.rhash[key] is not None:

            key=key+1

        self.rhash[key]=element

    def **search**(self, element): *# Поиск элемента*

        key=self.keys(element)

        while self.rhash[key] is not None:

            if self.rhash[key]==element:

                return key

            else:

                key=key+1

        return None

    def **deleted**(self, element): *# Удаление элемента*

        key=self.search(element)

        while key is not None and self.rhash[key] is not None:

            if self.rhash[key]==element:

                del self.rhash[key]

                key=int(key+1)

                while key<**len**(self.rhash) and self.rhash[key] is not None:

                    el=self.rhash.pop(key)

                    self.add(el)

                    key=key+1

                return 1

            else:

                key=key+1

        return -1

    def **pr**(self): *#вывод на экран*

        for  key, i in **enumerate**(self.rhash):

            if self.rhash[key] is not None:

**print**(key, " ", i)

a=prost\_rehash()

a.add("qwe")

a.add("qwq")

a.add("qws")

a.add("qwm")

a.add("qwo")

a.pr()

s=a.deleted("qwq")

**print**(s)

a.pr()

*#метод цепочек*

class **chain\_rehash**:

*# Конструктор, создание словаря*

    def **\_\_init\_\_**(self):

        self.rhash=[[]\*0 for i in **range**(10)]

    def **add**(self, element):  *# Добавление элемента*

        key=int(0)

        for i in **range**(**len**(element)):

            key=key+**ord**(element[i])

        key=key%10

        self.rhash[key].append(element)

    def **search**(self, element): *#Поиск*

            key=int(0)

            for i in **range**(**len**(element)):

                key=key+**ord**(element[i])

            key=key%10

            if self.rhash[key] is not None:

                for i in **range**(**len**(self.rhash[key])):

                    if self.rhash[key][i]==element:

                        return key, i

            return None, None

    def **deleted**(self, element):

            key, i=self.search(element)

            if key is not None:

                del(self.rhash[key][i])

**print**("Элемент успешно удален")

            else:

**print**("Элемент не найден")

                return -1

    def **pr**(self): *#вывод*

        for key in **range**(**len**(self.rhash)):

            for i in **range**(**len**(self.rhash[key])):

                if self.rhash[key][i] is not None:

**print**("(ключ)", key, "- (Элемент)", self.rhash[key][i])

a=chain\_rehash()

a.add("qwe")

a.add("qwe")

a.add("qwe")

a.pr()

a.deleted("qwe")

a.pr()

<!DOCTYPE *HTML*>

<html>

    <style>

        .board { border:2px solid black; margin:4px; display:inline-block; }

        .cell  { border:1px solid black; width:20px; height:20px; font-size: 20px; display:inline-block; }

        .white { background-color:white; color:black;}

        .black { background-color:#555;  color:white;}

     </style>

<pre>

<script>

    var size = 8;  *// размер доски*

var queens = **Array**(size);  *// положение ферзей*

for(var c=0; c < size; c++)

   queens[c] = 0;  *// всех ставим на первую строку (это не решение*

function **Show**() *// функция печати доски*

{

   document.**write**('<div class="board">');         *// открываем тег div*

   for(var r=0; r < size; r++){                   *// по строкам*

      for(var c=0; c < size; c++){                *// по колонкам*

         var ch  = queens[c]===r ? "&#9819;" : " "; *//если значение массива queens[c] совпадает с номером строки r, выводится ферзь, иначе - пробел*

         var knd = 'class="cell ' + ( (r+c) % 2 ? 'black"': 'white"'); *//выясняем чётность суммы r+c. При чётной сумме выбирается класс стиля class="cell white", а для нечётной: class="cell black"*

         document.**write**('<div ', knd, '>', ch, '</div>');

      }

      document.**write**('<br>');                     *// переход на новую строку*

   }

   document.**write**('</div>');                      *// закрываем тег div*

}

</script>

</pre>

<script>

    var nSolutions = 0;  *// число найденных решений*

 function **Solve**(n)

 {

    if(n===undefined) n = 0;  *//если не нашли еще ферзей*

    if(n >= size){  *// всех расставили*

       if(nSolutions++ < 5)   *// подсчитываем число решений*

**Show**();  *// и выводим первые 5*

       return;  *// перебор окончен*

    }

    for(var r=0, c; r < size; r++){  *// бежим по строчкам сверху-вниз*

       for(c=0; c < n; c++)  *// перебираем уже поставленных ферзей*

          if(   queens[c] === r  *// если они стоят на этой строке*

             || **Math**.**abs**(queens[c]-r) === n-c ) *// или находятся с новым на одной диагонали*

             break;     *// вариант не подходит - выходим из цикла*

       if(c === n){   *//если ни кто не бьет ферзя*

          queens[n] = r; *// ставим его туда*

**Solve**(n+1);  *// и подбираем следующего*

       }

    }

 }

**Solve**();

</script>

</html>

# **4. Результат работы программы**

Результат работы программы представлен на Рисунках 1-2.

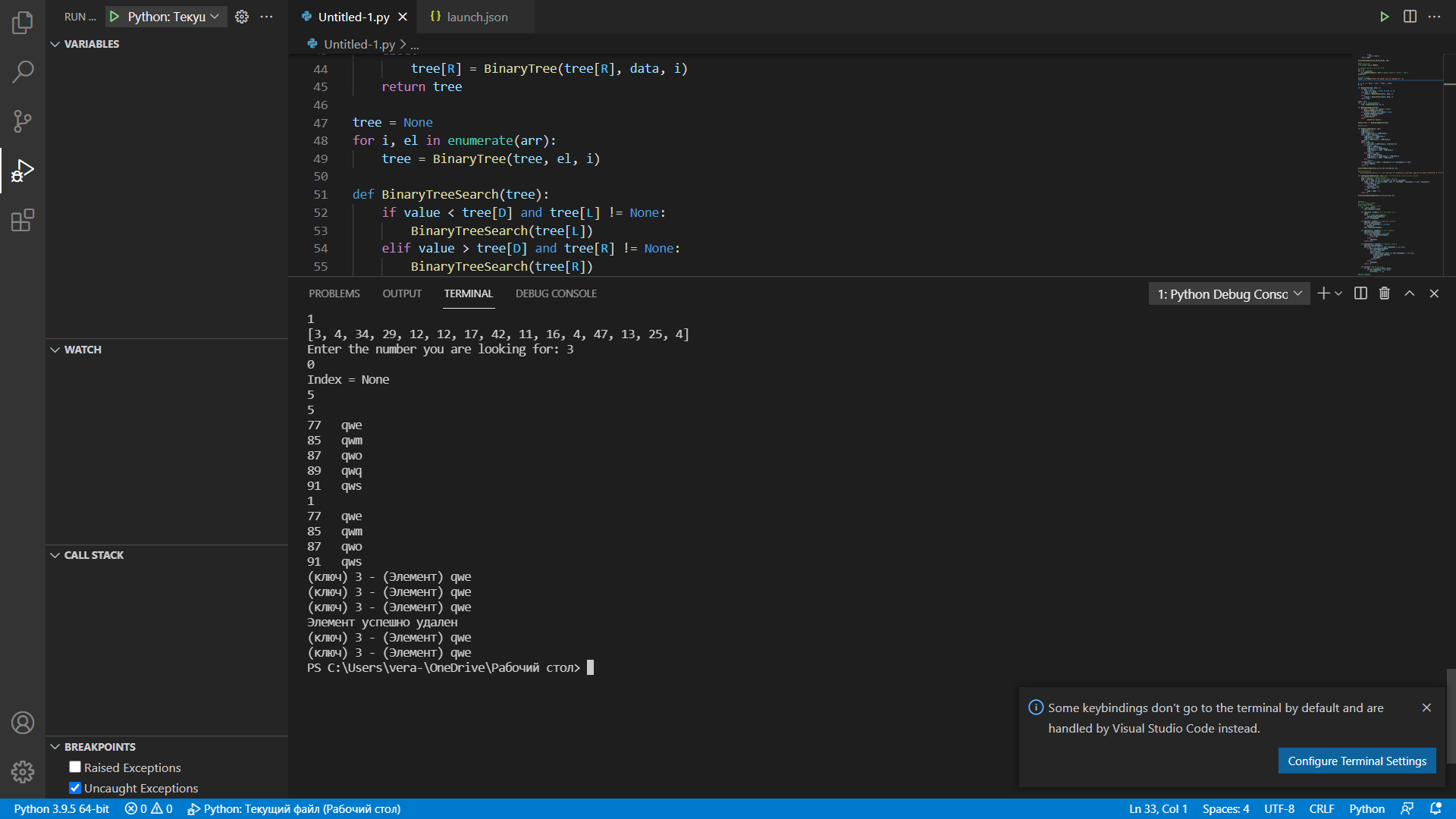


Рисунок 1 - Задание 1 и 2

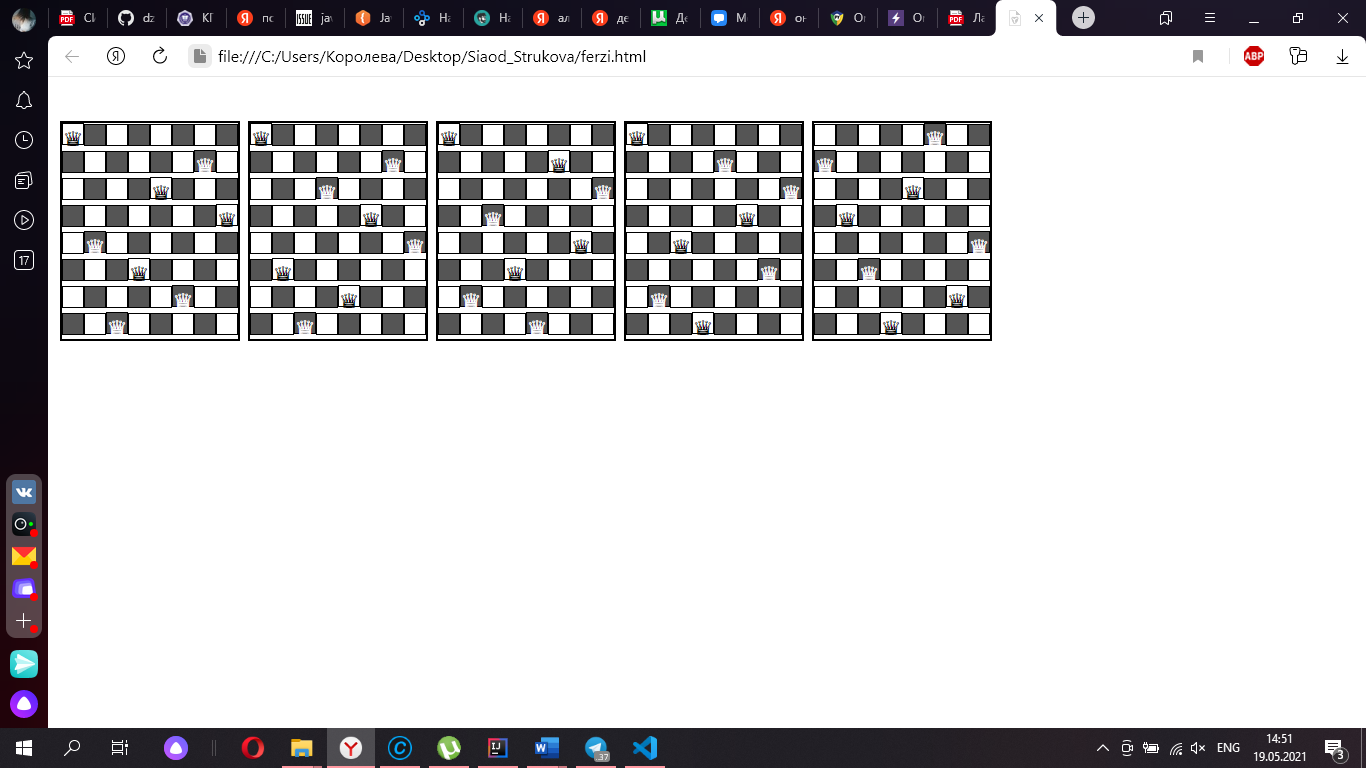


Рисунок 2 - Задание 3

**Вывод:** мы рассмотрели и изучили основные методы поиска, такие как бинарный, Фибоначчиев и тд.

# **Список использованных источников**

1) ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления

2) ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления