МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра

«Математическая кибернетика и информационные технологии»

Лабораторная работа №2

**«Методы поиска»**

по дисциплине

**«**Структуры и алгоритмы обработки данных**»**

Выполнил:

студент гр. БСТ2004

Шадюк М.Р.

Вариант №22

Москва, 2022 г.

Оглавление

[Цель лабораторной работы 3](#_Toc102095644)

[Задание 1 3](#_Toc102095645)

[Бинарный поиск 3](#_Toc102095646)

[КОД 3](#_Toc102095647)

[Выполнение 4](#_Toc102095648)

[Бинарное дерево 4](#_Toc102095649)

[КОД 4](#_Toc102095650)

[Выполнение 6](#_Toc102095651)

[Фибоначчиев поиск 7](#_Toc102095652)

[Код 7](#_Toc102095653)

[Выполнение 9](#_Toc102095654)

[Интерполяционный поиск 10](#_Toc102095655)

[Код 10](#_Toc102095656)

[Выполнение 12](#_Toc102095657)

[Задание 2 13](#_Toc102095658)

[Простое рехеширование 13](#_Toc102095659)

[Рехеширование с помощью псевдослучайных чисел 13](#_Toc102095660)

[Метод цепочек 15](#_Toc102095661)

[Задание 3 16](#_Toc102095662)

[КОД 16](#_Toc102095663)

[Выполнение 18](#_Toc102095664)

# Цель лабораторной работы

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.

# Задание 1

Бинарный поиск, бинарное дерево, Фибоначчиев, Интерполяционный

## Бинарный поиск

### КОД

def binary\_search(arr, key):

  # Определяем верхнюю и нижнюю границы поиска

    low = 0

    high = len(arr)-1

  # Пока между ними не останется один элемент

    while low <= high:

    # Находим средний элемент

        middle = (low + high) // 2

    # Сравниваем с искомым

        if arr[middle] == key:

            return middle

        else:

            # Если не совпал изменяем либо нижнюю либо верхнюю границу

            if arr[middle]>key:

                high = middle-1

            else:

                low = middle+1

  # Значение не найдено

    return None

def addItem\_binSearch(arr, key):

    index = binary\_search(arr,key)

    if index != None and arr[index] == key:

        print("Такой элемент уже существует")

    else:

        i = 0

        while i<len(arr):

            # меньше первого

            if key<arr[0]:

                arr.insert(0,key)

            # больше последнего

            if key>arr[len(arr)-1]:

                arr.insert(len(arr),key)

            # где-то по середине

            if key>arr[i] and key<arr[i+1]:

                arr.insert(i+1,key)

            i+=1

    return arr

def deleteItem\_binSearch(arr,key):

    index = binary\_search(arr,key)

    if index == None:

        print("Такой элемент не существует")

    else:

        arr.pop(index)

    return arr

### Выполнение

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Результат работы программы

## Бинарное дерево

### КОД

class Node:

    # конструктор

    def \_\_init\_\_(self, key):

        self.key = key

        self.left = self.right = None

    # создание вершин

    def insert(self, key):

        # Если корневой элемент существует

        if self.key:

            # Если ключ меньше значения в узле добавляем его в левую ветку

            if key < self.key:

                # если ветка пустая то создаем в ней новый узел

                if self.left is None:

                    self.left = Node(key)

                else:

                    # если не пустая то вызываем для нового узла эту функцию

                    self.left.insert(key)

            elif key >= self.key:

                if self.right is None:

                    self.right = Node(key)

                else:

                    self.right.insert(key)

        #узел принимает знач ключа

        else:

            self.key = key

    def show(self):

        if self.left:

            self.left.show()

        print( self.key),

        if self.right:

            self.right.show()

    def search(self,key):

        if self.key:

            # если ключ совпадает со значением узла то элемент найден

            if self.key==key:

                print("Элемент найден")

                return

            if key< self.key:

                # если ветка пустая, то элемента в ней нет

                if self.left is None:

                    print("Элемент не найден")

                    return

                # Если не пустая то вызываем эту функцию для нового узла

                self.left.search(key)

            else:

                if self.right is None:

                    print("Элемент не найден")

                    return

                self.right.search(key)

        else:

            print("Дерево пустое")

    def delete(self,key):

        if self.key:

            # Если ключ меньше значения узла

            if key < self.key:

                # Вызываем функцию для нового узла

                self.left = self.left.delete(key)

            elif key > self.key:

                self.right = self.right.delete(key)

                # если у узла есть обе ветки, помещаем в узел наименьший элемент из правой ветки и удаляем его из ветки

            elif self.left != None and self.right != None:

                self.key = min(self.right).key

                self.right = self.right.delete(self.key)

            else:

                # если есть только левая ветка, элемент удаляетя при этом узел левой ветки становится на его место

                if self.left != None:

                    self = self.left

                elif self.right != None:

                    self = self.right

                else:

                    self = None

        return self

### Выполнение

Tree = Node(5)

Tree.insert(8)

Tree.insert(6)

Tree.insert(2)

Tree.insert(-1)

Tree.insert(99)

Tree.show()

print()

Tree.insert(2)

Tree.show()

print()

Tree.delete(-1)

Tree.show()

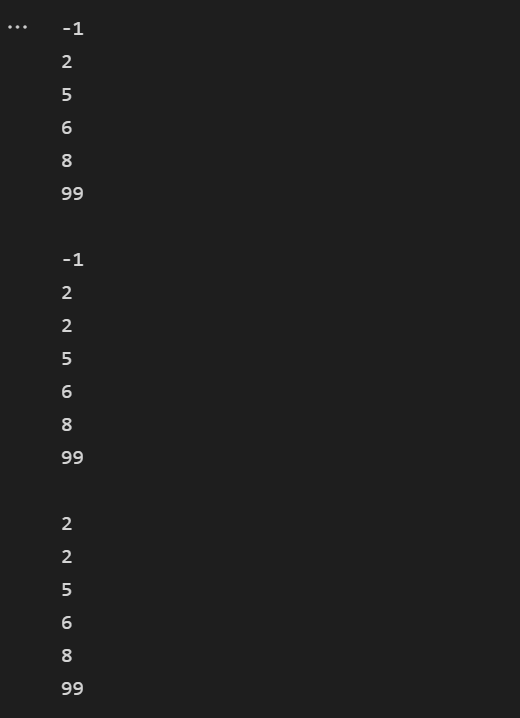


Рисунок 2 – Результат выполнения программы

## Фибоначчиев поиск

### Код

def fibonacciSearch(arr,key):

    f1=0

    f2=1

    summa = 0

    # пока числа фибоначчи не выходят за пределы массива и пока ключ больше элемента в массиве

    while summa<len(arr) and key>=arr[summa]:

        # если они равны - элемент найден

        if arr[summa]==key:

            print("Элемент найден")

            return 1

        # находим следующее число фибоначчи

        summa = f1+f2

        f1=f2

        f2=summa

    # если массив стал одним эл-м, то ничего не найдено

    if summa==f1:

        print("Элемент не найден")

        return None

    # если число фибоначчи больше массива, то берем самый крайний

    if summa>len(arr):

        summa = len(arr)

    i=0

    # нижняя граница не считается

    j=f1+1

    newArr = [0]\*(summa-f1-1)

    # новый массив из промежутков фибаначчи

    while j>f1 and j< summa:

        newArr[i] = arr[j]

        j+=1

        i+=1

    # в новом промежутке все по новой

    index = fibonacciSearch(newArr,key)

    if index:

        return index

# Добавление значений в массив

def addItem\_fibonacciSearch(arr, key):

    index = fibonacciSearch(arr,key)

    if index != None and arr[index] == key:

        print("Такой элемент уже существует")

    else:

        i = 0

        while i<len(arr):

            # если эл-т меньше первого, то ставим его первым

            if key<arr[0]:

                arr.insert(0,key)

            # если эл-т больше последнего, ставим на последнее место

            if key>arr[len(arr)-1]:

                arr.insert(len(arr),key)

            # что-то между

            if key>arr[i] and key<arr[i+1]:

                arr.insert(i+1,key)

            i+=1

    return arr

# тут удаляем, что хочется

def deleteItem\_fibonacciSearch(arr,key):

    index = fibonacciSearch(arr,key)

    if index:

        for m in range(len(arr)):

            if key == arr[m]:

                arr.pop(m)

                return arr

# на случай, если нет того, что хотим удалить

    else:

        print("Такого элемента не существует")

    return arr

### Выполнение

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Результат выполнения 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Результат выполнения 2

## Интерполяционный поиск

### Код

def interpolation\_search(arr, key):

  # Определяем границы поиска

    low = 0

    high = len(arr)-1

  # Пока между ними не останется один элемент

    while low <= high and arr[low] <= key and key <= arr[high]:

    # Находим средний элемент

    #сама формула

        middle = low+((key-arr[low])\*(high-low)//(arr[high]-arr[low]))

        if arr[middle]==key:

            return middle

        elif arr[middle]<key:

            low = middle+1

        else:

             high = middle-1

    return None

# для добавления значения

def addItem\_interpolationSearch(arr, key):

    index = interpolation\_search(arr,key)

    if index != None and arr[index] == key:

        print("Такой элемент уже существует")

    else:

        i = 0

        while i<len(arr):

            # если данный эл-т меньше первого, ставим его первым

            if key<arr[0]:

                arr.insert(0,key)

            # если данный эл-т больше последнего, ставим его последним

            if key>arr[len(arr)-1]:

                arr.insert(len(arr),key)

            # что-то между

            if key>arr[i] and key<arr[i+1]:

                arr.insert(i+1,key)

            i+=1

    return arr

# удаление

def deleteItem\_interpolationSearch(arr,key):

    index = interpolation\_search(arr,key)

    if index == None:

        print("Такой элемент не существует")

    else:

        arr.pop(index)

    return arr

### Выполнение

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

Рисунок 4 – Результат 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Результат 2

# Задание 2

## Простое рехеширование

#Простое рехэширование

class HashMap:

    def \_\_init\_\_(self):

        self.size = 0

        self.data = []

        self.\_resize()

    #сама ф-я, что бы не словить коллизию

    def \_hash(self, key, i):

        return (hash(key) + i) % len(self.data)

    def \_find(self, key):

        i = 0

        index = self.\_hash(key, i)

        while self.data[index] is not None and self.data[index][0] != key:

            i += 1

            index = self.\_hash(key, i)

        return index

    def \_resize(self):

        temp = self.data

        self.data = [None] \* (2\*len(self.data) + 1)

        for item in temp:

            if item is not None:

                self.data[self.\_find(item[0])] = item

    def \_\_setitem\_\_(self, key, value):

        if self.size + 1 > len(self.data) // 2:

            self.\_resize()

        index = self.\_find(key)

        if self.data[index] is None:

            self.size += 1

        self.data[index] = (key, value)

    def \_\_getitem\_\_(self, key):

        index = self.\_find(key)

        if self.data[index] is not None:

            return self.data[index][1]

        raise KeyError()

## Рехеширование с помощью псевдослучайных чисел

class RandomHashMap:

    def Fhash2 (a, id, i):

        import random

        i=i+1

        if i<63:

            n=64

            hash = (ord(id[0])+ord(id[len(id)//2])+ord(id[len(id)-1])+random.randint(0, 1500))//n

            if a[hash] == None:

                a[hash]=id

            else:

                k=hash

                while (a[hash]!=None):

                    hash = (ord(id[0])+ord(id[len(id)//2])+ord(id[len(id)-1])+random.randint(0, 1500))//n

        else:

            print ("места нет")

        return a

    def Fhashsearch2(a, id):

        i=0

        while i<63 and a[i]!=id:

            i=i+1

        if a[i]==id:

            return i

        else:

             print ("Идентификатор не найден")

    def FhashDel2(a, id):

        i=0

        while i<63 and a[i]!=id:

            i=i+1

        if a[i]==id:

            a[i]=None

            return a

        else:

             print ("Идентификатор не найден")

a=[None]\*64

i=0

print ("Хотите добавить новый идентификатор в таблицу?")

s=(input())

if s == "да":

    while s =="да":

        id=(input())

        RandomHashMap.Fhash2 (a, id, i)

        print ("Хотите добавить новый идентификатор в таблицу?")

        s=(input())

print (a)

print ("Хотите найти идентификатор в таблице?")

k=(input())

if k == "да":

    while k =="да":

        id=(input())

        hash =RandomHashMap.Fhashsearch2 (a, id)

        print (hash)

        print ("Хотите найти идентификатор в таблице?")

        k=(input())

    if hash!= None:

        print ("Хотите удалить идентификатор из таблицы?")

        k=(input())

        if k == "да":

            RandomHashMap.Fhashdel2 (a, id)

            print (a)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 -- Результат

## Метод цепочек

class ChainMap:

    def Chainmethod (arr, id):

        h=0

        for i in range (0, len(id)):

            h=ord(id[i])+h

        key=h%10

        arr[key].append(id)

        return key

    def ChainmethodSearch (a, id):

        h=0

        for i in range (0, len(id)):

            h=ord(id[i])+h

        key=h%10

        k=0

        for i in range (len(a[key])):

            if a[key][i]==id:

                print("Строка ", key, "Элемент ", i)

                k=1

        if k==0:

            print("Идентификатор не найден")

            return (1)

    def ChainmethodDeletions (a, id):

        h=0

        for i in range (0, len(id)):

            h=ord(id[i])+h

        key=h%10

        for i in range (len(a[key])):

            if a[key][i]==id:

                del(a[key][i])

        return (key)

key=[[]\*1 for i in range (10)]

print ("Хотите добавить новый идентификатор в таблицу?")

s=(input())

if s == "да":

    while s =="да":

        id=(input())

        x=ChainMap.Chainmethod (key, id)

        print ("Хотите добавить новый идентификатор в таблицу?")

        s=(input())

print (key)

print ("Хотите найти идентификатор в таблице?")

k=(input())

if k == "да":

    while k =="да":

        id=(input())

        hash =ChainMap.ChainmethodSearch (key, id)

        print ("Хотите найти идентификатор в таблице?")

        k=(input())

    if hash!= 1:

        print ("Хотите удалить идентификатор из таблицы?")

        k=(input())

        if k == "да":

            hash =ChainMap.ChainmethodDeletions (key, id)

            print (key)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 -- Результат

# 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

#Проверяет поле на атаку другими ферзями

def checkField(i,j,Queens):

    r=i

    c=j

    #В столбце

    for k in range(i):

        if j == Queens[k]:

            return False

    # В ниспадающей диагонали

    while i>=0 and j>=0:

        if Queens[i]==j:

            return False

        i-=1

        j-=1

    # В растущей диагонали

    while r>=0 and c<=14:

        if Queens[r]==c:

            return False

        r-=1

        c+=1

    return True

 # Queens - массив, в котором индекс это строка, значение это столбец, где стоит ферзь

def findQueens(Queens=[0]\*15, i=0):

    # Если на доске уже 8 ферзей, то рекурсия останавливается

    if i ==15:

        arr = [[0 for i in range(15)] for j in range(15)]

        for i in range(15):

            for j in range(15):

                arr[i][Queens[i]] = 1

        print(Queens)

        return arr

    else:

        for j in range(15):

            # Проверяем свободно ли поле

            if checkField(i,j,Queens):

                # Если да, записываем координату ферзя

                Queens[i]=j

                # Снова вызываем функцию со следующей строки

                chessBoard = findQueens(Queens,i+1)

                # если переменная не пустая, выходим в предыдующую

                #  функцию с этой перемееной

                if chessBoard:

                    return chessBoard

findQueens()

### Выполнение

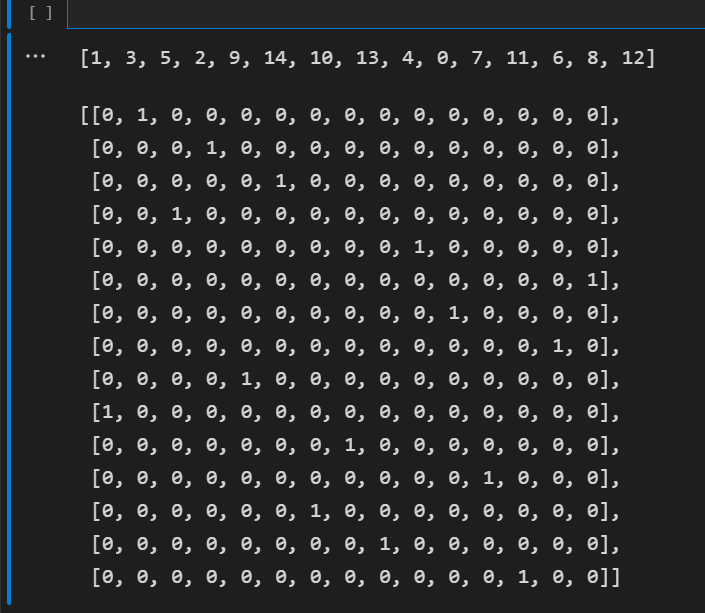


Рисунок 8 -- Результат