Федеральное агентство связи

Ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №2

по дисциплине «Структура и алгоритмы обработки данных»

«Методы поиска»

Выполнил студент

группы БФИ1902

Кочеринский Н.В.

Проверил: МкртчянГ.М.

Москва 2021

**Оглавление**

[1 Задание на лабораторную работу. 3](#_Toc72398868)

[2 Решение лабораторной работы 3](#_Toc72398869)

[2.1 Задание 1. 3](#_Toc72398870)

[2.2 Задание 2. 8](#_Toc72398871)

[2.3 Задание 3. 12](#_Toc72398872)

# 1 Задание на лабораторную работу.

А) 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.

Б) 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.

# 2 Решение лабораторной работы

## 2.1 Задание 1.

Необходимо реализовать 4 метода поиска: бинарный поиск, бинарное древо, Фебонначиев поиск и интерполяционный поиск. На рисунке 1 показан результат работы программы.

Листинг 1

from random import randint  
  
print("\nБинарный поиск\n")  
  
  
arr = []  
for i in range(15):  
 arr.append(randint(1, 50))  
arr.sort()  
print(arr)  
  
  
value = int(input("искать "))  
  
value\_delete = int(input("удалить "))  
  
value\_add = int(input("добавить "))  
  
def BinarySearch(lys, val):  
 first = 0  
 last = len(lys)-1  
 index = "not found"  
 while (first <= last) and (index == "not found"):  
 mid = (first+last)//2  
 if lys[mid] == val:  
 index = mid  
 else:  
 if val<lys[mid]:  
 last = mid -1  
 else:  
 first = mid + 1  
 return index  
  
  
def BinarySearchDelete(arr, val\_d):  
 arr.pop(BinarySearch(arr, val\_d))  
  
def BinarySearchAdd(arr, val\_a):  
 arr.append(val\_a)  
 arr.sort()  
  
print("ячейка", BinarySearch(arr, value))  
  
BinarySearchDelete(arr, value\_delete)  
print("новый массив", arr)  
  
BinarySearchAdd(arr, value\_add)  
print("новый массив", arr)  
  
  
  
print("\nБинарное дерево\n")  
  
  
arr = []  
for i in range(15):  
 arr.append(randint(1, 50))  
print(arr)  
  
  
value = int(input("искать "))  
  
value\_delete = int(input("удалить "))  
  
value\_add = int(input("добавить "))  
  
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")  
  
  
def BinaryTreeDelete(tree, arr, value):  
 arr.pop(value)  
 tree = None  
 for i, el in enumerate(arr):  
 tree = BinaryTree(tree, el, i)  
  
  
def BinaryTreeAdd(tree, arr, val\_a):  
 arr.append(val\_a)  
 BinaryTree(tree, val\_a, len(arr))  
  
  
print("ячейка", BinaryTreeSearch(tree))  
  
BinaryTreeDelete(tree, arr, value\_add)  
print("новый массив", arr)  
  
BinaryTreeAdd(tree, arr, value\_add)  
print("новый массив", arr)  
  
  
  
print("\nПоиск Фибоначчи\n")  
  
  
  
arr = []  
for i in range(15):  
 arr.append(randint(1, 50))  
arr.sort()  
print(arr)  
  
  
value = int(input("искать "))  
  
value\_delete = int(input("удалить "))  
  
value\_add = int(input("добавить "))  
  
  
def FibonacciSearch(lys, val):  
 fibM\_minus\_2 = 0  
 fibM\_minus\_1 = 1  
 fibM = fibM\_minus\_1 + fibM\_minus\_2  
 while (fibM < val):  
 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 "not found"  
  
  
def FibonacciAdd(arr, val\_a):  
 arr.append(val\_a)  
 arr.sort()  
  
  
def FibonacciDelete(arr, val\_d):  
 arr.pop(BinarySearch(arr, val\_d))  
  
  
print("ячейка", FibonacciSearch(arr, value))  
  
FibonacciAdd(arr, value\_add)  
print("новый массив", arr)  
  
FibonacciDelete(arr, value\_delete)  
print("новый массив", arr)  
  
print("\nИнтерполяционный поиск\n")  
  
arr = []  
for i in range(15):  
 arr.append(randint(1, 50))  
arr.sort()  
print(arr)  
  
  
value = int(input("искать "))  
  
value\_delete = int(input("удалить "))  
  
value\_add = int(input("добавить "))  
  
  
def InterpolationSearch(lys, val):  
 low = 0  
 high = (len(lys) - 1)  
 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 "not found"  
  
  
print("ячейка", InterpolationSearch(arr, value))  
  
  
def InterpolationAdd(arr, val\_a):  
 arr.append(val\_a)  
 arr.sort()  
  
  
def InterpolationDelete(arr, val\_d):  
 arr.pop(BinarySearch(arr, val\_d))  
  
  
print("Index =", FibonacciSearch(arr, value))  
  
InterpolationAdd(arr, value\_add)  
print("новый массив", arr)  
  
InterpolationDelete(arr, value\_delete)  
print("новый массив", arr)

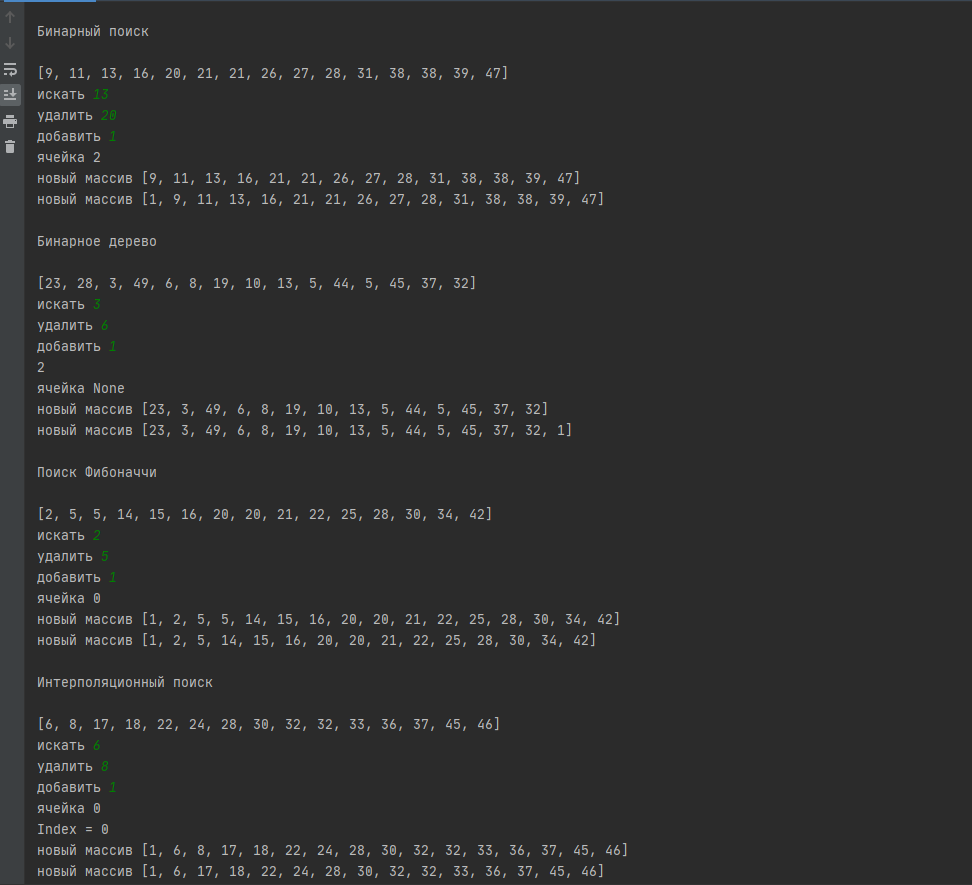


Рисунок 1 – Методы поиска

## 2.2 Задание 2.

Далее, по по плану лабораторной работы, необходимо реализовать просто рехеширование, рехеширование с помощью псевдослучайных чисел и метод цепочек. На рисунке 2 показан результат работы программы.

Листинг 2

import random  
  
print("\nПростое рехеширование\n")  
  
  
class prost\_rehash:  
 # Конструктор, создание словаря  
 def \_\_init\_\_(self):  
 self.rhash = [None] \* 256  
  
 def keys(self, element):  
 key = 0  
 for i in range(len(element)):  
 key = key + ord(element[i])  
 return int(key % 256)  
  
 def add(self, element):  
 key = self.keys(element)  
 while self.rhash[key] is not None:  
 key = key + 1  
 self.rhash[key] = element  
  
 def search(self, element):  
 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):  
 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):  
 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()  
  
print("\nРехэширование с помощью псевдослучайных чисел\n")  
  
  
class random\_rehash():  
  
 def \_\_init\_\_(self):  
 self.rhash = [None] \* 256  
  
 def rand(self, element):  
 key = int(0)  
 for i in range(len(element)):  
 key = key + ord(element[i])  
 return key  
  
 def keys(self, key, l):  
 random.seed(l)  
 return int(key + ((random.random() \* 10000000000000000) % 1000))  
  
 def add(self, element):  
 l = int(0)  
 key = self.rand(element)  
 key1 = self.keys(key, l) % 256  
 while key1 < len(self.rhash) and self.rhash[key1] is not None:  
 l = l + 1  
 key1 = self.keys(key, l) % 256  
 if key1 < len(self.rhash):  
 self.rhash[key1] = element  
 else:  
 print("Таблица заполнена")  
  
 def search(self, element):  
 l = int(0)  
 key = self.rand(element)  
 key1 = self.keys(key, l) % 256  
 while key1 < len(self.rhash) and self.rhash[key1] is not None:  
 if self.rhash[key1] == element:  
 return key1  
 else:  
 l = l + 1  
 key1 = self.keys(key, l)  
 return None  
  
 def deleted(self, element):  
 l = int(0)  
 keyn = self.rand(element)  
 key1 = self.keys(keyn, l) % 256  
 key = self.search(element)  
 if key is not None:  
 while key is not key1:  
 l = l + 1  
 key1 = self.keys(keyn, l) % 256  
 self.rhash[key] = None  
 l = l + 1  
 key1 = self.keys(keyn, l) % 256  
 while key1 < len(self.rhash) and self.rhash[key1] is not None:  
 el = self.rhash[key1]  
 self.rhash[key1] = None  
 self.add(el)  
 l = l + 1  
 key1 = self.keys(keyn, l) % 256  
 return "Элемент удален"  
 else:  
 return "Элемент не найден"  
  
 def pr(self): #  
 for key, i in enumerate(self.rhash):  
 if self.rhash[key] is not None:  
 print(key, " ", i)  
  
  
a = random\_rehash()  
a.add("qwe")  
a.add("qwe")  
a.add("qwe")  
a.add("qwe")  
a.add("qwe")  
a.pr()  
s = a.deleted("qwe")  
print(s)  
a.pr()  
print(a.search("qwe"))  
  
print("\nМетод цеопчек\n")  
  
  
class chain\_rehash:  
  
 def \_\_init\_\_(self):  
 self.rhash = [[] \* 0 for i in range(10)]  
  
 def add(self, element):  
 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):  
 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()

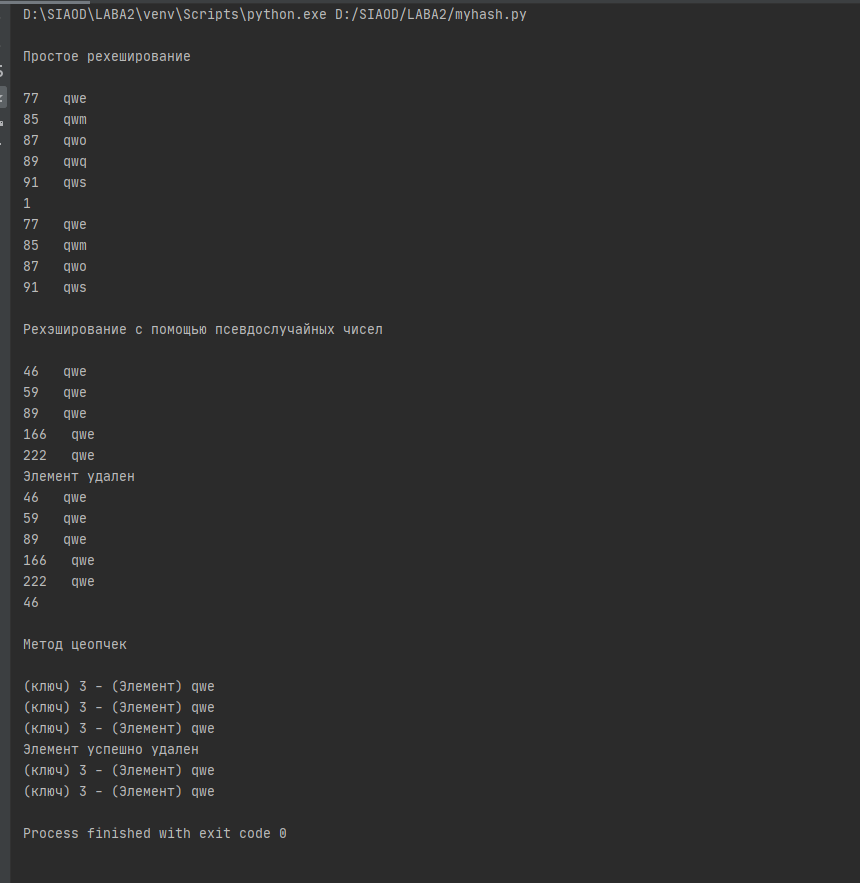


Рисунок 2 - Рехеширование

## 2.3 Задание 3.

В конце лабораторной работы не обходимо решить задачу, которая описана в пункте 1, подпункте Б. На рисунке 3 показан результат работы программы.

Листинг 3

def findQueens(Queens=[0] \* 8, i=0):  
  
 if i == 8:  
 arr = [[0 for i in range(8)] for j in range(8)]  
 for i in range(8):  
 for j in range(8):  
 arr[i][Queens[i]] = 1  
 print(Queens)  
 return arr  
 else:  
 for j in range(8):  
  
 if checkField(i, j, Queens):  
  
 Queens[i] = j  
  
 chessBoard = findQueens(Queens, i + 1)  
  
 if chessBoard:  
 return chessBoard  
  
  
  
def checkField(i, j, Queens):  
 r = i  
 c = j  
  
 for k in range(i):  
 if j == Queens[k]:  
 return False  
  
 while i >= 0 and j >= 0:  
 if Queens[i] == j:  
 return False  
 i -= 1  
 j -= 1  
  
 while r >= 0 and c <= 7:  
 if Queens[r] == c:  
 return False  
 r -= 1  
 c += 1  
 return True  
  
  
findQueens()

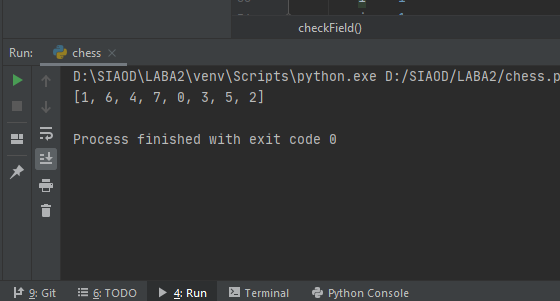


Рисунок 3 – Результат работы программы

Вывод: в данной лабораторной работе были изучены и применены на практики различные методы поиска и рехеширование.