

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: Мкртчян Г.М.

Москва 2021

Оглавление

| | |
|---------------------------------------|----|
| 1 Задание на лабораторную работу..... | 3 |
| 2 Решение лабораторной работы..... | 3 |
| 2.1 Задание 1..... | 3 |
| 2.2 Задание 2..... | 8 |
| 2.3 Задание 3..... | 12 |

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 дерево, Фибоначчиев 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)

```

```
Бинарный поиск
[9, 11, 13, 16, 20, 21, 21, 26, 27, 28, 31, 38, 38, 39, 47]
искать 13
удалить 20
добавить 1
ячейка 2
новый массив [9, 11, 13, 16, 21, 21, 26, 27, 28, 31, 38, 38, 39, 47]
новый массив [1, 9, 11, 13, 16, 21, 21, 26, 27, 28, 31, 38, 38, 39, 47]

Бинарное дерево
[23, 28, 3, 49, 6, 8, 19, 10, 13, 5, 44, 5, 45, 37, 32]
искать 3
удалить 4
добавить 1
2
ячейка None
новый массив [23, 3, 49, 6, 8, 19, 10, 13, 5, 44, 5, 45, 37, 32]
новый массив [23, 3, 49, 6, 8, 19, 10, 13, 5, 44, 5, 45, 37, 32, 1]

Поиск Фибоначчи
[2, 5, 5, 14, 15, 16, 20, 20, 21, 22, 25, 28, 30, 34, 42]
искать 2
удалить 3
добавить 1
ячейка 0
новый массив [1, 2, 5, 5, 14, 15, 16, 20, 20, 21, 22, 25, 28, 30, 34, 42]
новый массив [1, 2, 5, 14, 15, 16, 20, 20, 21, 22, 25, 28, 30, 34, 42]

Интерполяционный поиск
[6, 8, 17, 18, 22, 24, 28, 30, 32, 32, 33, 36, 37, 45, 46]
искать 8
удалить 8
добавить 1
ячейка 0
Index = 0
новый массив [1, 6, 8, 17, 18, 22, 24, 28, 30, 32, 32, 33, 36, 37, 45, 46]
новый массив [1, 6, 17, 18, 22, 24, 28, 30, 32, 32, 33, 36, 37, 45, 46]
```

Рисунок 1 – Методы поиска

2.2 Задание 2.

Далее, по плану лабораторной работы, необходимо реализовать простое 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 цепочек\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()

```

```
D:\SIA0D\LABA2\venv\Scripts\python.exe D:/SIA0D/LABA2/myhash.py

Простое рехеширование

77 qwe
85 qwm
87 qwo
89 qwq
91 qws
1
77 qwe
85 qwm
87 qwo
91 qws

Рехеширование с помощью псевдослучайных чисел

46 qwe
59 qwe
89 qwe
166 qwe
222 qwe
Элемент удален
46 qwe
59 qwe
89 qwe
166 qwe
222 qwe
46

Метод цеопчек

(ключ) 3 - (Элемент) qwe
(ключ) 3 - (Элемент) qwe
(ключ) 3 - (Элемент) qwe
Элемент успешно удален
(ключ) 3 - (Элемент) qwe
(ключ) 3 - (Элемент) qwe

Process finished with exit code 0
```

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

```

```

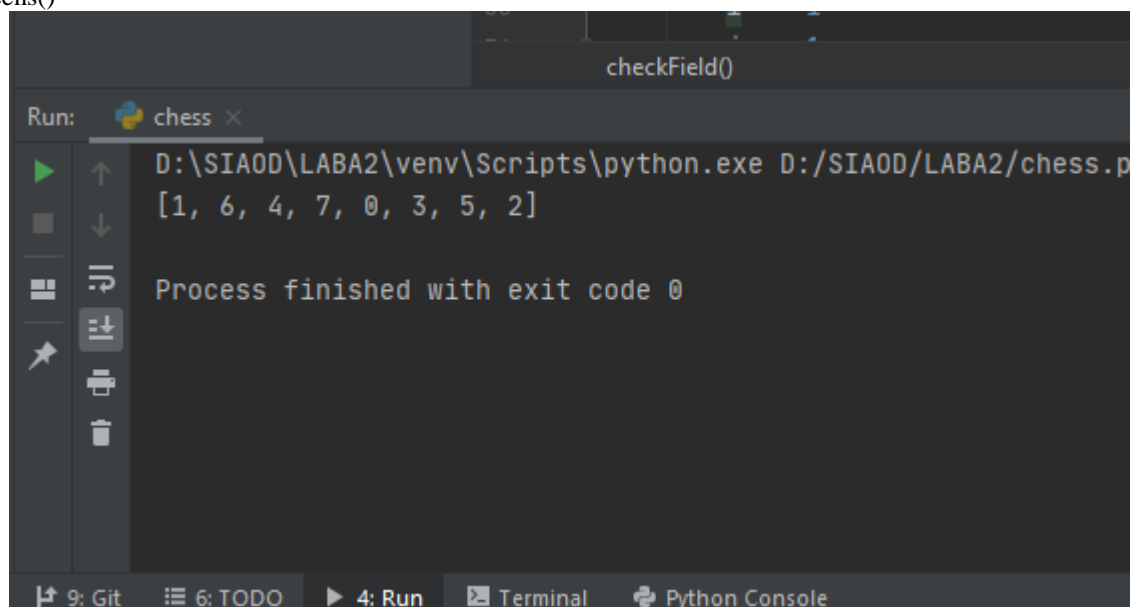
    while r >= 0 and c <= 7:
        if Queens[r] == c:
            return False
        r -= 1
        c += 1
    return True

```

```

findQueens()

```



The screenshot shows a Python IDE interface. At the top, there's a tab labeled 'chess'. Below it, the 'Run' window is open, showing the command executed: 'D:\SIA0D\LABA2\venv\Scripts\python.exe D:/SIA0D/LABA2/chess.p'. The output of the program is displayed as '[1, 6, 4, 7, 0, 3, 5, 2]'. Below the output, it says 'Process finished with exit code 0'. The IDE has a dark theme and includes a sidebar with icons for file operations and a bottom status bar with tabs for 'Git', 'TODO', 'Run', 'Terminal', and 'Python Console'.

Рисунок 3 – Результат работы программы

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