Национальный исследовательский университет «Высшая школа экономики» Специальность «Компьютерная безопасность»

Отчет
По лабораторной работе №2
«Поиск»

по дисциплине «Методы программирования»
направления «Компьютерная безопасность»

Вариант 16

Шаплавский Л.П.
СКБ-182

Москва, 2021

Практическое задание по теме «Алгоритмы поиска данных»

1) Реализовать прямой и бинарный поиск заданного элемента в массиве объектов по ключу в соответствии с вариантом (ключом является первое НЕ числовое поле объекта).
2) Входные данные для поиска обязательно считывать из внешних источников: текстовый файл, файл MS Excel, MS Access, данные из СУБД (любое на выбор).
3) Выполнить поиск 7-10 раз на массивах разных размерностей от 100 и более (но не менее 100000). Засечь (программно) время поиска для следующих способов: прямой поиск, бинарный поиск в заранее отсортированном массиве, сортировка массива (наиболее эффективным методом из работы 2) и бинарный поиск в нем. По полученным точкам построить графики зависимости времени поиска от размерности массива.
4) Записать входные данные в ассоциативный массив multimap<key, object> и сравнить время поиска по ключу в нем с временем поиска из
   п.3. Добавить данные по времени поиска в ассоциативном массиве в общее сравнение с остальными способами и построить график зависимости времени поиска от размерности массива.
5) Сделать отчет, содержащий титульный лист, код программы со спецификациями каждого метода и подробными комментариями, графики скоростей поиска и выводы.

main.py

```
import data_modul
from StudentClass import Student,Multimap
import time
import SortingModule
import datetime
import SearchModul
import copy

print(datetime.datetime.now())
findword = "Шварев"
sampl = [100,500,1000,5000,10000,50000,100000]

for i in range(len(sampl)):
    print('\n' + '-----' + str(sampl[i]) + '-----')
    data_modul.creation_of_data(sampl[i]) # создаем файл с данными
    data_array = data_modul.creation_array(sampl[i])




    start2 = time.time()
    b = SearchModul.direct_search(data_array,findword)
    end2 = time.time()

    data_modul.write_data_info('direct_search ', start2 - end2, sampl[i])
    data_modul.write_data_info_live('direct_search ', start2 - end2, sampl[i])




    start3 = time.time()
    c = SearchModul.binar_search_with_sortind(data_array,findword)
    end3 = time.time()

    data_modul.write_data_info('binar_search_with_sortind ', start3 - end3, sampl[i])
    data_modul.write_data_info_live('binar_search_with_sortind ', start3 - end3, sampl[i])






    start1 = time.time()
    a = SearchModul.binar_search(data_array,findword)
    end1 = time.time()

    data_modul.write_data_info('binar_search', start1 - end1, sampl[i])
    data_modul.write_data_info_live('binar_search', start1 - end1, sampl[i])




    merge = SortingModule.merge_sort(copy.deepcopy(data_array))
    multik = Multimap(merge)

    start4 = time.time()
    multik.find(findword)
    end4 = time.time()

    data_modul.write_data_info('multimap', start4 - end4, sampl[i])
    data_modul.write_data_info_live('multimap', start4 - end4, sampl[i])


print(datetime.datetime.now())
```

StudentClass.py

```
import time


class Student:
    level: int
    name: str
    fac: str
    spec: str

    def __init__(self,line):
        level,name,fac,spec = line.split('|')
        level = int(level)
        self.level = level
        self.name = name
        self.fac = fac
        self.spec = spec

    def PrintInfo(self):
        print(self.level,'|',self.name,'|',self.fac,'|',self.spec)

    def PrintInfoSTR(self):
        return str(str(self.level) + '|' + self.name + '|' + self.fac + '|' + self.spec)

    def __lt__(self, other):  #  <
        if self.name < other.name:
            return True
        else:
            return False

    def __le__(self, other):  #  <=
        if self.name == other.name:
            return True
        elif self.name < other.name:
            return True
        else:
            return False

    def __gt__(self, other):  #  >
        if isinstance(other, str):

            if self.name > other:
                return True
            else:
                return False
        else:
            if self.name > other.name:
                return True
            else:
                return False

    def __ge__(self, other):  #  >=
        if self.name == other.name:
            return True
        elif self.name > other.name:
            return True
        else:
            return False

    def __eq__(self, other):  #  ==
        if self.name == other:
            return True
        else:
            return False
```

```
class Multimap():
    multarr = []


    def __init__(self, array):
        self.multarr = []
        self.keys=[]
        for elem in array:
            self.multarr.append((elem.name, elem))

    def PrintInfoMM(self):
        print(self.multarr)

    def find(self, find_word):
        midel = len(self.multarr) // 2
        start = 0
        end = len(self.multarr) - 1
        while find_word != self.multarr[midel][0] and start <= end:
            if self.multarr[midel][0] > find_word:
                end = midel - 1
            else:
                start = midel + 1
            midel = (end + start) // 2
        time.sleep(0.001)
        if end >= start:
            return midel
        else:
            return -1
```

data_modul.py

```
import random
import StudentClass
import datetime

def creation_of_data(N):
    name = ['Шварев','Римский', 'Чайковский', 'Мусоргский', 'Бородин', 'Глинка',
'Скрябин','Рахманинов',
'Стравинский','Прокофьев','Шостакович','Шаплавский','Шимановский','Смирнов','Иванов','Кузнецо
в','Соколов','Попов','Лебедев','Козлов','Новиков','Морозов','Петров','Волков','Соловьёв','Вас
ильев','Зайцев','Павлов','Семёнов','Голубев']
    fak = ['Факультет экономических наук','Московский институт электроники и математики
им.А.Н. Тихонова','Факультет компьютерных наук','Высшая школа бизнеса','Факультет
права','Высшая школа юриспруденции и администрирования','Факультет
гуманитарныхнаук','Факультет социальных наук']
    spec = [['Департамент теоретической экономики','Департамент
прикладнойэкономики','Департамент математики'],['Департамент электронной
инженерии','Департамент компьютернойинженерии','Департамент прикладной
математики'],['Департамент программной инженерии','Департамент анализа данных иискусственного
интеллекта','Департамент больших данных и информационного поиска'],['Департамент бизнес-
информатики','Департамент маркетинга','Департаментоперационного менеджмента и
логистики'],['Департамент теории права и межотраслевых юридических
дисциплин','Департаментпубличного права','Департамент частного права'],['Институт
юридического менеджмента','Институт кадровогоадминистрирования','Институт спортивного
менеджмента и права'],['Школа философии и культурологии','Школа филологических
наук','Институтклассического Востока и античности'],['Департамент социологии','Департамент
политики и управления','Департаментпсихологии']]
    f1=open('datd.txt','w')#создание файла с данными

    b = random.randint(0, N - 1)

    for i in range(N):
        fak_num = random.randint(0, len(fak) - 1)
        if i == b:
            f1.write('{}'.format(random.randint(100, 300)) + '|' + name[0] + '|' +
fak[fak_num] + '|' + spec[fak_num][random.randint(0, 2)] + '\n')
        else:
            f1.write('{}'.format(random.randint(100,300))+ '|' +
name[random.randint(1,len(name)-1)] + '|' + fak[fak_num] + '|'
+spec[fak_num][random.randint(0,2)] + '\n')
    f1.close()

    return 0

def creation_array(N): #передача данных файла в массив
    f1 = open('datd.txt','r')
    data_array = []
    for i in range(N):
        line = f1.readline()
        data_array.append(StudentClass.Student(line))
    f1.close()
    return data_array

def array_info(array): #вывод массива
    for i in range(len(array)):
        array[i].PrintInfo()
    return 0

def output_file(array,Time): #запись данных в файл
    f1 = open('data_output.txt','w')
    for i in range(len(array)):
        f1.write(array[i].PrintInfoSTR())
        f1.write("time:"+str(Time))
        f1.close()
    return 0

def write_data_info(word,time,sampl):
```

```
    f1=open('datd_info.txt','a')
    f1.write(str(word)+ '|' + str(time) + '|' + str(sampl) + '|' +
str(datetime.datetime.now()) + '\n'+'\n')
    f1.close()
    return 0

def write_data_info_live(word,time,sampl):
    print(str(word) + '|' + str(time) + '|' + str(sampl) + '|' + str(datetime.datetime.now())
+ '\n' +'_________')
    return 0
```

SortingModul.py

```
import StudentClass
import copy

def selection_sort(data):
    array = copy.deepcopy(data)
    i = 0
    while i < len(array) - 1:
        smal = i
        j = i + 1
        while j < len(array):
            if array[j].__lt__(array[smal]):
                smal = j
            j += 1
        array[i], array[smal] = array[smal], array[i]
        i += 1
    return array

def HeapSort(lst):
    data=copy.deepcopy(lst)
    for i in range((len(data) - 2) // 2, -1, -1):
        HeapSift(data, i, len(data) - 1)
    for end in range(len(data) - 1, 0, -1):
        data[end], data[0] = data[0], data[end]
        HeapSift(data, 0, end - 1)
    return data

def HeapSift(data, start, end):
    root = start
    while True:
        child = root * 2 + 1
        if child > end:
            break
        if ((child + 1) <= end) and (data[child].__lt__(data[child + 1]) ):
            child += 1
        if data[root].__lt__(data[child]):
            data[root], data[child] = data[child], data[root]
            root = child
        else:
            break


def merge(left, right): #сортировка слиянием
    res = []
    i = 0
    j = 0
    while i < len(left) and j < len(right):
        if left[i].__le__(right[j]):
            res.append(left[i])
            i += 1
        else:
            res.append(right[j])
            j += 1
    res += left[i:] + right[j:]
    return res
```

```
def merge_sort(lst):
    array = copy.deepcopy(lst)
    if len(array) <= 1:
        return array
    else:
        left = array[:len(array) // 2]
        right = array[len(array) // 2:]
    return merge(merge_sort(left), merge_sort(right))
```

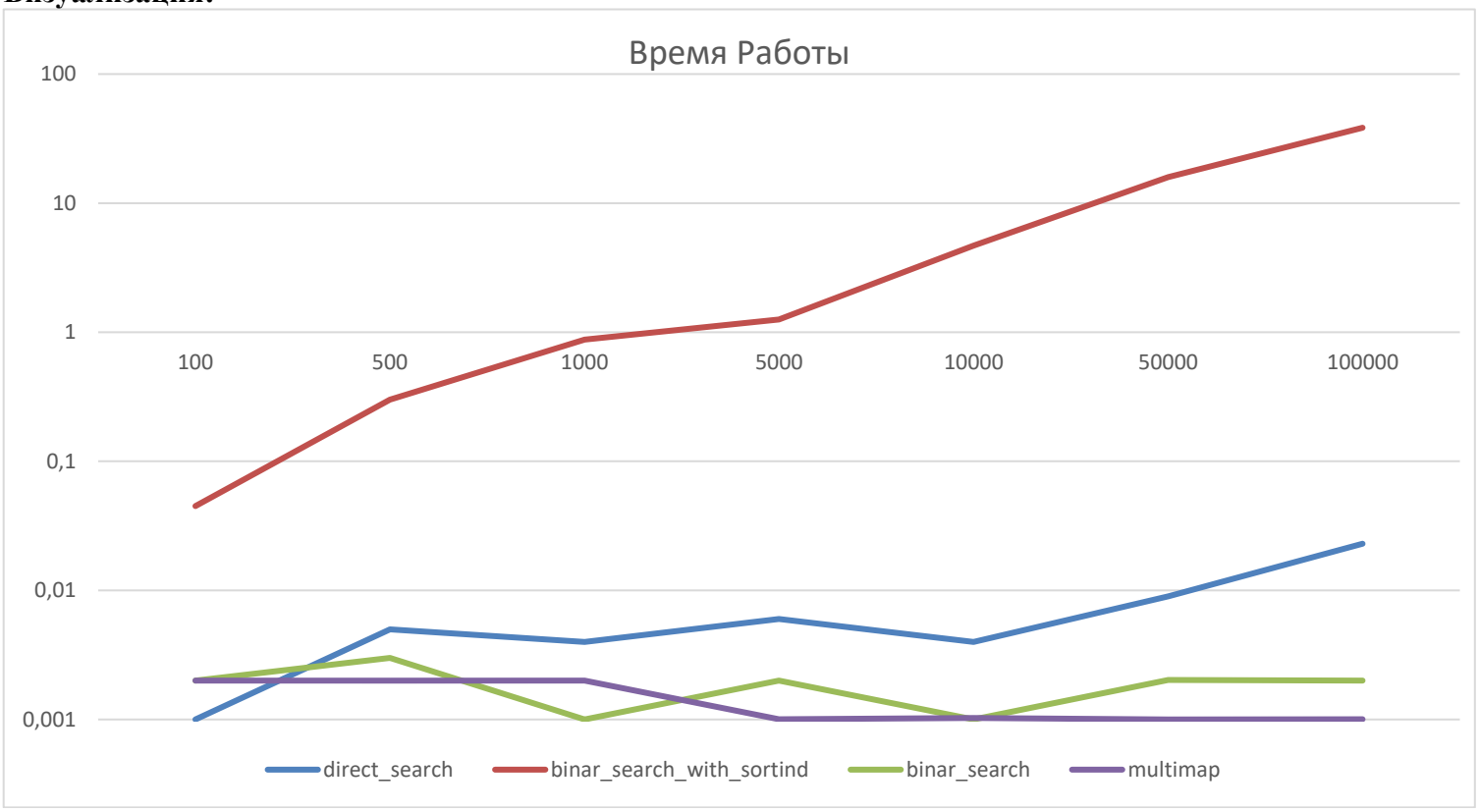
SearchModul.py

```
import copy
import SortingModule
import time


def direct_search(array,find_word):
    f = 0
    for i in range(len(array)):
        if array[i] == find_word:
            f = i
            break
    time.sleep(0.0001)
    return f


def binar_search(array, find_word):
    midel = len(array) // 2
    start = 0
    end = len(array) - 1
    while find_word != array[midel] and start <= end:
        if array[midel] > find_word:
            end = midel - 1
        else:
            start = midel + 1
        midel = (end + start) // 2
    time.sleep(0.0001)
    if end >= start:
        return midel
    else:
        return -1


def binar_search_with_sortind(array, find_word):
    merge = SortingModule.merge_sort(copy.deepcopy(array))

    midel = len(merge) // 2
    start = 0
    end = len(merge) - 1
    while find_word != merge[midel] and start <= end:
        if merge[midel] > find_word:
            end = midel - 1
        else:
            start = midel + 1
        midel = (end + start) // 2
    time.sleep(0.0001)
    if end >= start:
        return midel
    else:
        return -1
```

Результат:

	100	500	1000	5000	10000	50000	100000
direct_search	0,001000404	0,004999638	0,003998041	0,00599885	0,003998756	0,008996964	0,022992849
binarsearchwithsortind	0,044986963	0,300907373	0,874727726	1,255609512	4,680536747	15,93309188	38,40539026
binar_search	0,002000093	0,002998829	0,001000404	0,001999855	0,001000404	0,002023458	0,001998425
multimap	0,001999617	0,001999855	0,001998663	0,001003742	0,001026392	0,001000404	0,001004457

Визуализация:



Вывод:

По результатам исследования видим, что бинарный поиск с сортировкой работает медленнее, чем остальные алгоритмы поиска