Национальный исследовательский университет
Высшая школа экономики
Московский институт электроники и математики

Департамент прикладной математики
кафедра компьютерной безопасности

Отчет по лабораторной работе №2 по дисциплине "Методы программирования"

Вариант: 26
Массив данных генеологического фонда: ФИО - поле для сравнения
Сортировка: пирамидальная

Выполнил:
Фролов О.В. СКБ201

Москва, 2023

Оглавление

Задание 1.

Задание

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 сортировки массива считывать из внешних источников.
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 прошлой работы) 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, состоящий из: документации (doxygen), ссылки на исходный код программы в репозитории, графики зависимости времени поиска от размера массива.

1. Подготовка входных данных

Воспользуемся инструментарием языка программирования Python для генерации файлов со случайными записями формата: "Фамилия Имя Отчество год_рождения год_смерти кол-во_детей".
Пример полученного входного файла:

```
Жашлыков(-а) Мария Эдуардович(-вна) 1905 2027 0
Зролов(-а) Вадим Александрович(-вна) 1906 2054 2
Краско Ангелина Николаевич(-вна) 1907 2147 1
Лестеренко Бенито Георгиевич(-вна) 1908 2013 3
Никитченко Рауль Эдуардович(-вна) 1910 2070 2
Ассолини Вадим Александрович(-вна) 1901 2043 0
Дикитченко Рауль Львович(-вна) 1904 1998 3
Браско Илья Григорьевич(-вна) 1902 2054 4
Гуссолини Рауль Максимович(-вна) 1903 1943 0
Краско Олег Артемович(-вна) 1975 1995 3
Муссолини Ангелина Олегович(-вна) 2018 2112 4
Воловик Анастасия Артемович(-вна) 2000 2091 4
Ушаков(-а) Олег Артемович(-вна) 1986 2022 2
Муссолини Бенито Георгиевич(-вна) 1941 1985 2
Онищенко Олег Александрович(-вна) 1927 1986 4
Нестеренко Олег Георгиевич(-вна) 2013 2087 4
Воловик Анастасия Олегович(-вна) 1967 2058 1
Онищенко Рауль Олегович(-вна) 1955 1992 3
Краско Мария Александрович(-вна) 2011 2066 1
Краско Мария Николаевич(-вна) 1997 2065 4
Никитченко Илья Эдуардович(-вна) 2011 2059 3
Муссолини Анастасия Витальевич(-вна) 1967 2051 1
Онищенко София Олегович(-вна) 2011 2038 2
Воловик София Эдуардович(-вна) 2042 2140 3
Никитченко Анастасия Александрович(-вна) 1991 2026 4
Краско Владислав(-а) Григорьевич(-вна) 1980 2006 4
Бронштейн Вадим Эдуардович(-вна) 1935 2044 3
Фролов(-а) Анастасия Григорьевич(-вна) 2022 2114 4
```

Программа для генерации, написанная на Python'е:

In [15]:
```
import random

familii=["Фролов(-а)", "Онищенко", "Воловик", "Ушаков(-а)", "Никитченко", "Бронштейн", "Муссолини", "Краско", "Ташлыков(-а)", "Не
imena=["Олег", "Мария", "Ангелина", "Илья", "Владислав(-а)", "Вадим", "Рауль", "Анастасия", "Бенито", "София"]
otchestva=["Витальевич(-вна)", "Максимович(-вна)", "Георгиевич(-вна)", "Олегович(-вна)", "Николаевич(-вна)", "Александрович(-вна)

def data_create(n: int):
    res=[]
    for i in range(n):
        result=""
        child_amount=random.randrange(5)
        year=random.randint(1900, 2050)
        year_death=year+random.randint(18, 110)
        fio=familii[random.randint(0, len(familii)-1)]+" "+imena[random.randint(0, len(imena)-1)]+" "+otchestva[random.randint(0,
        result=str(fio)+" "+ str(year)+" "+str(year_death)+" "+str(child_amount)
        res.append(result)

    return res
```

In [22]:
```
def gen_values():
    cycle=random.randint(7, 11)
    for i in range(cycle):
        amount=random.randint(100, 100100)
        data=data_create(amount)
        #исправить utf-8 на ansi и проверить!
        file=open("data_"+str(i)+".txt", 'w', encoding='ansi')
        for elem in data:
            file.write(elem)
            file.write('\n')
        file.close()
gen_values()
```

Далее сгенерируем 13 файлов с подобными записями размерности от 100 до 100100. Пример названия файла: "data_1.txt"
Отсортированный массив будет выведен в отдельный файл. Пример названия файла: "out_1.txt".

2. Реализация собственного класса

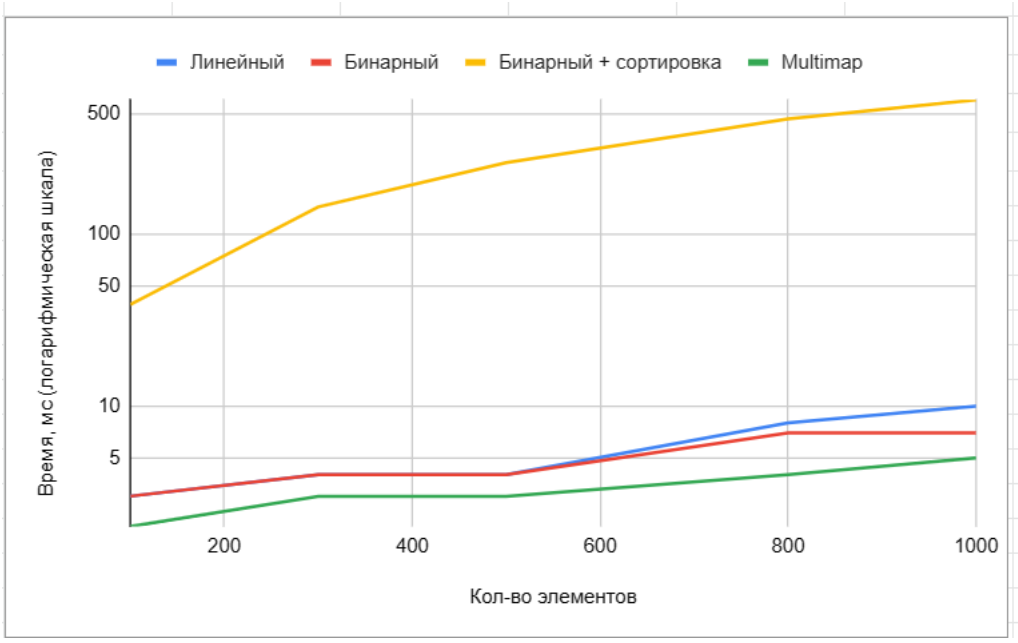
Для дальнейшей работы потребуется реализовать класс, который будет хранить в себе данные из данных записей, т.е. в нем будут следующие поля: фамилия, имя, отчество, год рождения, год смерти и кол-во детей. Данный класс у меня называется - Element.

3. Реализация алгоритмов поиска

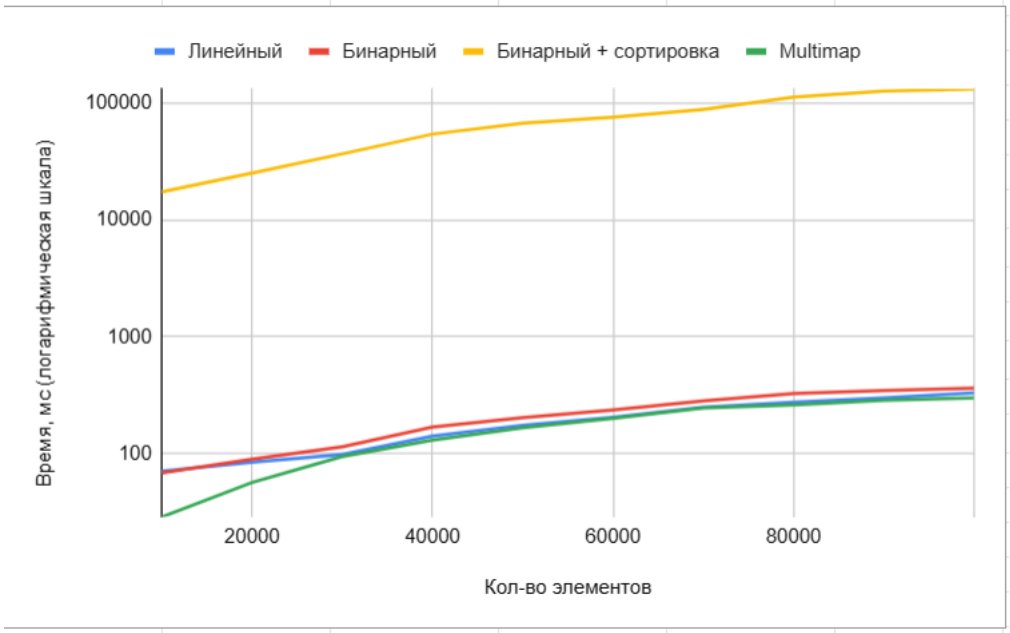
В Main'е мною были реализованы следующие функции поиска: linearSearch - прямой (линейный) поиск, binarySearch1 - бинарный поиска, findMultimap - поиск в multimap'е. heapSort1 - пирамидальная сортировка, был выбран данный алгоритм сортировки, т.к. он лучше всего показал себя в предыдущей работе. Сортировка для бинарного поиска происходит по ФИО, т.к. поиск происходит по ФИО (первое не числовое поле объекта!), для бинарного поиска массив должен быть отсортирован, иначе данный алгоритм нельзя применить.

4. Сравнение скоростей сортировок

Кол-во элемент	Линейный	Бинарный	Бинарный + пир	multimap
100	3	3	39	2
300	4	4	144	3
500	4	4	260	3
800	8	7	467	4
1000	10	7	604	5



Кол-во элемент	Линейный	Бинарный	Бинарный + пир	multimap
10000	71	69	17322	29
20000	85	90	25030	57
30000	99	115	36530	95
40000	142	170	53891	131
50000	176	204	66593	167
60000	205	237	75032	201
70000	251	283	87321	247
80000	276	328	111378	263
90000	301	348	125217	288
100000	332	365	131024	301



5. Выводы

Полученные результаты времени поиска согласуются с теоретической оценкой среднего времени работы алгоритмов. Для бинарного - O(logn) и сортировка с теоретической оценкой среднего времени работы O(nlogn), для линейного - O(n). Линейный и бинарный дают практически одинаковый результат как при малых, так и при больших размерностях массивов данных. Однако стоит заметить, что быстрее всего работает поиск в хеш-таблице (multimap), но на практике требуется ещё занести данные в хеш-таблицу. Поэтому в обычном случае эффективнее всего будет иметь отсортированный массив и использовать бинарный поиск - по моим данным - если массив не превышает размерность в 20000 элементов, в ином случае самый эффективный метод - линейный поиск. Если же время заполнения хеш-таблицы не превышает и значительно меньше времени линейного поиска или времени бинарного поиска в заранее отсортированном массиве, то поиск в хеш-таблице окажется гораздо эффективнее для работы с большим количеством данных.

6. Пример работы программы

```
Choose search algorithm (0 - linear (already sorted), 1 - binar (already sorted), 2 - sort and binar, 3 - multimap (sea
ch by key)):
0
Choose file(0-7)
3
17281

Parsing done!



Input your search query, please!
Бронштейн Ангелина Георгиевич

Бронштейн Ангелина Георгиевич


Linear searching started!
Бронштейн Ангелина Георгиевич was found at index 4570
The time: 85 ms
Linear searching ended!



End!

Would you like to search again (0 - linear (already sorted), 1 - binary (already sorted), 2 - sort and binary, 3 - multi
map and search by key,  -2 - end)?
```

При запуске программы нужно выбрать алгоритм поиска: 0 - линейный, 1 - бинарный в заранее отсортированном пирамидальной сортировкой массиве, 2 - бинарный + пирамидальная сортировка массива, 3 - поиск в multimap. Далее выбирается файл с данными: 0-7 (имеют названия data_0.txt, data_1.txt). Дальше выводится количество записей в файле, и начнается процесс поиска. Когда процесс будет завершен, пользователь будет уведомлен о завершении поиска, увидит, имеется ли в данном массиве запись, соответствующая его запросу, и на каком месте (в какой строке) она находится. А также сможет ознакомиться с затраченным на эту процедуру временем. Далее пользователь может завершить выполнение программы, отправив -2, или же выбрать и запустить иной алгоритм поиска.