Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет ПИиКТ

ОТЧЁТ

По лабораторной работе № 1

По предмету: Низкоуровневое программирование

Вариант: 1

Студент:

Андрейченко Леонид Вадимович

Группа P33301

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт – Петербург

2022

Цель

Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего варианту вида.

Задачи

- Спроектировать структуры данных для представления информации в оперативной памяти
- Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним:
- Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс с операциями удаления, добавления, поиска, изменения.
- Реализовать тестовую программу для демонстрации работоспособности решения
- Результаты тестирования приложить в виде графиков

Описание работы

Программа представляет из себя приложение, которое получает данные из текстового файла (генерируется отдельно), производит операции над данными и хранить их в формате документного дерева. Глобально проект поделен на части:

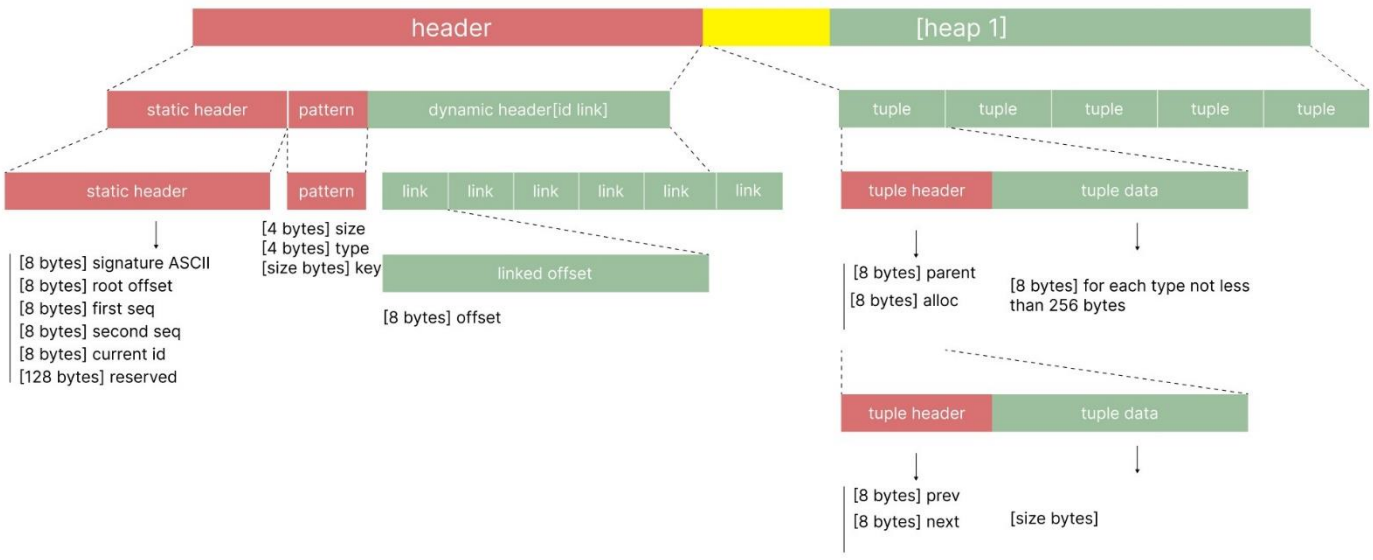
- data_generator – в данной директории хранятся тестовые данные и скрипт, который их генерирует.
- include – хранит заголовочные файлы проекта
- src
    - data – содержит функции для работы с тестовыми данными, их обработкой.
    - filetools – базовые функции для работы с файлами.
    - generator – нужен для инициализации структур программы
    - interface – содержит базовые функции в файле crud_interfaces, а их реализация и “низкоуровневая” составляющая в файле basic_crud.
    - perf – Содержит функции обертки для базовых функций, с помощью которых собирается статистика.

Приложение кроссплатформенное, использовал make, в последствии приложение.

Также разработан скрипт на python который генерирует тестовые данные.

Аспекты реализации

Модель документа



Глобально модель поделена на две части

- header – заголовок программы хранит мета информацию
    - static header – статические, не расширяемые служебные данные.
    - pattern – содержит информацию о типе входных данных
    - dynamic header – массив ссылка на данные (tuple) динамически расширяется
- heap
    - base_tuple – структура, которая позволяет хранить в себе целые числа, дробные числа, булевы значения
    - string_tuple – хранит только строки

Пример реализации структур

```
struct tree_subheader {
    uint64_t ASCII_signature;
    uint64_t root_offset;
    uint64_t first_seq;
    uint64_t second_seq;
    uint64_t cur_id;
    uint64_t pattern_size;
};
```

```
struct tree_header {
    struct tree_subheader *subheader;
    struct key **pattern;
    uint64_t *id_sequence;
};
```

```
struct tuple {
    union tuple_header header;
    uint64_t *data;
};
```

```
#pragma pack(push, 4)
struct key_header {
    uint32_t size;
    uint32_t type;
};
struct key {
    struct key_header *header;
    char *key_value;
};
#pragma pack(pop)
```

Базовые операции

- Добавление: при добавлении элемента мы добавляем его в конец файла, и добавляем ссылку на его место в памяти в dynamic header. Благодаря этому отсутствует внешняя фрагментация
- Удаление: при удалении элемента мы сначала находим его id в dynamic header, рекурсивно начинаем искать всех его потомков, удаляя и их. При удалении tuple, также рекурсивно, удаляются все хранящиеся в нем строки и данные.
- Обновление: при обновлении элемента мы находим id элемента, вытаскиваем его из памяти, обновляем значения, и кладем обратно. Если новые данные больше предыдущих, то добавляются новые tuple.
- Поиск: при поиске по id просто ищем его в dynamic header и достаем, при поиске по полям, заглядываем в поля.

Семантика базовых функций

- Добавление элемента O(1)
- Поиск по индексу O(1)
- Поиск по соответствию поля O(n)
- Удаление элемента по индексу O(m)
- Обновление элемента по индексу O(1)
- По количеству занимаемой памяти O(n)

Результаты

- Описаны ключевые структуры для работы программы.
- Была разработана тестирующая выборка данных
- Были разработаны тесты, и получены их результаты, которые совпали с оценкой производительности программы

Заполнение header *NIX

```
--- SUBHEADER ---
ASCII Signature:    65534
Root Offset:        0
First Sequence:     0
Second Sequence:    0
Current ID:         600
Pattern Size:       4
--- PATTERN ---
Key   8 [Type   3]: name
Key   8 [Type   1]: age
Key   8 [Type   2]: height
Key   8 [Type   0]: male
```

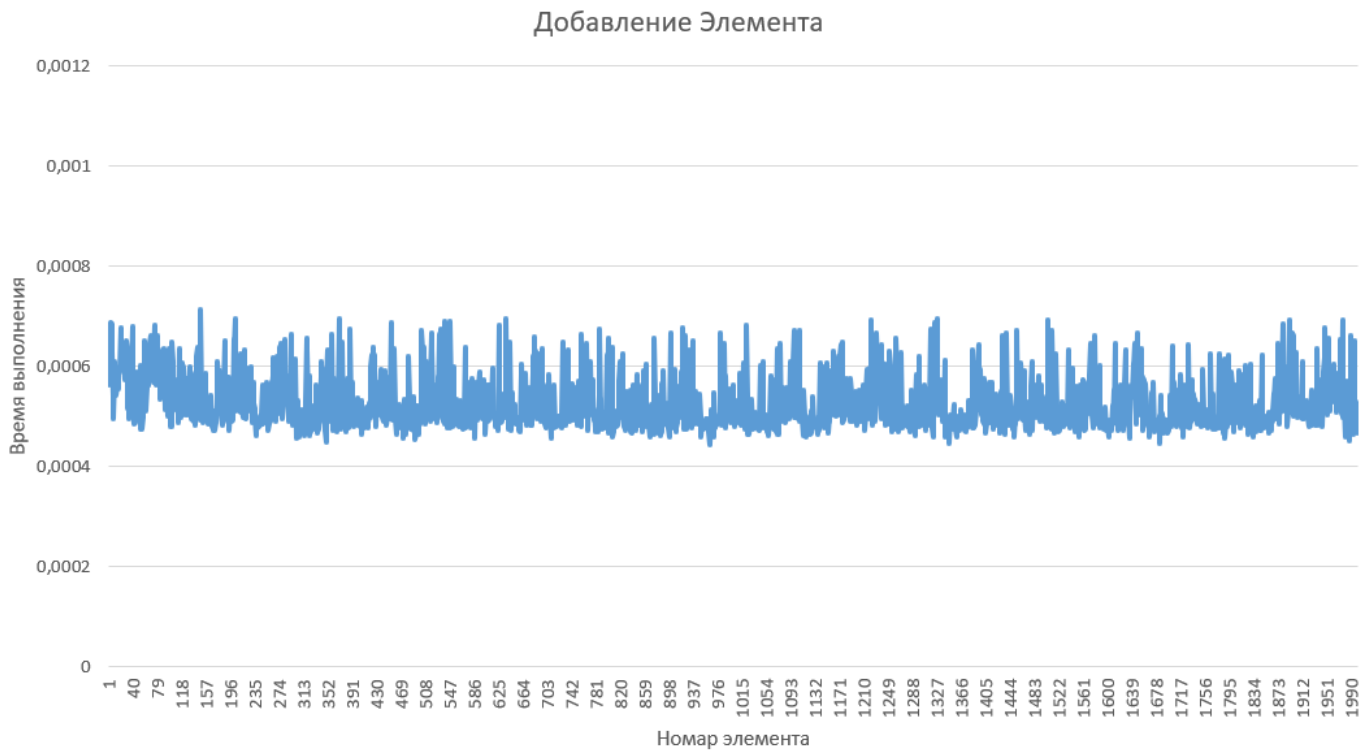
Заполнение header Windows

```
--- SUBHEADER ---
ASCII Signature:    65534
Root Offset:        0
First Sequence:     0
Second Sequence:    0
Current ID:         5
Pattern Size:       4
--- PATTERN ---
Key   8 [Type   3]: name
Key   8 [Type   1]: age
Key   8 [Type   2]: height
Key   8 [Type   0]: male
```

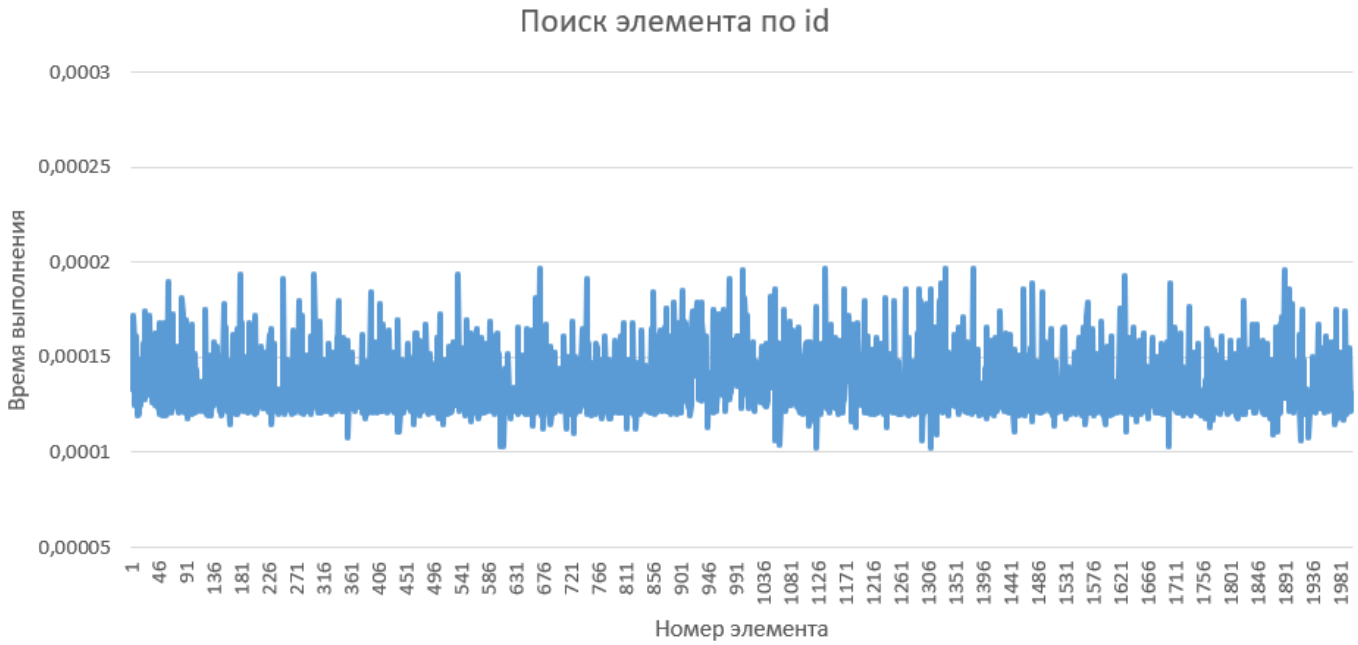
Массив данных после заполнения

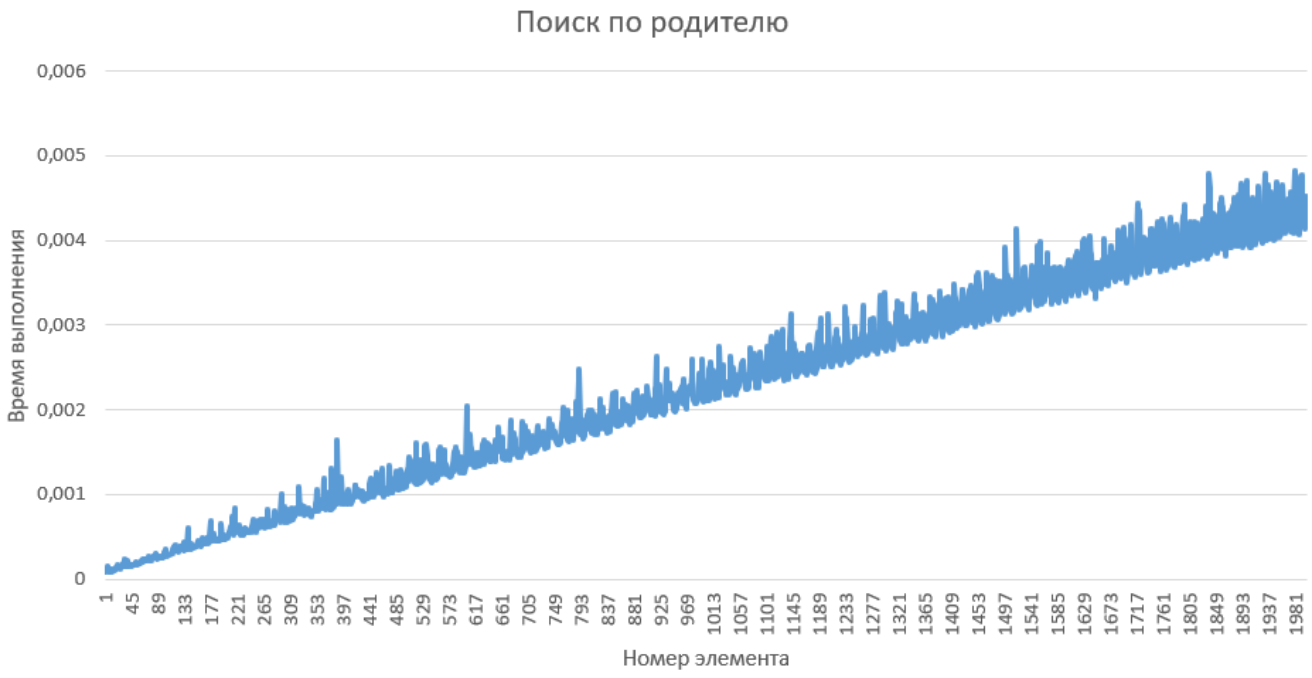
```
--- TUPLE 997 ---
name                imznmkcepbsybfanltoj
age                 22421
height              75.522330
male
                    0
--- TUPLE 998 ---
name                ehzrbpegvcalkodaxppg
age                 27305
height              32.798530
male
                    0
--- TUPLE 999 ---
name                awblfjggeschrfqyuvta
age                 61766
height              99.567860
male
                    0
```

Графики производительности




Поиск элемента по полю
Время выполнения
0,01
0,008
0,006
0,004
0,002
0
0
500
1000
1500
2000
Номар элемента


Поиск элемента по id
Время выполнения
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
Номер элемента


Поиск по родителю
Время выполнения
0,006
0,005
0,004
0,003
0,002
0,001
0
Номер элемента


Обновление элемента
Время выполения
0,0004
0,00035
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
0
1 57 113 169 225 281 337 393 449 505 561 617 673 729 785 841 897 953 1009 1065 1121 1177 1233 1289 1345 1401 1457 1513 1569 1625 1681 1737 1793 1849 1905 1961
Номер элемента


Удаление элемента без зависимостей
Время выполнения
0,0004
0,00035
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
0
1 39 77 115 153 191 229 267 305 343 381 419 457 495 533 571 609 647 685 723 761 799 837 875 913 951 989 1027 1065 1103 1141 1179 1217 1255 1293 1331 1369 1407 1445 1483 1521 1559 1597 1635 1673 1711 1749 1787 1825 1863 1901 1939 1977
Номер элемента


Зависимость объема ОЗУ от добавления элементов
Объем памяти
9450
9400
9350
9300
9250
9200
9150
9100
9050
9000
0 200 400 600 800 1000 1200 1400 1600 1800 2000
Количество элементов

Выводы

В ходе выполнения данной лабораторной работы я разработал программу, которая хранит данные, и удовлетворяет требованиям по памяти и производительности. Большим плюсом было решение хранить массив ссылок на соответствующие кортежи, благодаря этому можно итерироваться по массиву ссылок, что в контексте данной задачи равно константе. Также решения добавлять новые кортежи в конец файла было полезно. При хранении длинных строк в структуре кортежной строки мы храним адрес следующего кортежа, который хранит её продолжение. Из за этого мы имеем не большую внутреннюю фрагментацию в виде размером корежа и длинной последнего куска строки, однако размер выбран исходя из баланса между количеством ссылок в массиве указателей и фактическим размером кортежа. Недостатком данной модели является переписывание ссылок (из за обновления элемента большего размера).