Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Низкоуровневое программирование

Лабораторная работа №1

Вариант №4

Группа: P33302

Выполнил: Варюхин И.А.

Проверил: Кореньков Ю.Д.

Санкт-Петербург

2023

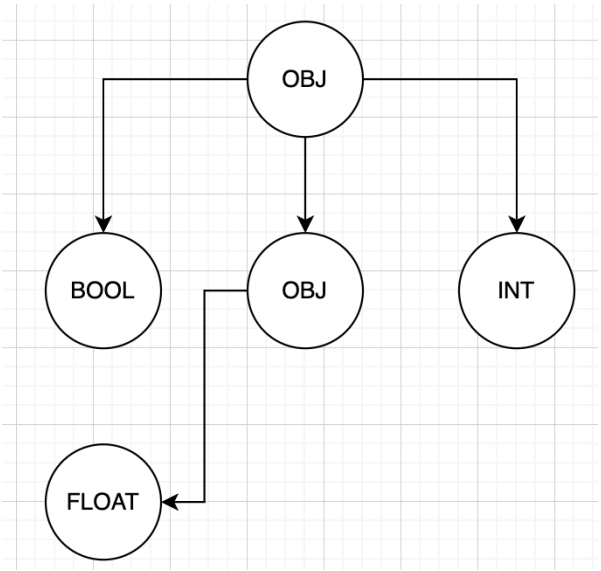
1. Задача

Основная цель лабораторной работы - Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объемом от 10GB соответствующего варианту вида:

- Спроектировать структуры данных для представления информации в оперативной памяти
- Спроектировать физическую организацию данных и архитектуру приложения
- Спроектировать прикладные интерфейсы модулей
- Сделать имплементацию интерфейсов посредством TDD
- Покрыть реализацию модульными тестами
- Написать стресс тесты для приложения на полученном прикладном интерфейсе

2. Описание работы

Данные хранятся как деревья узлов. Каждый узел содержит свой ключ и значение, которое является одним из 4 примитивов. Объекты тоже являются узлами, примитивных значений они не хранят, только ссылку на узел ребенка.



Также есть запросы. Запросы состоят из частичек запросов, которые содержат путь и условие выборки. Таким образом запросы могут быть составными.

3. Аспекты реализации

Фактическая реализация базы данных

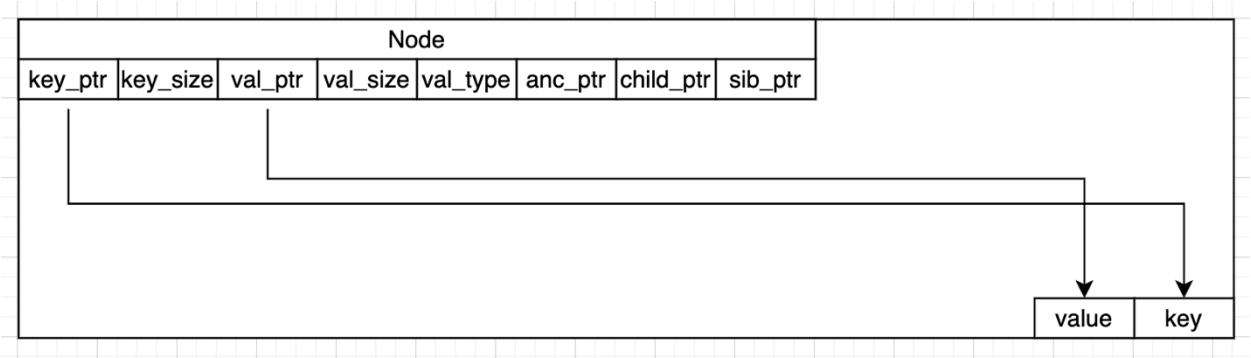
Файл для размещения данных использует секции, секции представляют из себя куски файла размером 16384 байта.

У каждой секции есть header, который хранит метаинформацию для обслуживания.

```
struct section_region {
    struct section_header *header;
    struct section_region *next;
};

struct section_header {
    size_t free_space;
    section_off first_free_cell;
    section_off last_free_cell;
    file_off section_addr; // offset from file start
};
```

Размещение сущностей в секциях:



Далее мы работаем с этими секциями через file.
У него тоже есть свой header, который хранит метаинформацию.

```
struct file {
    struct file_header *header;
    struct sections_heap *sections;
};


struct file_header {
    int file_desc;
    size_t file_size;
    file_off root_object_addr;    // the first object without ancestor
    file_off last_root_layer_addr;// the last object without ancestor
};
```

Для хранения указателей на секции организована куча (sections_heap), что обеспечивает возможность вставки элемента за константное время.

Sections API

```
enum status section_add_entity(struct entity *entity, struct string *key, union
raw_value *val, file_off *file_off_addr, struct file *file);

enum status section_write_sib_ptr(file_off prev_sibling_ptr, file_off
sibling_ptr, struct file *file);

enum status section_write_child_ptr(file_off ancestor_ptr, file_off child_ptr,
struct file *file);

struct entity section_find_entity(struct file *file, file_off addr);

struct string section_get_entity_key_by_ptr(struct file *file, file_off addr);

struct entity section_find_entity_with_values(struct file *file, file_off
entity_addr, struct string **key, union raw_value *value);

void section_update_entity_value(struct file *file, file_off entity_addr, union
raw_value value, enum value_type type);

void section_delete_entity(struct file *file, file_off entity_addr);
```

File API

```
enum status file_add_entity(struct object *obj, struct file *file, file_off
ancestor_ptr, struct object *prev_sibling, bool isChild);

enum status file_read(struct file *file, const struct query *query, struct
query_result *result);

file_off file_find_siblings_on_layer_by_key(struct file *file, struct string
*key, file_off first_sibling);

enum status file_find_obj_addr(struct file *file, const struct query *query,
file_off *addr);

enum status file_update_obj_value(struct file *file, file_off addr, union
raw_value value, enum value_type type);

enum status file_delete_object(struct file *file, file_off obj_addr);
```

User API

```
enum status user_insert_object(struct file *file, struct object *obj);

enum status user_read_object_property(struct file *file, struct string *path,
struct query_result *res);

enum status user_modify_object_property(struct file *file, struct string *path,
union raw_value value, enum query_value_type type);

enum status user_delete_object(struct file *file, struct string *path);
```

4. Результаты

Разработка велась по технике TDD, исходники представлены по ссылке:
https://github.com/vuhtang/low-level-programming

5. Выводы

Асимптотики основных операций:

Write

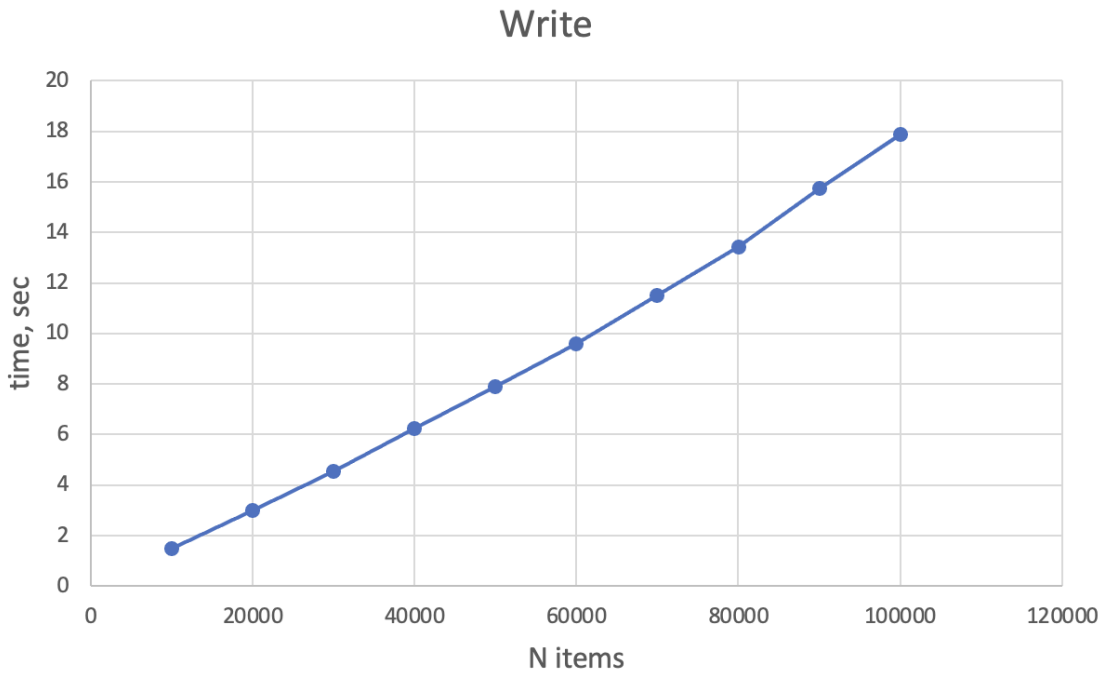


График зависимости времени вставки от количества элементов вставки. Время замерялось через каждые 10000 элементов. Как видно из графика, на каждом интервале зависимость линейная. Таким образом я доказываю, что время выполнения операции вставки зависит только от количества вставляемых данных и не зависит от количества данных в файле.

Read

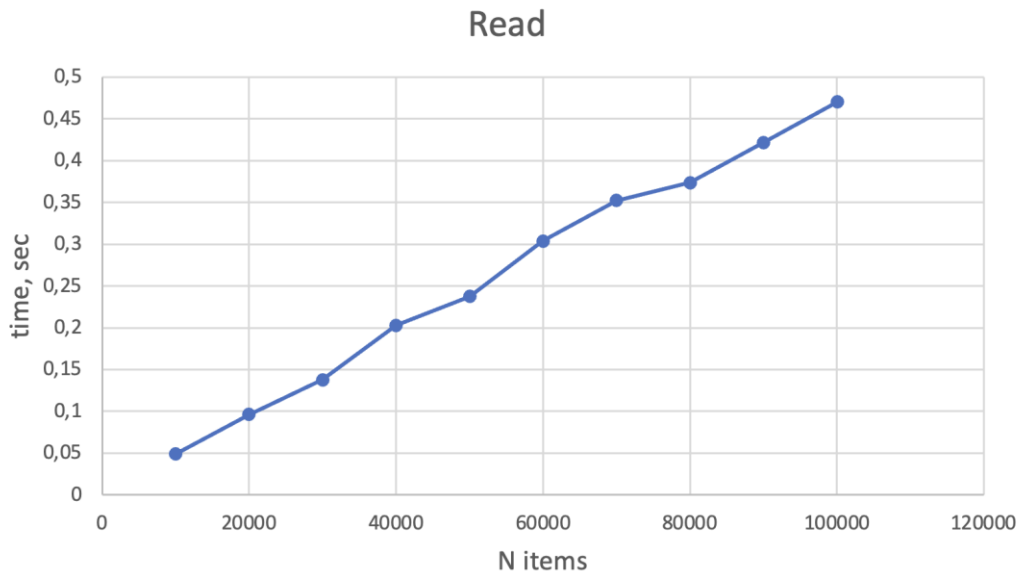


График зависимости худшего времени поиска элемента (без учета отношений) от количества элементов, представленных в файле. Из графика видно, что зависимость линейная.

Update

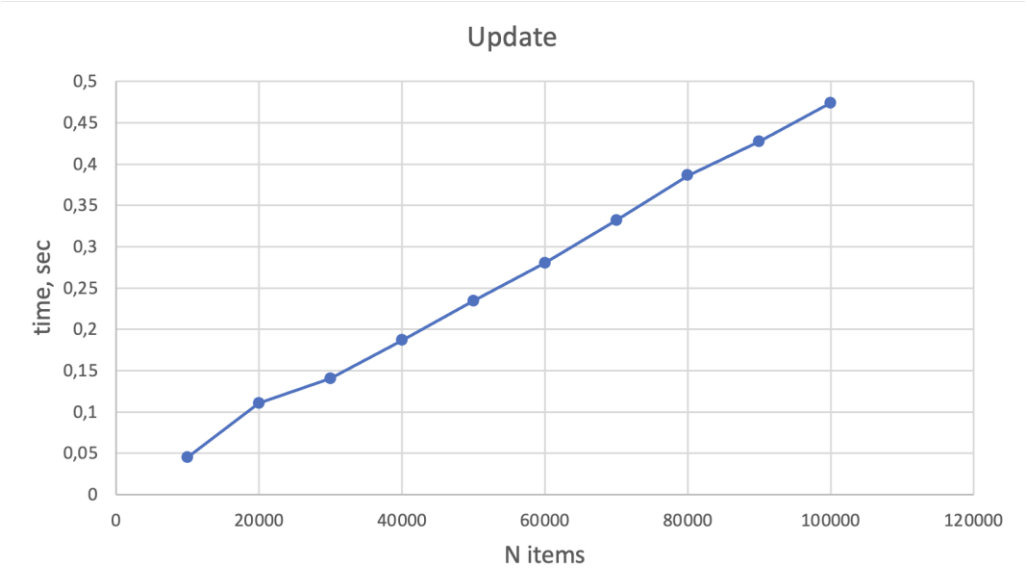


График зависимости худшего времени выполнения операции обновления элемента от количества элементов, представленных в файле. Во время каждой операции при замере выполнялось обновление элемента одной глубины (M). Из графика видно, что зависимость линейная. Таким образом я доказываю, что обновление элемента данных выполняется не более чем за O(N+M).

Delete

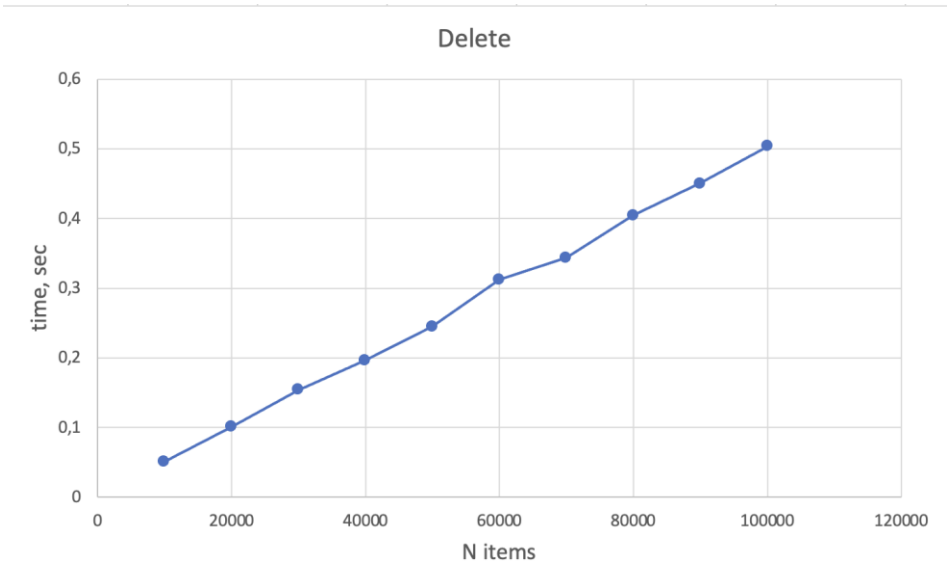
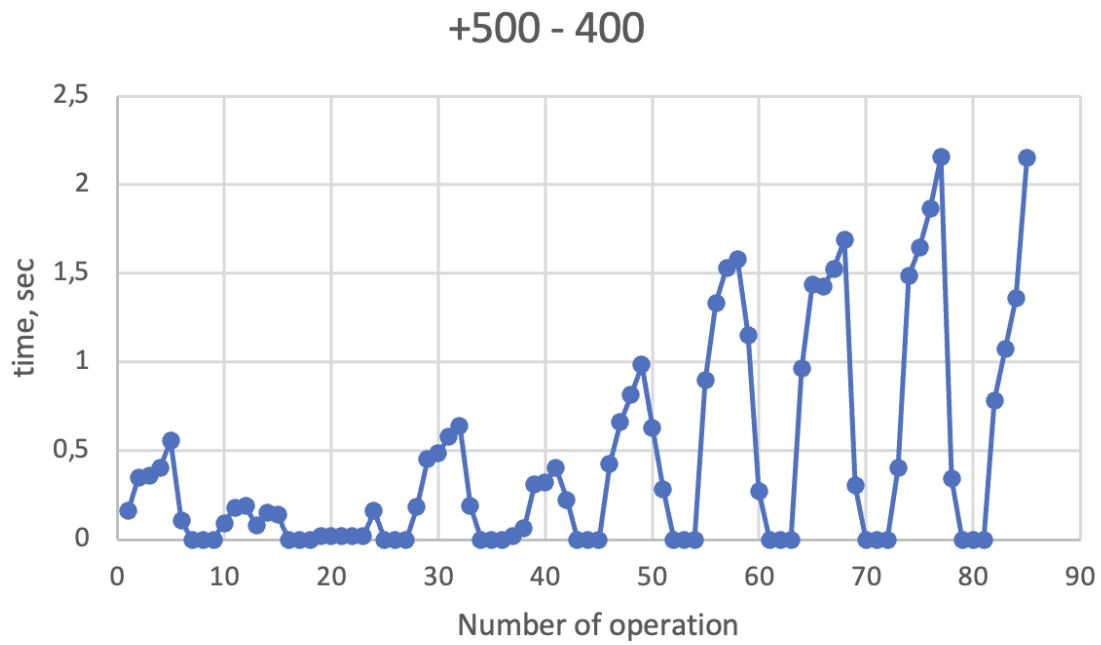


График зависимости худшего времени выполнения операции удаления элемента от количества элементов, представленных в файле. Во время каждой операции при замере выполнялось удаление элемента одной глубины (M). Из графика видно, что зависимость линейная. Таким образом я доказываю, что удаление элемента данных выполняется не более чем за O(N+M).

Mega Test



Тест заключается в повторении следующих операций: добавления 500 сущностей (глубина атрибутов - 3 уровня) в случайное место дерева базы данных, то есть как ребенка случайного атрибута любой сущности в базе, затем удаления случайного элемента (глубина атрибута – 3) из базы данных. Как и ожидалось, вставка элементов происходит за константное время, независимо от места вставки – это видно по группе точек, образующих линию у нуля. Удаление элементов также ожидаемо происходит за время, прямо пропорциональное количеству элементов в базе, в силу поиска объекта удаления.