

федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 1
по дисциплине «Низкоуровневое программирование»

Выполнил:
Студент группы Р33302
Русинов Д. С.
Преподаватель:
Кореньков Юрий Дмитриевич

Санкт-Петербург
2023

Содержание

1	Цель	2
2	Задачи	2
3	Описание работы	2
3.1	Публичный интерфейс взаимодействия с базой данных	2
3.2	Описание разработанного модуля	4
4	Амортизированные показатели ресурсоемкости	5
5	Вывод	7

1 Цель

Выданный вариант - 5 (реляционные таблицы, отображение файла)

Целью является создать модуль, реализующий хранение в одном файле данных информации общим объёмом от 10GB в виде реляционных таблиц.

2 Задачи

Для достижения поставленной цели я выделил для себя следующие задачи:

- Реализовать аллокатор для выделения памяти в файле на основе запросов
- На основе аллокатора реализовать базовые операции работы с данными вида реляционных таблиц.
- Для тестирования работоспособности модуля на разных этапах проводить тестирование.
- После завершения работы над модулем провести тестирование производительности различных операций.

3 Описание работы

3.1 Публичный интерфейс взаимодействия с базой данных

Для взаимодействия с модулем пользователю предложен следующий интерфейс:

```

enum PredicateOperator {
    PO_EQUAL,
    PO_NOT_EQUAL,
    PO_GREATER_THAN,
    PO_LESS_THAN,
};

struct OperationPredicateParameter {
    struct OperationPredicateParameter *next;
    TableColumnSchemaName column_name;
    enum PredicateOperator predicate_operator;
    uint64_t value_size;
    void *value;
};

struct SelectResultIterator
operation_select(char *table_name, struct OperationPredicateParameter *parameters);

struct SelectResultIterator {
    bool has_element;
    bool has_more;
    uint64_t current_element_offset;
    uint64_t table_metadata_offset;
    struct OperationPredicateParameter *parameters;
};

struct TableField *get_by_iterator(struct SelectResultIterator *iterator);
|
struct SelectResultIterator get_next(struct SelectResultIterator *iterator);

int operation_insert(char *table_name, struct TableField *first_table_field);

int operation_truncate(char *table_name);

int operation_delete(char *table_name, struct OperationPredicateParameter *parameters);

int operation_update(char *table_name,
                    struct OperationPredicateParameter *parameters,
                    struct TableField *first_table_field);

```

```

int init_db(const char *filename);

int close_db();

int delete_db_file(const char *filename);

```

```

#define MAX_TABLE_NAME_LENGTH 255
#define MAX_TABLE_COLUMN_NAME_LENGTH 255

typedef char TableSchemaName[MAX_TABLE_NAME_LENGTH];
typedef char TableColumnSchemaName[MAX_TABLE_COLUMN_NAME_LENGTH];

enum TableDatatype {
    TD_INT64,
    TD_FLOAT64,
    TD_STRING,
    TD_BOOL,
};

struct TableField {
    uint64_t size;
    struct TableField *next;
    void *value;
};

void free_table_row(struct TableField *table_field);

void free_table_row_without_values(struct TableField *table_field);

struct TableField *create_table_row(uint64_t first_field_size, void *first_field_value, ...);

struct TableColumn {
    TableColumnSchemaName name;
    enum TableDatatype type;
};

```

```

int operation_create_table(char *table_name, struct TableColumn *columns, uint64_t columns_count);

int operation_inner_join(char *left_table_name,
    struct OperationPredicateParameter *left_table_parameters,
    char *left_table_column_name,
    char *right_table_name,
    struct OperationPredicateParameter *right_table_parameters,
    char *right_table_column_name,
    char *result_table_name);

int operation_drop_table(char *table_name);

```

3.2 Описание разработанного модуля

Для хранения данных в файле была взята за основу следующая идея:

- Хранить в файле связный список из Элементов
- Элемент - это набор данных о содержании таблицы (строка таблицы) или набор данных, описывающий схему таблицы и хранящий ее метаданные.
- Помимо двух представленных выше типов, элемент также может быть помечен как удален
- Элементы ссылаются не только на непосредственных соседей по расположению в файле, но и на соседей по типу.

Таким образом, создание таблицы это создание Элемента типа Метадаты таблицы. Добавление строки в таблицу это создание Элемента типа Данные таблицы и заполнение соответствующего Элемента-Метаданных таблицы. Остальные

операции сводятся к итерации по Элементам на основе ссылок на их непосредственных соседей и соседей по типу.

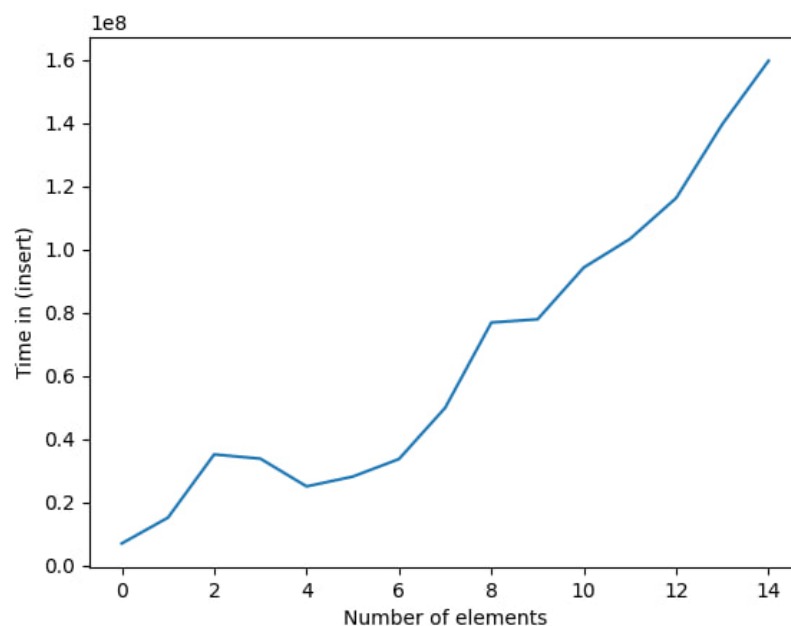
Для выделения памяти в файле была создана отдельная абстракция, аллокатор:

```
int allocate_element(uint64_t requested_element_size, enum ElementType element_type, uint64_t *element_offset);  
int delete_element(uint64_t element_offset);
```

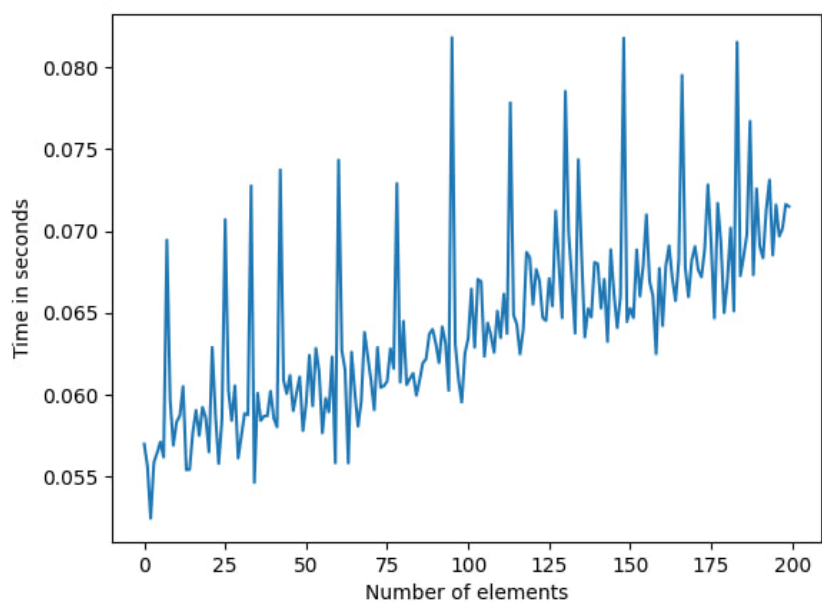
Помимо выделения памяти, аллокатор также может и удалять Элементы. Для этого он помечает их как удаленные, а также сливает непосредственных соседей-удаленных элементов, чтобы в последующем переиспользовать эту память эффективно.

4 Амортизированные показатели ресурсоемкости

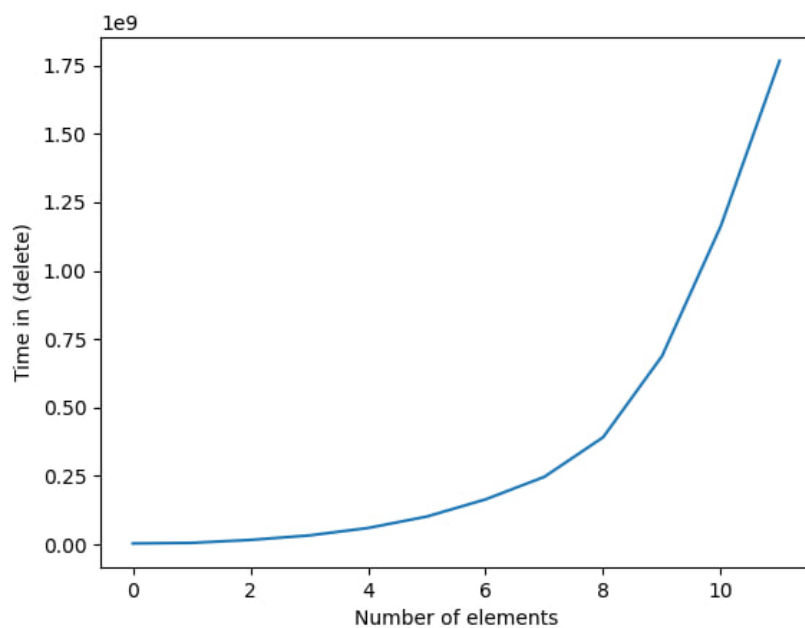
Операция вставки



Операция выборки



Операция обновления/удаления



Также был проведен опыт с повторяющимся запуском программы, которая обновляет существующие данные (обновление сводится к удалению старых и вставке новых), чтобы показать, что занимаемая программой память при этом не увеличивается.

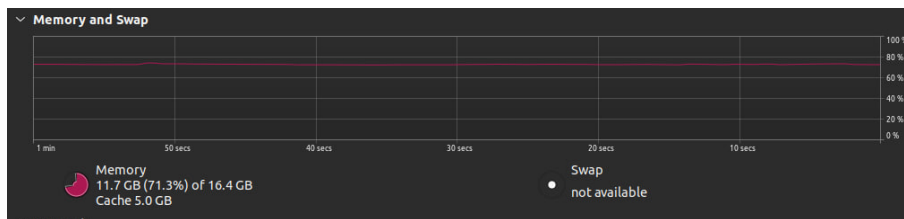
Здесь, перед каждым вызовом ls была запущена эта программа.

```

ruskaof@ruska-zenbook:~/Desktop$ ls -la
total 1044
drwxr-xr-x  4 ruskaof ruskaof   4096 Oct 11 01:51 .
drwxr-x--- 54 ruskaof ruskaof   4096 Oct 10 19:08 ..
drwxrwxr-x  2 ruskaof ruskaof   4096 Jun 18 00:19 .ipynb_checkpoints
drwxrwxr-x 10 ruskaof ruskaof   4096 Oct 11 01:22 llp_database
-rw-----  1 ruskaof ruskaof 1179865 Oct 11 01:51 testdb2
ruskaof@ruska-zenbook:~/Desktop$ ls -la
total 2068
drwxr-xr-x  4 ruskaof ruskaof   4096 Oct 11 01:51 .
drwxr-x--- 54 ruskaof ruskaof   4096 Oct 10 19:08 ..
drwxrwxr-x  2 ruskaof ruskaof   4096 Jun 18 00:19 .ipynb_checkpoints
drwxrwxr-x 10 ruskaof ruskaof   4096 Oct 11 01:22 llp_database
-rw-----  1 ruskaof ruskaof 2294194 Oct 11 01:51 testdb2
ruskaof@ruska-zenbook:~/Desktop$ ls -la
total 2068
drwxr-xr-x  4 ruskaof ruskaof   4096 Oct 11 01:51 .
drwxr-x--- 54 ruskaof ruskaof   4096 Oct 10 19:08 ..
drwxrwxr-x  2 ruskaof ruskaof   4096 Jun 18 00:19 .ipynb_checkpoints
drwxrwxr-x 10 ruskaof ruskaof   4096 Oct 11 01:22 llp_database
-rw-----  1 ruskaof ruskaof 2294194 Oct 11 01:51 testdb2
ruskaof@ruska-zenbook:~/Desktop$ ls -la
total 2068
drwxr-xr-x  4 ruskaof ruskaof   4096 Oct 11 01:51 .
drwxr-x--- 54 ruskaof ruskaof   4096 Oct 10 19:08 ..
drwxrwxr-x  2 ruskaof ruskaof   4096 Jun 18 00:19 .ipynb_checkpoints
drwxrwxr-x 10 ruskaof ruskaof   4096 Oct 11 01:22 llp_database
-rw-----  1 ruskaof ruskaof 2294194 Oct 11 01:51 testdb2
ruskaof@ruska-zenbook:~/Desktop$

```

Также была запущен скрипт, вызывающий программу на вставку на все большее и большее количество элементов с каждой итерацией, а во время него было изучено состояние оперативной памяти системы: оно не расло с ростом затрагиваемых программой элементов.



5 Вывод

Тесты показали, что операция вставки имеет сложность $O(1)$, операция выборки - за время, не большее $O(n)$, операция удаления/обновления за $t < O(n * m) \rightarrow O(n + m)$