Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

«Низкоуровневое программирование»

Вариант 2

Выполнил:
Студент группы P33102
Гулямов Т.И.

Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург
2023

Цель:

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

Задачи:

1. Спроектировать структуры данных для представления информации в оперативной памяти
    a. Для порции данных, состоящий из элементов определённого рода (см форму данных), поддержать тривиальные значения по меньшей мере следующих типов: четырехбайтные целые числа и числа с плавающей точкой, текстовые строки произвольной длины, булевские значения
    b. Для информации о запросе
2. Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним:
    a. Операции над схемой данных (создание и удаление элементов схемы)
    b. Базовые операции над элементами данных в соответствии с текущим состоянием схемы (над узлами или записями заданного вида)
        i. Вставка элемента данных
        ii. Перечисление элементов данных
        iii. Обновление элемента данных
        iv. Удаление элемента данных
3. Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных:
    a. Добавление, удаление и получение информации о элементах схемы данных, размещаемых в файле данных, на уровне, соответствующем виду узлов или записей
    b. Добавление нового элемента данных определенного вида
    c. Выборка набора элементов данных с учётом заданных условий и отношений со смежными элементами данных (по свойствам/полями/атрибутам и логическим связям соответственно)
    d. Обновление элементов данных, соответствующих заданным условиям
    e. Удаление элементов данных, соответствующих заданным условиям
4. Реализовать тестовую программу для демонстрации работоспособности решения
    a. Параметры для всех операций задаются посредством формирования соответствующих структур данных
    b. Показать, что при выполнении операций, результат выполнения которых не отражает отношения между элементами данных, потребление оперативной памяти стремится к O(1) независимо от общего объема фактического затрагиваемых данных
    c. Показать, что операция вставки выполняется за O(1) независимо от размера данных, представленных в файле
    d. Показать, что операция выборки без учета отношений (но с опциональными условиями) выполняется за O(n), где n – количество представленных элементов данных выбранного вида

    e. Показать, что операции обновления и удаления элемента данных выполняются не более чем за O(n*m) > t O(n+m), где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных
    f. Показать, что размер файла размещенных элементов данных
    g. Показать работоспособность решения под управлением ОС семейств Windows и *NIX
5. Результаты тестирования по п.4 представить в составе отчета, при этом:
    a. В части 3 привести описание структур данных, разработанных в соответствии с п.1
    b. В части 4 описать решение, реализованное в соответствии с пп.2-3
    c. В часть 5 включить графики на основе тестов, демонстрирующие амортизированные показатели ресурсоемкости по п. 4

Описание работы:

Сборка и запуск:

```
git clone git@github.com:tplaymeow/itmo-low-level-programming-lab1.git
git submodule init
git submodule update
```

Для сборки и запуска необходимо использовать Cmake.

Тестовая программа

```
cmake -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build
build/database_test_app/database_test_app ...
```

Бенчмарки

```
cmake -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build
build/database_benchmarks/database_benchmarks
```

Модули:

- utils – вспомогательный модуль с небольшими вспомогательными функциями м макросами (MIN, MAX и тп.)
- logger – вспомогательный модуль с функциями логирования
- paging – модуль, отвечающий за страничную работу с файлами. Предоставляет функции для страничной записи/удаления/чтения данных.
- database – модуль, отвечающий за функциональность базы данных. Позволяет создавать таблицы, удалять таблицы (с автоматическим удалением всех связанных строк), добавлять строки в таблицу, выбирать строки из таблицы по условию, удалять строки из таблицы.
- database_test_app – тестовое приложение, демонстрирующие основные возможности созданного решения.

- database_benchmarks – бенчмарки, проверяющие время выполнения и затрачиваемое дисковое пространство основных операций.

Создание таблицы:

```
database_create_table_request_create("Table", 3);
database_create_table_request_set(
  table_request, 0,
  (struct database_attribute){.name = "ATTR1",
                              .type = DATABASE_ATTRIBUTE_INTEGER});
database_create_table_request_set(
  table_request, 1,
  (struct database_attribute){.name = "ATTR2",
                              .type = DATABASE_ATTRIBUTE_FLOATING_POINT});
database_create_table_request_set(
  table_request, 2,
  (struct database_attribute){.name = "ATTR3",
                              .type = DATABASE_ATTRIBUTE_STRING});

const struct database_create_table_result create_table_result =
    database_create_table(database, table_request);
```

Удаление таблицы:

```
const struct database_drop_table_result result =
    database_drop_table(database, table);
```

Вставка строк:

```
struct database_insert_row_request insert_request =
    database_insert_row_request_create(get_table_result.table);
database_insert_row_request_set(
    insert_request, 0, (union database_attribute_value){.integer = 1});
database_insert_row_request_set(
    insert_request, 1,
    (union database_attribute_value){.floating_point = 2.5});
database_insert_row_request_set(
    insert_request, 2, (union database_attribute_value){.string = "Hello"});

database_insert_row(database, get_table_result.table, insert_request);
```

Выбор строк (с условием):

```
const struct database_where where_first = {
    .operation = DATABASE_WHERE_OPERATION_EQUAL,
    .attribute_position = 0,
    .value = 0};
database_select_row_first(database, get_table_result.table, where_first);
```

Удаление строк:

```
database_remove_row(database, select_result.row);
```

Объединение:

```
const struct database_join join = {.left_attribute_position = 0,
                                   .right_attribute_position = 4};
struct database_select_join_result select_result =
    database_select_join_first(database, left_table, right_table,
                               left_where, right_where, join);
```

Аспекты реализации:

Файл с данными разбит на страницы равного размера. Страницы могут быть разного типа. Все типы страниц связаны в списки, указатели на начало списков лежат в заголовке файлов.

Заголовок имеют следующую структуру:

```
struct paging_file_header {
  uint64_t first_free_page_number;
  uint64_t first_page_type_1_page_number;
  uint64_t first_page_type_2_page_number;
  uint64_t first_page_type_3_page_number;
};
```

first_free_page_number – указатель на список свободных страниц
first_page_type_(n)_page_number – указатель на список страниц n-го типа

Страницы имеют следующую структуру:

```
struct paging_file_page_header {
  uint64_t next_page_number;
  uint64_t next_continuation;
};
```

next_page_number – указатель на следующую страницу этого типа
next_continuation – признак, имеет ли страница продолжение (один кусок данных может быть разбит на несколько страниц в зависимости от размера)

Выбранная структура позволяет:

- переиспользовать освобожденную память
- добавлять данные конкретного размера в начало списка за O(1) по времени
- удалять данные конкретного размера зная номер начальной страницы за O(1) по времени. Для удаления, связываем соседние страницы, а удаленную страницу добавляем в начало списка свободных страниц.
- искать данные конкретного размера и типа за O(n) по времени. Обычной итерацией по списку страниц этого типа

Результаты:

Тестирование:
Для тестирования приложения в разных конфигурациях используется GitHub Actions. Результаты.

Создан workflow, который проверяет основную функциональность приложения в следующих конфигурациях:

- Ubuntu, GCC, Release
- Ubuntu, Clang, Release
- MacOS, GCC, Release
- MacOS, Clang, Release
- Windows, CL, Release
- Ubuntu, GCC, Debug + Sinitizers
- Ubuntu, Clang, Debug + Sinitizers
- MacOS, GCC, Debug + Sinitizers
- MacOS, Clang, Debug + Sinitizers

Создан workflow, который проверяет кросс платформенность файла бд (файл создается на одной ОС и читается/изменяется на другой) в следующих конфигурациях:

- Ubuntu (GCC, Release) to MacOS (GCC, Release)
- Ubuntu (GCC, Release) to Windows (CL, Release)
- MacOS (GCC, Release) to Ubuntu (GCC, Release)
- MacOS (GCC, Release) to Windows (CL, Release)
- Windows (CL, Release) to Ubuntu (GCC, Release)
- Windows (CL, Release) to MacOS (GCC, Release)

Бенчмарки:
График зависимости времени выполнения вставки от количества данных. Время выполнения стремится к O(1).

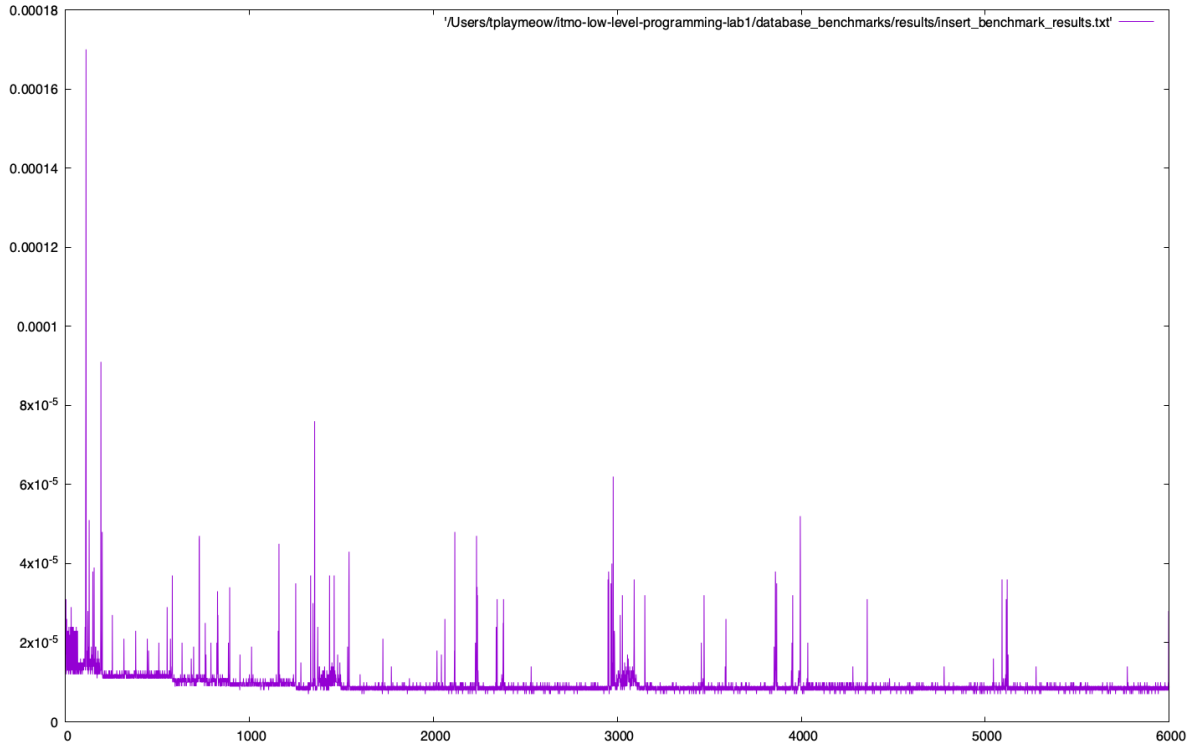


График зависимости времени выполнения выборки от количества данных. Время выполнения стремится к O(n).

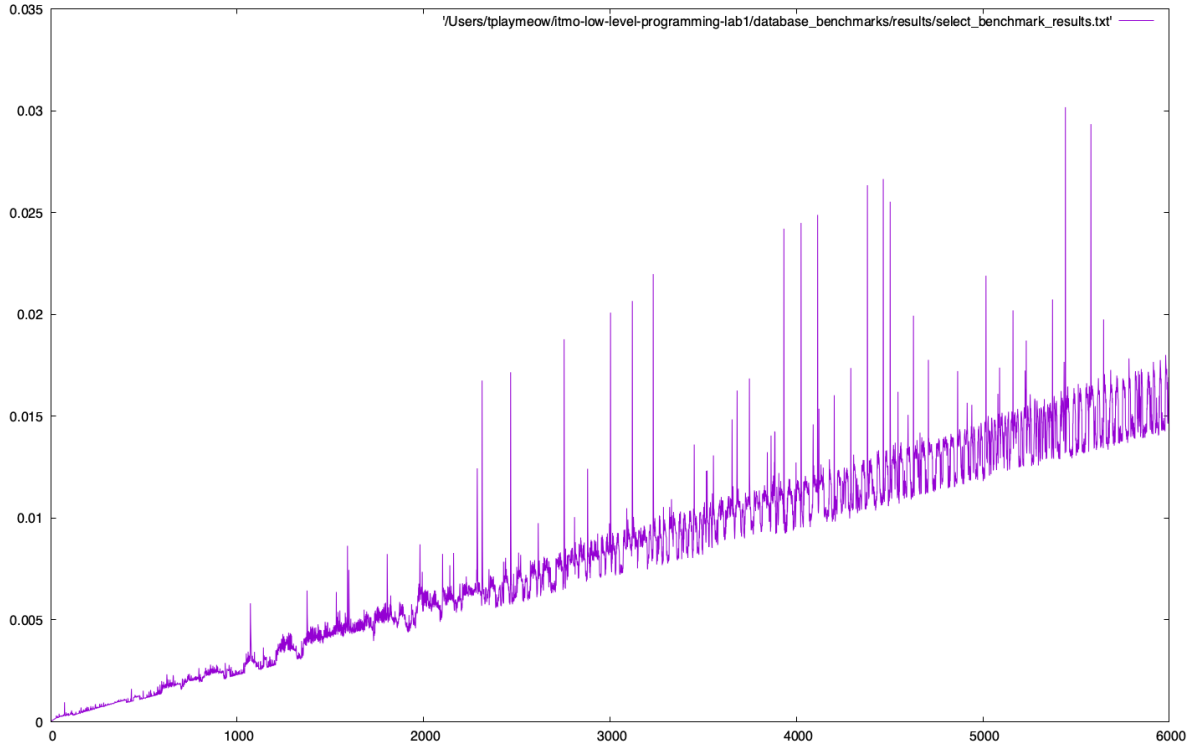


График зависимости времени выполнения удаления от количества данных. Время выполнения стремится к O(1).

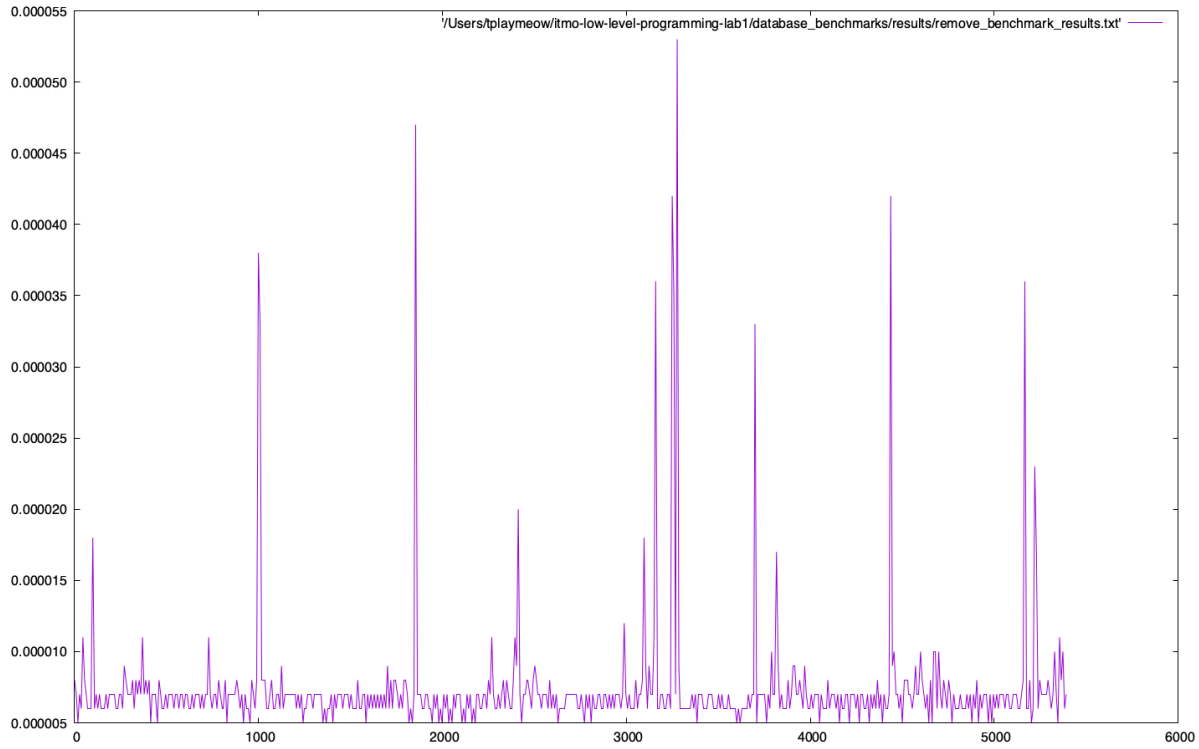
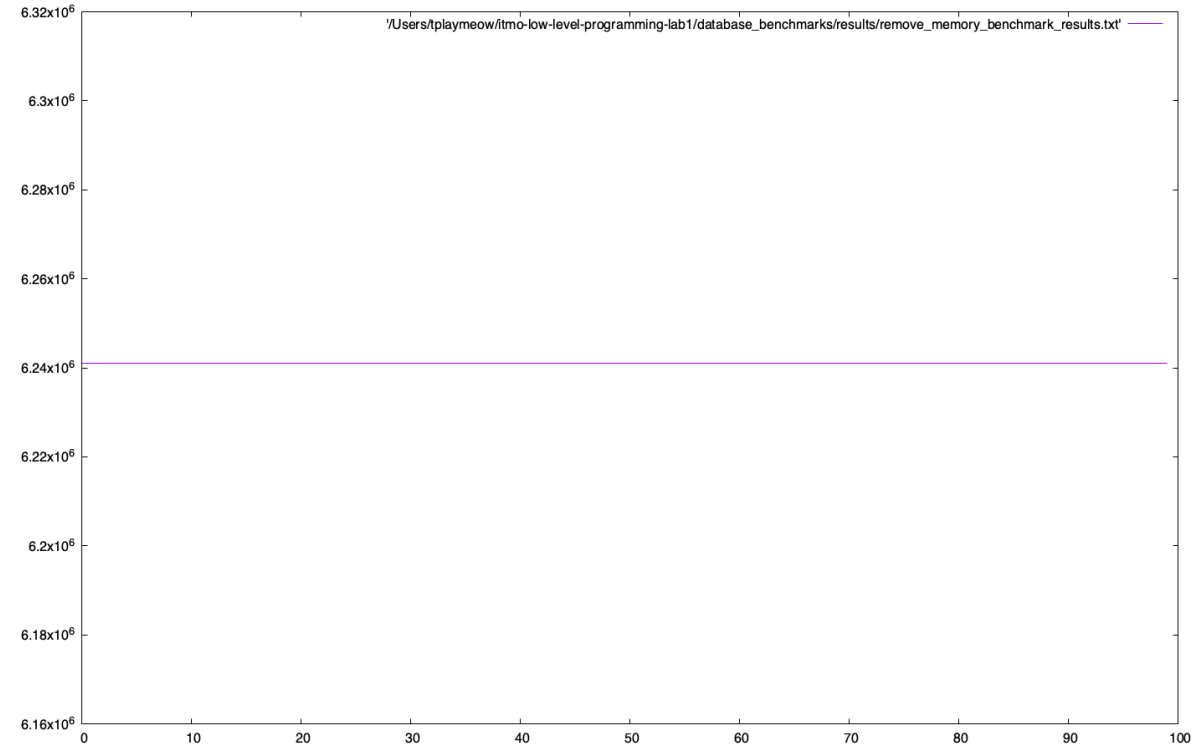


График изменения количества памяти. При записи и удалении строк таблицы. Освободившиеся память переиспользуется.



Результаты:

Во время выполнения работы, реализовал структурированное хранение данных в файле, поддерживающее эффективную вставку, удаление, выборку, попрактиковался в создание кросс платформенных приложений на языке C, попрактиковался в создание

CI для тестирования CMake приложений. Получил бесценный опыт, который, несомненно, пригодится мне в будущем!