Университет ИТМО
Факультет программной инженерии и вычислительной техники

Лабораторная работа №1 по дисциплине «Низкоуровневое программирование»

Вариант №3, Граф

Выполнил:
Ерехинский Андрей Владимирович
Студент группы P33312

Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург
2023

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

Порядок выполнения:

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: четырёхбайтовые 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 об элементах схемы данных, размещаемых в файле данных, на уровне, соответствующем виду узлов или записей
   b. Добавление нового элемента данных определённого 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 объёма 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 учёта отношений (но с опциональными условиями) выполняется за O(n), где n – количество представленных элементов данных выбираемого 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 данных всегда пропорционален количеству фактически размещённых элементов данных

g. Показать работоспособность решения под управлением ОС семейств Windows и *NIX

5. Результаты тестирования по п.4 представить в составе отчёта, 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 ресурсоёмкости по п. 4

Описание:

Схема:

```
typedef struct db_scheme {
    node_scheme *first_node_scheme;
    node_scheme *last_node_scheme;
} db_scheme;
```

Узел схемы:

```
typedef struct node_scheme {
    char *type_in_string;
    int root_offset;
    int first_offset;
    int last_offset;
    char *buffer;
    int buffer_count;
    int new_node_flag;
    int prev_offset;
    int this_offset;
    node_link *link_to_first;
    node_link *link_to_last;
    struct attribute_entry *attribute_first;
    struct attribute_entry *attribute_last;
    struct node_scheme *next_node_scheme;
} node_scheme;
```

Атрибут:

```
typedef struct attribute_entry {
    char *string_name;
    unsigned char type;
    struct attribute_entry *next;
} attribute_entry;
```

БД представляет схему, в которой находятся узлы разных типов. У узлов есть именованные атрибуты и связи с другими узлами. Структура данных для связи между собой узлов, атрибутов: связный список.

Показательный пример:

Создаем схему с двумя типами узлов: email и message

```
Sender:dron12301230@yandex.ru Recipient:IS@mail.ru length:50
Sender:IS@mail.ru Recipient:BA@ya.ru length:51
Sender:BA@ya.ru Recipient:GA@ya.ru length:52
Sender:GA@ya.ru Recipient:TV@ya.ru length:53
Sender:TV@ya.ru Recipient:KV@ya.ru length:54
Sender:KV@ya.ru Recipient:IK@ya.ru length:55
Sender:IK@ya.ru Recipient:MP@ya.ru length:56
Sender:MP@ya.ru Recipient:MD@ya.ru length:57
Sender:MD@ya.ru Recipient:IT@ya.ru length:58
Sender:IT@ya.ru Recipient:dron12301230@yandex.ru length:59
Name:Andrey Surname:Erekhinsky Email_addres:dron12301230@yandex.ru
Name:Sonya Surname:Inglikova Email_addres:IS@mail.ru
Name:Sasha Surname:Bogatov Email_addres:BA@ya.ru
Name:Anvar Surname:Gazizov Email_addres:GA@ya.ru
Name:Veronica Surname:Troynikova Email_addres:TV@ya.ru
Name:Vitaliy Surname:Kiyko Email_addres:KV@ya.ru
Name:Kirill Surname:Ievlev Email_addres:IK@ya.ru
Name:Polina Surname:Morgunova Email_addres:MP@ya.ru
Name:Danya Surname:Menkov Email_addres:MD@ya.ru
Name:Tanya Surname:Ivanova Email_addres:IT@ya.ru
```

Первый запрос select:

```
MATCH (e:Email)-[:SEND]->(m:Message) WHERE (e.Name != Sonya) AND (e.Surname != Bogatov)  AND (m.Length < 55) RETURN e;
Name:Andrey Surname:Erekhinsky Email_addres:dron12301230@yandex.ru
Name:Anvar Surname:Gazizov Email_addres:GA@ya.ru
Name:Veronica Surname:Troynikova Email_addres:TV@ya.ru
```

Второй запрос update:

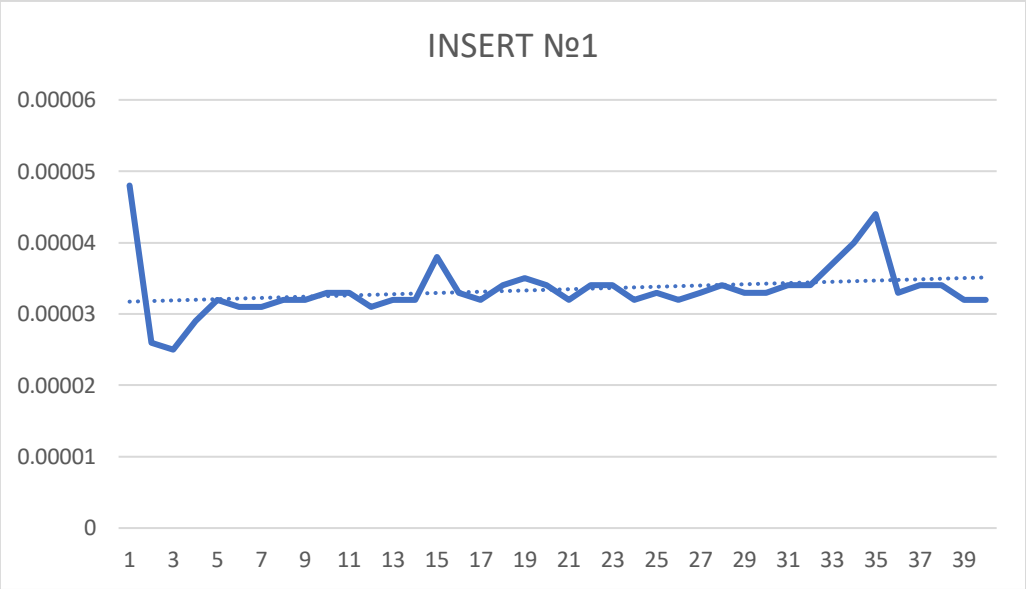
```
MATCH (e:Email)-[:SEND]->(m:Message) WHERE (e.Name != Sonya) AND (e.Surname != Bogatov) SET m.Surname=Erekhinsky RETURN
e;

MATCH (e:Email)-[:SEND]->(m:Message) WHERE (e.Name != Sonya) AND (e.Surname != Bogatov) RETURN e;
Name:Andrey Surname:Erekhinsky Email_addres:dron12301230@yandex.ru
Name:Anvar Surname:Erekhinsky Email_addres:GA@ya.ru
Name:Veronica Surname:Erekhinsky Email_addres:TV@ya.ru
Name:Vitaliy Surname:Erekhinsky Email_addres:KV@ya.ru
Name:Kirill Surname:Erekhinsky Email_addres:IK@ya.ru
Name:Polina Surname:Erekhinsky Email_addres:MP@ya.ru
Name:Danya Surname:Erekhinsky Email_addres:MD@ya.ru
Name:Tanya Surname:Erekhinsky Email_addres:IT@ya.ru
```

Третий запрос delete (после запроса вывод оставшихся email’ов):

```
MATCH (e:Email)-[:SEND]->(m:Message) WHERE (e.Surname == Erekhinsky) DELETE e;
Name:Sonya Surname:Inglikova Email_addres:IS@mail.ru
Name:Sasha Surname:Bogatov Email_addres:BA@ya.ru
```

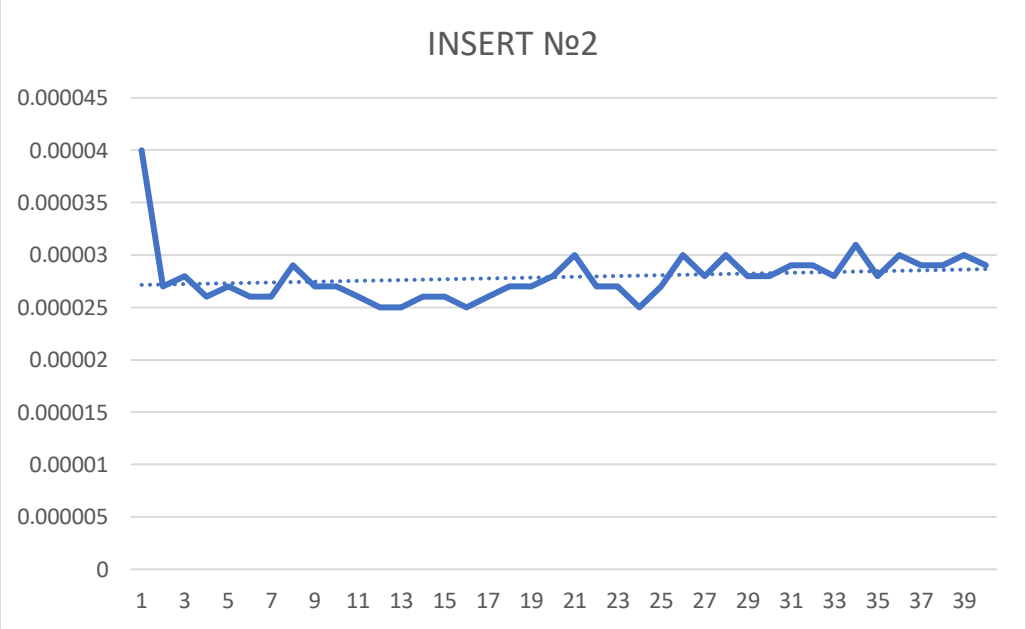
Измерения:

Вновь добавленные данные добавляются к старым. На оси Ox представлен номер итерации, в каждой итерации добавляется 40 узлов.



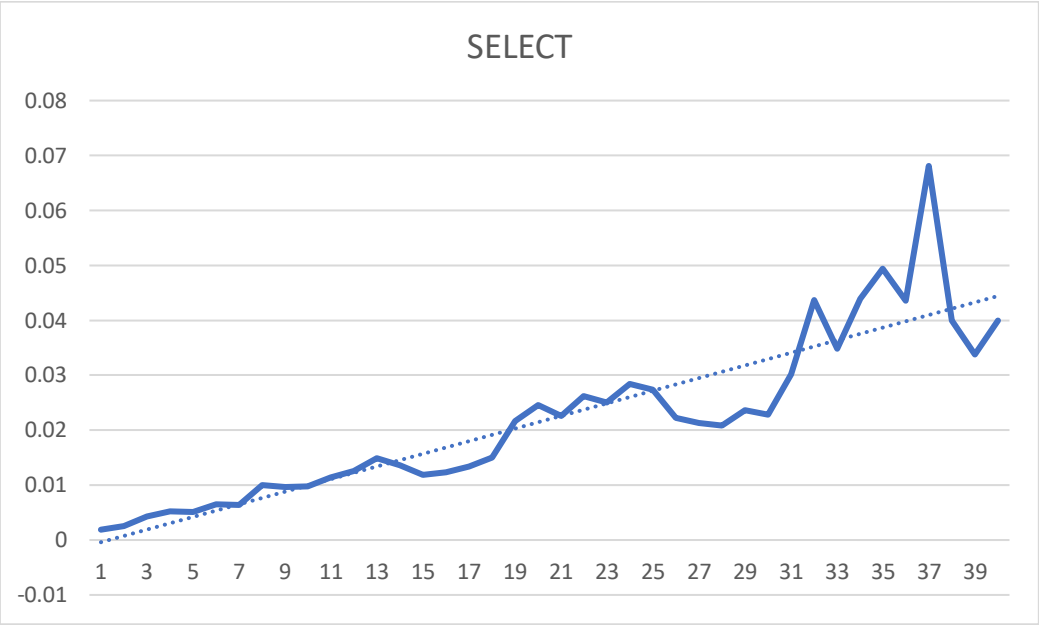
Аппроксимирующая указывает на то, что время вставки O(1).

Также был произведен второй замер insert’ов для вставки другого типа узлов:

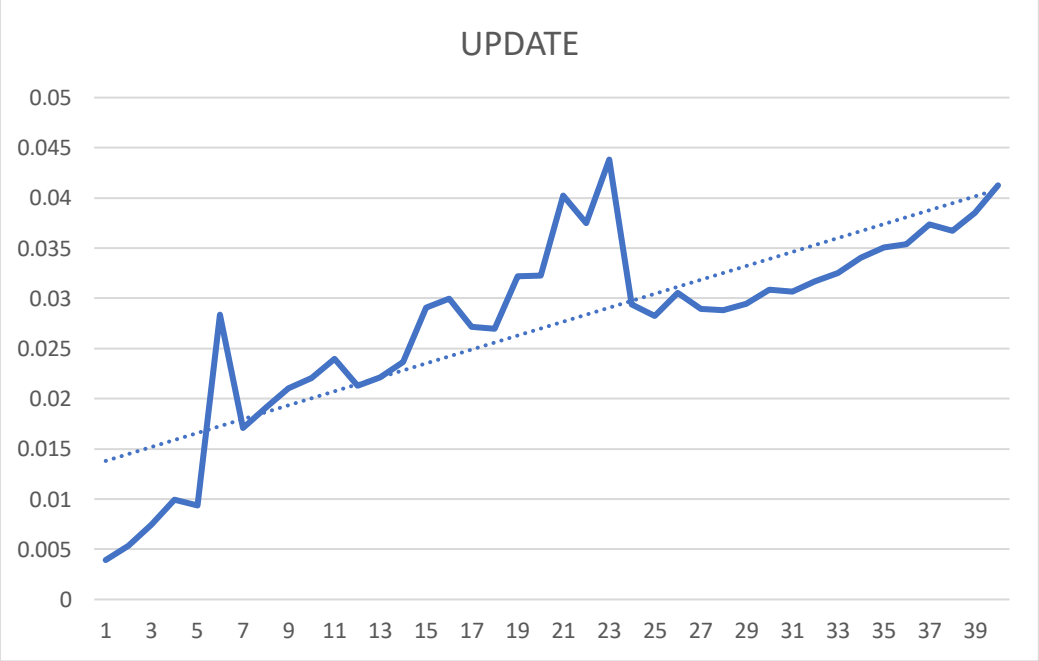


Аппроксимирующая также указывает на то, что время вставки O(1).

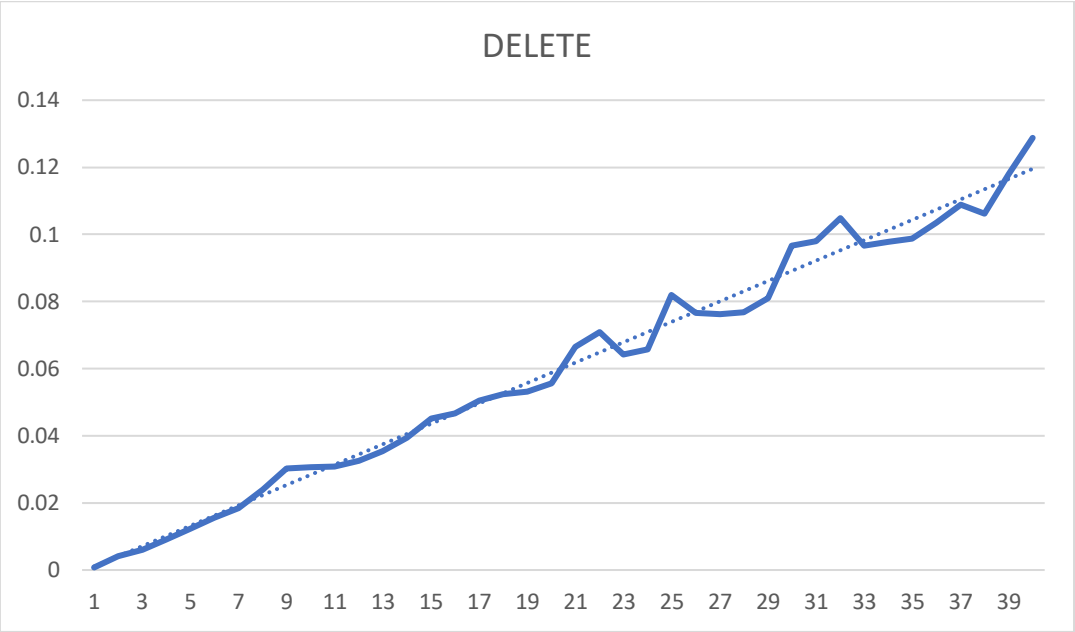
В запросах select, update, delete с каждой новой итерацией добавляется 40 узлов, с которыми работает запрос. Это показывает на то, что с каждым новым запросом в цикле, запрос работает с 40*i узлами.



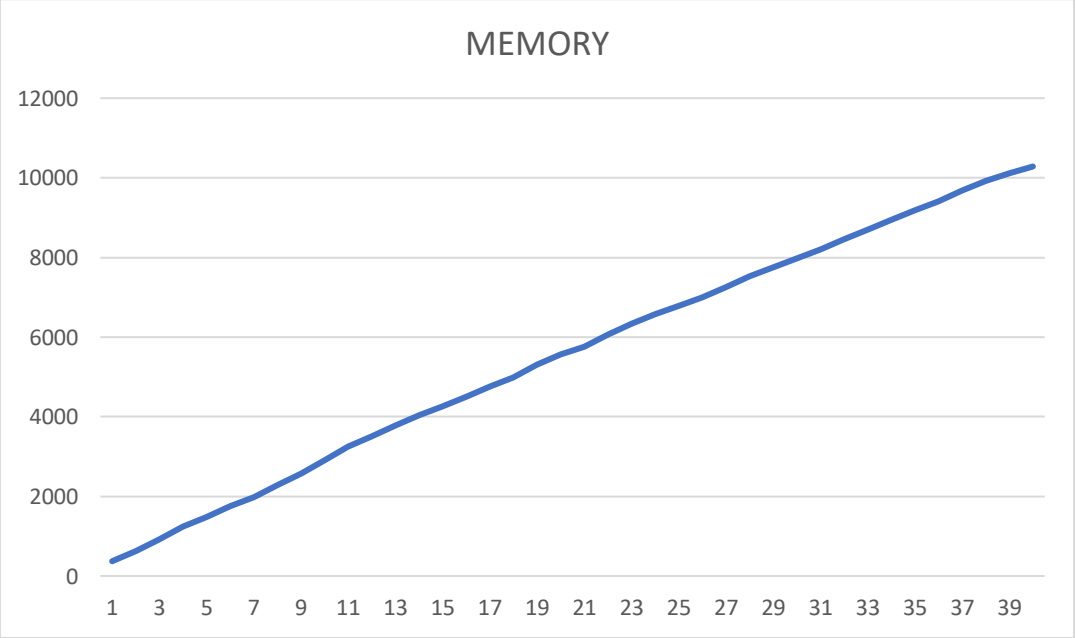
Аппроксимирующая указывает на то, что время выборки примерно O(n).



Аппроксимирующая указывает на то, что время обновления примерно O(n).



Аппроксимирующая указывает на то, что время удаления примерно O(n).



На графике видно, что с увеличением количества узлов в БД, память увеличивается пропорционально (O(n)).

Вывод: В ходе выполнения работы был реализован модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB. Модуль может работать с UNIX Windows, так как в нем не использовались функции ОС.