Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Низкоуровневое программирование»

Вариант № 1

Выполнил:

Студент группы P33301

Акимов Роман Иванович

Преподаватель:

Кореньков Юрий Дмитриевич

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 о 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 $O(n*m) > t \rightarrow O(n+m)$, где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных

f. Показать, что размер файла данных всегда пропорционален 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

Вариант

Форма данных – Документное дерево
Способ работы с файлом – Чтение-запись
Базовый язык запросов – XPath
Формат транспортного протокола – Xml
UI API - 1

Описание работы

Include – заголовочные файлы

src\utils – crud.c интерфейс для взаимодействия с данными

src\utils – data_manager.c & low_data_manager.c отвечают за манипуляцию с данными

src\utils – file_manager.c работа с файлом (data.txt)

src\utils – wrapper.c нужен для замеров времени выполнения операций

Пример работы программы

Linux

```
################################ FIND BY PARENT ID 1 ################################
{
   id - 4
   parent - 1
   name - new
   age = 22
   height = 170.000
   healthy = 0
}
################################ REMOVE TUPLE BY ID 4 ################################
{
   id = 1
   parent = 0
   name = roma
   age = 21
   height = 177.000
   healthy = 1
}
{
   id = 2
   parent = 0
   name = sanya
   age = 22
   height = 166.000
   healthy = 1
}
{
   id = 3
   parent = 0
   name = igor
   age = 20
   height = 220.000
   healthy = 0
}
```

```
File header - {
   Current ID: 4
   Fields:
      Type 0 - name
      Type 1 - age
      Type 3 - height
      Type 2 - healthy
}
{
   id = 1
   parent = 0
   name = roma
   age = 21
   height = 177.000
   healthy = 1
}
{
   id = 2
   parent = 0
   name = sanya
   age = 22
   height = 166.000
   healthy = 1
}
{
   id = 3
   parent = 0
   name = igor
   age = 20
   height = 220.000
   healthy = 0
}
```

```
################################ ADD TUPLE ################################
{
   id = 1
   parent = 0
   name = roma
   age = 21
   height = 177.000
   healthy = 1
}
{
   id = 2
   parent = 0
   name = sanya
   age = 22
   height = 166.000
   healthy = 1
}
{
   id = 3
   parent = 0
   name = igor
   age = 20
   height = 220.000
   healthy = 0
}
{
   id = 4
   parent = 1
   name = new
   age = 30
   height = 170.000
   healthy = 0
}
```

```
################################ FIND TUPLE BY ID 4 ################################
{
   id - 4
   parent - 1
   name - new
   age = 30
   height = 170.000
   healthy = 0
}
################################ FIND TUPLE BY FIELD INT = 21 ################################
{
   id - 1
   parent - 0
   name - roma
   age = 21
   height = 177.000
   healthy = 1
}
```

```
################################ UPDATE TUPLE'S FIELD BY ID 4 INT = 22 ################################
{
   id = 1
   parent = 0
   name = roma
   age = 21
   height = 177.000
   healthy = 1
}
{
   id = 2
   parent = 0
   name = sanya
   age = 22
   height = 166.000
   healthy = 1
}
{
   id = 3
   parent = 0
   name = igor
   age = 20
   height = 220.000
   healthy = 0
}
{
   id = 4
   parent = 1
   name = new
   age = 22
   height = 170.000
   healthy = 0
}
```

Windows

```
File header - {
   Current ID: 4
   Fields:
      Type 0 - name
      Type 1 - age
      Type 3 - height
      Type 2 - healthy
}
{
   id = 1
   parent = 0
   name = roma
   age = 21
   height = 177.000
   healthy = 1
}
{
   id = 2
   parent = 0
   name = sanya
   age = 22
   height = 166.000
   healthy = 1
}
{
   id = 3
   parent = 0
   name = igor
   age = 20
   height = 220.000
   healthy = 0
}
```

Аспекты реализации

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

struct tree_subheader {
    uint64_t ASCII_signature;
    uint64_t cur_id;
    uint64_t pattern_size;
};
```

```c
union tuple_header {
    struct {
        uint64_t parent;
        uint64_t alloc;
    };
    struct {
        uint64_t prev;
        uint64_t next;
    };
};

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

В нашем файле есть tree_header, который содержит информацию о коллекции. В нем хранится текущий id, последовательность id-шников, а также название полей и их типы.

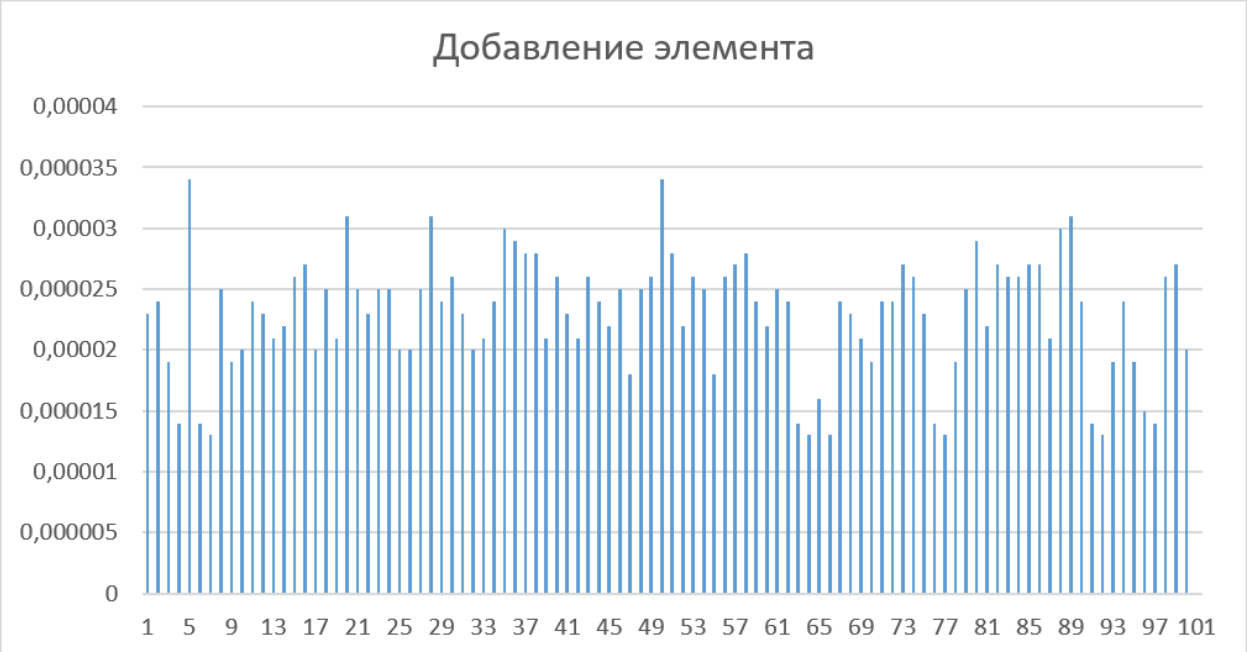
Возникает проблема со структурой tuple, а именно со значением типа string и его разноразмерностью. Мы не сможем удалить элемент из середины и на его место поставить другой с размером строки больше, чем изначальный. Чтобы это исправить, храним отдельно строку, а в самой структуре tuple мы лишь храним ссылку на неё, лежащую в файле. Остальные поля int, float и bool имеют фиксированный размер, поэтому в файле они занимают одинаковое количество места.

При удалении элемента из середины на его место встает крайний элемент, чтобы избежать фрагментации файла.

- Поиск по id – с помощью id и массива идентификаторов находим и достаем тот кортеж, который нам нужен.
- Обновление по id – находим элемент по id, считываем его, меняем нужное поле и кладем обратно. Но в случае обновления поля string нам может понадобиться обновить двухсвязный список.
- Добавление – кортеж вставляем в конец файла, а его id в конец массива идентификаторов.
- Поиск по полям – идем по массиву идентификаторов, смотрим и сравниваем каждое поле с тем, что нам нужно.
- Удаление по id – удаляем нужный кортеж, а на его место ставим последний, затем ищем всех его детей с помощью поиска по полю и точно так же удаляем. Процесс рекурсивный.

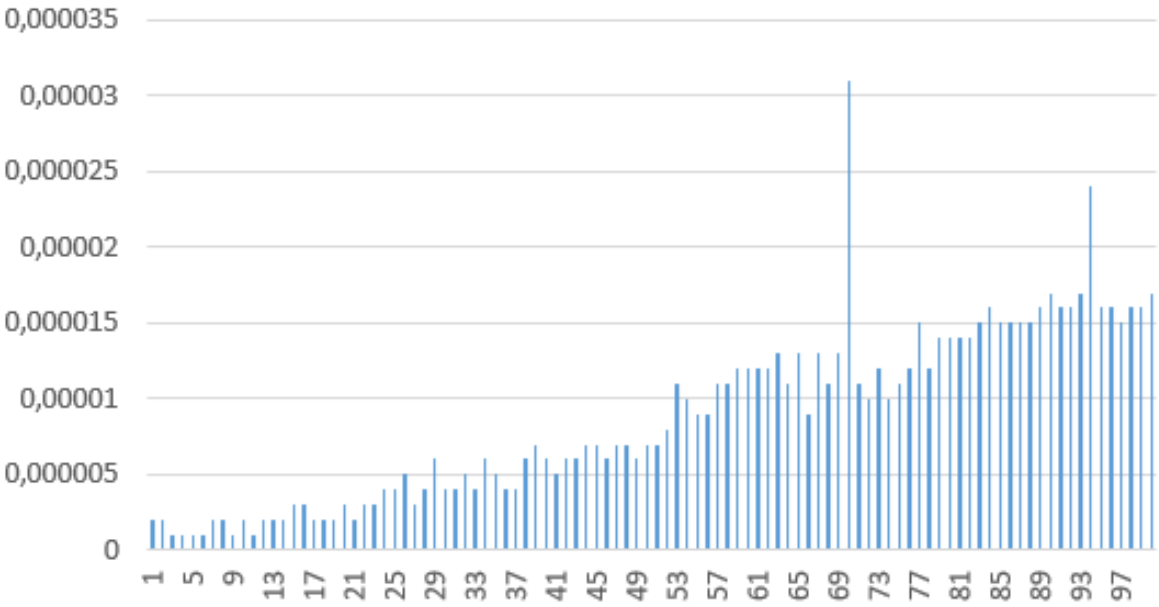
Результаты

- Поиск по id O (n)
- Обновление элемента O (n)
- Добавление O (1)
- Поиск по полю O (n)
- Удаление O (n*m)


Добавление элемента
0,00004
0,000035
0,00003
0,000025
0,00002
0,000015
0,00001
0,000005
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101

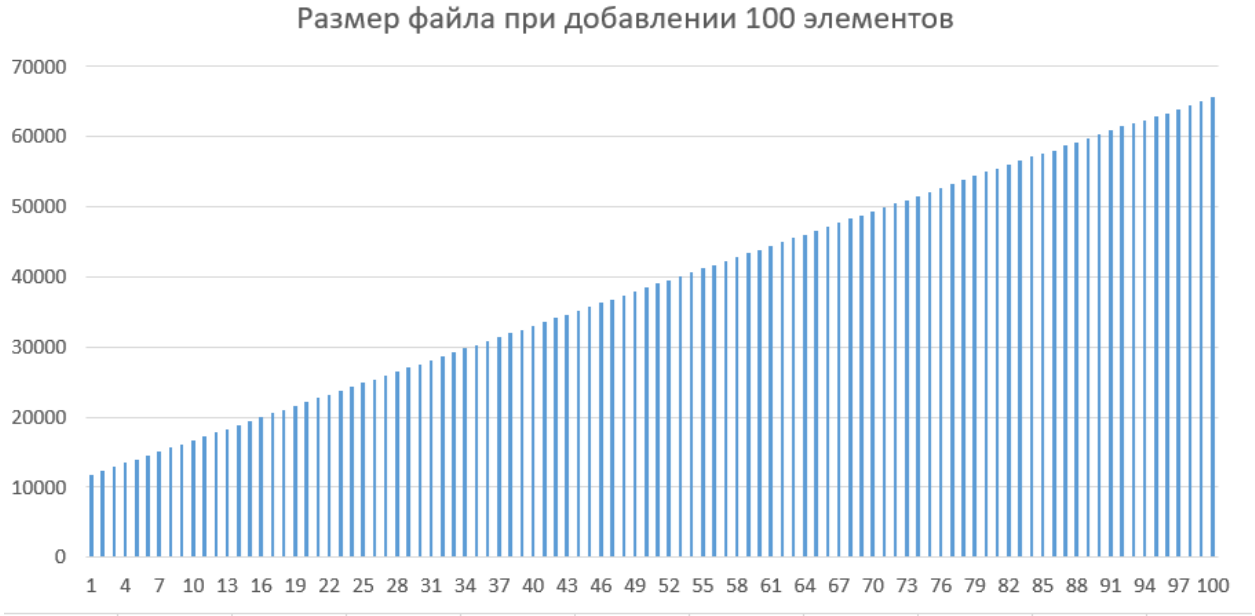

Удаление элемента по id
0,000025
0,00002
0,000015
0,00001
0,000005
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97


Обновление элемента по id
0,00004
0,000035
0,00003
0,000025
0,00002
0,000015
0,00001
0,000005
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97

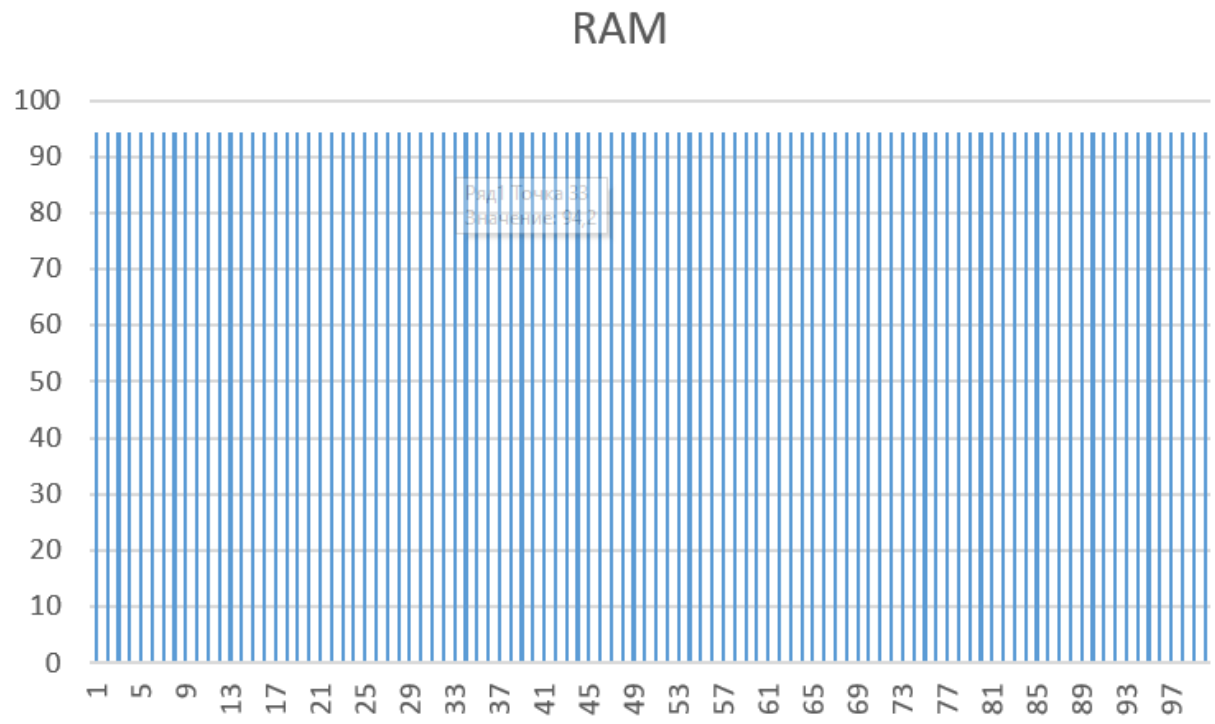

Поиск элемента по id
0,000035
0,00003
0,000025
0,00002
0,000015
0,00001
0,000005
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97


Поиск элемента по значению поля
0,00008
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100


Поиск по id родителя
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100


Размер файла при добавлении 100 элементов
70000
60000
50000
40000
30000
20000
10000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100

gnome-system-monitor – утилита для проверки ram



Вывод:

В ходе выполнения лабораторной работы я реализовал хранение документного дерева в файле и базовые операции для работы с ним. Также я убедился в работоспособности своего решения, проведя замеры времени работы.