Отчет по Лабораторной работе №
по курсу “Низкоуровневое программирование”

Вариант №3

Выполнил:
Студент группы P33082
Дробыш Дмитрий Александрович

Преподаватель:
Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023

Моя реализация ЛР1

Описание заданий

Задание 1

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: цетырёхбайтовые 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
    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

Идея:
Приложение работает на двух уровнях. Нижний уровень, который занимается записью в файл, работой над удалениями, чтением. И уровень пользователя, где происходит связь узлов графа, их создание, изменение и удаление. При удалении данных из файла образуются пустые места «Дырки». Они связываются в Linked List. На нижнем уровне узлы графа тоже представлены как Linked List. Это решает проблему медленной работы с записью и удалением (и поиском соотв.) узлов, а также позволяет еще раз использовать это место в файле. У самих же узлов нет ограничений на количество строк, целых чисел и строк. Их можно добавлять в процессе работы программы.

Структуры:

Заголовок файла:
```
struct header{
        uint32_t signature; - сигнатура
        uint64_t first_hole_ptr; - указатель на первую «Дырку»
        uint64_t first_node_ptr; - указатель на первую «Ноду»
        uint64_t last_node_ptr; - указатель на последнюю «Ноду»
        uint64_t node_id; - id узла. Тут понятно.
} __attribute__((packed));
```

Дырки для вторичного использования данных.
```
struct hole{
        uint64_t hole_ptr; - указатель на дырку
        uint64_t size_of_hole; - размер дырки
        uint64_t prev_ptr; указатель на пред. Дырку
        uint64_t next_ptr; - указатель на след. Дырку
} __attribute__((packed));
```

Строки бывают разного размера, поэтому их трудно положить в один массив. Решение этой проблемы — структура, которая хранит указатель на строку и размер этой строки.

```
struct string_save{
        uint64_t size_of_string; -  размер строки
        uint64_t string_line_ptr; - указатель на строку
}; __attribute__((packed));
```

Ну и сам узел:
```
struct node {
        uint64_t id; —  node_id
        uint64_t d; — степень вершины графа
        uint64_t nodes; — указатель на соседей
        uint64_t prev_ptr; — указатель на прошлую ноду
        uint64_t next_ptr; — указатель на следующую ноду

        uint64_t n_ints; — сколько целых чисел в ноде
        uint64_t ints_array_ptr; — указатель на массив целых чисел

        uint64_t n_doubles; сколько чисел с плавающей точкой в ноде
        uint64_t doubles_array_ptr;  — указатель на массив double

        uint64_t n_strings; — количество строк в нодею
        uint64_t strings_array_ptr; — указатель на struct string_save, о ктором я говорил ранее
} __attribute__((packed));
```

Описание функций:
From file.h:
FILE* open_file(char* name); — открывает файл name
void* read_file(uint64_t offset, uint64_t length); - по offset читает length байт
uint64_t write_file(void* data_ptr, uint64_t size_of_data); - вернет адрес в файле, куда получилось записать данные
void delete_from_file(uint64_t offset, uint64_t length); - производит удаление из файла. В некоторых случаях с освобождением памяти, но, чаще всего, резервируя место для дальнейшей работы.

From node.h:

char* read_var_string(void); — производит чтение строки любого размера.
uint64_t find_node_by_id(uint64_t id); — производит поиск ноды по id, вернет указатель на ноду в файле.
void delete_node_by_id(uint64_t id); — производит поиск и удаление ноды по id.

void create_node(void); — функция создания ноды, попросить ввести количество uint, double, string и их значения.

void connect_nodes(uint64_t id1, uint64_t id2); — создает связь между узлами. (Изменит массив соседей и степень вершины графа)

void print_nodes(void); — вывод статистики по узлам (кол-во, свойства)

void print_info_node(uint64_t id); — вывод ноды и ее содержания на экран

void append_int(uint64_t id);
void append_double(uint64_t id);
void append_string(uint64_t id); — Эти три функции производят добавление данных в узел.

void remover(void); — удалит все ноды в файле.

У пользователя есть свой формат взаимодействия с файлом:
h -- help
q -- exit
i -- show all nodes
p / p uint64 -- detailed info for concrete node
n -- add new node
u / u uint64 -- append new unsigned integer to node with id = param
d / d uint64 -- append new double to node with id = param
s / s uint64 -- append new string to node with id = param
c / c uint64 uint64 -- connect 2 nodes by id
r / r uint64 -- remove node by id
t -- remove nodes

Тесты:

Не очень много смысла в простом прогоне программы (1 раз), поэтому при помощи hyperfine я сделал это 600 раз.

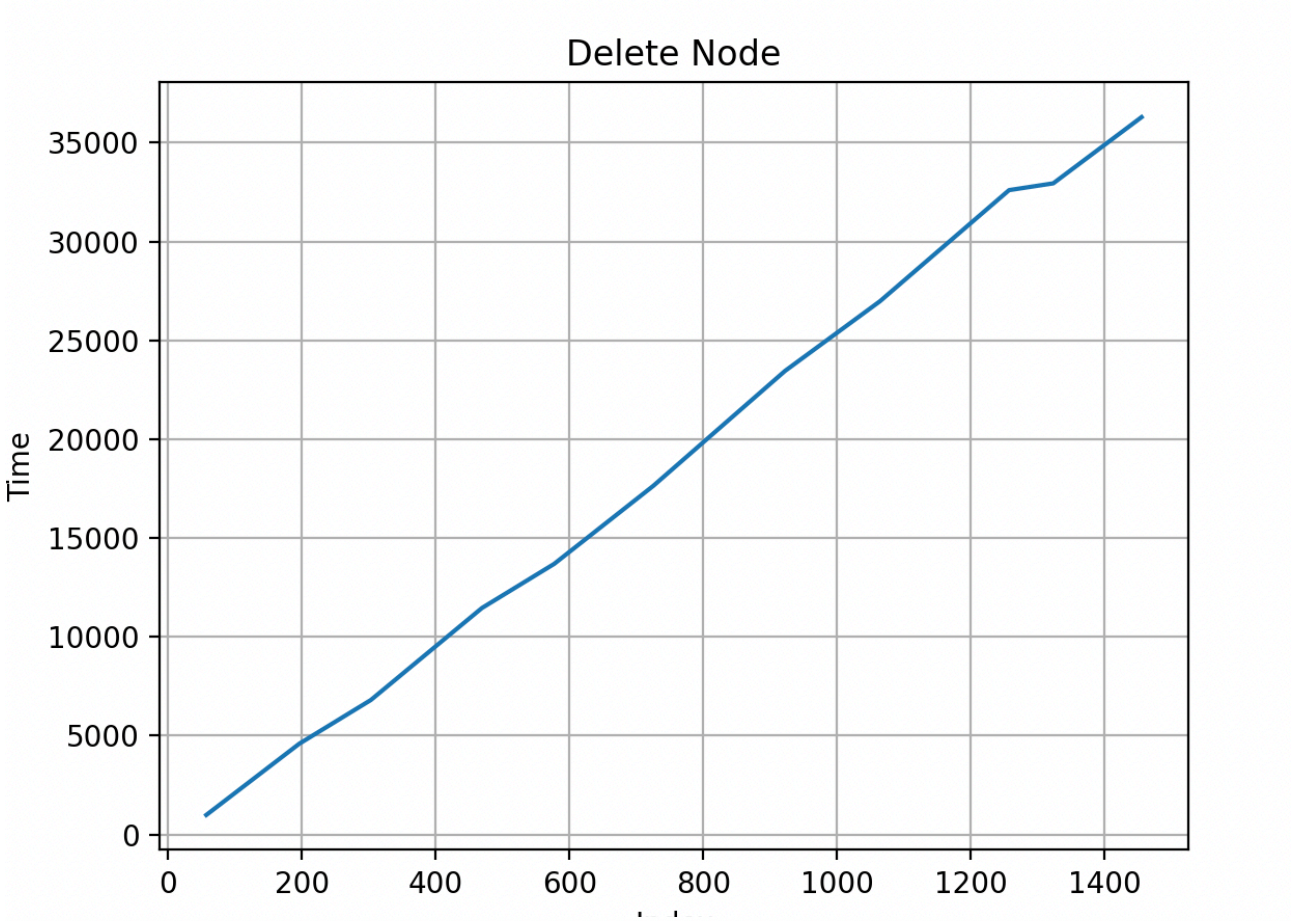
В файл input написал скрипт для создания узлов с числами и строками.

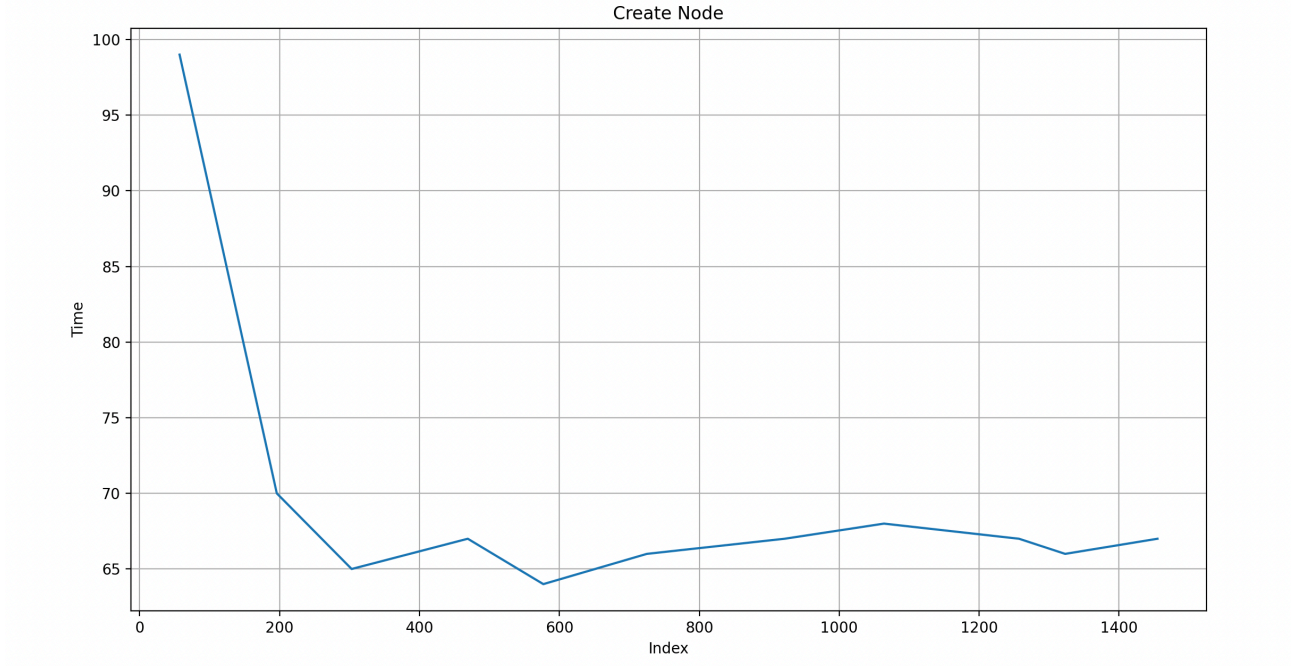
Так как удалений не было, запись будет только за O(1).
Смотрим на результат:

Теперь удалим все ноды и прогоним алгоритм еще один раз. Так мы получим значения для записи с дырками :

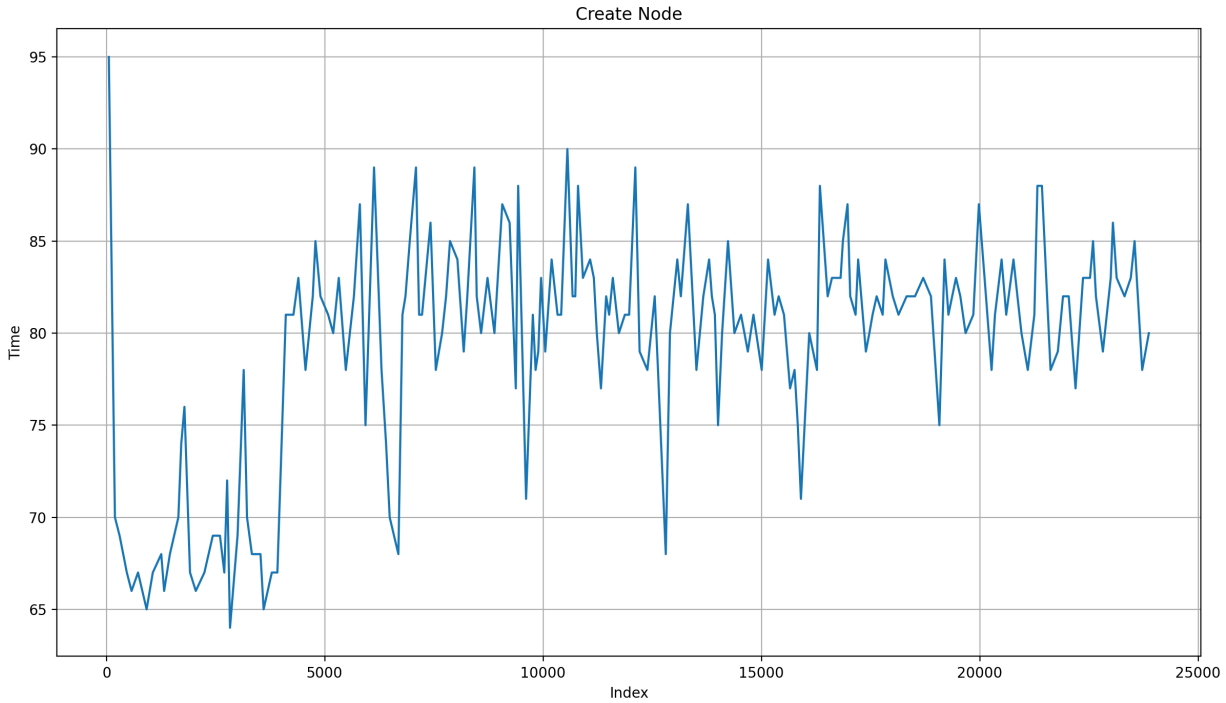
Tests:

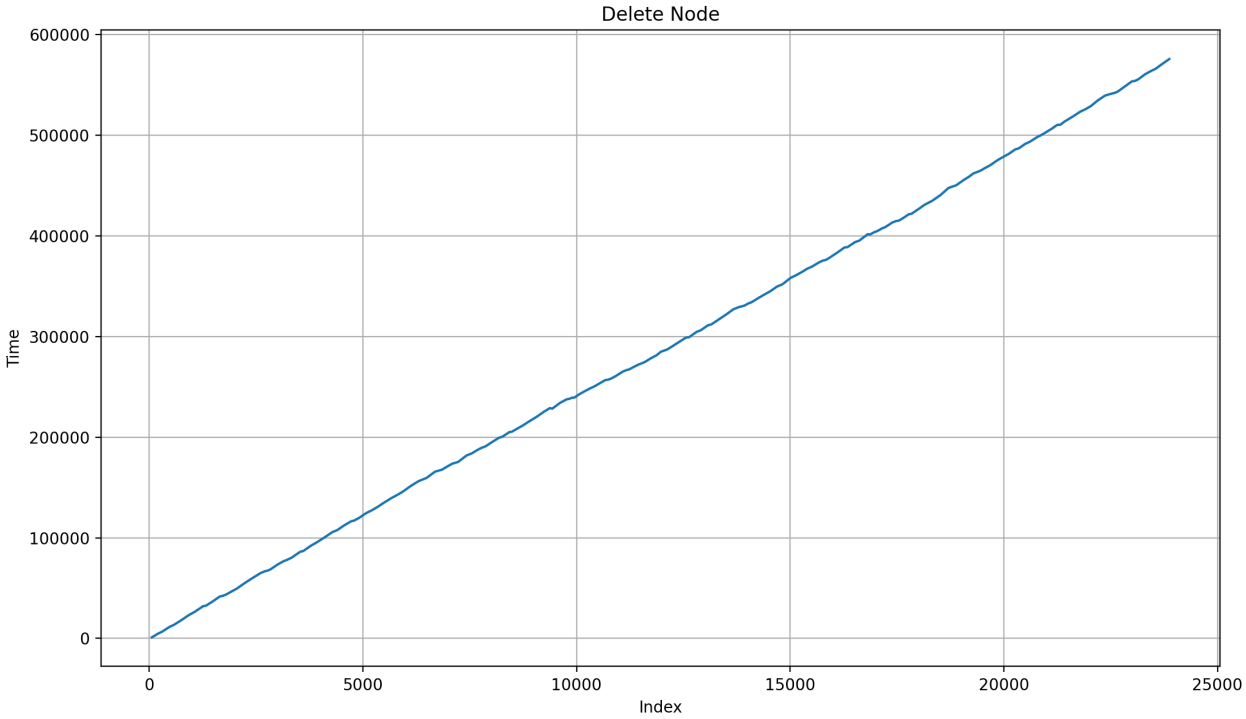
Small data:





Big data


Create Node
Time
Index
95
90
85
80
75
70
65
0
5000
10000
15000
20000
25000


Delete Node
Time
Index
600000
500000
400000
300000
200000
100000
0
0
5000
10000
15000
20000
25000

Да, конечно, время записи выросло, но требовалось написать за опт. O(1). А это она и есть!!)