ИТМО

Лабораторная работа №1

по дисциплине: Низкоуровневое программирование

Вариант: 3

Выполнил: Неграш Андрей, P33301

Преподаватель: Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023

Задание

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

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 $O(n*m) > t \rightarrow O(n+m)$, где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных
   f. Показать, что размер файла данных всегда пропорционален 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 работы

Основной модуль состоит из двух файлов – заголовочного файла и файла с кодом. Сам модуль предоставляет для использования 3 функции:

```
struct database_struct* open_database(char* filename);

void close_database(struct database_struct* db);

struct user_answer do_request(struct database_struct* db, struct
user_request* ur);
```

Open_database получает на вход строку с именем файла и путь до него. Если файла не существует, он создается, если существует, то открывается. Close_database закрывает базу данных и освобождает ресурсы. Запросы к базе данных осуществляются через функцию do_request, которой подается на вход открытая база данных и структура с запросом, которая выглядит следующим образом:

```
struct user_request{
    enum actions act;
    struct node* node;
    struct request* req;
};
```

Имеются следующие действия: ADD, REMOVE, UPDATE, GET
Конкретные параметры запроса задаются ссылкой на граф в памяти (актуально для действия ADD) или через набор условий, например величины параметров узла.

Функция возвращает ответ в виде следующей структуры:

```
struct user_answer{
    bool is_success;
    struct node* node;
    char* message;
};
```

Эта структура содержит информацию об успехе, сообщение в случае ошибки, и ссылку на узел графа, если это запрос GET.
Также к программе написан набор тестов, лежащий в файле main.c, которые измеряют время работы запросов и пишут файл с временем выполнения каждого запроса (при помощи этого файла и были получены графики в пункте ниже).

Реализация

Вся база данных хранится в одном текстовом файле, разбитом на блоки трёх видов:

Первый – таблица, где ID узла сопоставлены адреса мест, где лежит сам узел и его связи

Второй – блоки, где лежит содержимое самих узлов.

Третий – блоки, где для каждого узла перечислены все его связи и типы связей.

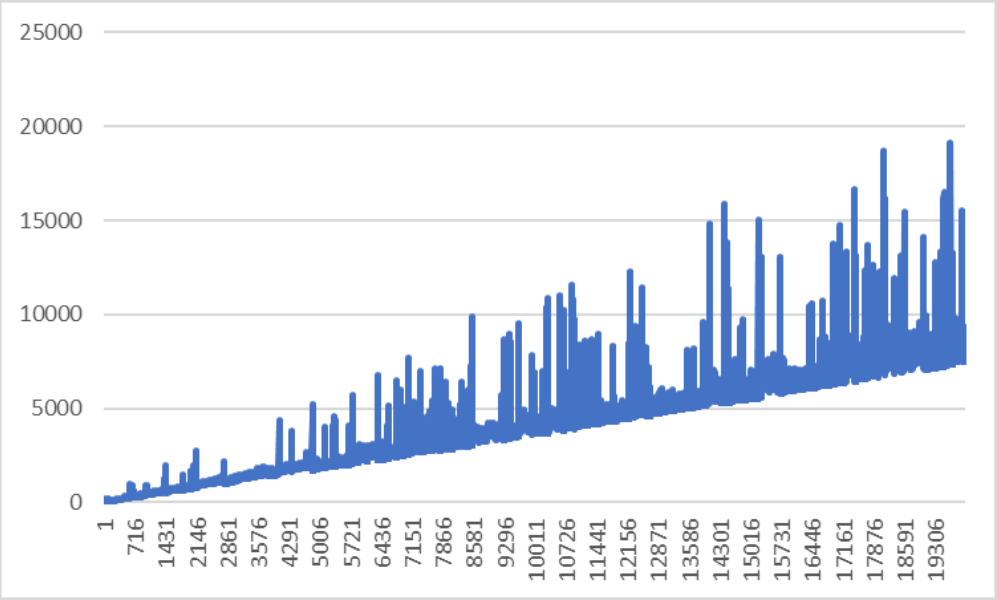
При переполнении каждого из блоков в конец файла добавляется новый блок данного типа. Каждый блок имеет заголовок, состоящий из трёх чисел: тип блока, его размер, адрес следующего блока такого же типа.

Написаны два аллокатора:

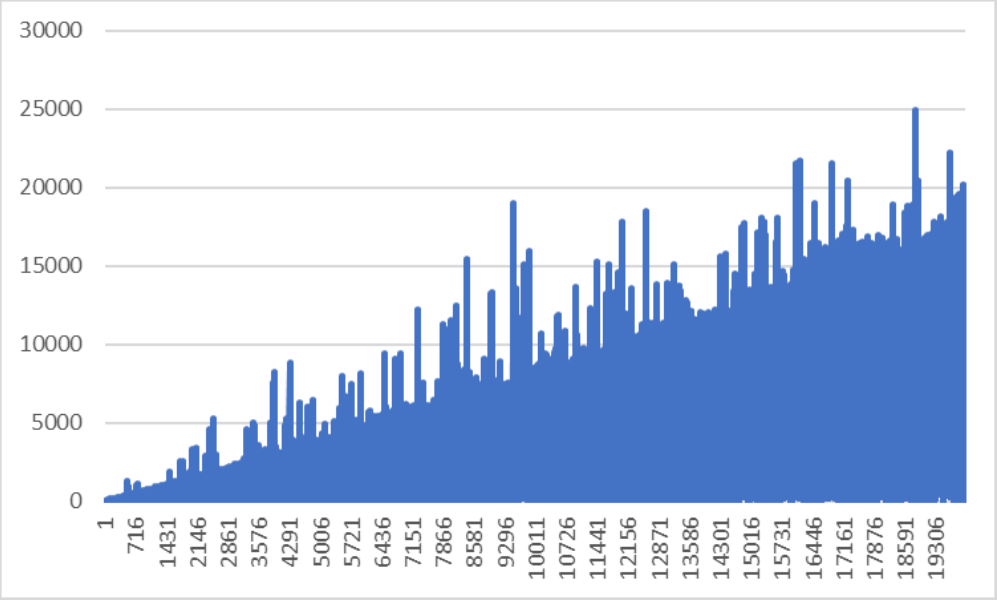
1) Для первого типа блоков. Так как размер блока фиксированный, то вместо полноценного аллокатора в заголовок добавляется еще одно число, обозначающее, сколько блоков занято. Таким образом можно идти по блокам, пока не найдётся свободное место, и потом добавляться туда.
2) Для остальных блоков. В памяти храниться двухсвязный массив, в котором хранится информация о размере свободного блока, о его положении, и при добавлении новой записи в БД ищется самый маленький подходящий блок.

Результат

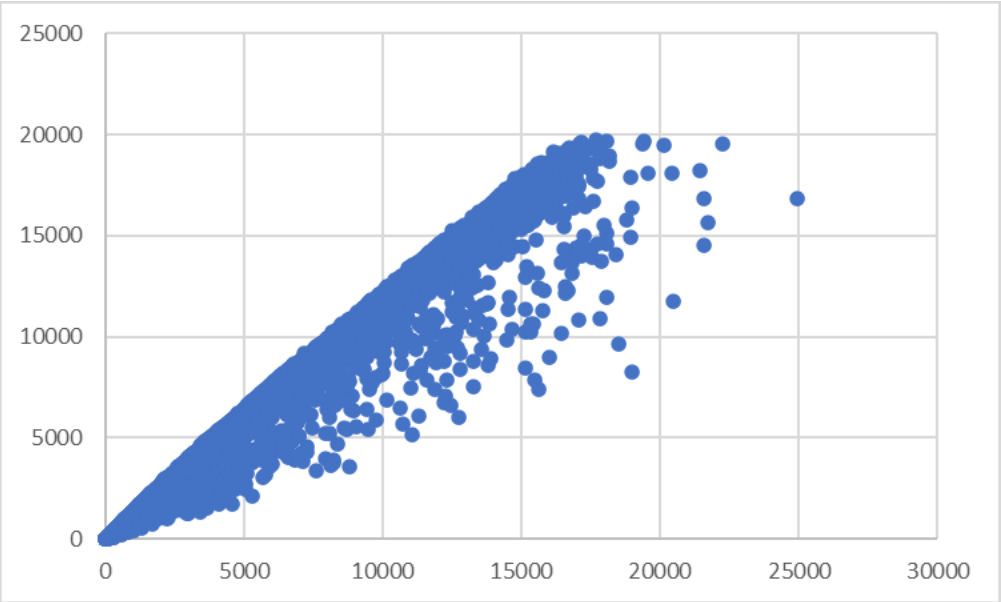
В результате тестовой программой получилась следующая зависимость времени добавления нового элемента от размера всей БД. Время выполнения запроса зависит от размера БД как O(n), иногда время становится больше, чем обычно. Это происходит из-за того, что периодически требуется выделить новый блок, так как в старых закончилось место.



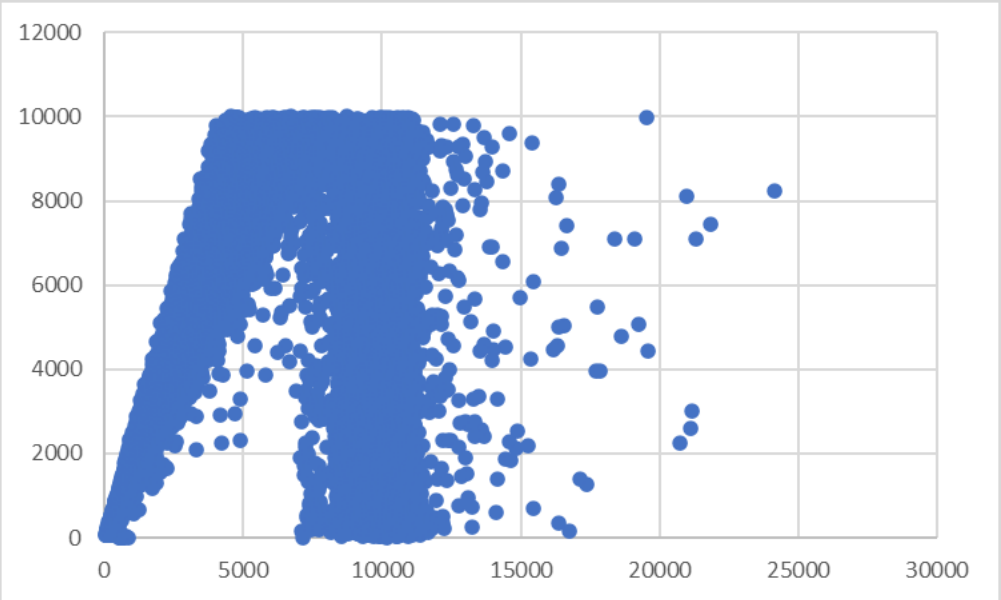
Зависимость времени выбора элемента по ID от размера БД:



Некоторые элементы берутся быстрее, некоторые медленнее, это зависит от того, в какой части БД они находятся. Если в начале, то быстрее, если дальше, то медленнее. Это можно увидеть, если построить зависимость времени от ID записи:



Удаление происходит также за O(n), что видно по следующему графику:



Вывод

Итак, в ходе выполнения данной лабораторной работы мне удалось создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего варианту вида: Граф узлов с атрибутами.