Университет ИТМО
Факультет ПИиКТ

Низкоуровневое программирование
Лабораторная №1
Вариант 6

Выполнил: Рябоконь А.Б.
Группа: P33302
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.

Задачи:

1. Спроектировать структуры данных для представления информации в оперативной памяти
2. Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним:
3. Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных
4. Реализовать тестовую программу для демонстрации работоспособности решения
5. Результаты тестирования по п.4 представить в составе отчёта, при этом

Описание работы

Данные организованы в виде графа, где каждая вершина представляет собой запись данных и имеет фиксированный набор атрибутов, таких как порядок, типы и названия. Взаимоотношения между вершинами представлены рёбрами, которые могут быть направленными или ненаправленными и иметь вес. Схема данных определяется структурой графа, а доступ к вершинам может осуществляться через фильтры на их атрибуты или смежность с другими вершинами.

Основные структуры

Основные структуры, использующиеся для представления графа, выглядят следующим образом

Структура ноды:

```
struct Node {
  size_t Id;
  bool Deleted;
  struct AddrInfo Previous;
  struct AddrInfo Attributes;
  struct AddrInfo Next;
};
```

Структура ребра связи между нод:

```
struct NodeLink {
  size_t Id;
```

```
    bool Deleted;
    size_t LeftNodeId;
    size_t RightNodeId;
    enum ConnectionType Type;
    float Weight;
    struct AddrInfo Next;
    struct AddrInfo Previous;
};
```

Структура графа:

```
struct Graph {
    size_t Id;
    size_t NodeCounter;
    size_t LinkCounter;
    size_t AttributeCounter;
    size_t LazyDeletedNodeCounter;
    size_t LazyDeletedLinkCounter;
    size_t NodesPlaceable;
    size_t LinksPlaceable;
    size_t PlacedNodes;
    size_t PlacedLinks;
    struct MyString Name;
    struct AddrInfo Nodes;
    struct AddrInfo AttributesDecription;
    struct AddrInfo LastNode;
    struct AddrInfo Links;
    struct AddrInfo LastLink;
    struct AddrInfo Next;
    struct AddrInfo Previous;
};
```

Для хранения данных о размещении объектов используется структура:

```
struct AddrInfo {
    bool HasValue;
    size_t BlockOffset;
    size_t DataOffset;
};
```

File API:

```
struct FileAllocator *initFileAllocator(char *FileName);
void shutdownFileAllocator(struct FileAllocator *allocator);
void dropFileAllocator(struct FileAllocator *allocator);
struct AddrInfo allocate(struct FileAllocator *const allocator, size_t Size);
struct AddrInfo getFirstBlockData(const struct FileAllocator *const allocator);
void deallocate(const struct FileAllocator *const allocator, struct AddrInfo Addr);
int fetchData(const struct FileAllocator *const allocator, const struct AddrInfo Addr,
        const size_t Size, void *const Buffer);
```

```
int storeData(const struct FileAllocator *const allocator, const struct AddrInfo Addr,
        const size_t Size, const void *const Buffer);

Graph API:
void endWork(struct StorageController *Controller);
void dropStorage(struct StorageController *Controller);
void deleteString(const struct StorageController *const Controller, struct MyString String);
struct MyString createString(struct StorageController *const Controller,
                    const char *const String);

size_t createGraph(struct StorageController *const Controller,
            const struct CreateGraphRequest *const Request);
size_t createNode(struct StorageController *const Controller,
            const struct CreateNodeRequest *const Request);
size_t createNodeLink(struct StorageController *const Controller,
            const struct CreateNodeLinkRequest *const Request);

struct NodeResultSet *readNode(const struct StorageController *const Controller,
                    const struct ReadNodeRequest *const Request);
struct NodeLinkResultSet *readNodeLink(const struct StorageController *const Controller,
                        const struct ReadNodeLinkRequest *const Request);
struct GraphResultSet *readGraph(const struct StorageController *const Controller,
                    const struct ReadGraphRequest *const Request);

size_t updateNode(const struct StorageController *const Controller,
            const struct UpdateNodeRequest *const Request);
size_t updateNodeLink(const struct StorageController *const Controller,
            const struct UpdateNodeLinkRequest *const Request);

size_t deleteNode(const struct StorageController *const Controller,
            const struct DeleteNodeRequest *const Request);
size_t deleteNodeLink(const struct StorageController *const Controller,
            const struct DeleteNodeLinkRequest *const Request);
size_t deleteGraph(struct StorageController *const Controller,
            const struct DeleteGraphRequest *const Request);

Storage API:
size_t increaseGraphNumber(struct StorageController *Controller);
size_t decreaseGraphNumber(struct StorageController *Controller);
size_t increaseNodeNumber(struct StorageController *Controller);
size_t increaseNodeLinkNumber(struct StorageController *Controller);
void updateLastGraph(struct StorageController *const Controller,
            struct AddrInfo LastGraphAddr);
void updateFirstGraph(struct StorageController *const Controller,
            struct AddrInfo FirstGraphAddr);
```

Выполнение

Программа состоит из нескольких частей. Часть работы с файлом, часть работы с графом и часть со структурами данных, запросов и ответов. Для проверки соответствия необходимым показателям используются бенчмарки.

Замер временных показателей программы

График зависимости времени вставки от количества элементов вставки. Время замерялось на вставке каждых 10000 элементов. Как видно из графика вставка происходит за O(1) (Погрешность в районе 10%, но при таком объёме данных и условиях проведения расчётов это допустимо). Таким образом мы видимо что время никак не зависит от количество данных в файле

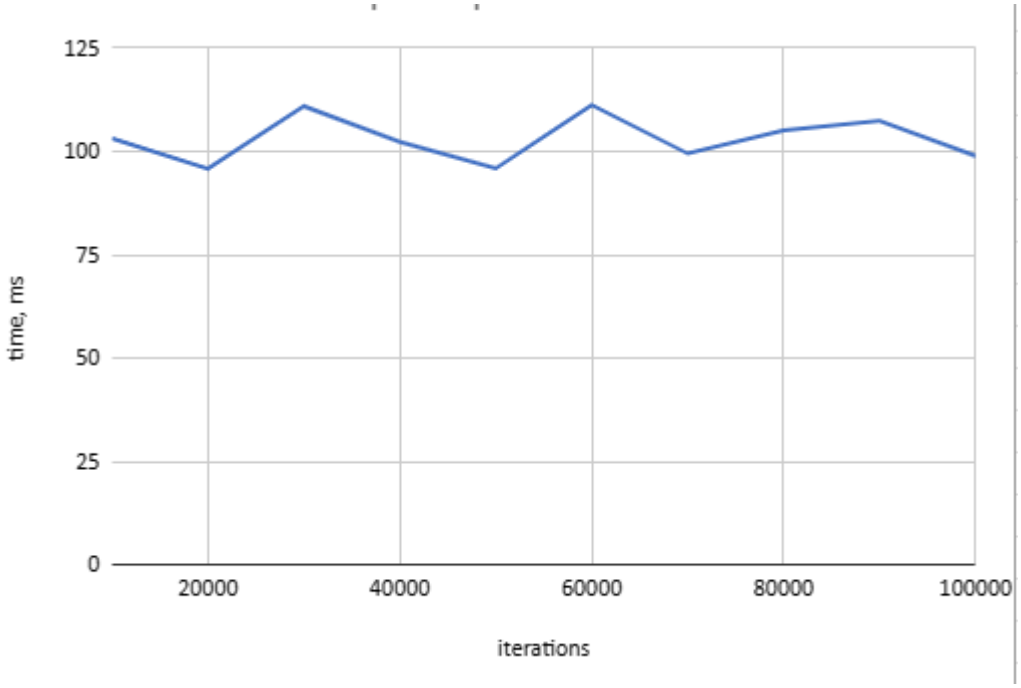
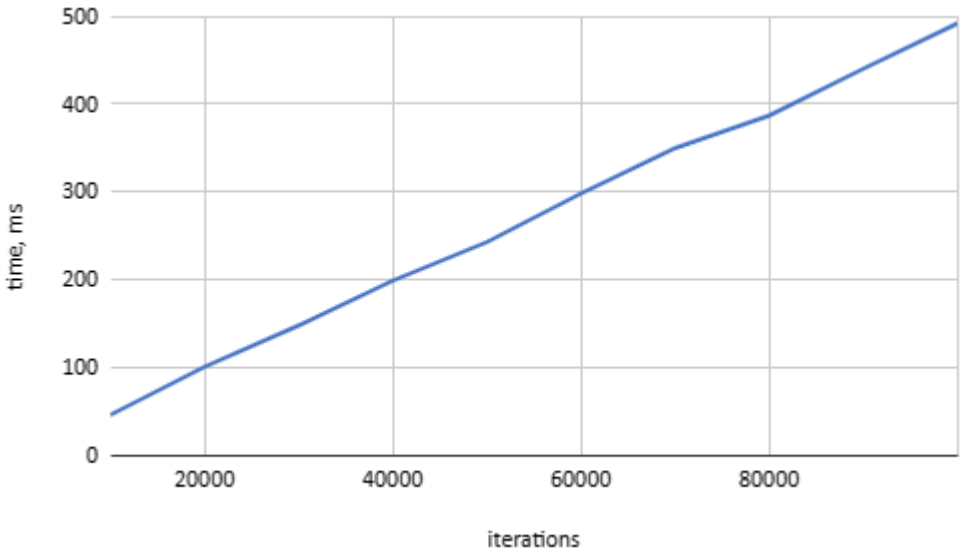
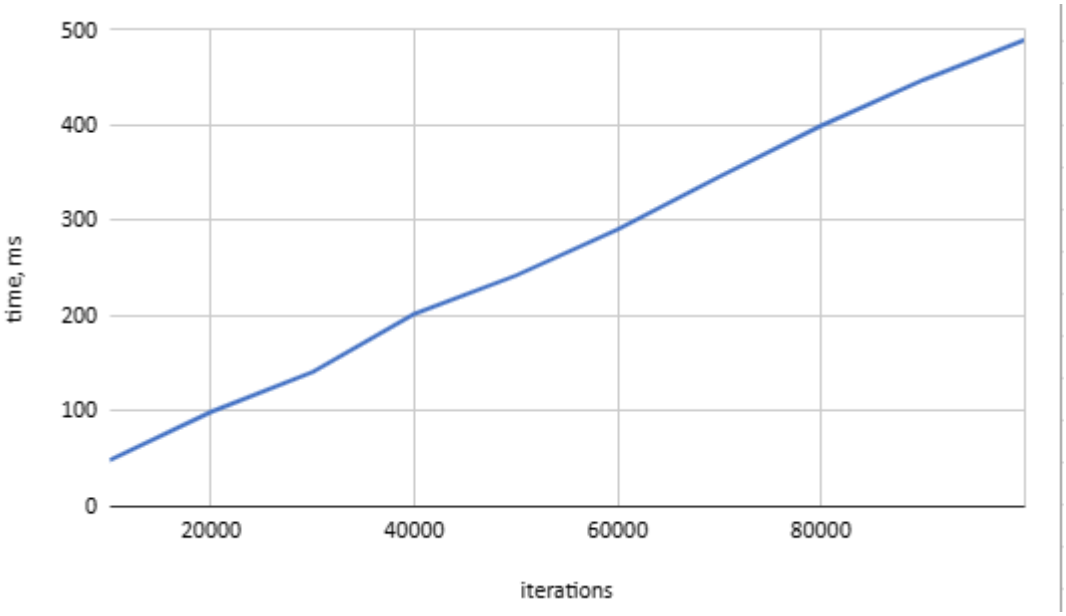


График зависимости худшего времени поиска элемента (без учета отношений) от количества элементов, представленных в файле. Из графика видно, что зависимость линейная.



Обновление элементов в зависимости от количества. Видно что зависимость линейная.



Размер файла в зависимости от количества элементов

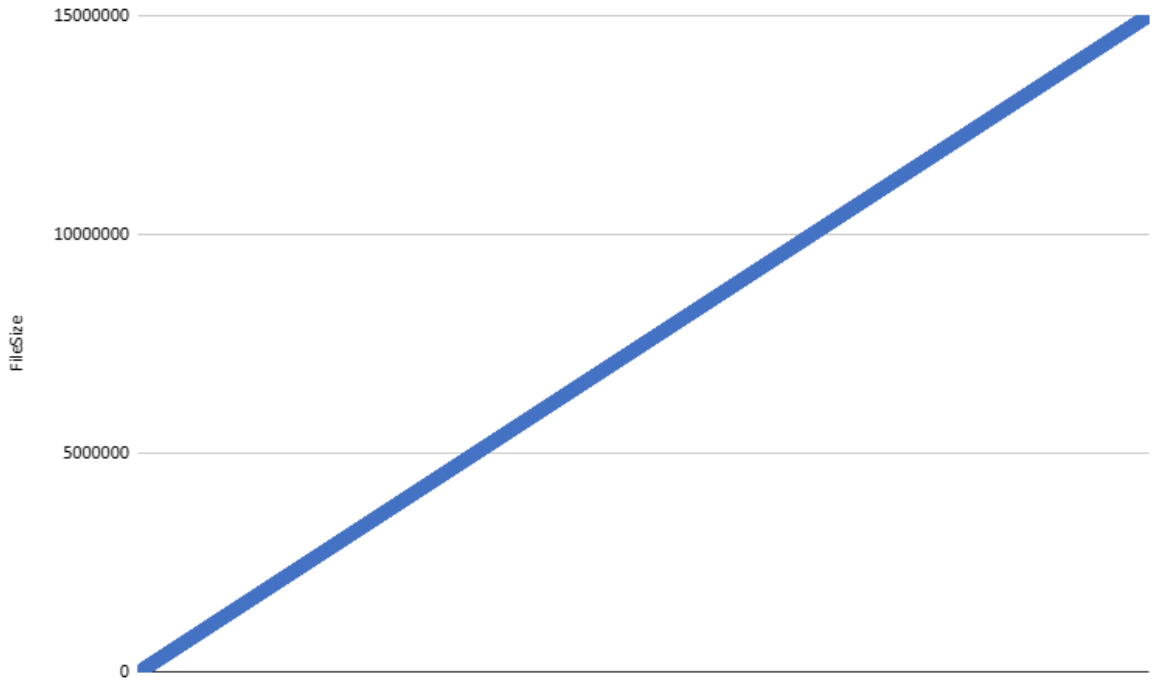
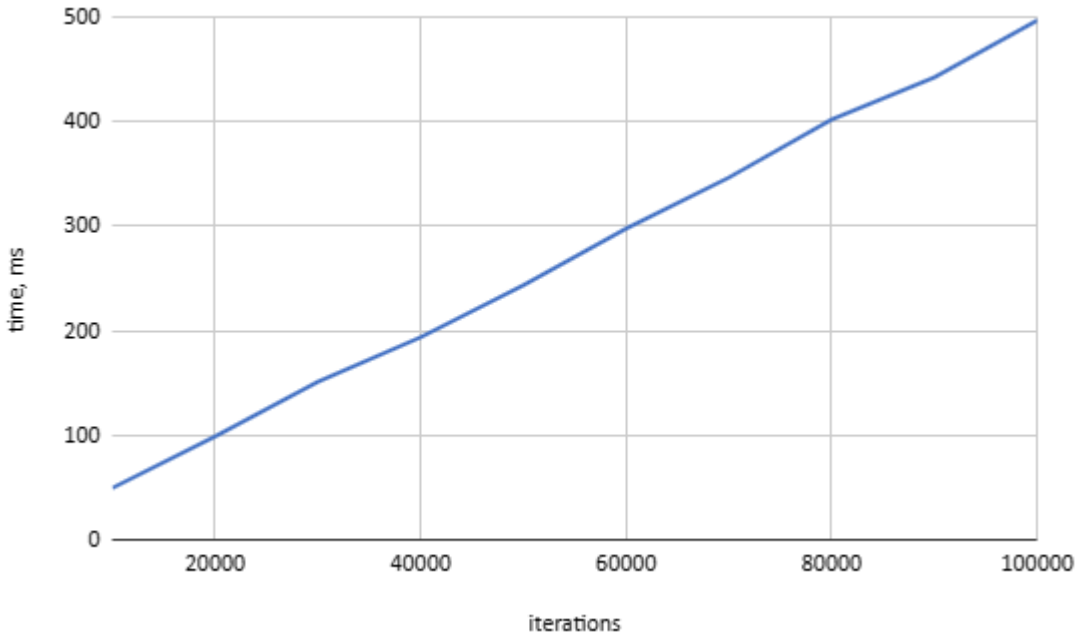
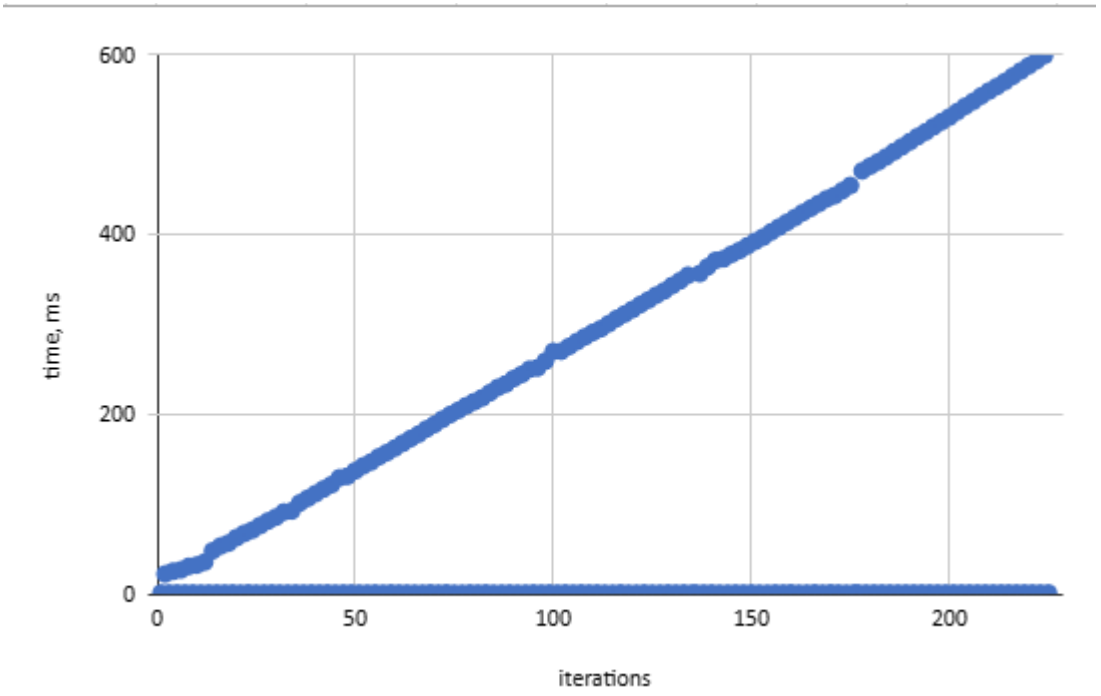


График зависимости худшего времени выполнения операции удаления элемента от количества элементов, представленных в файле. Во время каждой операции при замере выполнялось удаление элемента одной глубины (M). Из графика видно, что зависимость линейная. Таким образом я доказываю, что удаление элемента данных выполняется не более чем за O(N+M).



Дополнительный бенчмарк:

Добавление 1000 и удаление 800 элементов и всё это в цикле. Видим что добавление происходит за константное время, а удаление увеличивается в зависимости от количества элементов в графе, как и должно быть.



Вывод

В ходе выполнения данной лабораторной работы был разработан модуль, реализующий хранение данных в виде реляционных таблиц. Размер данных может достигать 10 GB. В модуле реализованы операции выборки, вставки, обновления и удаления элементов, а также возможность соединения таблиц. Модуль может работать под управлением ОС семейств Windows и *NIX