ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники
Системное программное обеспечение

Дисциплина «Низкоуровневое программирование»

Отчет к лабораторной работе №1

Выполнил:
Студент группы P33302
Владыкина Карина Кирилловна
Проверил:
Кореньков Юрий Дмитриевич

Санкт-Петербург
2022 год

Вариант 3: Граф узлов с атрибутами (Cypher)

Репозиторий:
https://github.com/1KarinaV/llp_lab1

1.Цели

Создать модуль, реализующий хранение в одном файле данных в виде графа узлов с атрибутами общим объёмом от 10GB.

2.Задачи

- Спроектировать структуры данных для представления информации в оперативной памяти
- Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним:
    - Операции над схемой данных (создание и удаление элементов)
    - Базовые операции над элементами данными (вставка, перечисление, обновление, удаление)
- Реализовать публичный интерфейс для приведенных выше операций
- Реализовать тестовую программу для демонстрации работоспособности решения

3. Описание работы

Программа представляет собой следующие модули:

- graphdb (содержит все операции: над элементами данных, над схемой, взаимодейтсвие с БД, и работа с файлом БД)
- functions_demo (иллюстрируется все функции)
- measurings (выполняются измерения)

Подключение к БД

- создание бд с помощью метода createNewDBbyScheme:

```
DB = createNewDBbyScheme(Scheme, FileName: "graphs.mydb");
```

- закрытие с помощью метода closeDB

```
closeDB(DB);
```

Создание и заполнение схемы таблицы:

```
Scheme = createDBScheme();

MovieNodeType = addNodeTypeToScheme(Scheme, TypeName: "Movie");
addAttrToNodeScheme( NodeScheme: MovieNodeType, Name: "Title", Type: tpString);
addAttrToNodeScheme( NodeScheme: MovieNodeType, Name: "Year", Type: tpInt32);
ActorNodeType = addNodeTypeToScheme(Scheme, TypeName: "Actor");
addAttrToNodeScheme( NodeScheme: ActorNodeType, Name: "Family", Type: tpString);
addAttrToNodeScheme( NodeScheme: ActorNodeType, Name: "Name", Type: tpString);
addAttrToNodeScheme( NodeScheme: ActorNodeType, Name: "Oscar", Type: tpBoolean);
addAttrToNodeScheme( NodeScheme: ActorNodeType, Name: "Year_of_birthday", Type: tpInt32);

addDirectedToNodeScheme( NodeScheme: MovieNodeType, ToNodeScheme: ActorNodeType);
```

Реализация запроса:

Все запросы построены по типу Cypher-запроса

- пример запроса:

```
// MATCH (j:Movie)-[:DIRECTED]->(a:Actor) WHERE (j.Year < 2004) AND (a.Family != 'Pitt') AND (a.Family != 'Hamatova') RETURN a;
printf("MATCH (j:Movie)-[:DIRECTED]->(a:Actor) WHERE (j.Year < 2004) AND (a.Family != 'Pitt') AND (a.Family != 'Hamatova') RETURN a;  =>\n");
ns2 = queryCypherStyle(DB, nLinks: 2, MovieNodeType, cond, ActorNodeType, cond2);
ns12 = ns2;
i = 0;
while (ns12 != NULL) {
    navigateByNodeSetItem(DB, NodeSet: ns12);
    if (openNode(DB, NodeScheme: ActorNodeType)) {
        char * Family = getString(DB, Offset: getNodeAttr(DB, NodeScheme: ActorNodeType, AttrName: "Family"));
        printf("%s [%i]\n", Family, (int)getNodeAttr(DB, NodeScheme: ActorNodeType, AttrName: "Year_of_birthday"));
        register_free( amount: strlen( s: Family)+1);
        free(Family);
        cancelNode(DB, NodeScheme: ActorNodeType);
    } else
        printf("Can't open actor node!\n");
    ns12 = ns12->next;
    i++;
}
freeNodeSet(DB, NodeSet: ns2);
printf("%i actors selected!\n", i);
```

- пример вывода:

```
Stepanov [1980]
Churikova [1982]
2 actors selected!
```

- Результатом запроса может быть:

```
typedef struct memNodeSetItem { // Структура-элемент результата запроса к базе данных
    memNodeSchemeRecord * NodeScheme; // Ссылка на тип узла
    int PrevOffset; // Смещение предыдущего узла этого типа
    int ThisOffset; // Смещение текущего узла этого типа
    struct memNodeSetItem * next; // Следующий элемент
    struct memNodeSetItem * prev; // Предыдущий элемент
} memNodeSetItem;
```

Используемые структуры:

- Схема:

```
typedef struct { // Структура-описатель структуры базы данных
    memNodeSchemeRecord * FirstSchemeNode; // Указатель на описатель первого типа узла
    memNodeSchemeRecord * LastSchemeNode; // Указатель на описатель последнего типа узла
} memDBScheme;

typedef struct { // Структура-описатель открытой базы данных
    memDBScheme * Scheme;
    char * WriteBuffer; // Буфер записи
    int nWriteBuffer;
    char * ReadBuffer; // Буфер чтения
    int nReadBuffer;
    int iReadBuffer;
    FILE * FileDB;
} memDB;
```

- Атрибут:

```
typedef struct memAttrRecord { // Структура-описатель атрибута узла
    char * NameString; // Запись с именем атрибута
    unsigned char Type; // Тип атрибута
    struct memAttrRecord * next; // Указатель на следующий атрибут
} memAttrRecord;
```

- Тип узла:

```
typedef struct memNodeSchemeRecord { // Структура-описатель типа узла в памяти
    char * TypeString; // Запись с именем типа узла
    int RootOffset; // Смещение от начала файла корневого указателя на список узлов
    int FirstOffset; // Смещение от начала файла корневого указателя на первый элемент списка узлов
    int LastOffset; // Смещение от начала файла корневого указателя на последний элемент списка узлов
    char * Buffer; // Буфер с данными текущего узла
    int nBuffer; // Число заполненных байт в буфере
    int added; // Флаг того, что создается новый узел
    int PrevOffset; // Смещение от начала файла предыдущего узла
    int ThisOffset; // Смещение от начала файла текущего узла
    memNodeDirectedTo * DirectedToFirst; // Указатель на начало списка с типами узлов
    memNodeDirectedTo * DirectedToLast; // Указатель на конец списка с типами узлов
    struct memAttrRecord * AttrsFirst; // Указатель на начало списка атрибутов
    struct memAttrRecord * AttrsLast; // Указатель на конец списка атрибутов
    struct memNodeSchemeRecord * NextNodeScheme; // Указатель на следующий тип узла
} memNodeSchemeRecord;
```

- Операнд

```
typedef struct { // Операнд в операции условия
    unsigned char OperandType; // тип операнда
    union {
        struct memCondition * opCondition; // Другое условие
        char * opString; // Строка
        float opInt_Bool_Float; // Целое число или логическое значение или вещественное число
        char * opAttrName;
    };
} memConditionOperand;
```

- Элемент условия:

```
typedef struct memCondition { // Элемент условия
    unsigned char OperationType; // Операция
    memConditionOperand * Operand1; // Первый операнд
    memConditionOperand * Operand2; // Второй операнд (или NULL, если операция унарная)
} memCondition;
```

- Возможные типы данных:

```
enum { tpInt32 = 0, tpFloat, tpString, tpBoolean } tpDataItems; // Типы данных атрибутов
```

4. Аспекты реализации

База данных хранится в файле, объемом до 10ГБ. Бд построена на графах - узлах и дугах.

Объектом БД является Node. У узлов есть разные типы. Так, например, ноды типа movie связаны с нодами типа actor. В схему новый тип узла добавляется с помощью:

```
memNodeSchemeRecord * addNodeTypeToScheme(memDBScheme * Scheme, char * TypeName);
```

А для того, чтобы показать связь между узлами используются дуги.

У каждого узла есть набор атрибутов (свойств), каждый из которых содержит в себе запись с именем, тип и указатель на следующий атрибут.

2. Для создания БД сначала нужно создать схему БД, которая содержит указатели на первый и последний типы узла. После того, как была создана бд, функция возвращает указатель на структуру данных, представляющую созданную базу.

3. Когда создаётся новый экземпляр узла вызовом функции createNode,он записывается в БД.

4. Для этого создаётся строка (после будет возвращено ее смещение от начала файла), в нее записывается экземпляр узла со всеми атрибутами. И данная строка записывается в файл

```
createNode(DB, NodeScheme: MovieNodeType);
setNodeAttr(DB, NodeScheme: MovieNodeType, AttrName: "Title", Value: createString(DB, S: Titles[i/200]));
setNodeAttr(DB, NodeScheme: MovieNodeType, AttrName: "Year", Value: 2000 + i);
postNode(DB, NodeScheme: MovieNodeType);
```

В случае, если не удастся провести дугу между двумя узлами, будет выведено соотвествующее сообщение:

```
    if (!LinkCurrentNodeToCurrentNode(DB,  NodeSchemeFrom: MovieNodeType,  NodeSchemeTo: ActorNodeType))
        printf("Can't connect!\n");
    postNode(DB,  NodeScheme: MovieNodeType);
}
```

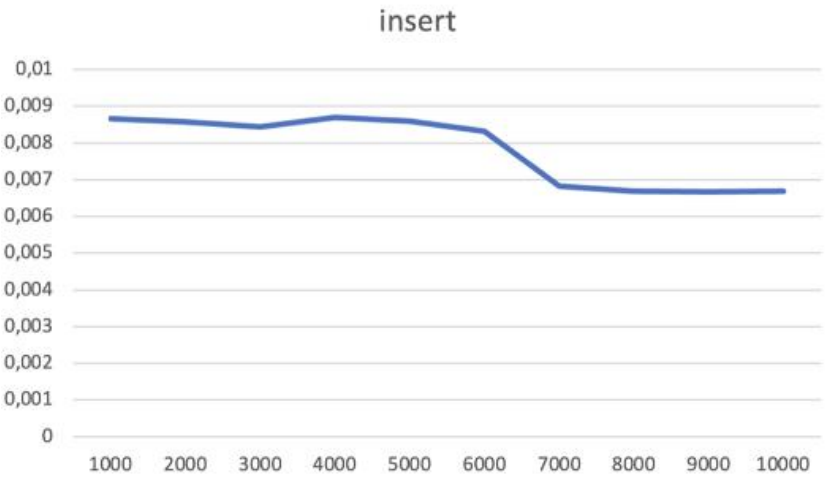
5. После этого внутренний указатель множества узлов перемещается на первый узел

```
rewindFirstNodes(DB,  NodeScheme: MovieNodeType);
```

5. Результаты

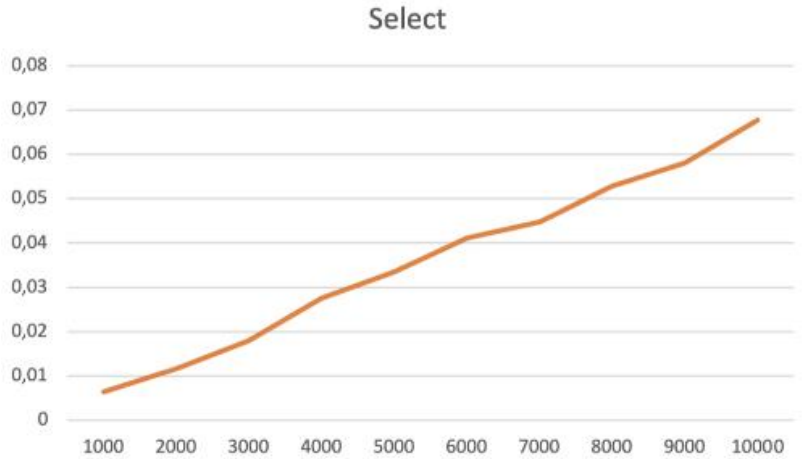
Insertion

выполняется за O(1) независимо от размера данных, представленных в файле



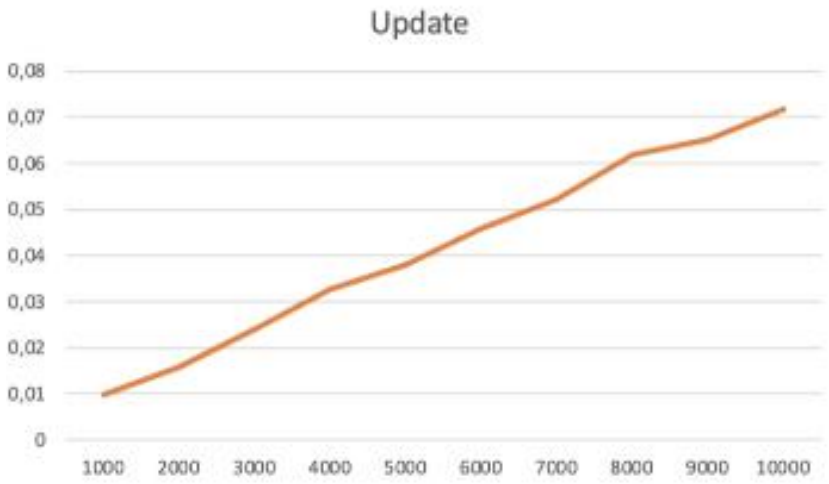
Select

Операция выборки без учёта отношений (но с опциональными условиями) выполняется за O(n), где n – количество строк



Update and Delete

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



5.1 Сборка

- Linux

```
git clone https://github.com/1KarinaV/llp_lab1
cd llp_lab1
make all
./functions_demo
./measurings
```

- Windows

```
git clone https://github.com/1KarinaV/llp_lab1
cd llp_lab1
```

Необходимо оставить файл Makefile.win и удалить расширение

```
make all
functions_demo.exe && measurings.exe
```

6. Вывод

В ходе работы была реализована графовая база данных ,поддерживающая операции Create, Insert, Delete, Update между вершинами и ребрами. Так же были разработаны тесты, в результате выполнения которых, было продемонстрировано, что реализация операция была выполнена с помощью правильных алгоритмов, потому что по графикам видно, что алгоритмическая сложность совпала с ожидаемой.