Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №1**

по дисциплине

‘Низкоуровневое программирование’

Вариант №1

*Выполнил:*

Студент группы P33312

Соболев Иван Александрович

*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.

**Задачи:**

**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 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  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

**Исходный код проекта:**

[Ivanio1/LLP-1 (github.com)](https://github.com/Ivanio1/LLP-1)

Изображение выглядит как шаблон, пиксель

Автоматически созданное описание

**Описание работы:**

Программа представляет собой консольное приложение, позволяющее хранить, редактировать и просматривать данные в формате документного дерева.

Программа состоит из следующих модулей:

* File\_managing – модуль для работы с файлом (Открытие, закрытие, чтение, запись, инициализация).
* Data\_managing – модуль для работы с основными структурами данных (Чтение из файла, запись в файл).
* Crud – основной модуль, в котором описаны базовые операции с данными и публичный интерфейс для работы с этими операциями.
* String\_managing – вспомогательный модуль для работы со строками.
* Test – модуль, в котором содержится тестовая программа.
* User\_interface – вспомогательный модуль для удобного создания структур данных и работы с ними из консоли.

Примеры работы программы:

Изображение выглядит как текст, Шрифт, графический дизайн, снимок экрана

Автоматически созданное описание

Рисунок 1-Выбор режима работы с файлом

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2-Заполнение паттерна

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 3-Работа с существующим файлом

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Рисунок 4-Вывод доступных команд

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

Рисунок 5-Добавление элемента

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 6-Вывод всех элементов

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 7-Обновление элемента

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 8-Удаление элемента

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 9-Поиск по id

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10-Поиск по полю

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 11-Поиск по родителю

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Автоматически созданное описание

Рисунок 12-Завершение работы

**Аспекты реализации:**

Внутреннее описание созданной программы:

Для обеспечения хранения всей информации в одном файле было решено разделить его на две части – метаданные и основные данные.

Секция с метаданными хранит в себе – информацию о типах и названии полей, массив индексов и текущий указатель кортежа (вспомогательный элемент, который нужен для отслеживания позиции последнего модифицированного элемента). Массив индексов (идентификаторов) для каждого кортежа содержит отступ до него от начала файла, что позволяет за O(1) находить любой элемент по id.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Секция с данными хранит элементы с их значениями. При проектировании и первых попытках реализации возникла проблема размера кортежа. Хотелось хранить кортежи одинаковых размеров, чтобы в случае удаления можно было на место «дырки» просто переместить новый кортеж. С типами int, bool и float было все ясно, а вот со строковым типом было непонятно, как разместить строки различной длины, при этом имея фиксированный размер кортежа. Для решения данной проблемы были введены два типа кортежей – обычный и строковый. Обычный кортеж хранит отступ до родительского и непосредственно массив данных кортежа, в котором все типы данных, кроме строкового, лежат в явном виде. Строковый тип хранится в виде ссылки на строковый кортеж. Сам же строковый кортеж хранится в виде двусвязного списка. Пример в коде:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Данное решение помогло справиться с проблемой разно размерных кортежей.

**Особенности алгоритмов с примерами кода:**

Для того, чтобы различать обычные и строковые кортежи необходимо просто проверить наличие индекса в массиве индексов равного положению кортежа в файле.

**Добавление** элемента не представляет из себя ничего интересного, узел просто добавляется в конец файла, а его идентификатор – в конец массива идентификаторов. Задачи фрагментации памяти делегирована удалению.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Обновление** элемента тоже происходит довольно тривиально – просто обновляем поле внутри узла. Однако в случае обновления строкового поля могут понадобиться новые строковые узлы, то есть необходимо расширить двусвязный список.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

**Поиск** элемента также является довольно интуитивной операцией: пробегаемся по массиву идентификаторов, при необходимости заглядывая внутрь каждого узла, и ищем совпадение по нужному полю.

Самой интересной и сложной операцией является **удаление**. Так как наши кортежи одинакового размера, то при удалении нам необходимо взять крайний элемент и переместить его на место удаляемого.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Но при этом возникает необходимость изменения ссылок на кортежи в массиве индексов. Для обычных кортежей требуется просто поменять индекс в массиве. Для строковых же был применен иной подход. Так как строковый кортеж представляет из себя двусвязный список, то у первого элемента этого списка в качестве указателя на предыдущий элемент мы указали ссылку на обычный кортеж, к которому принадлежит строчный. В момент удаления нам требуется просто поменять данную ссылку.

Так же так как основной структурой данных является документное дерево, то при удалении элемента было необходимо удалять его дочерние элементы. Для этого был реализован алгоритм поиска в глубину:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Также необходимо было расширять массив индексов, для этого мы использовали тот же функционал, но перемещали элемент из начала в конец, тем самым расширяя массив.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

**Результаты:**

Для выполнения задач было:

* Созданы соответствующие структуры данных
* Описаны базовые операции с данными
* Создан публичный интерфейс для работы с операциями
* Созданы тестовые сценарии для демонстрации работоспособности и эффективности решения

Алгоритмическая сложность:

Вставка – O(1)

Обновление – O(1)

Удаление – O(m), где m – количество зависимых элементов

Поиск по id – O(1)

Поиск по родителю – O(1)

Поиск по полю - O(m), где m - количество представленных элементов данных выбираемого вида

Изображение выглядит как текст, снимок экрана, График, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Изначально было 1000 элементов, каждый зависит от каждого. Чем меньше зависимых элементов, тем выше скорость удаления.

Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, График, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Чем больше элементов, тем ниже скорость поиска.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

**Выводы:**

Тесты показали, что предложенная структура удовлетворяет требованиям производительности.

Такие показатели достигаются благодаря идеи хранения массива индексов – отступов. Это позволяет за константу находить элемент по id. Также добавление занимает константное время, так как элемент добавляется в конец. Идея с разделением кортежей на строковые и обычные даёт потенциальную возможность хранить в файле строки любого размера. Есть один минус данной реализации – при перемещении кортежей необходимо обновлять индексы.