

Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.01 Информатика и вычислительная техника
Дисциплина «Низкоуровневое программирование»

Отчет

По лабораторной работе №1

Вариант 1

Выполнил:

Казаченко Р. О.

Преподаватель:

Кореньков Ю. Д.

Санкт-Петербург, 2022 г.

Цели:

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 с операциями добавления, обновления, удаления данных.
4. Реализовать тестовую программу для демонстрации работоспособности решения
5. Построить графики зависимости скорости выполнения операций и размера файлов в зависимости от количества элементов в файле.

Описание работы:

Программа представляет собой консольное приложение, позволяющее хранить, редактировать и просматривать данные в формате документного дерева. Каждый узел дерева задан определенным паттерном, присущим всему файлу.

В программе присутствует несколько модулей:

- Filetools – содержит низкоуровневые функции для работы с памятью.
- Generator – содержит функции для инициализации ключевых структур документного файла.
- Interface – содержит базовые CRUD-функции и высокоуровневые интерфейсы для работы с ними из интерактивного режима.
- Ui – содержит интерфейс для пользователя и обертки команд.
- Commands – содержит функции пользовательских команд.
- Tools – содержит вспомогательные функции (для работы со строками).

Примеры использования:

- Заполнение паттерна (*nix)

```
Initializing pattern.
Input the number of fields in pattern:2
2
--- Field 0 ---
Enter field name:code
code
0. Boolean
1. Integer
2. Float
3. String
Choose field type:1
1
```

```
--- Field 1 ---
Enter field name:name
name
0. Boolean
1. Integer
2. Float
3. String
Choose field type:3
3
File opened successfully!
Type 'help' for available commands info.
```

- Добавление и поиск элемента (win*)

```
add 123 code=1.32 name=Donald
find_by field code 1.32
--- FIND RESULT ---
id: 495
```

- Вывод доступных команд (*nix)

```
help
add <parent_id> <key1>=<value1> <key2>=<value2> ...
Adds the specified node to tree. Given arguments must match the full pattern.

update <node_id> <key1>=<upd_value1> <key2>=<upd_value2> ...
Updates some (one or more) fields of specified node.

remove <node_id>
Removes specified node with all descendants.

find_by field <field name> <value>
Finds node(s) with specified field.

find_by id <id>
Finds node with specified id.

find_by parent <parent id>
Finds children of specified parent.

print header/array
Prints to stdout fields of header or array instances.

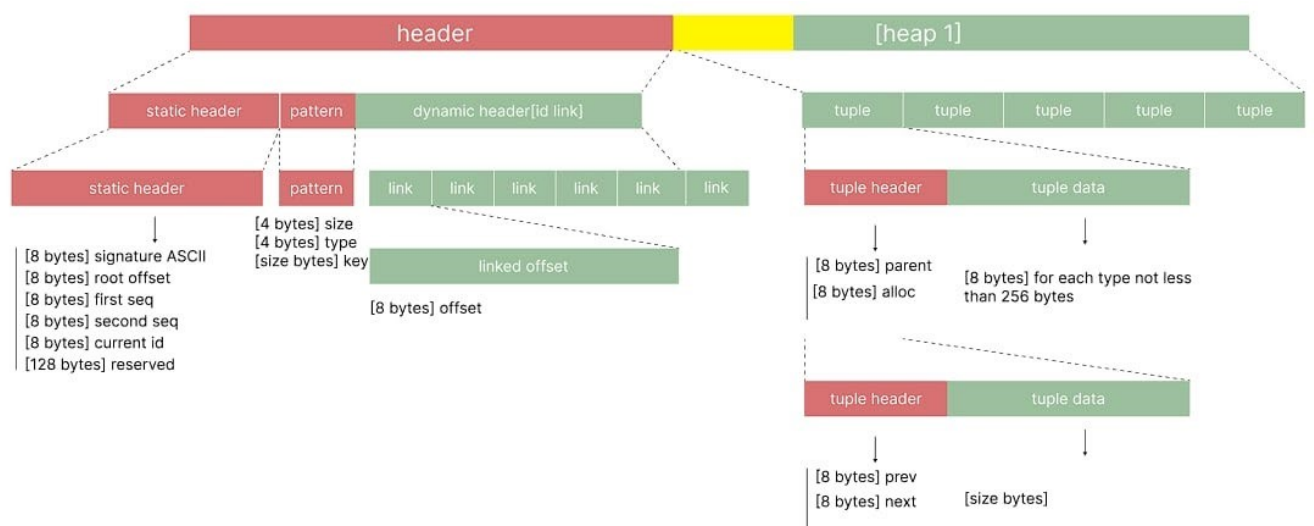
exit
Exit the program.
```

- Вывод массива элементов (win*)

```
code 11.541108
name exbceumugmkrv
--- TUPLE 492 ---
code 44.365032
name tjqeoqzaxhydpowon
--- TUPLE 493 ---
code 29.844508
name fbcpvve
--- TUPLE 494 ---
code 3.095983
name piocacxnx
--- TUPLE 495 ---
code 1.320000
name Donald
EXIT CODE H
```

```
--- SUBHEADER ---
ASCII Signature: 65534
Root Offset: 0
First Sequence: 0
Second Sequence: 0
Current ID: 496
Pattern Size: 2
--- PATTERN ---
Key 8 [Type 2]: code
Key 8 [Type 3]: name
--- ID ARRAY ---
3aae0 3ad00
3c020 3c240
3cce0 3cf00
3e220 3e440
3eee0 3f100
40420 40640
410e0 41300
```

Аспекты реализации



Файл делится на две части: заголовок и массив кортежей с данными. Каждому кортежу соответствует идентификатор, который хранит отступ до его кортежа от начала файла, что позволяет за $O(1)$ находить любой элемент по id. Сами кортежи имеют фиксированный размер и бывают двух типов: обычные и «строковые». Обычный кортеж хранит отступ до родительского и непосредственно массив данных кортежа, в

котором все типы данных, кроме строкового, лежат в явном виде. На строковое поле там будет лежать только ссылка, ссылка уже на «строковый» кортеж, который полностью посвящен хранению строки непосредственно (набор «строковых» кортежей одной строки представляет собой двусвязный список). Каждому элементу данных соответствует минимум один обычный кортеж и N «строковых», в зависимости от наполнения кортежа.

Добавление элемента реализовано максимально эффективно: узел просто добавляется в конец файла, а его идентификатор – в конец массива идентификаторов. Это достигается за счет того, что задача дефрагментации декларируется операции удаления.

Удаление элемента происходит следующим образом: по идентификатору находим нужный элемент в массиве узлов и удаляем его и все связанные с ним «строковые» узлы. Особенность в том, что каждая подоперация удаления сопровождается единичной дефрагментацией – на место удаленного узла помещается последний узел файла – за счет чего «дырок» в массиве узлов не будет. При удалении любого элемента также рекурсивно удаляются все его дочерние узлы.

Обновление элемента не представляет из себя ничего особенного – просто обновляем поля внутри необходимого узла. Однако если необходимо обновить строковое поле, то может понадобиться добавить дополнительные «строковые» узлы, если места в старых не хватит.

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

Пример реализации в коде:

```
59  /**
60   * Заголовок кортежа
61   */
62  union tuple_header {
63      struct {
64          uint64_t parent;
65          uint64_t alloc;
66      };
67      struct {
68          uint64_t prev;
69          uint64_t next;
70      };
71  };
```

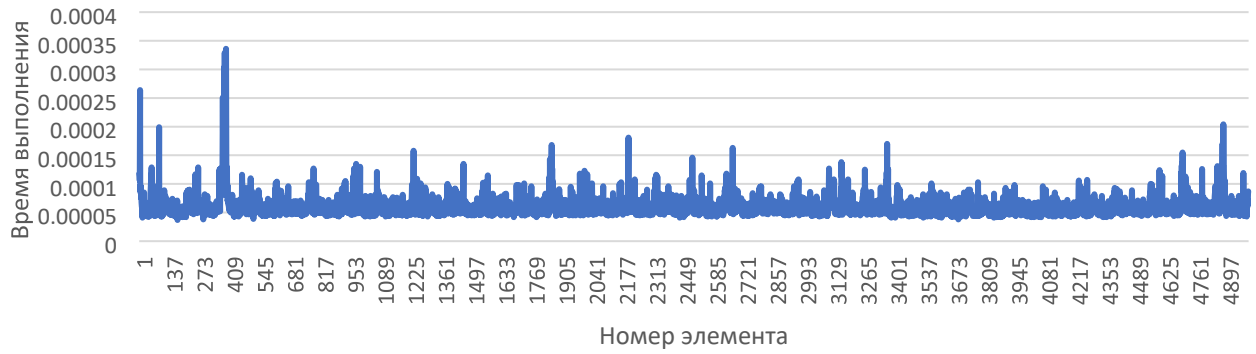
```
73  /**
74   * Кортеж
75   */
76  struct tuple {
77      union tuple_header header;
78      uint64_t *data;
79  };
80
81  /**
```

Результаты:

1. Были спроектированы структуры данных.
2. Было спроектировано представление схемы данных и описаны базовые операции работы над ним.
3. Был реализован публичный интерфейс с операциями добавления, обновления, удаления данных.
4. Были реализованы тестовые сценарии для демонстрации производительности программы.
5. По результатам тестовых сценариев построены графики производительности.



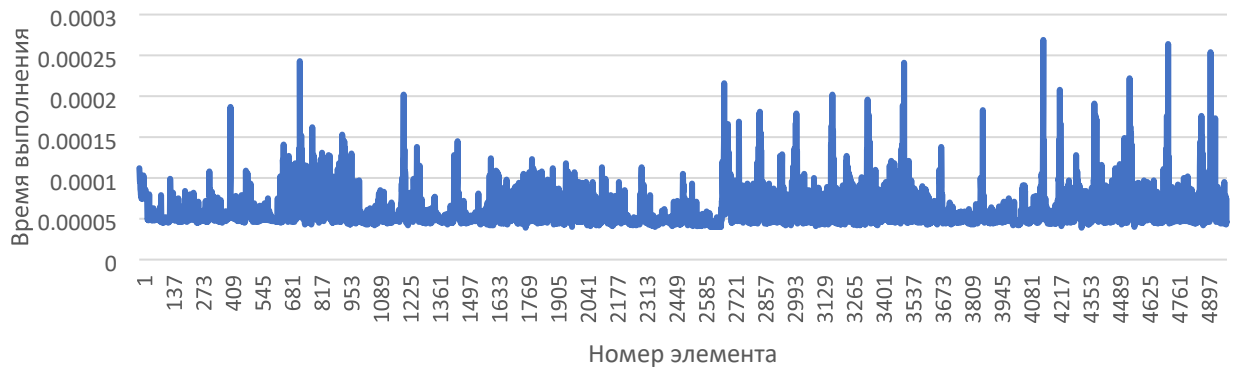
поиск элемента по id

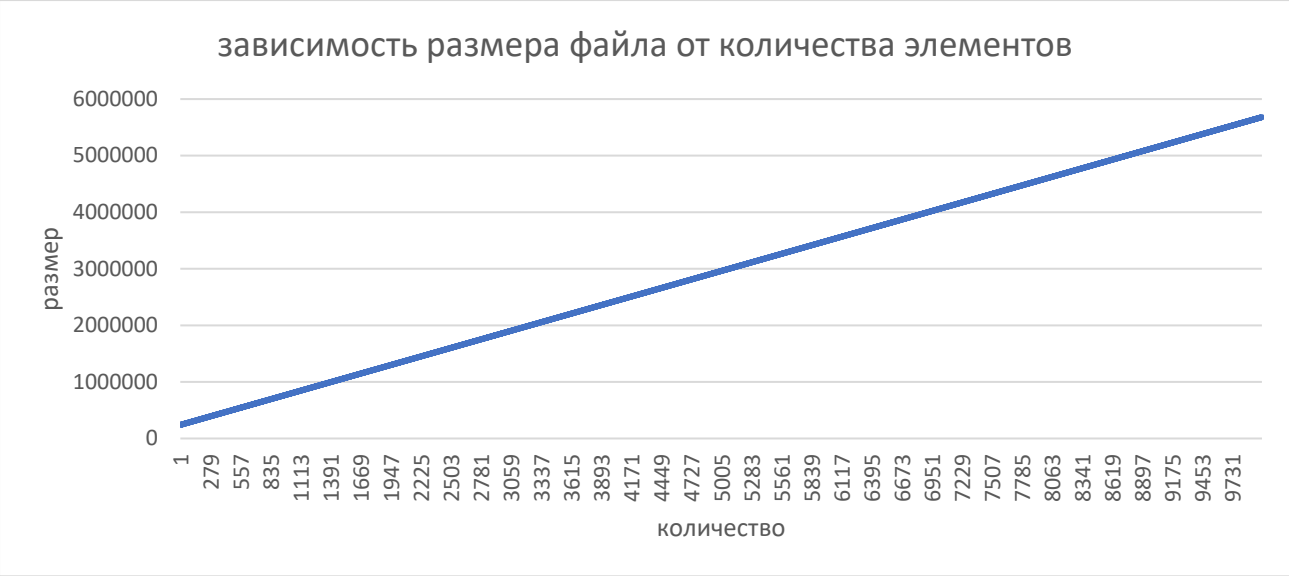


Поиск элемента по полю



Обновление элемента по id







Выводы

По результатам тестов видно, что предложенная структура удовлетворяет требованиям производительности. Хорошие показатели, в основном, благодаря концепции с хранением массива отступов (они же идентификаторы) до каждого кортежа, что способствует быстрому поиску элементов. Добавление элементов происходит за единицу, так как в любом случае добавление происходит в конец файла, так как вся работа по дефрагментации возлагается на операцию удаления. Благодаря гибкому хранению строк мы в теории можем уместить в узел строку любой длины – для нее просто будут создано много «строковых» кортежей, которые просто хранят ссылки друг на друга. Однако у такой реализации есть и проблема – при любых перемещениях кортежей в памяти приходится перезаписывать все ссылки на этот кортеж во всех связанных с ним.