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 древа. 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.
```

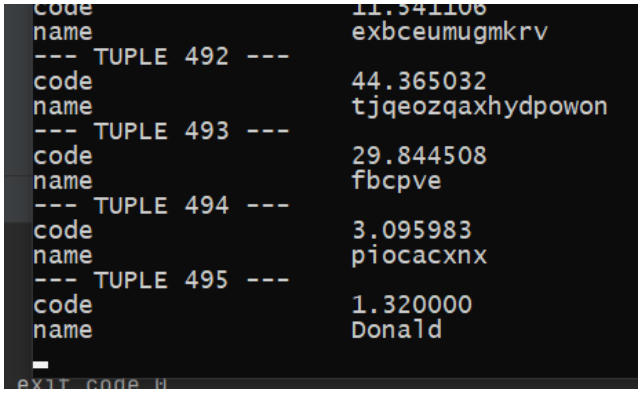
- Добавление и поиск элемента (win*)

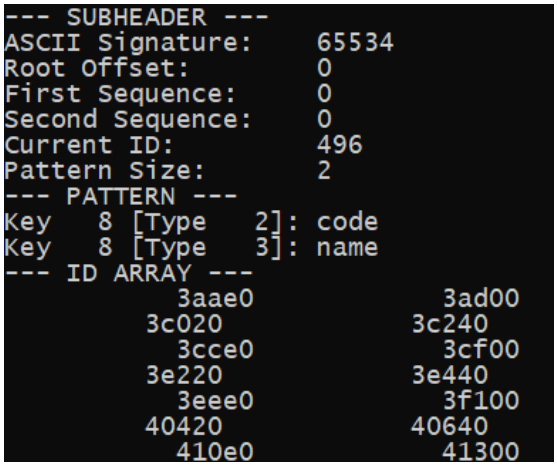
```
add 123 code=1.32 name=Donald
find_by field code 1.32
--- FIND RESULT ---
id: 495
```

- Вывод доступных команд (*nix)

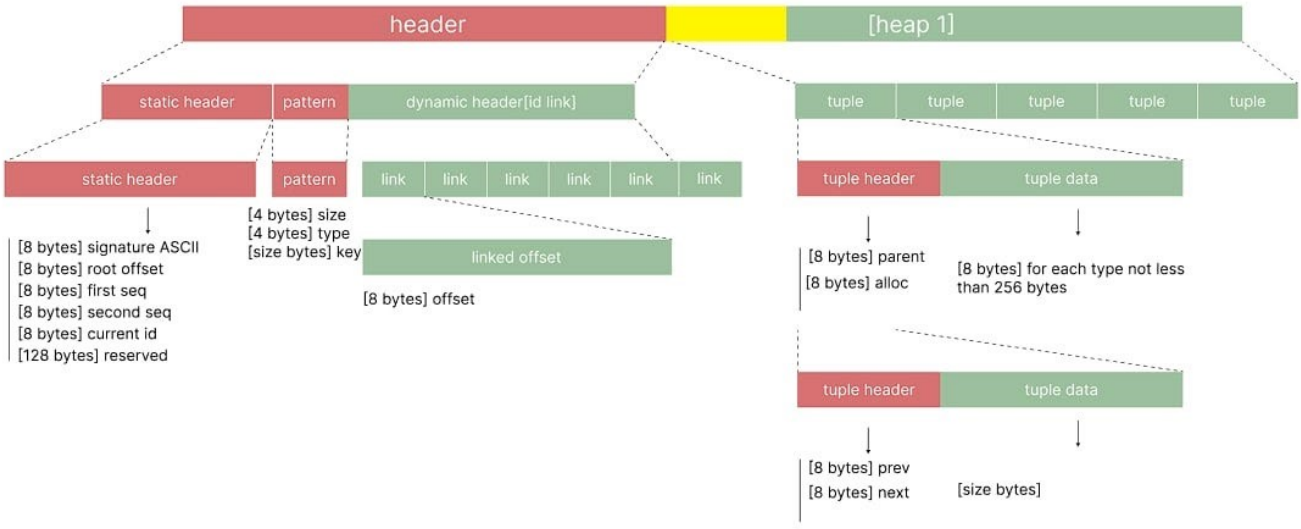
```
help
add <parent_id> <key1>=<value1> <key2>=<value2> ...
Adds the specified node to tree. Given arguments must match the full pattern.

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*)





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:

```c
/**
 * Заголовок кортежа
 */
union tuple_header {
    struct {
        uint64_t parent;
        uint64_t alloc;
    };
    struct {
        uint64_t prev;
        uint64_t next;
    };
};
```

```c
/**
 * Кортеж
 */
struct tuple {
    union tuple_header header;
    uint64_t *data;
};

/**
```

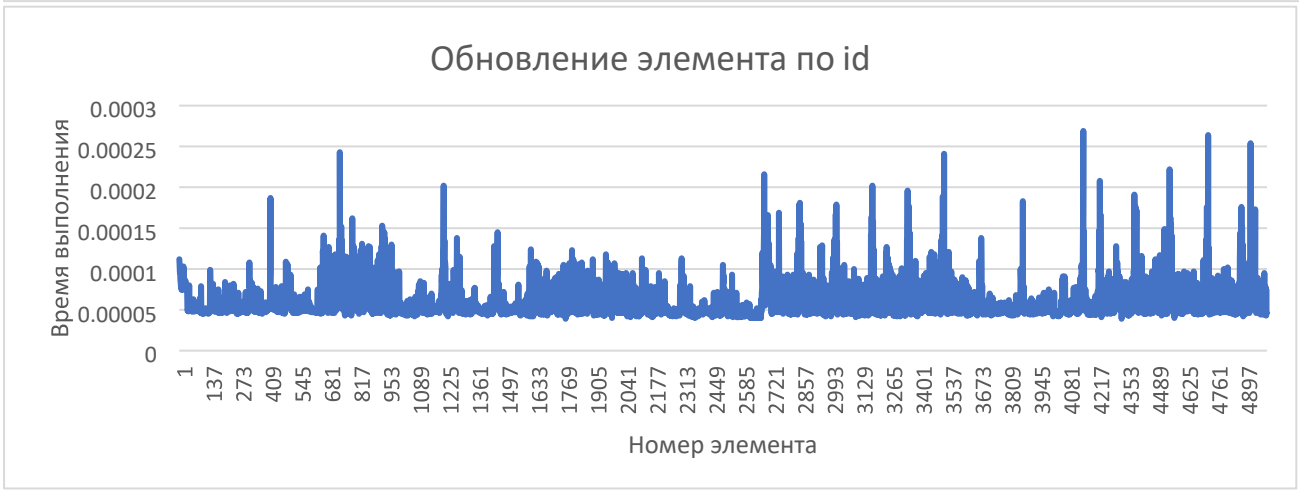
Результаты:

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.

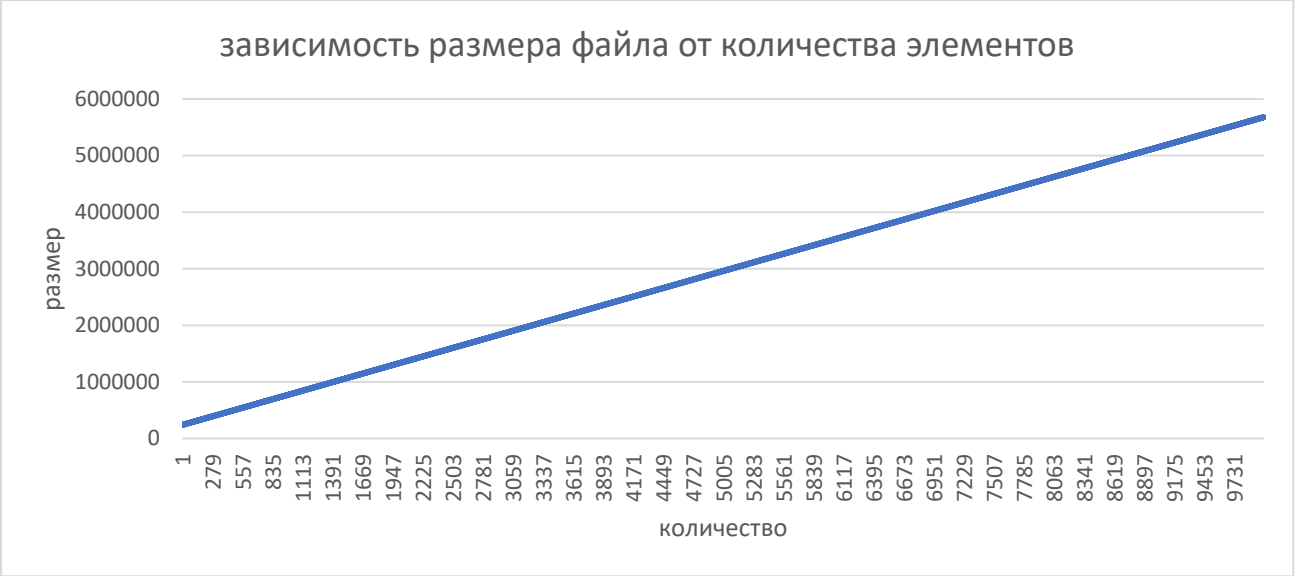

Добавление элемента
Время выполнения
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
1
133
265
397
529
661
793
925
1057
1189
1321
1453
1585
1717
1849
1981
2113
2245
2377
2509
2641
2773
2905
3037
3169
3301
3433
3565
3697
3829
3961
4093
4225
4357
4489
4621
4753
4885
Номер элемента

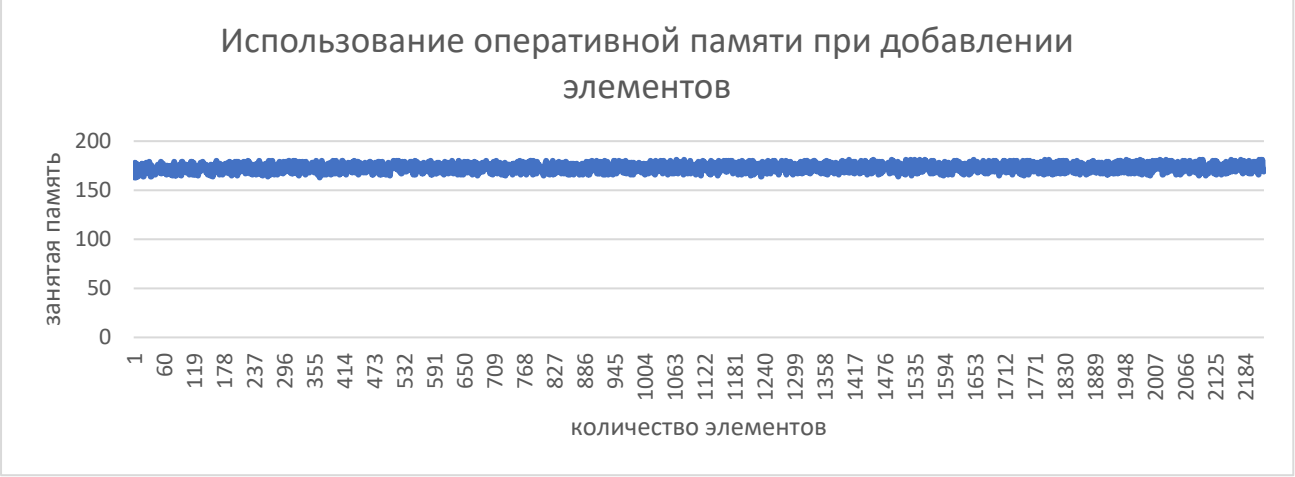

поиск элемента по id
Время выполнения
0.0004
0.00035
0.0003
0.00025
0.0002
0.00015
0.0001
0.00005
0
1 137 273 409 545 681 817 953 1089 1225 1361 1497 1633 1769 1905 2041 2177 2313 2449 2585 2721 2857 2993 3129 3265 3401 3537 3673 3809 3945 4081 4217 4353 4489 4625 4761 4897
Номер элемента


Поиск элемента по полю
Время выполнения
0.005
0.004
0.003
0.002
0.001
0
1 133 265 397 529 661 793 925 1057 1189 1321 1453 1585 1717 1849 1981 2113 2245 2377 2509 2641 2773 2905 3037 3169 3301 3433 3565 3697 3829 3961 4093 4225 4357 4489 4621 4753 4885
Номер элемента


Обновление элемента по id
Время выполнения
0.0003
0.00025
0.0002
0.00015
0.0001
0.00005
0
1 137 273 409 545 681 817 953 1089 1225 1361 1497 1633 1769 1905 2041 2177 2313 2449 2585 2721 2857 2993 3129 3265 3401 3537 3673 3809 3945 4081 4217 4353 4489 4625 4761 4897
Номер элемента


удаление элементов с n зависимостями
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49


зависимость размера файла от количества элементов
6000000
5000000
4000000
3000000
2000000
1000000
0
размер
1 279 557 835 1113 1391 1669 1947 2225 2503 2781 3059 3337 3615 3893 4171 4449 4727 5005 5283 5561 5839 6117 6395 6673 6951 7229 7507 7785 8063 8341 8619 8897 9175 9453 9731
количество


Использование оперативной памяти при добавлении элементов
200
150
100
50
0
занятая память
1 60 119 178 237 296 355 414 473 532 591 650 709 768 827 886 945 1004 1063 1122 1181 1240 1299 1358 1417 1476 1535 1594 1653 1712 1771 1830 1889 1948 2007 2066 2125 2184
количество элементов

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.