НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ КОРПОРАЦИЯ ИТМО

Факультет программной инженерии и компьютерной техники

Низкоуровневое программирование

Лабораторная работа № 1

Вариант: 1

Выполнил
студент

Нестеров Иван Алексеевич

Группа P33302

Преподаватель: Кореньков Юрий Дмитриевич

г. Санкт-Петербург

2022

Задание:

Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объемом от 10 GB, соответствующих варианту вида «документное дерево». Спроектировать структуры для представления информации, реализовать публичный интерфейс, предоставляющий операции обновления, удаление, добавления данных, проверить решение и построить графики, демонстрирующие скорость выполнения операций и потребление оперативной памяти в зависимости от объема файла с данными.

Ход работы:

В ходе работы была реализована указанная программа, код был опубликован на GitHub.

Был написан makefile, позволяющий собрать программу на операционных системах семейства UNIX и на Windows. Сборка и запуск осуществляются следующим образом.

- `make clean build`
- `./main <flag> <file> <old-file>` или `./main -o <file> <old-file>`

Для операционной системы Windows рекомендуется использовать обратный слеш, однако при тестировании command prompt заменяла слеш на нужный при автодополнении команды сама.

Флаги:

1. -c – для создания нового файла и работы с ним
2. -o – для работы с существующим файлом (в <file> будут скопированы данные из <old_file>

Файл с данными состоит из двух частей: заголовка и непосредственно данных, которые представляют собой массив кортежей элементов. В заголовке хранится информация о типе элементов, названии их полей, кроме того, оттуда программа берет указатель на память, где хранятся сами данные. У каждого кортежа есть идентификатор, хранящий смещение от начала файла до него самого. Данные всех типов хранятся в этих кортежах непосредственно, кроме строк, для которых заведен отдельный кортеж для реализации возможности хранить там строки произвольной длины (а, значит, и «веса»). Размеры остальных типов данных (целые, нецелые числа и логические значения) фиксированы.

Реализация операций:

1. Вставка. В конец файла добавляется новый узел, а его идентификатор добавляется в конец соответственно массива идентификаторов;
2. Обновление. Реализовано в виде обновления полей внутри узла (перезапись старого значения новым);
3. Выборка: итерация по массиву идентификаторов в поиске нужного;
4. Удаление: нахождение узла по его ID, нахождение всех связанных с ним строковых узлов (которые хранятся в другом кортеже), удаление их всех.

В описанных массивах не будет «дыр», так как при удалении элементов (из-за того что элементы занимают одинаковое кол-во памяти), осуществлена возможность взять крайний правый элемент массива, и передвинуть его на место возникшей после удаления узла

«дыры». Поэтому данные в файле хранятся плотно, что позволяет сделать вывод о том, что количество «мусорных» байт в нем стремится к минимуму.

Приведем графики, отражающие скорость описанных выше операций и расхода памяти:

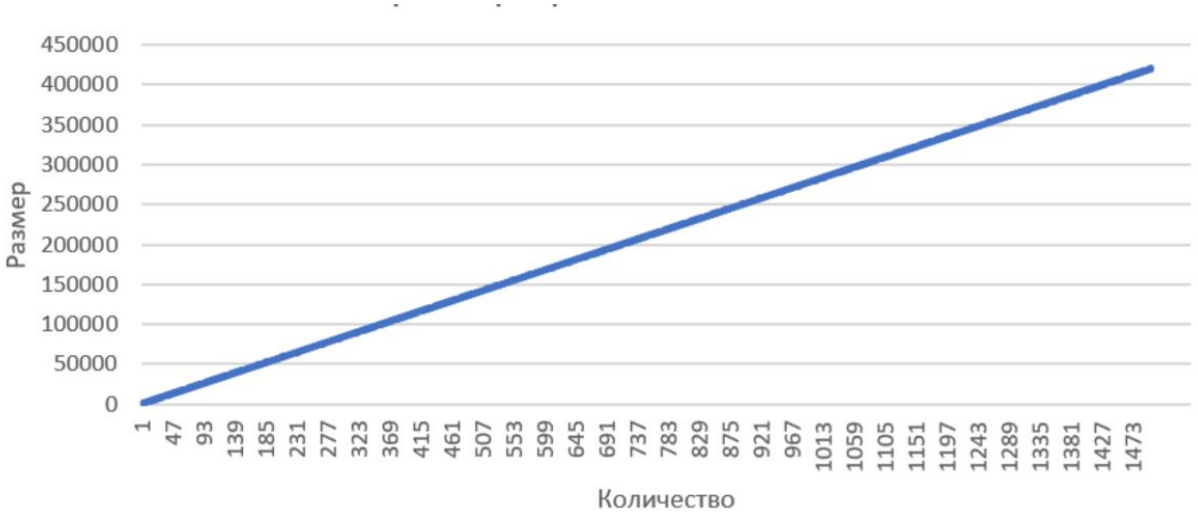


Рисунок 1. Зависимость размера файла от кол-ва элементов данных в нем

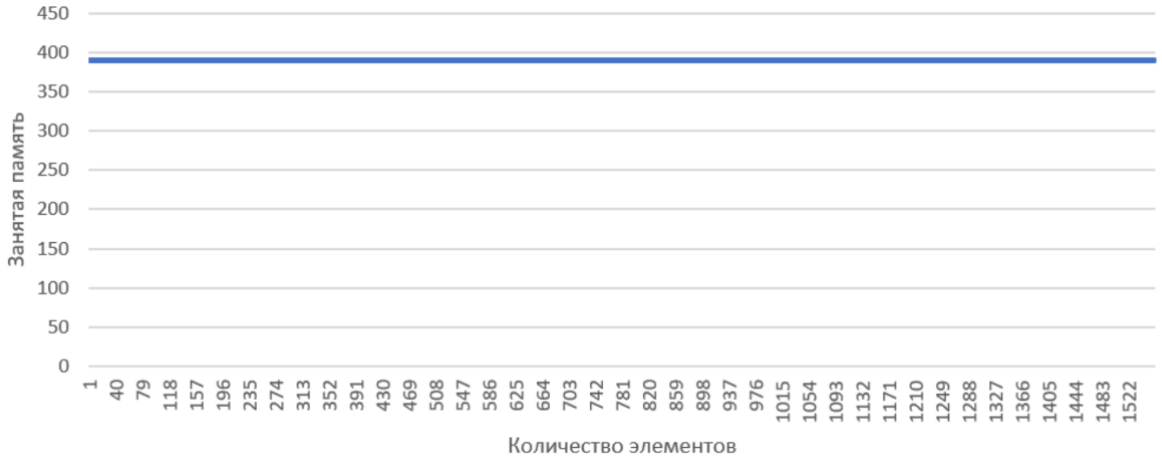


Рисунок 2. Потребление оперативной памяти

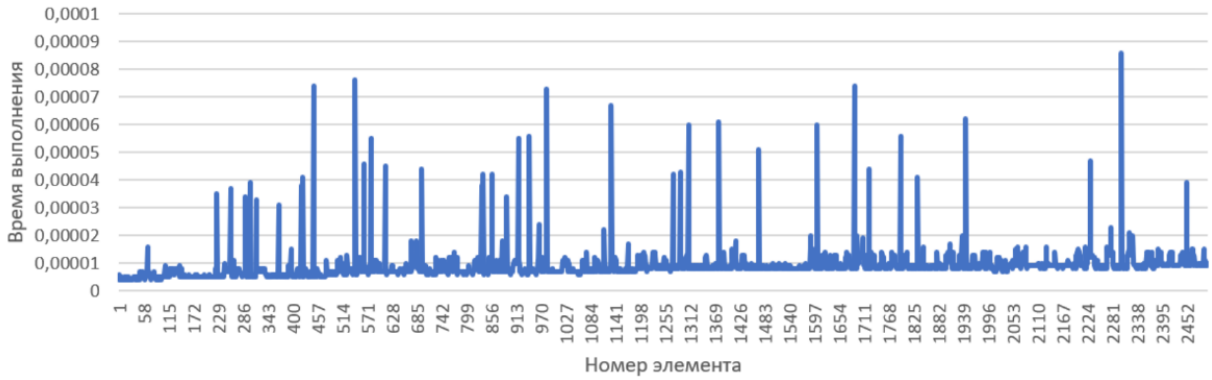


Рисунок 3. Выборка элемета

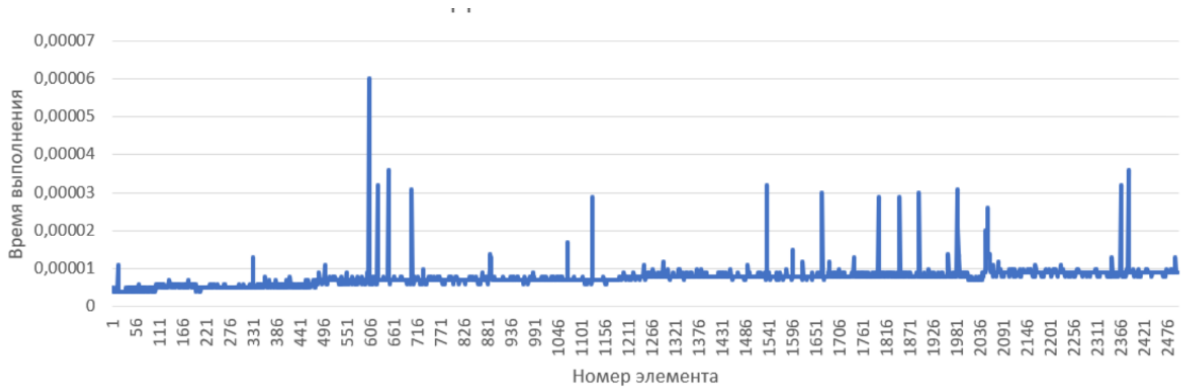


Рисунок 4. Вставка элемента

Рисунок 5. Выборка элемента с использованием родителя в дереве

Рисунок 6. Выборка элемента по значению поля

Рисунок 7. Обновление элемента (с использованием ID)

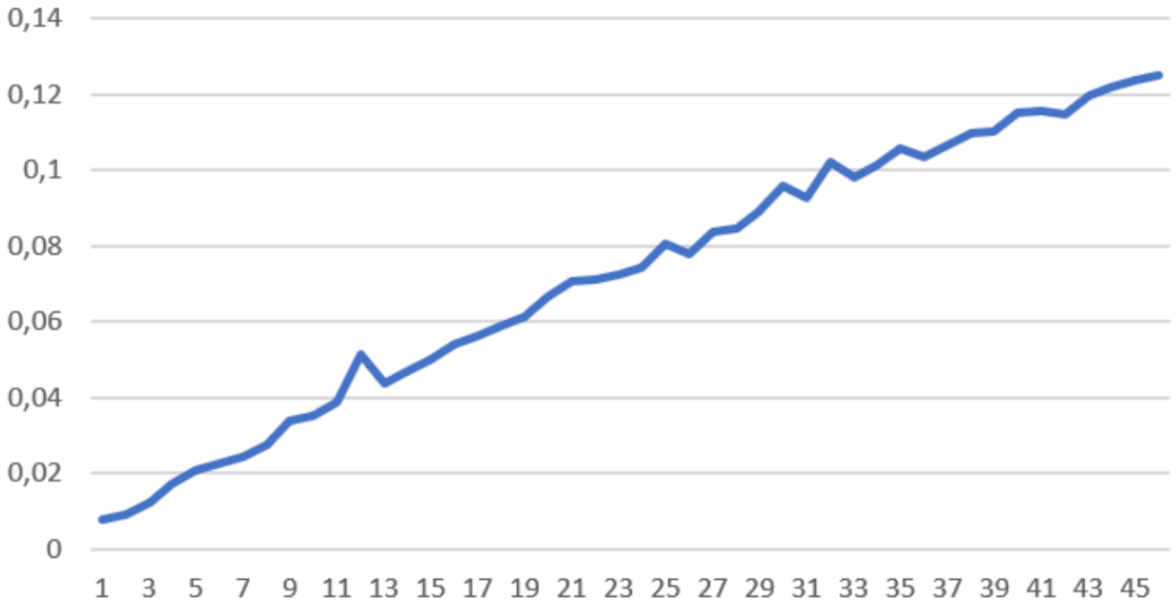


Рисунок 8. Удаление элемента с анализом зависимостей из дерева

Вышеуказанные графики были построены в Excel, а замер времени производился при помощи стандартной функции clock() языка C.

Демонстрация работы:

Запуск программы:

```
C:\Users\Иван Нестеров\Documents\GitHub\low-level-programming\lab1>main.exe -c tree.bin
### Welcome to my program! ###
Type the number of fields in data(int):4
### Field 0  ###
Input the name of the field:int
Choose type:
0. Boolean
1. Integer
2. Float
3. String
1
### Field 1  ###
Input the name of the field:bool
Choose type:
0. Boolean
1. Integer
2. Float
3. String
0
### Field 2  ###
Input the name of the field:float
Choose type:
0. Boolean
1. Integer
2. Float
3. String
2
### Field 3  ###
Input the name of the field:str
Choose type:
0. Boolean
1. Integer
2. Float
3. String
3
+++ File initiated successfully! +++
Enter 'help' to see what can you do
Enter 'exit' to exit the program, if you're don't wanna do something
Enter the command:
```

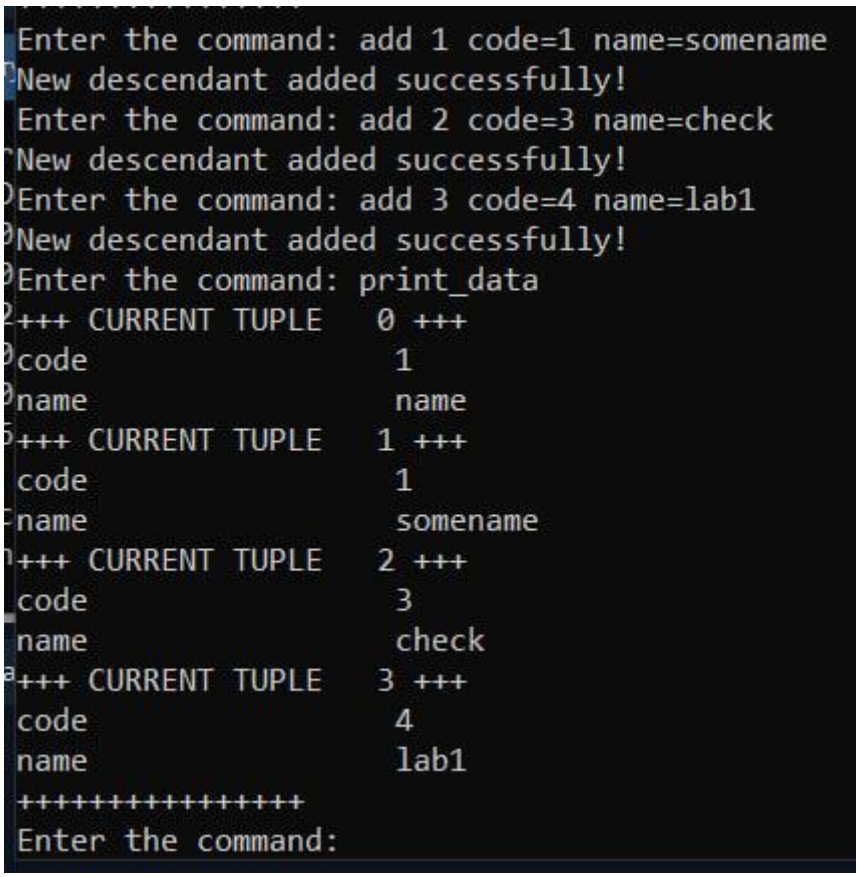
Проверка команд help и exit:

```
Enter the command: help
++++++++++++++++++++++++++++++++++++++
'DATA PRINTING'
PRINTING CURRENT DATA FROM ARRAY.

'HEADER PRINTING'
HEADER FIELDS PRINTING

'syntax: add <parent_id> <key #1>=<value #1> <key #1>=<value #2> <key #3>=<value #3> etc...'
ADDING TO THE TREE NEW NODE WITH PARAMETERS WHICH WERE WRITTEN JUST NOW...

'syntax: update <id> <key #1>=<new_value #1> <key #2>=<new_value #2> <key #3>=<new_value #3> etc...'
UPDATING OF SELECTED FIELD OF FOUND NODE...

'syntax: delete <node_id>'
DELETING THE NODE WITH GIVEN ID. CHILDREN OF THIS NODE WILL BE DELETED TOO...

'syntax: select_by field <field name> <field value>'
FINDING NODE VIA GIVEN CONDITIONS...

'syntax: select_by id <id>'
FINDING THE NODE BY ID...

'select_by parent <parent id>'
Finding children of written parent.

'exit'
Exit the program.

++++++++++++++++++++++++++++++++++++++
Enter the command:
```

Проверка команды на вывод информации о заголовке файла:

```
++++++++++++++++
Enter the command: print_header
+++ CHILDREN HEADER DESCRIPTION +++
Current ID:         0
Root Offset:        0
Pattern Size:       2
First Sequence:     0
Second Sequence:    0
ASCII Signature:    65534
--- PATTERN ---
Key   8 [Type   1]: code
Key   8 [Type   3]: name
Enter the command:
```

Добавление элементов в файл и вывод информации о созданных узлах:



Вывод: в ходе работы была разработана программа, позволяющая хранить большое количество данных, организованных в виде документного дерева. Закреплены на практике навыки организации данных с указанными в задании ограничениями и работа с ними, с использованием построенных данных.