Цель

Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное
обновление) информации общим объёмом от 10GB соответствующего варианту вида.

Порядок выполнения

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: четырёхбайтовые 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 о 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 $O(n*m) > t \rightarrow O(n+m)$, где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных

f. Показать, что размер файла данных всегда пропорционален 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:

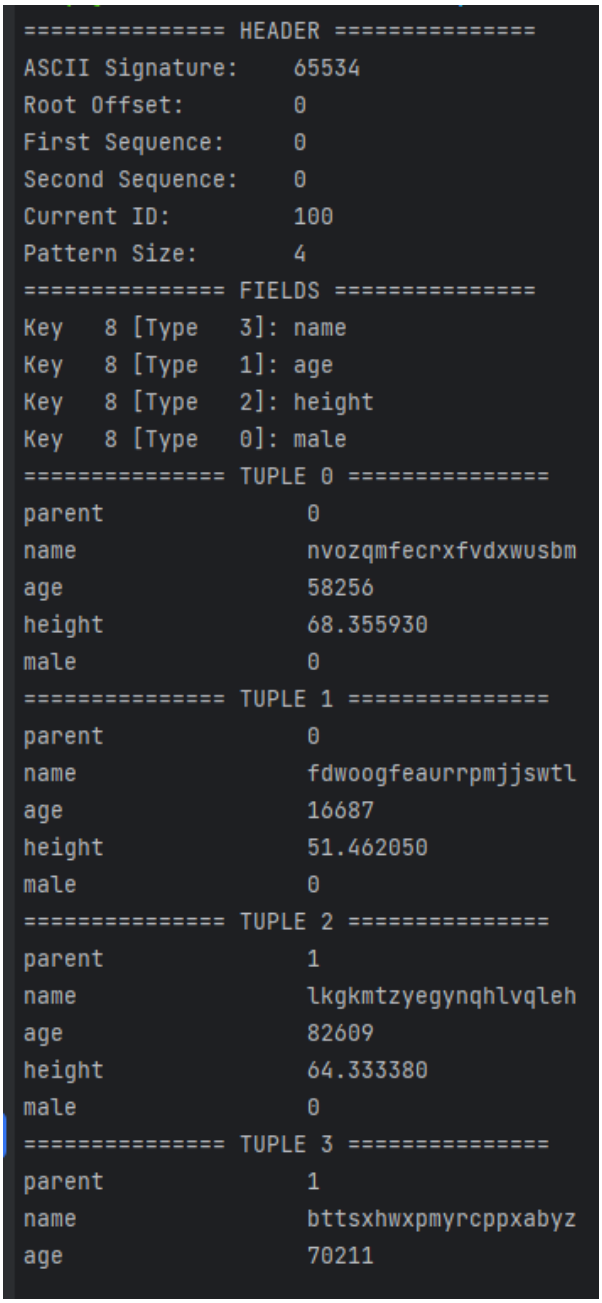
a. В части 3 привести описание структур данных, разработанных в соответствии с п.1

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

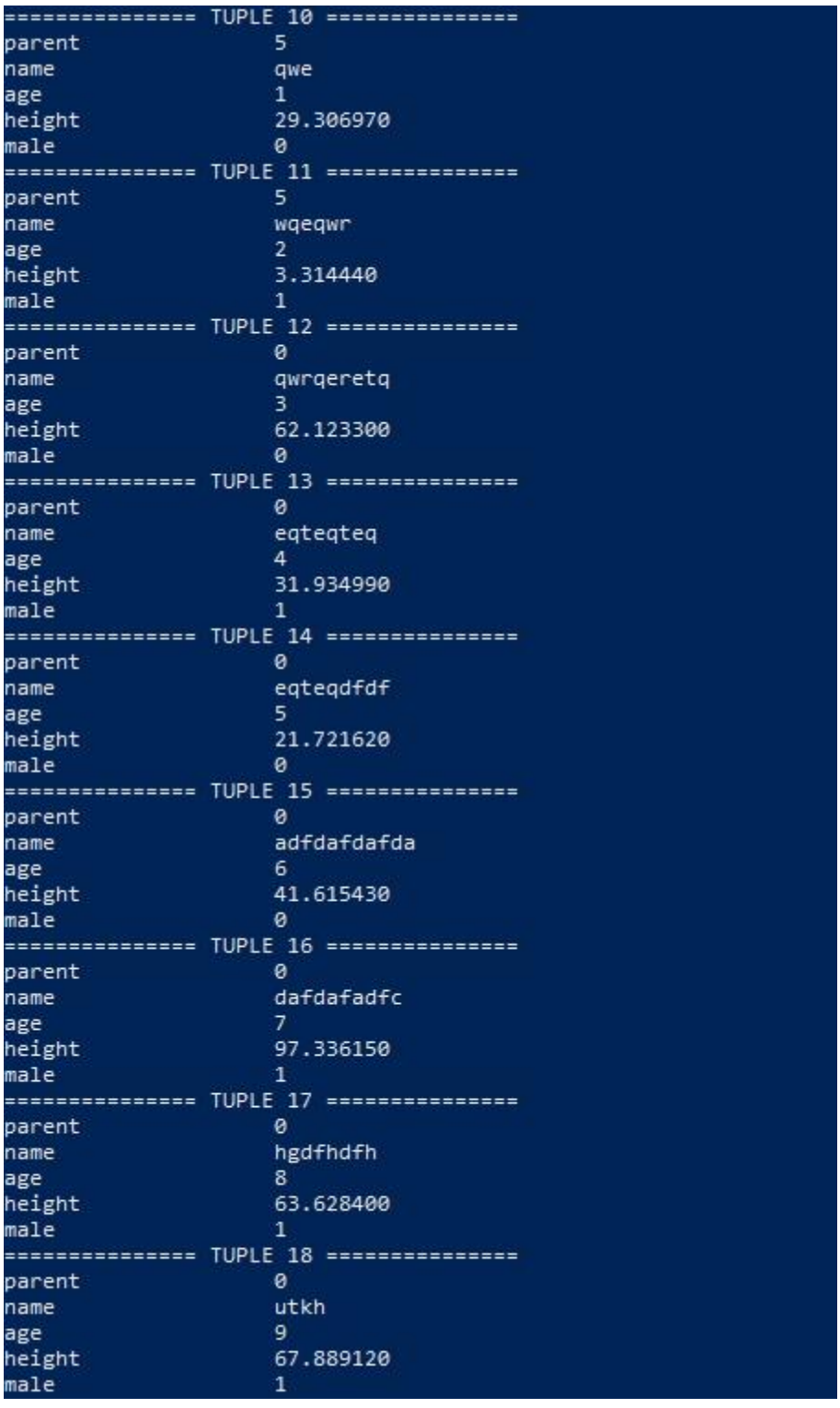
Пример работы программы

Linux



```
=============== HEADER ===============
ASCII Signature:    65534
Root Offset:        0
First Sequence:     0
Second Sequence:    0
Current ID:         100
Pattern Size:       4
=============== FIELDS ===============
Key   8 [Type   3]: name
Key   8 [Type   1]: age
Key   8 [Type   2]: height
Key   8 [Type   0]: male
=============== TUPLE 0 ===============
parent              0
name                nvozqmfecrxfvdxwusbm
age                 58256
height              68.355930
male                0
=============== TUPLE 1 ===============
parent              0
name                fdwoogfeaurrpmjjswtl
age                 16687
height              51.462050
male                0
=============== TUPLE 2 ===============
parent              1
name                lkgkmtzyegynqhlvqleh
age                 82609
height              64.333380
male                0
=============== TUPLE 3 ===============
parent              1
name                bttsxhwxpmyrcppxabyz
age                 70211
```

Windows –



Аспекты реализации

```c
/**
 * Контейнер, хранящий параметры файла
 * ASCII_signature - 0xFFFE или 0xFEFF символы для определения little- или big- endian порядок байтов
 * root_offset     - отступ корневого элемента
 * first_seq       - последовательность, увеличивающаяся при открытии файла
 * second_seq      - последовательность, увеличивающаяся при закрытии файла
 * cur_id          - последовательность для отслеживания текущего id
 * pattern_size    - количество полей в шаблоне вершин
 */
struct tree_subheader {
    uint64_t ASCII_signature;
    uint64_t root_offset;
    uint64_t first_seq;
    uint64_t second_seq;
    uint64_t cur_id;
    uint64_t pattern_size;
};
```

```c
/**
 * Контейнер для ключа шаблона
 */
#pragma pack(push, 4)
struct key_header {
    uint32_t size;
    uint32_t type;
};
struct key {
    struct key_header* header;
    char* key_value;
};
#pragma pack(pop)

/**
 * Полный заголовок файла
 * id_sequence - массив ссылок на кортежи
 */
struct tree_header {
    struct tree_subheader* subheader;
    struct key** pattern;
    uint64_t* id_sequence;
};
```

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

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

В нашем файле хранится хидер. Он содержит метаинформацию о БД. В нем хранится текущий id и последовательность id-шников. А так же название полей и их типы.

Есть проблема разноразмерности наших элементов из за строчного типа данных, поэтому было решено хранить его отдельно от элемента.

Например -> если у нас есть 3 элемента, мы удаляем 2, то на его местро мы сможем положить элемент только с такой же длиной строки, иначе никак. Остальные поля фиксированного размера. И чтобы не было фрагментации при удалении элемента мы на его место ставим последний в БД.

А доступ к эелементам мы получаем благодаря массиву айдишников, т е чтобы найти элемент по айди, мы ищем его айди в этом массиве и уже оттуда имеем отступ от начала файла, по которому лежит тот самый элемент с нужным айди.

Операции:

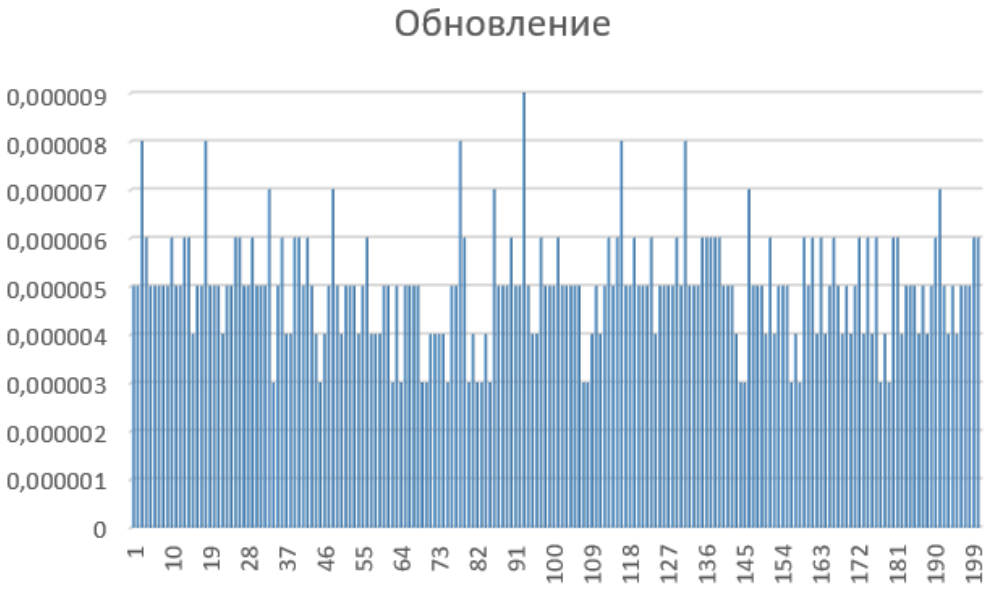
- Добавление – элемент добавляется в конец файла, и обновляется последовательность айдишников.
- Поиск по айди – мы ищем айди в последовательности и как находим берем нужный элемент.
- Поиск по полям – считываем элементы последовательно, смотрим и сравниваем поля с тем, которое нам нужно.
- Обновление по айди – находим элемент по айди, меняем нужное поле и кладем обратно.
- Удаление по айди – удаляем нужный кортеж, а на его место ставим последний, затем ищем всех его детей (рекурсивно).

Результаты

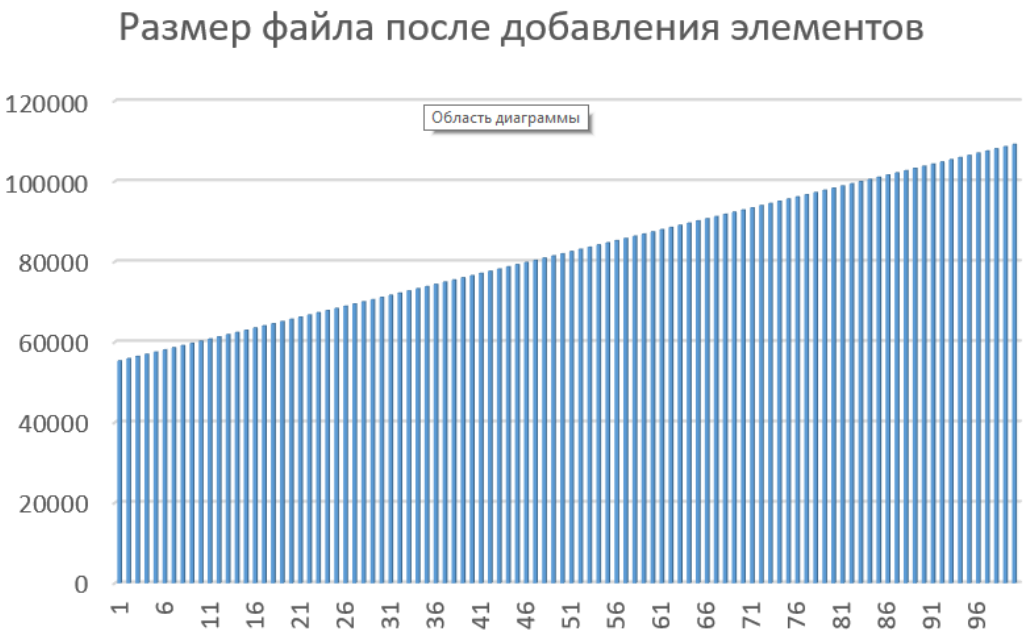
- Вставка O (1)
- Поиск по айди O (n)
- Обновление элемента O (n)
- Поиск по полю O (n)
- Удаление O (n*m)


Поиск по айди
0,000012
0,00001
0,000008
0,000006
0,000004
0,000002
0
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96


Поиск по полю
Область диаграммы
0,00035
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181
190
199


Обновление
0,000009
0,000008
0,000007
0,000006
0,000005
0,000004
0,000003
0,000002
0,000001
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181
190
199


Поиск по родителю
0,00045
0,0004
0,00035
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181
190
199


Размер файла после добавления элементов
Область диаграммы
120000
100000
80000
60000
40000
20000
0
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96


Оперативная память
120
100
80
60
40
20
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97

Вывод: в результате выполнения данной лабораторной работы мной была разработана эффективная программа, которая способна хранить данные, обеспечивая оптимальное использование памяти и высокую производительность. Один из важных моментов в решении было использование массива ссылок на соответствующие кортежи. Это позволило эффективно выполнять итерации по массиву ссылок, что в контексте данной задачи является константной операцией. Кроме того, добавление новых кортежей в конец файла также оказалось полезным решением.

Одна из ключевых особенностей структуры кортежной строки заключается в хранении длинных строк с помощью адресации следующего кортежа, который содержит их продолжение. Это позволяет уменьшить фрагментацию внутренней памяти, так как размер кортежа и длина последнего куска строки выбраны таким образом, чтобы достичь оптимального баланса между количеством ссылок в массиве указателей и фактическим размером кортежа.

Следует отметить, что недостатком данной модели является необходимость переписывания ссылок при обновлении элемента, особенно в случае, если его размер увеличивается. Однако этот недостаток можно минимизировать с помощью оптимизации алгоритма обновления, например, используя индексы, чтобы быстро находить нужную ссылку и выполнять обновление только в необходимых случаях.

В целом, разработанная программа предоставляет эффективное решение для хранения и обработки данных, при этом учитывая особенности и требования данной задачи.