|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| Факультет | «Информатика и системы управления» (ИУ) |
| Кафедра | «Программное обеспечение ЭВМ и информационные технологии» (ИУ7) |

**Лабораторная работа №6**

**“Деревья”**

**Вариант №8**

|  |  |
| --- | --- |
| Студент: Князев Дмитрий Юрьевич, группа ИУ7-33Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |
| Преподаватель: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Оглавление

Условия задачи 3

Входные и выходные данные 4

Аварийные ситуации и особенности реализации 5

Структуры данных 6

Сравнение эффективности 7

Вывод 13

Контрольные вопросы 14

**Условия задачи:**

Реализовать следующие функции:

* Построение АВЛ дерева и двоичного дерева поиска на основе текстового файла, содержащего целые числа.
* Вывод деревьев на экран.
* Определение количества узлов деревьев на каждом уровне.
* Добавить число в деревья и в файл.
* Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов.
* Сравнить время добавления и поиска чисел в ДДП, АВЛ дерево и файл.

**Входные данные:**

* Пункт меню (число от 0 до 8 включительно)
* В зависимости от выбранного пункта меню:
  + Файл с последовательностью целых чисел
  + Целое число

**Выходные данные (в зависимости от выбранного пункта меню):**

* Графическое изображение АВЛ дерева или ДДП в формате .svg файла в отдельном окне путём вызова xdg-open
* Количество узлов с определённой высотой в АВЛ дереве или ДДП
* Информация об успешном или неудачном удалении, добавлении, поиске, загрузке элементов из файла в АВЛ дерево или ДДП
* Результаты сравнения эффективности алгоритмов сортировки и поиска с использованием ДДП, АВЛ дерева и файла

**Функции меню**

0. Выход

1. Загрузить числа из файла в АВЛ дерево и в ДДП

2. Вывести ДДП

3. Вывести АВЛ дерево

4. Добавить элемент в АВЛ дерево, в ДДП и в файл

5. Удалить элемент из АВЛ дерева, из ДДП и из файла

6. Найти элемент в АВЛ дереве и в ДДП

7. Определить количество узлов АВЛ дерева и ДДП на каждом уровне

8. Сравнить эффективность алгоритмов сортировки и поиска с использованием ДДП, АВЛ дерева и файла

**Аварийные ситуации:**

1. Считываемый файл кроме чисел и переносов строк содержит другие символы

**Особенности реализации:**

1. Если файл содержит несколько одинаковых значений, они будут перезаписаны

2. Имена используемых файлов находятся в заголовочных файлах:

|  |
| --- |
| #define DATABASE\_FILENAME "database/numbers.txt" |

В файле presets.h

|  |
| --- |
| #define ITERATIONS 10  #define SMALL\_FILE\_NAME "database/numbers\_small.txt"  #define MEDIUM\_FILE\_NAME "database/numbers\_medium.txt"  #define BIG\_FILE\_NAME "database/numbers\_big.txt"  #define SORTED\_FILE\_NAME "database/numbers\_sorted.txt"  #define SORTED\_REVERSE\_FILE\_NAME "database/numbers\_sorted\_reverse.txt" |

В файле efficiency.h

**Структуры данных:**

|  |
| --- |
| **typedef** **struct** tree\_node  {  **int** value;  **unsigned** **char** height;  **struct** tree\_node \*left;  **struct** tree\_node \*right;  } **tree\_node\_t**; |

Структура узла дерева

value — значение узла, информация пользователя дерева

height — высота узла, игнорируется в функциях для работы с деревом двоичного поиска

left — указатель на левое поддерево

right — указатель на левое поддерево

|  |
| --- |
| **typedef** **struct** array  {  **int** \*data;  **size\_t** size;  **size\_t** max\_size;  } **array\_t**; |

Структура массива чисел, используемая для обработки файла

data — указатель на динамический массив, хранящий числа из файла

size — текущий размер массива

max\_size — максимальный размер массива

**Сравнение эффективности**:

Сортировка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Количество элементов | АВЛ дерево | Дерево двоичного поиска | Файл |
| Маленький файл | 200 | 97 | 50 | 71 |
| Средний файл | 20000 | 15034 | 8736 | 7470 |
| Большой файл | 200000 | 246266 | 160422 | 82270 |
| Отсортированный файл | 20000 | 8001 | 1146641 | 3127 |
| Файл, отсортированный в обратном порядке | 20000 | 7082 | 1039986 | 2921 |

Поиск

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Количество элементов | АВЛ дерево | Дерево двоичного поиска | Файл |
| Маленький файл | 200 | 3 | 3 | 55 |
| Средний файл | 20000 | 413 | 454 | 4223 |
| Большой файл | 200000 | 11482 | 12623 | 40691 |
| Отсортированный файл | 20000 | 123 | 170 | 2516 |
| Файл, отсортированный в обратном порядке | 20000 | 128 | 179 | 2344 |

Сортировка

Для самого большого файла (на 200000 элементов) создание АВЛ дерева оказалось на 53% дольше создания дерева двоичного поиска и на 199%дольше сортировки в файле.

Дерево двоичного поиска на основе отсортированного файла (на 20000 элементов) создавалось в 143 раза дольше АВЛ дерева и в 366 раз дольше, чем массив чисел на основе того же файла.

Дерево двоичного поиска на основе отсортированного в обратном порядке файла (на 20000 элементов) создавалось в 146 раз дольше АВЛ дерева и в 356 раз дольше, чем массив чисел на основе того же файла.

Поиск

Для самого большого файла (на 200000 элементов) поиск в файле оказался на 254% дольше поиска в дереве двоичного поиска П и на 222% дольше поиска в двоичном дереве поиска.

Для отсортированного в обратном порядке файла (на 20000 элементов) поиск в файле в 20 раз дольше поиска в АВЛ дереве и в 15 раз дольше, чем поиск в дереве двоичного поиска.

Для отсортированного в обратном порядке файла (на 20000 элементов) поиск в файле в 18 раз дольше поиска в АВЛ дереве и в 13 раз дольше, чем поиск в дереве двоичного поиска.

Память

Для файла из 200 элементов:

Размер дерева в байтах: 24 \* 200 = 4800

Размер массива в байтах: 24 + 4 \* 200 = 824

То есть для данного файла дерево занимает в 5.8 раз больше места

Для файла из 20000 элементов:

Размер дерева в байтах: 24 \* 20000 = 480000

Размер массива в байтах: 24 + 4 \* 20000 = 80024

То есть для данного файла дерево занимает в 5.9 раз больше места

**Вывод**

Поиск в АВЛ дереве происходит быстрее, чем в дереве двоичного поиска и быстрее, чем поиск в файле

Высота АВЛ дерева в среднем меньше высоты ДДП, поиск в первом осуществляется быстрее, что подтверждается экспериментально.

Создание АВЛ дерева дольше, чем создание ДДП и больше, чем создание и сортировка массива из файла

Деревья занимают почти в 6 раз больше памяти, чем массив, создаются они дольше, но поиск в них происходит быстрее на 200-250%.

**Контрольные вопросы:**

1. Что такое дерево?

Дерево – это нелинейная структура данных, используемая для представления

иерархических связей, имеющих отношение «один ко многим».

2. Как выделяется память под представление деревьев?

Память (динамическая или статическая) выделяется отдельно под каждый узел, образуя связный список.

3. Какие бывают типы деревьев?

Двоичные деревья (обычное. АВЛ дерево, красно-чёрное дерево), Сильно ветвящиеся (B-дерево, 2-3 дерево)

4. Какие стандартные операции возможны над деревьями?

Обход дерева с применением операции к элементам, поиск элемента в дереве, удаление и вставка элементов в дерево.

5. Что такое дерево двоичного поиска?

Это такое дерево, коэффициент ветвления которого равен двум.