|  |  |
| --- | --- |
| 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

Описание задачи 3

Входные и выходные данные 4

Аварийные ситуации и особенности реализации 5

Структуры данных 6

Сравнение эффективности 7

Вывод 10

Контрольные вопросы 11

**Цель работы:**

получить навыки применения двоичных деревьев

**Общее задание:**

Построить дерево двоичного поиска.

Вывести его на экран в виде дерева.

Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов.

Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

Написать функцию для определения количества узлов дерева на каждом уровне.

**Задание по варианту:**

В текстовом файле содержатся целые числа. Построить двоичное дерево из чисел файла. Вывести его на экран в виде дерева.

Используя процедуру, определить количество узлов дерева на каждом уровне.

Добавить число в дерево и в файл.

Сравнить время добавления чисел в указанные структуры.

**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 (на основе десяти замеров)**:

Сортировка (под структурами данных указано среднее время в микросекундах)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Количество элементов | Сортировка массива с помощью АВЛ дерева | Массив из файла с помощью treesort (на основе ДДП) | Массив из файла с помощью qsort |
| Маленький файл | 200 | 173 | 50 | 82 |
| Средний файл | 20000 | 13211 | 7177 | 8076 |
| Большой файл | 200000 | 205455 | 128045 | 112677 |
| Отсортированный файл | 20000 | 7387 | 1149976 | 6819 |
| Файл, отсортированный в обратном порядке | 20000 | 6337 | 999465 | 3345 |

Поиск (под структурами данных указано среднее количество сравнений)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Количество элементов | АВЛ дерево | Дерево двоичного поиска | Файл |
| Маленький файл | 200 | 5 | 7 | 69 |
| Средний файл | 20000 | 11 | 13 | 10780 |
| Большой файл | 200000 | 15 | 20 | 97587 |
| Отсортированный файл | 20000 | 11 | 10063 | 10063 |
| Файл, отсортированный в обратном порядке | 20000 | 11 | 9936 | 9936 |

Средние коэффициенты ветвления деревьев

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Структура | Маленький файл | Средний файл | Большой файл | Отсортированный файл | Файл, отсортированный в обратном порядке |
| АВЛ дерево | 1,809 | 1,752 | 1,750 | 2,000 | 2,000 |
| ДДП | 1,508 | 1,497 | 1,499 | 1,000 | 1,000 |

**Сортировка**

До 20 тысяч элементов сортировка с помощью ДДП была на 12,5% быстрее, чем qsort массива из файла. Сортировка с помощью АВЛ дерева заняла на 63% больше времени, чем qsort.

При работе с 200 тысячами чисел количество сравнений при создании ДДП стало настолько велико, что время на сортировку стало занимать на 13,6% больше, чем на qsort. Сортировка с АВЛ деревом потребовала на 60% больше времени, чем treesort, и на 82% больше, чем qsort.

Treesort занимает намного больше времени, если пытаться применить её на уже отсортированный (в прямом или обратном порядке) исходный файл, в этом случае отношение скоростей работы увеличится до 300 раз по сравнению с быстрой сортировкой массива из файла и до 157 раз по сравнению с сортировкой с помощью АВЛ дерева (в случае файла, отсортированного в обратном порядке). Если числа в исходном файле были отсортированы, то время на добавления новвх узлов в ДДП с каждым элементом увеличивается, также много времени будет затрачено на обход по такому дереву. Видно, что при изначально отсортированных числах время на балансировку АВЛ дерева оказалось сравнительно небольшим по сравнению с временем, затраченным на прохождение по узлам при добавлении элемента в ДДП и на обход по нему.

**Поиск**

Для самого большого файла (на 200000 элементов) поиск в файле занял в 4879 раз больше сравнений, чем ДДП и в 6506 раз больше, чем в АВЛ дереве.

Файлы, отсортированные в прямом и обратном направлениях, оказались худшими случаями для массива и ДДП. Видно, что поиск в массиве и ДДП занял одинаково большое количество сравнений и по сложности равен O(N), а поиск в АВЛ дереве использовал в 914 раз меньше сравнений и по сложности составил O(log2(N)).

**Память**

Размер узла дерева в байтах: 24

Размер структуры массива в байтах: 24

Размер элемента массива в байтах: 4

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, чем массив.

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, чем массив.

**Вывод**

До 20 тысяч элементов treesort незначительно выигрывает во времени у qsort и заметно выигрывает у сортировки с помощью АВЛ дерева. При большем количестве элементов, оно начинает проигрывать qsort, но всё ещё выигрывает у АВЛ дерева. А при отсортированных изначальных данных, он заметно проигрывает qsort и созданию АВЛ дерева.

Если данные в файле не были изначально отсортированы, поиск в АВЛ дереве занимает примерно одинаковое количество сравнений с ДДП и зависимость количества сравнений от количества элементов логарифмическая.

На отсортированных входных данных количество сравнений при поиске в ДДП зависит от количества элементов линейно, как и при поиске в массиве.

Деревья занимают почти в 6 раз больше памяти, чем массив, но поиск в них может занимать до 6 тысяч раз меньшее количество сравнений.

**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.