|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство образования и науки 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)

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5  
«ОБРАБОТКА ДЕРЕВЬЕВ, ХЕШ-ФУНКЦИЙ»**

Название предмета: Типы и структуры данных

Студент: Малышев Иван Алексеевич

Группа: ИУ7-31Б

*2020 г.*

**Цель работы**: получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов; построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах.

**Описание условия задачи**

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Добавить указанное слово, если его нет в дереве (по желанию пользователя) в исходное и сбалансированное дерево. Сравнить время добавления и объем памяти. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана, используя метод цепочек для устранения коллизий. Вывести построенную таблицу слов на экран. Осуществить добавление введённого слова, вывести таблицу. Сравнить время добавления, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев, хеш-таблиц и файла.

**Техническое задание**

**Входные данные**

1. **Целое число, представляющее собой номер команды:** целое число от 0 до 12.
2. **Строки с названием файла с данными или с искомым словом:** непустые строки.

**Выходные данные**

1. Файлы с изображением деревьев.
2. Результаты обхода деревьев.
3. Результаты поиска слова в деревьях или в хеш-таблице.
4. Статус выполнения команды.
5. Сравнительная характеристика ДДП, сбалансированных деревьев, хеш-таблиц и файла согласно условию задачи.

**Функция программы:** программа выполняет ряд функций, указанных при её первом запуске. Она позволяет:

1. Создать дерево и сбалансированное дерево по данным из файла
2. Вывести деревья в файлы .png
3. Произвести прямой обход по деревьям
4. Добавить слово в деревья
5. Удалить из деревьев слово
6. Найти слово в дереве
7. Создать хеш-таблицу по данным из файла
8. Вывести хеш-таблицу на экран
9. Добавить слово в хеш-таблицу
10. Найти слово в хеш-таблице
11. Изменить хеш-функцию и (пере)создать таблицу
12. Сравнить эффективность работы обычных и сбалансированных деревьев, хеш-таблицы и файла

В случае, когда в деревьях не удалось найти искомое слово, по желанию пользователя это слово можно добавить в деревья.

**Способ обращения к программе:**

К программе можно обратиться через терминал ОС по её имени «app.exe». Дальнейшие инструкции будут выведены после запуска.

**Аварийные ситуации:**

* Некорректный номер операции, т. е. число меньше нуля или больше 12
* Ввод пустой строки
* Ввод названия несуществующего файла

Во всех указанных случаях программа завершится корректно или сообщит об ошибке.

**Структуры данных**

Общая реализация ДДП и АВЛ-деревьев

typedef struct tree\_node

{

char \*word; - слово

size\_t height; - высота узла

struct tree\_node \*left; - левое поддерево

struct tree\_node \*right; - правое поддерево

} tree\_t;

Реализация хеш-таблицы с внешним хешированием:

typedef struct hash\_node

{

char \*value; - слово

struct hash\_node \*next\_elem; - указатель на следующее слово в ячейке таблицы

} hash\_node\_t;

typedef struct

{

hash\_node\_t \*table\_ptr; - указатель на начало массива хеш-таблицы

size\_t size; - размерность таблицы

} hash\_table\_t;

**Описание функций**

tree\_t \*create\_node(char \*val) – создание узла дерева

tree\_t \*create\_tree(FILE \*f) – создание дерева по данным из файла

tree\_t \*insert\_to\_tree(tree\_t \*tree, char \*val, size\_t \*cmps) – вставка слова в дерево

tree\_t \*search\_in\_tree(tree\_t \*root, char \*word) – поиск слова в дереве

tree\_t \*remove\_from\_tree(tree\_t \*root, char \*val) – удаление слова из дерева

void vertex\_count(tree\_t \*root, size\_t \*n) – число вершин

tree\_t \*balance\_tree(tree\_t \*root) – балансировка дерева

tree\_t \*create\_balance\_tree(FILE \*f) – создание сбалансированного дерева по данным из файла

tree\_t \*insert\_to\_balanced\_tree(tree\_t \*node, char \*val, size\_t \*cmps) – вставка слова в сбалансированное дерево

tree\_t \*remove\_from\_balanced\_tree(tree\_t \*root, char \*val) – удаление слова из сбалансированного дерева

void free\_tree(tree\_t \*node) – освобождение памяти из под дерева

void print\_tree(tree\_t \*root, char \*name) – печать дерева в файл .png

void traversal\_tree(tree\_t \*root) – функция прямого обхода дерева

int insert\_to\_hash\_table(hash\_table\_t \*table, char \*word, hash\_func hash, size\_t \*cmps) – вставка слова в хеш-таблицу

hash\_table\_t \*create\_hash\_table(FILE \*f, size\_t size, hash\_func hash) – создание хеш-таблицы

hash\_node\_t \*search\_in\_hash\_table(hash\_table\_t \*table, char \*word, hash\_func hash) – поиск слова в хеш-таблице

void print\_hash\_table(hash\_table\_t \*table) – печать хеш-таблицы

void free\_table(hash\_table\_t \*table) – освобождение памяти из под хеш-таблицы

size\_t hash\_table\_count(hash\_table\_t \*table) – число элементов в хеш-таблице

size\_t hash\_by\_mod(char \*word, size\_t m) – хеш-функция (метод деления)

size\_t hash\_by\_ring\_polynom(char \*word, size\_t m) – хеш-функция (метод Горнера)

char \*search\_in\_file(FILE \*f, char \*word) – поиск слова в файле

int insert\_to\_file(FILE \*f, char \*word) – вставка слова в конец файла

int file\_size(FILE \*f, size\_t \*size) – размер файла

**Алгоритм**

1. Пользователь вводит номер операции из меню.
2. Пока пользователь не введёт 0 (выход из программы), ему будет предложено выполнять набор действий.
3. Дальнейшие указания будут выведены при выборе определённой команды.

**Алгоритмы работы хеш-функций**

*Алгоритм №1*. Хеш-функция подсчитывает сумму кодов символов исходной строки, и делит их на размер массива. Остаток от деления и будет ключом.

*Алгоритм №2.* Вычисляется очередной член полинома на основе кодов символов и берётся остаток от деления на размерность таблицы. Конечный результат и будет ключом.

**Тестирование**

Положительные тесты

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Создание дерева и сбалансированного дерева по данным из файла | 1  (название валидного файла) | Деревья успешно созданы |
| 2 | Печать деревьев | 2 | Если не возникло ошибок, то деревья успешно распечатаны в файлы "Tree.png" и "Balanced\_tree.png"  P. S. Красная дуга - левое поддерево, синяя дуга - правое поддерево |
| 3 | Прямой обход по деревьям | 3 | \*Вывод узлов деревьев в прямом порядке |
| 4 | Добавление слова в деревья | 4  абвгд | Слово «абвгд» успешно вставлено в деревья |
| 5 | Удаление слова из деревьев | 5  абвгд | Слово «абвгд» успешно удалено из деревьев |
| 6 | Поиск слова | 6  Иван | Слово «Иван» было найдено за 1486 тактов на высоте 3 |
| 7 | Создание хеш-таблицы | 7  (название валидного файла)  10 | Хеш-таблица успешно создана |
| 8 | Печать хеш-таблицы | 8 | \*Вывод хеш-таблицы |
| 9 | Вставка слова в хеш-таблицу | 9  абвгд | Слово «абвгд» успешно вставлено в хеш-таблицу |
| 10 | Поиск слова в хеш-таблице | 10  абвгд | Слово «абвгд» было найдено с хешем 3 за 486 тактов |
| 11 | Изменение хеш-функции и пересоздание таблицы | 11  Y  (название валидного файла)  10 | Хеш-функция успешно изменена  Хеш-таблица успешно создана |
| 12 | Сравнение эффективности работы структур данных | 12 | \*Вывод информации о результатах сравнения |

Негативные тесты

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Создание дерева и сбалансированного дерева по несуществующему или невалидному файлу | 1  (невалидный файл) | Возникли проблемы с созданием деревьев |
| 2 | Печать пустых деревьев | 2 | Попытка печати пустых деревьев |
| 3 | Прямой обход по пустым деревьям | 3 | Попытка печати пустых деревьев |
| 5 | Удаление слова из пустых деревьев | 5  абвгд | Такого слова в деревьях нет |
| 6 | Поиск несуществующего слова в дереве | 6  Иван | Такого слова в деревьях нет |
| 7 | Создание хеш-таблицы по невалидному файлу | 7  (название невалидного файла)  10 | Возникли проблемы с созданием хеш-таблицы |
| 8 | Печать пустой хеш-таблицы | 8 | Хеш-таблица пуста |
| 10 | Поиск несуществующего слова в хеш-таблице | 10  абвгд | Такого слова в таблице нет |
| 11 | Изменение хеш-функции и пересоздание таблицы с невалидным файлом | 11  Y  (название невалидного файла)  10 | Хеш-функция успешно изменена  Возникли проблемы с созданием хеш-таблицы |
| 12 | Неопределённый ответ | 11  А | Ошибка ввода ответа |

**Оценка эффективности**

Результаты с настройками: хеширование методом деления, кол-во элементов в структурах равно 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Добавление элемента | | | |
| ДДП | АВЛ | Хеш-таблица | Файл |
| Время\* (в тактах) | 435 | 752 | 352 | 49060 |
| Занимаемая память (в байтах) | 352 | 352 | 192 | 58 |
| Кол-во сравнений | 5 | 3 | 1 | 0 |

\*В таблице приводятся средние значения.

**Контрольные вопросы**

1. **Что такое дерево?**

Дерево – это рекурсивная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим»

1. **Как выделяется память под представление деревьев?**

В виде связного списка — динамически под каждый узел.

1. **Какие стандартные операции возможны над деревьями?**

Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

1. **Что такое дерево двоичного поиска?**

Двоичное дерево, для каждого узла которого сохраняется условие: левый потомок больше или равен родителю, правый потомок строго меньше родителя (либо наоборот).

1. **Чем отличается идеально сбалансированное дерево от АВЛ дерева?**

У АВЛ дерева для каждой его вершины высота двух её поддеревьев различается не более чем на 1, а у идеально сбалансированного дерева различается количество вершин в каждом поддереве не более чем на 1.

1. **Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?**

Поиск в АВЛ дереве происходит быстрее, чем в ДДП.

1. **Что такое хеш-таблица, каков принцип её построения?**

Хеш-таблицей называется массив, заполненный элементами в порядке, определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс, функция должна быть простой для вычисления, распределять ключи в таблице равномерно и давать минимум коллизий.

1. **Что такое коллизии? Каковы методы их устранения.**

Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие один и тот же индекс. Основные методы устранения коллизий: открытое и закрытое хеширование. При открытом хешировании к ячейке по данному ключу прибавляется связанны список, при закрытом – новый элемент кладётся в ближайшую свободную ячейку после данной.

1. **В каком случае поиск в хеш-таблицах становится неэффективен?**

Поиск в ХТ становится неэффективен при большом числе коллизий –сложность поиска возрастает по сравнению с О(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

1. **Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах.**

В хеш-таблице минимальное время поиска О(1). В АВЛ: О(log2n). В дереве двоичного поиска О(h), где h - высота дерева (от log2n до n).

**Вывод**

Основным преимуществом рассмотренных структур данных является возможная высокая эффективность реализации алгоритмов обхода, добавления, поиска и удаления элементов.

Среднее время выполнения этих операций для таблиц намного меньше (для добавления нужно в среднем *в 1.2 раз больше времени для ДДП, и в 2.1 раз больше для АВЛ-дерева*), чем для деревьев. Так же, для хранения хеш-таблицы нужно *в**2 раза меньше памяти*, чем для хранения деревьев. Но, у деревьев есть по крайней мере одно заметное преимущество по сравнению с хеш-таблицей: в них можно выполнить проход по возрастанию или убыванию ключей и сделать это быстро.

Такая разница (в *1.7 раз*) между *ДДП* и *АВЛ-деревьями* объясняется тем, что при добавлении узла в *АВЛ-дерево* оно заново «балансируется», что занимает ощутимое время, несмотря на относительно небольшую сложность алгоритма (он не рекуррентен и работает только с ближайшими узлами).

Стоит отметить, что время добавление в файл будет всегда константным, потому что перемещение каретки в конец файла происходит за константное время. Поэтому, выбор структур данных напрямую зависит от области их применения.