|  |  |
| --- | --- |
|  | **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»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**ОТЧЕТ К ЛАБОРАТОРНОЙ РАБОТЕ №7 ПО «ТИПАМ И СТРУКТУРАМ ДАННЫХ»**

**Тема: «Сбалансированные деревья, хеш-таблицы»**

Группа: ИУ7-32Б

|  |  |  |
| --- | --- | --- |
| Студент: |  | Сидоров Максим Михайлович |

Преподаватель: Никульшина Татьяна Александровна

**Цель работы:** научиться работать со сбалансированными деревьями и хэш-таблицами, а также сравнить время удаления элементов из них.

**Условие задачи:** построить дерево поиска из слов текстового файла, сбалансировать полученное дерево. Вывести его на экран в виде дерева. Удалить все слова, начинающиеся на указанную букву, в исходном и сбалансированном дереве. Сравнить время удаления, объем памяти. Построить хеш-таблицу из слов текстового файла. Вывести построенную таблицу слов на экран. Осуществить поиск и удаление введенного слова, вывести таблицу. Выполнить программу для различных размерностей таблицы и сравнить время удаления, объем памяти и количество сравнений при использовании сбалансированных деревьев и хеш-таблиц.

**Входные данные:** номер команды из меню, текстовый файл и терминальный ввод.

Команды:

1. Удалить слова, начинающиеся с буквы и вывести на экран
2. Сравнить удаление в исходном и сбалансированном дереве
3. Удалить слово из хеш-таблиц и вывести на экран
4. Сравнить удаление в сбалансированном дереве и в хэш-таблицах
5. Выйти

**Выходные данные:**

- ДДП в виде картинки

- сбалансированное дерево в виде картинки

- ДДП с удаленными словами в виде картинки

- сбалансированное дерево с удаленными словами в виде картинки

- хеш-таблица в виде картинки

- хеш-таблица с удаленным словом в виде таблицы

- сравнение удаления в ДДП и сбалансированном дереве

- сравнение удаления в сбалансированном дереве и хеш-таблице

**Аварийные ситуации:**

- ввод неправильной команды

- некорректный файл

**Описание СД:**

1. Для хранения узла обоих деревьев было выбрано следующее представление:

typedef struct tree tree;

struct tree

{

char \*word;

tree \*left;

tree \*right;

tree \*parent;

};

*Листинг 1. Представление узла дерева.*

1. Для хранения хеш-таблицы было выбрано следующее представление:

typedef struct list list\_t;

struct list

{

char word[WORD\_LEN];

list\_t \*next;

};

typedef struct

{

list\_t \*\*arr;

int len;

} opened\_hash;

*Листинг 2. Представление хэш-таблицы.*

**Алгоритм удаления из ДДП:** поиск узла, которые следует удалить. Если у данного элемента нет потомков, то освободить элемент и обнулить указатель родителя на этот элемент. Иначе от этого узла надо найти либо самый левый от правого потомка, либо самый правый от левого потомка. И поменять ссылки этого элемента на новый элемент, а текущий элемент удалить. Рекурсивно повторять удаления, пока все элементы не удалятся.

**Алгоритм удаления из сбалансированного дерева:** поиск узла, которые следует удалить. Далее следует найти минимальный узел от данного (самый левый). Заменить текущий узел на минимальный, а текущий удалить. После каждого удаления балансировать дерево, то есть совершать повороты поддеревьев, чтобы уменьшить, если возможно высоту дерева. Рекурсивно производить удаление, пока все элементы не удалятся.

**Алгоритм балансировки:** балансировка требуется, когда высота левого и правого поддеревьев отличаются на 2. В этом случае производят поворот вокруг корня дерева, то есть новой головой становится левый или правый узел, в зависимости от поворота, а также переставляются указатели нового и старого корней. Балансировка не зависит от размера АВЛ-дерева.

**Алгоритм хеширования:** хеш-функция – функция, позволяющая по определенному правилу для исходных данных получить некоторое значение. Хеш-функция должна создавать уникальные значения, чтобы минимизировать количество коллизий, для этого в основном используют методы деления. По входным данным с помощью внутреннего алгоритма функция выдает результат в виде ключа, с помощью которого можно однозначно обнаружить элемент в хеш-таблице, если для данного ключа нет коллизий.

**Алгоритм реструктуризации:** если при создании хеш-таблицы количество коллизий велико, то производится реструктуризация, а именно подбор новой хеш-функции с помощью деления, умножения, прибавления констант и переопределение ключей. Реструктуризация производится, пока количество переборов для доступа к элементу будет не более 3-4.

**Алгоритм удаления из открытой хеш-таблицы:** высчитывание хэша для искомого элемента. По этому хешу обратиться к элементу массива с номером хеша, если данному хешу соответствуют несколько элементов, то перебрать элементы списка, пока не найдется нужный. Удаление производится как удаление из списка.

**Алгоритм удаления из закрытой хеш-таблицы:** высчитывание хэша для искомого элемента. По этому хэшу обратиться к элементу массива с номером хеша, если данному хешу соответствует другой элемент, то перебрать элементы списка с заданным шагом, пока не найдется нужный. Удаление производится как удаление элемента массива.

**Хеш-функция:** в данной лабораторной работе использовалась функция, которая в слове складывает значения каждой буквы из ASCII таблицы и полученную сумму делит на простое число.

**Тестовые данные:**

|  |  |  |
| --- | --- | --- |
| **Описание** | **Вход** | **Выход** |
| Некорректная команда | А, -1, 15 | Некорректная команда |
| Запуск программы с некорректным файлом | Dsfsj.chh | Некорректный файл |
| Удалить слова, начинающиеся с буквы | а | Удаление слов и вывод на деревьев экран |
| Запуск программы с корректным файлом | - | Создание и показ деревьев и хеш-таблицы |
| Сравнение удаления из деревьев | - | Время удаления и объем памяти для деревьев |
| Удалить слово из хеш-таблицы | - | Удаление слова и вывод хеш-таблицы на экран |
| Сравнение удаления из хеш-таблицы и из сбалансированного дерева | - | Время удаления, объем памяти, количество переборов для сбалансированного дерева и для хеш-таблицы |

**Функции:**

void print\_menu() – вывод меню

static\_arr read\_words(FILE \*f) – запись слов в массив из файла

void compare\_trees(FILE \*f) – сравнения удаления из деревьев

void compare\_hash(FILE \*f) – сравнение удаления из сбалансированного дерева и из хеш-таблицы

void to\_dot(tree \*a, FILE \*f) – запись узлов в формате DOT

void apply\_pre(tree \*a, void (\*f)(tree\*, FILE\*), FILE \*arg) – префиксный обход дерева

void export\_to\_dot(tree \*a, char \*file\_name) – создание DOT-файла

void print\_hash(opened\_hash \*a, char \*file\_name) – вывод хеш-таблицы в DOT-формате

void free\_static(static\_arr \*a) – освобождение памяти из-под массива строк

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

void free\_list(list\_t \*a) – освобождение памяти из-под списка

void free\_hash(opened\_hash \*a) – освобождение памяти из-под хеш-таблицы

tree\_t \*init\_tree(char \*word) – инициализация узла дерева

tree\_t \*add\_block(tree\_t \*a, char \*word, tree\_t \*parent) – добавить узел в ДДП

tree\_t \*create\_tree(static\_arr \*a) – создать ДДП

int find\_height(tree\_t \*a) – определить высоту дерева

int balance\_count(tree\_t \*a) – определить разность высот поддеревьев

void correct\_height(tree\_t \*a) – скорректировать высоту дерева

tree\_t \*rotate\_right(tree\_t \*a) – повернуть поддерево вправо

tree\_t \*rotate\_left(tree\_t \*a) – повернуть поддерево влево

tree\_t \*balance\_root(tree\_t \*a) – сбалансировать дерево

tree\_t \*add\_avl\_block(tree\_t \*a, char \*word) – добавить узел в сбалансированное дерево

tree\_t \*create\_avl\_tree(static\_arr \*a) – создать сбалансированное дерево

int count\_hash(char \*s, int n) – вычислить хэш

list\_t \*find\_last(list\_t \*a) – поиск последнего элемента списка

opened\_hash create\_hash(static\_arr \*a) – создание хеш-таблицы

tree\_t \*most\_left(tree\_t \*a) – поиск самого левого элемента ДДП

tree\_t \*most\_right(tree\_t \*a) – поиск самого правого элемента ДДП

void delete\_tree(tree\_t \*a, char s) – удалить слова, начинающиеся на букву из ДДП

tree\_t \*find\_left(tree\_t \*a) – найти самый левый узел

tree\_t \*remove\_left(tree\_t \*a) – удалить самый левый узел

tree\_t \*delete\_avl\_tree(tree\_t \*a, char s, bool \*flag) – удалить слово, начинающееся на букву из сбалансированного дерева

tree\_t \*delete\_avl(tree\_t \*a, char s) – удалить все слова, начинающиеся на букву из сбалансированного дерева

tree\_t \*delete\_avl\_tree\_word(tree\_t \*a, char \*s, int \*count) – удалить слово из сбалансированного дерева

int delete\_hash\_word(opened\_hash \*a, char \*s) – удалить слово из хеш-таблицы

**Сравнение эффективности удаления из ДДП и сбалансированного дерева:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество элементов** | **Количество элементов, которые надо удалить** | **ДДП, мкс** | **Сбалансированное**  **дерево, мкс** | **Эффективность** |
| 1 | 1 | 10 | 9 | -10% |
| 5 | 1 | 14 | 11 | -21.4% |
| 10 | 1 | 14 | 12 | -14.3% |
| 5 | 20 | 24 | 20% |
| 10 | 22 | 32 | 45.5% |
| 20 | 1 | 16 | 15 | -16.7% |
| 10 | 23 | 44 | 91.3% |
| 20 | 30 | 60 | 100% |
| 40 | 1 | 19 | 17 | -10.5% |
| 20 | 29 | 64 | 120.6% |
| 40 | 46 | 115 | 150% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество элементов** | **ДДП, байты** | **Сбалансированное дерево, байты** | **Эффективность** |
| 1 | 56 | 56 | 0% |
| 5 | 280 | 280 | 0% |
| 10 | 560 | 560 | 0% |
| 20 | 1120 | 1120 | 0% |
| 40 | 2240 | 2240 | 0% |

Удаление элементов из сбалансированного дерева быстрее, если удаляемый элемент, иначе сбалансированное дерево будет работать намного медленнее ДДП. Затраты по памяти одинаковые.

**Сравнение эффективности удаления из хеш-таблицы и сбалансированного дерева:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество элементов** | **Хеш-таблица, мкс** | **Сбалансированное**  **дерево, мкс** | **Эффективность** |
| 1 | 0 | 1 | 100% |
| 5 | 4 | 10 | 150% |
| 10 | 4 | 14 | 250% |
| 20 | 4 | 19 | 375% |
| 40 | 5 | 31 | 520% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество элементов** | **Открытая хеш-таблица, перестановки** | **Закрытая хеш-таблица, перестановки** | **Сбалансированное**  **Дерево, перестановки** |
| 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 3 |
| 10 | 0 | 2 | 4 |
| 20 | 0 | 3 | 6 |
| 40 | 0 | 4 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество элементов** | **Открытая хеш-таблица, байты** | **Сбалансированное дерево, байты** | **Эффективность** |
| 1 | 40 | 56 | 40% |
| 5 | 200 | 280 | 40% |
| 10 | 400 | 560 | 40% |
| 20 | 800 | 1120 | 40% |
| 40 | 1600 | 2240 | 40% |

Удаление элементов из хеш-таблицы быстрее, чем сбалансированного дерева, объем памяти, занимаемый хеш-таблицей меньше, чем у сбалансированного дерева.

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество элементов** | **Открытая хеш-таблица, мкс** | **Закрытая хеш-таблица, мкс** | **Эффективность** |
| 1 | 0 | 0 | 0% |
| 5 | 4 | 1 | 300% |
| 10 | 4 | 3 | 33.3% |
| 20 | 4 | 6 | -33.3% |
| 40 | 5 | 9 | -80% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Количество элементов** | **Открытая хеш-таблица, байты** | **Закрытая хеш-таблица, байты** | **Эффективность** |
| 1 | 40 | 8 | 500% |
| 5 | 200 | 40 | 500% |
| 10 | 400 | 80 | 500% |
| 20 | 800 | 160 | 500% |
| 40 | 1600 | 320 | 500% |

Закрытая хеш-таблица проигрывает открытой при большом количестве коллизий, но выигрывает по занимаемой памяти.

**Вывод:** удаление одного элемента из сбалансированного дерева быстрее, чем из ДДП, так как высота сбалансированного дерева меньше. Удаление нескольких элементов из сбалансированного дерева дольше, чем из ДДП, так как при каждом удалении из сбалансированного дерева, его надо балансировать. Память, затрачиваемая на оба дерева одинакова.

Удаление элемента из хэш-таблицы быстрее, чем из сбалансированного дерева, так как операция удаления из хэш-таблицы О(1), в худшем случае линейна.

**Контрольные вопросы:**

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

Сбалансированное дерево – дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу.

АВЛ дерево – дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу.

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

В АВЛ-дереве поиск происходит быстрее за счет меньшей высоты дерева.

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

Хеш-таблица – массив, в котором индекс каждого элемента зависит от информации, занесенной в элемент (индекс равен хэш-функции от элемента).

Функция должна быть простой для вычисления, а также такой, чтобы давать наименьшее количество коллизий.

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

Коллизия – ситуация, при которой разным вводимым элементам в хэш-таблице ставится один и тот же индекс.

Методы устранения: открытое (список) и закрытое (на соседние места).

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

При большом числе коллизий. Чтобы повысить эффективность, нужно изменить хеш-функцию и реструктуризировать таблицу.

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

Хеш-таблица: О(1) – O(N).

АВЛ-дерево: O(log2N).

ДДП: O(log2N) – O(N).