Отчёт по лабораторной работе №6

“Деревья, хеш-таблицы”

Вариант 4

Выполнил: Соколов Ефим Маркович (группа ИУ7-33Б)

### ***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.

Техническое задание

В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать закрытое хеширование для устранения коллизий. Осуществить удаление введенного целого числа в ДДП, в сбалансированном дерево, в хеш-таблице и в файле. Сравнить время удаления, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного, то произвести реструктуризацию таблицы, выбрав другую функцию.

Типы входных и выходных данных

***Входные данные:***

* Ключ меню – целое число в промежутке [0 ; 10] представляющее собой команду меню
* Текстовый файл *source.txt,* в котором хранятся целые числа
* Максимальное количество сравнений для хеш-таблицы
* Целое число для удаления

**Выходные данные:**

* Графическая визуализация ДДП
* Графическая визуализация АВЛ-дерева
* Распечатанная в консоль хеш-таблица
* Статистические данные по времени, затраченной памяти и количеству сравнений для удаления элемента

***Возможные аварийные ситуации:***

* Попытка вывести дерево (ДДП или АВЛ), предварительно не создав его
* Попытка вывести хеш-таблицу, предварительно не создав её
* Некорректный ввод числа для удаления
* Число, которое следует удалить, отсутствует в данных
* Попытка обойти дерево, предварительно не создав его
* Некорректный ввод максимального числа сравнений
* Некорректная команда меню

Структуры данных и описание алгоритма

Хеш-таблица: массив структур

typedef struct

{

    int value; // значение

    int flag; // флаг состояния ячейки

} hashTable\_t;

Дерево двоичного поиска (ДДП)

typedef struct node\_t

{

    int value; // значение узла

    struct node\_t \*left; // указатель на узел слева

    struct node\_t \*right; // указатель на узел справа

} treeNode\_t;

АВЛ-дерево

typedef struct node

{

    int value; // значение узла

    int balance; // флаг, указывающий какое из поддеревьев больше

    struct node\_t \*left; // указатель на узел слева

    struct node\_t \*right; // указатель на узел справа

} treeNode\_t;

Структура для хранения статистических данных

typedef struct

{

    uint64\_t time\_del\_file;

    uint64\_t time\_del\_bst;

    uint64\_t time\_del\_avl;

    uint64\_t time\_del\_hash\_table;

    int cmps\_file;

    int cmps\_hash\_table;

    int cmps\_bst;

    int cmps\_avl;

    int avg\_file\_cmps;

    uint64\_t avg\_file\_time;

    int avg\_ht\_cmps;

    uint64\_t avg\_ht\_time;

    int avg\_bst\_cmps;

    uint64\_t avg\_bst\_time;

    int avg\_avl\_cmps;

    uint64\_t avg\_avl\_time;

    size\_t memory\_file;

    size\_t memory\_hash\_table;

    size\_t memory\_bst;

    size\_t memory\_avl;

} statData\_t;

Чтобы сбалансировать ДДП, оно вытягивается в односвязный линейный список, после чего рекурсивно составляется АВЛ-дерево. Чтобы удалить элемент из дерева, сначала находится искомый элемент для удаления, далее рекурсивно находится максимальный элемент в левом поддереве этого элемента и ставится на место удаляемого элемента. Если удаление происходит в АВЛ-дереве, то помимо описанного ранее алгоритма так же проверяется условие сбалансированности для нового дерева. В случае его невыполнения, дерево балансируется методом правого (или левого) поворота – все дерево не балансируется заново, только поддерево удалённого элемента.

В качестве хэш-функции используется деление с остатком (с постепенно увеличивающимися простыми делителями, если превышено максимальное количество сравнений).

В закрытом хешировании используется квадратичная адресация.

Тесты

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Тест | | Пользовательский ввод | | Результат | |
| 1 | Некорректный ввод команды меню | | Ключ меню:  -7 | | Сообщение “Введен недопустимый ключ! Повторите попытку” | |
| 2 | Некорректный ввод предельного числа сравнений для хеш-таблицы | | Предельное число сравнений:  -5 | | Сообщение “Введено недопустимое значение!” | |
| 3 | Некорректный ввод числа для удаления | | Число для удаления:  t | | Сообщение “Введено недопустимое значение! ” | |
| 4 | Число для удаления отсутствует в данных | | Число для удаления:  12 | | Сообщение “Введенное число не содержится в данных” | |
| 5 | Попытка распечатать АВЛ/ДП дерево или хеш-таблицу, не создав предварительно | | Попытка вывода | | Сообщение “Невозможно распечатать таблицу (дерево), так как оно не было создано предварительно” | |
| 6 | Попытка вывести обход дерева, не создав его предварительно | | Попытка  вывода | | Сообщение “Невозможно вывести обход. Сначала введите дерево” | |
| 7 | Тривиальное удаление | | Число для удаления:  56 | | Нет сообщений об ошибке, стандартное продолжение работы | |
| 8 | Тривиальный обход | | Попытка вывода | | Обход тремя разными способами: прямой, центрированный, обратный | |
| 9 | Реструктуризация  хеш-таблицы | | Файл с целыми числами | | Нет сообщений об ошибке, стандартное продолжение работы | |
| 10 | Хеш-таблицу невозможно построить | | Файл с целыми числами | | Сообщение  “Невозможно построить хеш-таблицу” | |

Оценка эффективности

**Среднее время удаления элемента (в тактах процессора):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Файл | Хэш-таблица | ДДП | АВЛ |
| 10 | 48549 | 813 | 1403 | 2635 |
| 50 | 281522 | 3383 | 7298 | 12494 |
| 100 | 504187 | 7801 | 15732 | 28197 |

**Среднее количество сравнений:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Файл | Хэш-таблица | ДДП | АВЛ |
| 10 | 7 | 0 | 4 | 5 |
| 50 | 24 | 2 | 9 | 8 |
| 100 | 52 | 2 | 14 | 11 |

**Занимаемая память (в байтах):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Файл | Хэш-таблица | ДДП | АВЛ |
| 10 | 41 | 88 | 240 | 240 |
| 50 | 84 | 424 | 1200 | 1200 |
| 100 | 414 | 808 | 2400 | 2400 |

Вывод

Самое быстрое удаление происходит из хеш-таблицы. Далее в порядке увеличения времени выполнения: удаление из ДДП, из АВЛ (учитывая время балансировки после удаления), из файла (поскольку происходит обращение к диску).

Контрольные вопросы

1. ***Что такое дерево?***

Дерево - это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

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

Память выделяется динамически. Зачастую создается структура, содержащая “полезную нагрузку” и две ссылки на левого и правого потомка.

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

Добавление узла, удаление узла, обход.

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

Это такое дерево, в котором все левые потомки меньше (моложе) предка, а все правые больше (старше).

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

В идеально сбалансированном дереве кол-во узлов в левом и правом поддеревьях отличается не более чем на единицу, а в АВЛ-дереве допускается различие высоты между правым и левым поддеревом каждого узла не более, чем на единицу.

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

Поиск в АВЛ-дереве выполняется быстрее, нежели чем в ДДП.

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

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей. Хеш-функция преобразует ключ в индекс его хранения. Существует закрытое и открытое хеширование.

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

Коллизия возникает, когда на разные ключи хеш-функция выдает один и тот же индекс. В открытом хешировании в список добавляется еще один элемент (метод цепочек); в закрытом хешировании просто ищется следующая свободная ячейка (можно изменить адресацию на, например, квадратичную, чтобы увеличить разброс).

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

Поиск в хеш-таблице становится неэффективным, когда количество сравнений превышает 3-4.

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

Эффективность поиска в дереве зависит от его высоты: у АВЛ-дерева наиболее эффективный поиск (сложность O(log(2))), медленнее работает поиск в ДДП (сложность O(n)). У хэш-таблицы без коллизий время поиска составляет O(1), с коллизиями – зависит от количества сравнений.