|  |  |
| --- | --- |
| **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**

КАФЕДРА **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04** Программная инженерия

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 7 |

Хеш-таблицы, ДДП, АВЛ-деревья

**Дисциплина:** Типы и структуры данных

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ7-35Б |  | А. В. Толмачев |
|  | (Группа) |  | (И.О. Фамилия) |
|  |  |  |  |

Москва, 2022

**Описание условия задачи**

Цель работы:

Построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем и внутреннем

хешировании.

Сбалансировать дерево, полученное из числе файла. Вывести его на экран в виде дерева. Построить хеш-таблицу из чисел файла. Осуществить поиск введенного целого числа в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных.

**Описание ТЗ**

1. **Описание исходных данных**

Исходными данными является путь до текстового файла с целыми числами, записанными каждый с новой строки.

Ограничения исходных данных:

1. Путь до файла должен быть указан как аргумент командной строки
2. Допускается указание пустого файла
3. Указание несуществующего файла является некорректным вводом
4. Если файл содержит любые символы, кроме цифр и переводов строк, то он считается некорректным.
5. Дубликаты элементов не поддерживаются
6. Недопустима работа с нецелыми числами
7. Выполнение любых операций (кроме вставки) над пустым деревом является невозможным.
8. Выполнение любых операций (кроме вставки) над пустой хеш-таблицей является невозможным.
9. Пункт меню задается целым числом. Возможен выбор одновременно только 1 пункта меню.
10. Ввод отсутствующего пункта меню является некорректным вводом.
11. **Описание задачи, реализуемой программой**

Программа позволяет выполнять операции над следующими структурами данных:

А) Деревья – двоичные деревья поиска и АВЛ-деревья. При работе с АВЛ-деревьями будет строиться сбалансированное дерево:

* + - Добавление узла дерева
    - Удаление узла дерева
    - Поиска узла дерева
    - Префиксный обход
    - Постфиксный обход
    - Инфиксный обход
    - Просмотр дерева
    - Определение количества узлов на каждом уровне дерева
    - Определение эффективности

Б) Хеш-таблицы – с разрешением коллизий методом цепочек и открытой адресацией

* Вывести таблицу
* Добавить элемент в таблицу
* Удалить элемент из таблицы
* Найти элемент в таблице

1. **Выходные данные**

* Сравнение эффективности
* Построенное дерево
* Результат добавления/удаления узла из дерева
* Результат поиска узла в дереве
* Последовательность вершин при префиксном/постфиксном/инфиксном обходе
* Количество узлов на каждом уровне
* Результат добавления/удаления элемента из хеш-таблицы
* Построенная хеш-таблица

1. **Способ обращения к программе**

Для запуска программы необходимо из консоли вызвать исполняемый файл ./app.exe.

1. **Описание возможных аварийных ситуаций и ошибок пользователя**

Аварийные ситуации:

1. Ошибка выделения памяти

Ошибки пользователя:

1. Ввод данных, не удовлетворяющих ограничениям исходных данных, описанных ранее
2. Попытка удалить узел из пустого дерева
3. Попытка найти узел в пустом дереве
4. Попытка посмотреть пустое дерево
5. Выбор несуществующего пункта меню
6. Ввод не целого числа при добавлении узла в дерево
7. Попытка найти элемент в пустой хеш-таблице
8. Попытка удалить элемент из пустой хеш-таблицы
9. Попытка посмотреть пустую хеш-таблицу

**Описание внутренних структур данных:**

Дерево поиска:

**struct** **tree\_node\_t**

{

**int** num;

**struct** **tree\_node\_t** \*left;

**struct** **tree\_node\_t** \*right;

};

АВЛ-дерево

**struct** **tree\_node\_t**

{

**int** num;

**unsigned** **char** height;

**struct** **tree\_node\_t** \*left;

**struct** **tree\_node\_t** \*right;

};

Хеш-таблица – метод цепочек

**typedef** **struct** **hash\_table\_t**

{

**node\_t** \*\*content;

**size\_t** count\_elements;

**size\_t** capacity;

**size\_t** mod;

**float** avarage\_count\_compare;

**size\_t** all\_count\_compare;

**size\_t** max\_count\_compare;

} **hash\_table\_t**;

Узлы списка для хеш-таблицы:

**typedef** **struct** **node\_t**

{

**int** num;

**struct** **node\_t** \*next;

} **node\_t**;

Хеш-таблица – открытая адресация

**typedef** **struct** **closed\_table\_t**

{

**cls\_cell\_t** \*content;

**size\_t** capacity;

**size\_t** count\_elements;

**size\_t** mod;

**float** avarage\_count\_compare;

**size\_t** max\_count\_compare;

**size\_t** all\_count\_compare;

} **closed\_table\_t**;

Ячейка таблицы с открытой адресацией:

**typedef** **struct** **cls\_cell\_t**

{

**int** num;

**char** status;

} **cls\_cell\_t**;

**Набор тестов**

1. Позитивные тесты
   1. Просмотр непустого дерева
   2. Добавление узла в дерево
   3. Удаление узла из дерева
   4. Просмотр непустой хеш-таблицы
   5. Добавление элемента в хеш-таблицу
   6. Поиск элемента в хеш-таблице
   7. Корректный выбор пункта меню
2. Негативные тесты

Тесты с входными данными, не удовлетворяющими ограничениям исходных данных, записанным ранее.

**Алгоритмы**

**1. Включение узла в АВЛ дерево**

1) Проход по пути поиска (пока не будем найдено место вставки – т. е. пустое поддерево)

2) Включение нового узла и определение показателя сбалансированности

3) Возврат по пути поиска, с проверкой показателей сбалансированности для каждой вершины, и проведение балансировки при необходимости.

**2. Правый поворот вокруг узла P**

1. Создать новую вершину дерева root
2. Проинициализировать новую вершину левым поддеревом вершины P
3. Установить правым поддеревом вершины root узел P
4. Выполнить корректировку значений высот для P и Q

Левый поворот выполняется аналогично

1. **Балансировка дерева**

Если высота правого поддерева узла P на 2 больше высоты левого поддерева:

1. Если высота левого поддерева узла Q больше высота его правого поддерева, то выполняется поворот влево вокруг p
2. Иначе выполняется сначала правый поворот вокруг q, после чего левый поворот вокруг p

Обратный случай реализуется симметрично.

1. **Удаление узла n с ключом k**

1) Ключ k сравнивается с ключом корневого узла

2) Если k>x — рекурсивно удалить k из правого поддерева, иначе из левого поддерева.

3) Если k=x:

а) Если обоих детей нет: удаляем узел, и обнуляем ссылку у родительского узла.

b) Если нет одного из детей — настроить ссылку родительского узла на ребенка, после чего удалить узел.

c) Если есть оба ребенка — найти самый левый узел правого поддерева узла. Скопируем данные в узел. Рекурсивно удалим найденный узел.

4) Вернуться по пути поиска и при необходимости сбалансировать дерево.

1. **Хеш-функция**

1) Вычисляется размер таблицы: R = 0.72 \* N/2, где N-количество элементов для метода цепочек, и R=1.2 \* N для метода открытой адресации.

2) Находится наименьшее простое число, большее размера таблицы - M

3) Хеш-функция h(k) — находит остаток от деления числа k на M, после чего находится остаток от деления на R

1. **Вставка в хеш-таблицу (открытая адресация)**

Пустая хеш-таблица T размера M, массив A размера N (M>=N), хеш-функция h ()

1. Для элемента xi с помощью хеш-функции вычисляется номер keyi
2. Если keyi - номер свободной ячейки таблицы T, то в нее записывается xi
3. Если keyi - номер уже занятой ячейки таблицы T, то на занятость проверяется соседняя ячейка. Если она свободна, то xi заносится в нее, иначе вновь проверяется другая ячейка, и так до тех пор, пока не найдется свободная или окажется, что все M ячеек таблицы заполнены.
4. **Удаление из хеш-таблицы (открытая адресация)**
5. Для удаляемого элемента xi с помощью хеш-функции вычисляется номер, начиная с которого может храниться данный элемент.
6. Выполняется поиск до тех пор, пока не будет найден искомый элемент, либо пока не будет достигнута свободная ячейка или ячейка, помеченная удаленной.
7. Если была достигнута свободная ячейка – нужный для удаления элемент в таблице отсутствовал.
8. Если значение какой-то ячейки совпало с искомым, ячейка помечается как удаленная.
9. **Поиск в хеш-таблице (открытая адресация)**
10. Для искомого элемента xi с помощью хеш-функции вычисляется номер, начиная с которого может храниться данный элемент.
11. Выполняется до тех пор, пока не будет встречена ячейка, помеченная как «свободная», или значение в какой-то ячейке не совпадет и искомым.

Элементы, помеченные удаленными, трактуются так же, как и занятые ячейки.

1. **Вставка в хеш-таблицу (метод цепочек)**

Данные хранятся в связанных списках. В таблице хранится ссылка на первый элемент каждого списка.

1. Для элемента xi с помощью хеш-функции вычисляется номер keyi
2. По номеру определяется ссылка на нужный список
3. Выполняется вставка в начала списка

Поиск и удаление происходят аналогично.

1. **Реструктуризация хеш-таблицы**

Если среднее число сравнений в хеш-таблице превышает заданное – 3 – то выполняется реструктуризация таблицы: создается новая таблица с удвоенным размером, выбирается новая хеш-функция.

Новая таблица заполняется элементами из старой таблицы, после чего старая хеш-таблица удаляется.

Для метода открытой адресации условием реструктуризации также является заполнение таблицы, т. е. если число элементов N равно размеру таблицы M.

**Эффективность**

Время поиска элемента при отсортированной последовательности чисел:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество элементов | Место | Дерево поиска, нс | АВЛ дерево, нс | Хеш таблица (метод цепочек), нс | Хеш таблица (открытая адресация), нс |
| 1000 | Середина | 1589.450000 | 36.310000 | 28.720000 | 27.750000 |
| Конец | 3997.450000 | 41.020000 | 28.970000 | 27.550000 |
| 10000 | Середина | 21503.390000 | 43.710000 | 28.810000 | 27.800000 |
| Конец | 46042.210000 | 55.720000 | 30.420000 | 233.930000 |

Размер структур данных:

Начальный размер хеш-таблицы (метод цепочек): 0.72 \* N/2

Начальный размер хеш-таблицы (открытая адресация): 1.2 \* N

N – кол-во элементов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Размер ДДП, байт | Размер АВЛ, байт | Размер хеш таблицы (метод цепочек), байт | Размер хеш таблицы (открытая адресация), байт |
| 1000 | 23000 | 24000 | 18944 | 9664 |
| 10000 | 230000 | 240000 | 188864 | 96064 |

Время поиска элемента при произвольной последовательности чисел:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Дерево поиска, нс | АВЛ дерево, нс | Хеш таблица (метод цепочек), нс | Хеш таблица (открытая адресация), нс |
| 100 | 36.580000 | 38.880000 | 32.430000 | 34.390000 |
| 1000 | 68.160000 | 42.450000 | 29.460000 | 27.870000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Размер ДДП, байт | Размер АВЛ, байт | Размер хеш таблицы (метод цепочек), байт | Размер хеш таблицы (открытая адресация), байт |
| 100 | 2300 | 2400 | 2240 | 1024 |
| 1001 | 23023 | 24024 | 21856 | 9680 |

Время поиска в таблице до и после реструктуризации

|  |  |  |
| --- | --- | --- |
| Количество элементов | Время в исходной таблице | Время в реструктуризированной таблице |
| 100000 | 36.750000 | 32.160000 |
| 100000 | 34.350000 | 28.400000 |

Эффективность устранения коллизий:

|  |  |  |
| --- | --- | --- |
| Количество элементов | Время (метод цепочек) | Время (открытая адресация) |
| 100000 | 28.820000 | 991.010000 |
| 37.090000 | 28.770000 |

Для файла, на котором производились замеры, среднее число сравнение при устранении коллизий методом цепочек составило 2.15. Максимальная длина цепочки: 7. При устранении коллизий методом открытой адресации среднее количество сравнений составило 26. Максимальное количество сравнений 279.

**Вывод:**

В данной лабораторной работе были реализованы несколько типов данных: деревья поиска, авл-деревья, хеш таблицы с 2 методами разрешения коллизий. Авл дерево является сбалансированным — в худшем случае поиск/вставка элемента займет O(logn). В простом же дереве поиска в худшем случае поиск/вставка займет O(n). Однако на поддержание дерева сбалансированным также расходуется некоторое время: после каждой операции вставки или удаления необходимо проверить, не был ли нарушен баланс, и при необходимости восстановить баланс. Также на хранение высоты дерева в каждом узле расходуется дополнительная память.

В хеш-таблицах вставка/поиск/удаление производятся за константное время - O (1); При этом, при работе с хеш-таблицами могут возникать коллизии — ситуации, когда h(k1) = h(k2) при k1!=k2; Для разрешение коллизий существует несколько методом: методы цепочек и открытая адресация. При этом время поиска элемента будет увеличиваться. Если количество сравнений для поиска элемента в среднем превышает 3–4, то поиск становится неэффективным и следует реструктурировать таблицу. реструктуризация — очень ресурсоемкая операция.

**Контрольные вопросы:**

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Дерево называется идеально сбалансированным, если число вершин в его левом и правом поддеревьях отличается не более, чем на 1

Дерево называется АВЛ деревом, если у каждого узла высота двух поддеревьев отличается не более, чем на 1.

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

В худшем случае поиск элемента в двоичном дереве поиска займет O(n), в АВЛ-дереве – O(log(n));

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

Хеш-таблица – массив, заполненный в порядке, определенным хеш-функцией.

Хеш-функция – функция, которая ставит в соответствие каждому ключу k индекс ячейки j, где расположен элемент с этим ключом.

1. Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам соответствует одно значение хеш-функции, т. е. h(k1) = h(k2) при k1! = k2

Методы устранения:

1. Открытое хеширование – метод цепочек
2. Закрытое хеширование – открытая адресация
3. В каком случае поиск в хеш-таблицах становится неэффективен?

При большом числе коллизий поиск в хеш-таблице становится не эффективен и требуется реструктуризация.

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

|  |  |  |
| --- | --- | --- |
| Структура данных | В среднем | В худшем |
| Авл дерево | O(log(n)) | O(log(n)) |
| Дерево поиска | O(log(n)) | O(n) |
| Хеш-таблица | O (1) | O(n) |
| Файл | O (n) | O (n) |