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»

Отчёт

по лабораторной работе №6

Название «Двоичные деревья поиска и хеш-таблицы»

Дисциплина «Типы и структуры данных»

Вариант 4

Студент ИУ7-31Б _____ Корниенко К.Ю.
(подпись, дата) (Фамилия И.О.)

Преподователь _____ Никульшина Т.А.
(подпись, дата) (Фамилия И.О.)

Москва, 2021

Содержание

Введение

Большинство современных информационных систем содержат в себе базы данных объем которых может превышать несколько гигабайт и к поиску данных в таких базах предъявляются особые требования, такие как максимальная скорость, прогнозируемость времени поиска и точность нахождения информации.

Простые алгоритмы перебора не способны обеспечить максимальную скорость и предоставить оценку времени выполнения операции поиска ввиду чего повсеместно заменяются на алгоритмы поиска с использованием двоичных деревьев. Доказательством этого служит сравнение скорости поиска в современных СУБД. Именно СУБД, использующие индексацию и двоичные деревья поиска показывают наибольшую производительность при поиске данных по таблице, содержащей большой объем данных. [1]

Целью данной работы является получение навыков применения двоичных деревьев, реализация основных операций над деревьями; построение и обработка хеш-таблицы; сравнение эффективности сбалансированных деревьев, двоичных деревьев поиска и хеш-таблиц.

Для выполнения поставленной цели необходимо решить следующие задачи:

- исследовать структуры данных: деревья и хеш-таблицы;
- сформулировать условие задачи;
- описать используемые структуры данных;
- привести схемы реализуемых алгоритмов;
- сформулировать требования к разрабатываемому программному обеспечению;
- определить средства программной реализации;
- реализовать алгоритм, решающий поставленную задачу;
- сравнить эффективность реализованного алгоритма для различных структур данных: сбалансированного дерева, двоичного дерева поиска, хеш-таблицы и файла.

1 Аналитический раздел

В данном разделе представлены основные теоритические сведения о двоичных деревьях и хеш-таблицах, а также описано техническое задание.

1.1 Основные теоритические сведения

Ниже представлены основные теоритические сведения, касаемые таких структур данных как двоичное дерево поиска, сбалансированное двоичное дерево поиска и хеш-таблица.

1.1.1 Двоичное дерево поиска

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

К основным операциям, выполняемым с различными структурами данных, можно отнести поиск среди набора данных. Один из широко используемых для этого методов — построение двоичного дерева поиска, которое ускоряет и упрощает задачу поиска данных в определенном наборе информации, структурирует заданную информацию для более эффективного ее дальнейшего использования, а при отсутствии необходимой информации возвращает указатель на пустой элемент [2].

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше. Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень. С учетом этого свойства поиск узла в двоичном дереве поиска можно осуществить, двигаясь от корня в левое или правое поддерево в зависимости от значения ключа поддерева.

1.1.2 Сбалансированное двоичное дерево поиска

Если при построении дерева поочередно располагать узлы слева и справа, то получится дерево, у которого число вершин в левом и правом поддеревьях отличается не более чем на единицу. Такое дерево называется идеально сбалансированным.

N элементов можно организовать в бинарное дерево с высотой не более $log_2(N)$, поэтому для поиска среди N элементов может потребоваться не больше $log_2(N)$ сравнений, если дерево идеально сбалансировано.

Адельсон-Вельский и Ландис сформулировали менее жесткий критерий сбалансированности таким образом: двоичное дерево называется сбалансированным, если у каждого узла дерева высота двух поддеревьев отличается не более чем на единицу. Такое дерево называется АВЛ-деревом.

Использование этого критерия приводит к легко выполняемой балансировке. При этом средняя длина поиска остается практически такой же, как и у идеально сбалансированного дерева. При включении узла в сбалансированное дерево возможны 3 случая: (рассматриваем включение в левое поддерево):

1) Левое и правое поддеревья становятся неравной высоты, но критерий сбалансированности не нарушается.

2) Левое и правое поддерево приобретают равную высоту и, таким образом, сбалансированность даже улучшается.

3) Критерий сбалансированности нарушается, и дерево надо перестраивать.

Алгоритм включения и балансировки существенно зависит от способа хранения информации о сбалансированности дерева. Одно из решений – хранить в каждой вершине показатель ее сбалансированности. В этом случае сбалансированность будет определяться как разность между высотой правого и левого поддеревьев

Отсюда следует, что дерево – это более подходящая структура для организации поиска, чем, например, линейный список.

1.1.3 Хеш-таблицы

Массив, заполненный в порядке, определенным хеш-функцией, называется *хеш-таблицей*. Хеш-функции позволяют по значению ключа определять сразу индекс элемента массива, в котором хранится информация. Минимальная трудоемкость поиска в хеш-таблице равна $O(1)$.

Принято считать, что хорошей является такая функция, которая удовлетворяет следующим условиям:

- функция должна быть простой с вычислительной точки зрения;
- функция должна распределять ключи в хеш-таблице наиболее равномерно.

Сформулируем понятие коллизии следующим образом. Пусть есть хеш-функция H. Если найдены две различных строки U, W со свойством $H(U) = H(W)$, то обнаружена коллизия при хешировании с помощью функции H [3].

Существует несколько возможных вариантов разрешения коллизий, которые имеют свои достоинства и недостатки.

Первый метод: внешнее (открытое) хеширование (метод цепочек).

В случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список. Таким образом, если для нескольких различных значений ключа возвращается одинаковое значение хеш-функции, то по этому адресу находится указатель на связанный список, который содержит все значения. Поиск в этом списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

Второй метод: внутреннее (закрытое) хеширование (открытая адресация).

Состоит в том, чтобы полностью отказаться от ссылок. В этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку (с шагом 1), до тех пор, пока не будет найден ключ K или пустая позиция в таблице. При этом, если индекс следующего просматриваемого элемента определяется добавлением

какого-то постоянного шага (от 1 до n), то данный способ разрешения коллизий называется линейной адресацией. Для вычисления шага можно также применить формулу:

$$h = h + a^2 \tag{1.1}$$

где a – это номер попытки поиска ключа. Этот вид адресации называется квадратичной или произвольной адресацией.

При любом методе разрешения коллизий необходимо ограничить длину поиска элемента. Если для поиска элемента необходимо более 3–4 сравнений, то эффективность использования такой хеш-таблицы пропадает и ее следует реструктуризировать (т.е. найти другую хеш-функцию), чтобы минимизировать количество сравнений для поиска элемента

1.2 Техническое задание

1.2.1 Общее задание

Построить дерево в соответствии с заданным вариантом задания. 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 дерева и степени его ветвления. Построить хеш-таблицу по указанным данным. Вывести на экран деревья и хеш-таблицу. Сравнить эффективность поиска в двоичном дереве поиска, в сбалансированном дереве поиска и в хеш-таблице. Вывести на экран измененные структуры. Подсчитать среднее количество сравнений для поиска данных в указанных структурах. Произвести реструктуризацию хеш-таблицы, если среднее количество сравнений больше указанного. Оценить эффективность использования этих структур (по времени и памяти) для поставленной задачи.

1.2.2 Задание варианта

Вариант 4:

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.

2 Конструкторский раздел

В данном разделе представлена информация об используемых структурах данных, а также приведены схемы реализуемых алгоритмов.

3 Технологический раздел

В данном разделе представлены требования к разрабатываемому ПО, а также представлены листинги кодов реализованных алгоритмов.

3.1 Средства реализации

Выбранным языком для реализации алгоритмов является структурный язык C, в соответствии с требованиями, предъявляемыми курсом «Типы и структуры данных».

Для получения исполняемого файла используется компилятор gcc [4]. В качестве инструмента сборки используется утилита make [5].

3.2 Требования к ПО

Ниже представлены требования к разрабатываемому ПО.

- на вход программа получает имя файла, в котором содержатся целые числа, элемент, который необходимо добавить в дерево, размерность хеш-таблицы и максимальное среднее количество сравнений для хеш-функции;
- на выходе – дерево из чисел файла, сбалансированное дерево, дерево после вставки элемента, сбалансированное дерево после вставки элемента, хеш-таблица, время добавления и поиска в двоичном дереве поиска, АВЛ-дереве, объемы занимаемой памяти каждой структурой.

4 Экспериментальный раздел

В данном разделе представлены результаты сравнения эффективности по времени реализованного алгоритма, при использования различных структур данных, таких как двоичное дерево поиска, АВЛ-дерево, хеш-таблица и файл.

5 Контрольные вопросы

1) Что такое дерево?

Дерево — рекурсивная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Эта структура описывается рекурентно как узел, у которого есть указатели на другие узлы (поддеревья).

2) Как выделяется память под представление деревьев?

Память для представления в виде связного списка выделяется динамически в момент добавления новых ключей.

3) Какие стандартные операции возможны над деревьями?

Стандартные операции над деревьями включают в себя вставку узла в дерево, поиск узла, балансировка дерева. Также возможно отделить поддерево в отдельное дерево.

4) Что такое дерево двоичного поиска?

Дерево двоичного поиска - это дерево, в котором для каждого узла задано отношение порядка таким образом, что этот узел меньше одного своего поддерева, но больше другого поддерева.

5) Чем отличается идеально сбалансированное дерево от АВЛ-дерева?

Идеально сбалансированное дерево определяется как дерево двоичного поиска, в котором у каждого узла количество узлов в обоих его поддеревьях отличается не более чем на единицу. В АВЛ деревьях это требование ослаблено. В них у каждого узла высоты обоих его поддеревьев отличаются не более чем на единицу.

6) Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?

Поиск в сбалансированном дереве зачастую происходит быстрее, так как высота несбалансированного дерева как правило превосходит высоту того же сбалансированного дерева.

7) Что такое хеш-таблица, каков принцип ее построения?

Хеш-таблица это структура для данных с произвольным доступом к ним. Принцип построения хеш-таблицы основан на особой функции, называемой хеш-функцией, которая сопоставляет уникальный ключ с его местом в таблице. Идеальная хеш-функция - это инъекция множества ключей во множество мест в таблице.

8) Что такое коллизии? Каковы методы их устранения.

Коллизии — это ситуации, когда для разных ключей выбранная хеш-функция возвращает одно и то же значение.

Коллизии могут возникать на этапе "упаковки"расчитанного большого хеша в размерность таблицы. То есть хеш-значения разных ключей могут быть разными, но при упаковке они получают одно и то же место в таблице. Такого рода коллизии могут быть устранены изменением размерности таблицы.

9) В каком случае поиск в хеш-таблицах становится неэффективен?

Поиск в хеш-таблице может становиться неэффективным в случаях большого числа коллизий, из-за которых нужно будет производить дополнительный последовательный поиск по ключам, имеющим одинаковый хеш.

10) Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах. В хеш-таблице минимальное время поиска $O(1)$. В АВЛ: $O(log_2 n)$. В дереве двоичного поиска $O(h)$, где h - высота дерева (от $log_2 n$ до n).

Заключение

Список использованных источников

1. Сергеев М.И., Янишевская А.Г. АВЛ-деревья, выполнение операций над ними // ИВД. 2016. №2 (41). — (дата обращения: 04.12.2021). https://cyberleninka.ru/article/n/avl-derevya-vypolnenie-operatsiy-nad-nimi.

2. *Вирт, Н.* Алгоритмы и структуры данных / Н. Вирт. — ДМК Пресс, 2010. — С. 272.

3. Исканцев Н. В. Математическая теория стойкости хеш-функций к коллизиям // Наука и современность. 2012. №16-2. — (дата обращения: 04.12.2021). https://cyberleninka.ru/article/n/matematicheskaya-teoriya-stoykosti-hesh-funktsiy-k-kolliziyam.

4. GCC, the GNU Compiler Collection. — (дата обращения: 04.12.2021). https://gcc.gnu.org/.

5. GNU Make. — (дата обращения: 04.12.2021). https://www.gnu.org/software/make/.