МГТУ им. Н.Э. Баумана

**Дисциплина Типы и Структуры данных.**

**Лабораторный практикум №6**

**по теме:** «Деревья, хеш–таблицы»

Работу выполнил:

студент группы ИУ7-35Б

Прянишников Александр

**Цель работы**: построить дерево, вывести его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; построить хеш-таблицу и вывести ее на экран, устранить коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировав таблицу; сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска (ДДП), в хеш-таблицах и в файлах. Сравнить эффективность реструктуризации таблицы для устранения коллизий и поиска в ней с эффективностью поиска в исходной таблице.

# Условие задачи. Вариант 3

# 

# Требования к задаче

**Входные данные**

Пользователь вводит количество сравнений в хэш–таблице. В ручном режиме – очередной элемент, который добавляется в структуры.

**Вывод данных**

К выходным данным относятся: дерево, АВЛ–дерево, хэш–таблица, количество коллизий, ключ для хэш–функции.

Также в программу был добавлен интерфейс, упрощающий работу для пользователей: есть пояснительные надписи, перед вводом данных написано, как правильно вводить данные.

**Описание задачи, реализуемой программой**

Программа предоставляет пользователю интерфейс для работы с деревом, АВЛ–деревом, хэш–таблицей. Она добавляет новые элементы во все структуры, в случае чего реструктуризует хэш–таблицу. Также она выводит таблицу сравнения эффективности четырёх способов хранения данных. Пользователь также может менять количество сравнений в хэш–таблице.

**Способ обращения к программе.**

Программу можно запустить через терминал.

**Описание возможных аварийных ситуаций и ошибок пользователя**

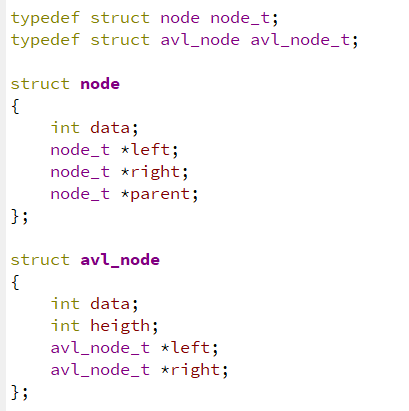
* Выбор отсутствующего пункта меню (от 0 до 4)  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести новое значение пункта меню.*
* Ввод букв при вводе числа  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести новое значение пункта меню.*
* Пустой ввод  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести новое значение пункта меню.*
* Количество сравнений превышено при добавлении элемента  
  *В этом случае программа переопределит новую хэш–функцию и добавит все элементы, включая новый, в новую таблицу.*
* Добавление нового элемента, который уже есть в структурах.  
  *В этом случае программа просто выдаст на экран, что элемент уже добавлен.*
* Ввод отрицательного числа сравнений  
  *В этом случае программа сообщит пользователю, что надо заново ввести число.*
* Удаление элемента, которого нет в структуре

*В этом случае программа сообщит пользователю, что удаление невозможно.*

# Описание внутренних структур данных.

Для решения этой задачи я решил использовать три структуры: обычное дерево, АВЛ–дерево и хэш–таблица.

Вот так описаны структуры:



Хэш–таблица в связи с методом «закрытого хэширования» представлена в виде статического массива элементов int.

# Описание алгоритма

Сначала опишем пункты меню, которые можно вызвать:

**1 – Вывод текущего состояния обычного дерева**Программа выводит на экран текущее состояние обычного дерева.

**2 – Вывод текущего состояния АВЛ–дерева**  
Программа выводит на экран текущее состояние АВЛ–дерева.

**3 – Вывод текущего состояния хэш–таблицы**Программа выводит на экран ненулевые элементы хэш–таблицы, а также текущий ключ для функции и текущее количество сравнений.

**4 – Добавить элемент**Программа принимает у пользователя элемент, смотрит, содержится ли он в структурах, и, если нет, добавляет в каждую.

**5 – Удалить элемент**Программа принимает у пользователя элемент, смотрит, содержится ли он в структурах, и, если есть, удаляет его из структур.

**6 – Сравнить эффективность структур**Программа считывает со специально заготовленных файлов числа, проводит операцию добавления и подсчитывает среднее время добавления в каждую структуру (файл, дерево, АВЛ–дерево, хэш–таблица).

**7 – Сменить количество сравнений**Пользователь вводит новое количество сравнений для хэш–таблицы, после чего вычисления происходят при новом значении.

**0 – Выход.**

Теперь обговорим алгоритмы добавления в структуры нового элемента.

В случае обычного дерева программа спускается по дереву, если новое значение больше значения в очередном узле – она проходит направо, иначе – налево. Когда достигнут узел без значения, туда записывается новый элемент.

В случае АВЛ–дерева сначала происходит всё также, как в обычном дереве, но затем происходит балансировка: если в одной вершине условие АВЛ–дерева не выполняется, то происходят операции поворота, и АВЛ–дерево балансируется.

В случае хэш–таблицы для элемента происходит расчёт значения индекса по формуле: **index = (сумма цифр + само число) % p**, где p – простое число, ключ хэш–функции. Изначально p = 31, затем оно меняется при реструктуризации до ближайшего простого. Если на том месте уже есть элемент, то программа использует квадратичную индексацию. Если количество сравнений превысило заданное, то происходит реструктуризация: выбирается новый ключ хэш–функции (ближайшее простое число к текущему ключу к хэш–функции), после чего всё повторяется.

Размер хэш–таблицы равняется текущему значению ключа хэш–функции, так что при реструктуризации хэш–таблица меняется по памяти.

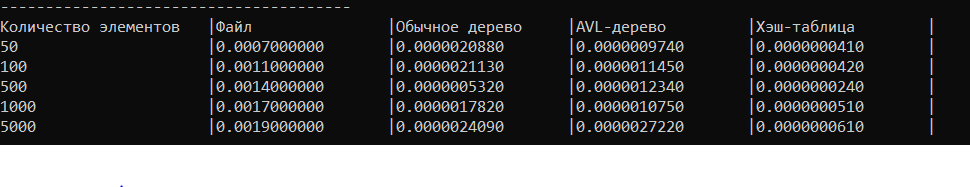
# Тесты

Тестирование происходило по такому принципу: каждый пункт тестировался отдельно, после чего контролировалась работа программы целиком. Поэтому здесь указаны только негативные тесты, которые в целом проверяли работоспособность модулей программы.

|  |  |
| --- | --- |
| **123345fd** | **Буквы при задании числа** |
|  | **Пустой ввод** |
| **–1** | **Выбор неправильного пункта меню** |
| **Элемент уже добавлен.** | **Добавление имеющегося элемента** |
| **Элемента нет в очереди** | **Удаление элемента, которого нет.** |

# Оценка эффективности

Сравнение эффективности по времени доступно в самой программе.



Как видно, хэш–таблицы работают в 10–20 раз быстрее, чем деревья, и в 10000 раз быстрее, чем файл. Обычное дерево при этом чаще работает медленнее, чем АВЛ–дерево, но также это зависит от того, требуется ли балансировка, и как расположились числа изначально. При этом от N зависимость есть.

Построим аналогичную таблицу для памяти, размер очереди – 10000 элементов.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Обычное дерево | АВЛ–дерево | Хэш–таблица |
| N = 50 | 1600 | 1200 | 212 |
| N = 100 | 3200 | 2400 | 404 |
| N = 500 | 16000 | 12000 | 2012 |
| N = 1000 | 32000 | 24000 | 4004 |
| N = 5000 | 160000 | 120000 | 20020 |

Как видно, обычное дерево проигрывает по памяти остальным типам данным. Хэш–таблица расширяется динамически, поэтому она является самым эффективным типом с точки зрения памяти.

Надо сказать, что хэш–таблице желательно, что данные отличались друг от друга, иначе может возникнуть ситуация, когда добавить элемент в таблицу станет невозможно. Я с этим столкнулся, когда добавлял элемент от 0 до 500, и на третьей сотне для некоторых чисел невозможно было подобрать новый элемент.

Также, построим таблицу среднего количества сравнений:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Обычное дерево | АВЛ–дерево | Хэш–таблица |
| N = 50 | 12 | 6 | 2 |
| N = 100 | 14 | 8 | 3 |
| N = 500 | 16 | 9 | 2 |
| N = 1000 | 22 | 10 | 2 |
| N = 5000 | 27 | 13 | 3 |

Последний пункт для изучения: зависимость времени работы хэш–таблицы от максимального количества сравнений. Возьмём таблицу для 100 элементов.

Время в секундах.

|  |  |
| --- | --- |
| N = 4 | 0.0000000360 |
| N = 10 | 0.0000000770 |
| N = 20 | 0.0000001470 |
| N = 50 | 0.0000003500 |
| N = 100 | 0.0000007260 |

Зависимость строго линейная, но хэш–таблицы быстро теряют свою эффективность.

# Выводы по проделанной работе

Сегодня я познакомился с тремя новыми методами хранения данных: обычным деревом, АВЛ–деревом и хэш–таблицей. Хэш–таблицы эффективны по времени работы и по памяти, при этом если данные слишком близки к друг другу, то хэш–таблица не всегда позволяет работать с такими данными. АВЛ–дерево всегда выигрывает по количеству сравнений у обычного дерева, но далеко не всегда – по времени.

# Ответы на контрольные вопросы

1. Что такое дерево?

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

2. Как выделяется память под представление деревьев?

Под каждый новый узел динамически выделяется новое количество памяти.

3. Какие стандартные операции возможны над деревьями?

Добавление элемента, удаление элемента, поиск элемента, обход дерева.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

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.

Двоичное дерево называется сбалансированным, если у каждого узла дерева высота двух поддеревьев отличается не более чем на единицу. Такое дерево называется АВЛ-деревом.

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

В АВЛ–дереве меньше количество сравнений.

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

Хеш-функцией ставит в соответствие каждому ключу ki индекс ячейки j, где расположен элемент с этим ключом. Массив, заполненный в порядке, определенным хеш-функцией, называется хеш–таблицей.

8. Что такое коллизии? Каковы методы их устранения.

Может возникнуть ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2. Такая ситуация называется коллизией.

Основные методы: метод цепочек, метод закрытого хэширования.

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

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.

10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш–таблицах

Хэш–таблица – О(1).

Дерево – О(h), где h – глубина дерева.

АВЛ–дерево – О(logN).