

*Отчет по дисциплине:*

**Типы и структуры данных**

*Лабораторная работа:*

**“Деревья, хэш-таблицы”**

Выполнил студент: Сироткина П.Ю.

Группа: ИУ7-36Б

Преподаватель: Силантьева А.В.

**Москва, 2020 г.**

1. **Условие задачи**

Построить двоичное дерево поиска (далее - ДДП), в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева.

Сбалансировать полученное дерево и вывести его на экран.

Удалить указанное слово в исходном и сбалансированном дереве. Сравнить время удаления и объем памяти.

Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана, используя метод цепочек для устранения коллизий. Вывести построенную таблицу слов на экран.

Осуществить удаление введенного слова, вывести таблицу.

Сравнить время удаления, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев, хеш-таблиц и файла.

1. **Описание ТЗ**

*А) Входные и выходные данные:*

Входные данные:

Целое число от 1 до 9, представляющее собой пункт меню программы.

В зависимости от выбранного пункта может потребоваться ввод дополнительных данных, информация о которых будет сообщена пользователю (например, ввод слова для удаления или указание максимального количества сравнений для хэш-таблицы).

Выходные данные:

Командно-зависимо:

* сообщение об успешности ввода данных;
* сообщение об успешности построения графа и вывод информации о том, как получить изображение графа (как для ДДП, так и для сбалансированного дерева);
* удаление элемента из графа и вывод соответствующей информации на экран, построение хэш-таблицы и ее вывод на экран;
* удаление слова из хэш-таблицы и вывод полученной таблицы на экран;
* вывод сравнительного анализа по времени выполнения базовых операций (добавление, удаление, поиск, создание) для следующих структур данных: ДДП, сбалансированное дерево, хэш-таблица и файл;
* вывод справки о программе на экран;

*Б) Задачи, реализуемые программой:*

Программа способна решать следующие задачи, которые запускаются посредством выбора соответствующего пункта меню:

**1) Ввести данные в систему из текстового файла** - сохранение информации об этом файле для дальнейшей работы с этими данными посредством разных структур данных.

**2) Построить ДДП** - построение ДДП с заранее введенными в систему данными из текстового файла. Затем выводится информация о том, как получить изображение полученного графа (для этого необходимо кликнуть на соответствующую иконку в текущей директории, после этого появится изображение графа в формате PNG).

**3) Построить сбалансированное дерево** - построение сбалансированного дерева с данными из заранее введенного текстового файла. Затем выводится информация о том, как получить изображение полученного графа (для этого необходимо кликнуть на соответствующую иконку в текущей директории, после этого появится изображение графа в формате PNG).

**4) Удалить указанное слово в исходном и сбалансированном дереве -** производится удаление введенного пользователем слова из исходного и сбалансированного деревьев. Если указанного слова нет в графе, выводится соответствующее сообщение, иначе - информация о том, как получить изображение полученного графа (для этого необходимо кликнуть на соответствующую иконку в текущей директории, после этого появится изображение графа в формате PNG).

**5) Построить хэш-таблицу -** построение хэш-таблицы с данными из заранее введенного текстового файла, затем вывод этой таблицы на экран. Если количество сравнений превышает указанное, хэш-функция автоматически меняется и таблица реструктуризируется. Если вновь не удалось добиться минимального количества сравнений, выводится соответствующая информация.

**6) Удалить слово из хэш-таблицы -** производится удаление указанного слова из хэш-таблицы. Если указанного слова нет, выводится соответствующее сообщение, иначе - выводится измененная таблица.

**7) Анализ эффективности реализованных структур данных** - на экран выводится сравнительный анализ по времени выполнения базовых операций (добавление, удаление, поиск, создание) для следующих структур данных: ДДП, сбалансированное дерево, хэш-таблица и файл.

**8) Справка о программе -** вывод на экран справки о программе.

**9) Выход -** завершение работы программы.

*В) Способы обращения к программе:*

Пользователь работает с программой посредством меню, команды которого описаны выше. Информация о необходимых дополнительных действиях будет своевременно и подробно выведена на экран (например, о том, как получить изображение графа в формате PNG).

1. **Описание внутренних структур данных**

Структура для хранения узла дерева (как ДДП, так и сбалансированного):

**typedef struct tree\_node**

**{**

**char data[MAX\_WORD\_LEN];**

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

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

**} tree\_node\_t;**

Поле **data** - информационная часть узла (слово), где MAX\_WORD\_LEN - максимально допустимая длина слова (словом считается любая непробельная последовательность символов).

Поле **left** - указатель на левого потомка узла дерева.

Поле **right** - указатель на правого потомка узла дерева.

Структура для хранения связного списка, который используется в хэш-таблице для устранения коллизий методом цепочек:

**typedef struct list**

**{**

**char data[MAX\_WORD\_LEN];**

**struct list \*next;**

**} list\_t;**

Поле **data** - информационная часть узла (слово), где MAX\_WORD\_LEN - максимально допустимая длина слова (словом считается любая непробельная последовательность символов).

Поле **next** - указатель на следующий элемент в списке.

1. **Описание алгоритма**

* Вставка ключа в ДДП

Возможны несколько ситуаций:

* Дерево пусто. Тогда вставляемый узел становится корнем дерева.
* Ключ не больше текущего узла. Тогда рекурсивно вставляем ключ в левое поддерево.
* Ключ больше ткущего узла. Тогда рекурсивно вставляем ключ в правое поддерево.
* Удаление ключа из ДДП:

Возможны несколько ситуаций:

* Узла с таким ключом нет в дереве - ничего делать не надо.
* Значение ключа совпадает с текущим узлом. Тогда, если у узла нет правого поддерева, заменить узел на левого потомка с наследованием. Если же правый узел есть, найти в правом поддереве минимальный элемент и текущий элемент заменить на него.
* Значение ключа не больше текущего узла. Тогда рекурсивно удаляем ключ из левого поддерева.
* Значение ключа больше текущего узла. Тогда рекурсивно удаляем ключ из правого поддерева.
* Поиск в ДДП (АВЛ - так же)

Возможны несколько ситуаций:

* + Узла с таким ключом нет в дереве - ничего делать не надо.
  + Значение ключа совпадает с текущим узлом. Поиск завершен.
  + Значение ключа не больше текущего узла. Тогда рекурсивно ищем ключ в левом поддереве.
  + Значение ключа больше текущего узла. Тогда рекурсивно ищем ключ в правом поддереве.

Удаление и вставка ключа в АВЛ-дереве осуществляется так же, как и в ДДП, за исключением того, что перед выходом из каждого рекурсивного вызова необходимо проводить балансировку узла.

Если балансирующий фактор (разница высот правого и левого поддерева с корнем в текущем узле) равна 2, то, если фактор правого поддерева равен -1, необходимо произвести большой левый поворот, иначе малый. Аналогично для -2.

Создание дерева - комбинация описанных выше функций.

* Хэш-таблица: реализованы 2 хэш-функции (использующие деление и умножение). Для каждого слова вычисляется значение хэш-функции и размещается по соответствующему адресу, используя метод цепочек для устранения коллизий.

**Тестирование программы**

|  |  |
| --- | --- |
| **Действие** | **Результат** |
| Ввод 1  Ввод wrong.txt (несуществующий файл) | Сообщение: “Ошибка: невозможно открыть файл с таким названием.  Внимательно проверьте название файла и попробуйте еще раз.” |
| Ввод 1  Ввод data.txt (файл с некорректными данными) | Сообщение: “Ошибка: файл содержит некорректные данные.” |
| Ввод 1  Ввод data.txt (пустой файл) | Сообщение: “Ошибка: файл содержит некорректные данные.” |
| Ввод 1  Ввод data.txt (корректный файл)  Ввод 2  Выполнение предложенного действия | Сообщение “Граф успешно считан. Для того, что получить представление графа в виде изображения, кликните по иконке с названием make\_image.cmd в текущей директории.  Png-файл с соответствующим названием появится в директории images.”  Изображение: |
| Ввод 1  Ввод data.txt (корректный файл)  Ввод 2  Ввод 3  Выполнение предложенных действий | Сообщение об успешности выполнения операций и инструкция для получения изображений. Изображения:  ДДП:    Сбалансированное дерево: |
| Ввод 1  Ввод data.txt (корректный файл)  Ввод 2  Ввод 3  Ввод 4  Удаление f | Сообщение об успешности выполнения операций и инструкция для получения изображений. Выполнение удаления.  Изображения:    Вывод оценки эффективности: |
| Аналогично прошлому пункту, при удалении нарушается сбалансированность сбалансированного графа | Сообщение об успешности выполнения операций и инструкция для получения изображений. Выполнение удаления.  Изображения:  До:    Удаление 2  После: |
| Аналогично прошлому пункту, указанного слова нет в графах | Сообщение: “Узел с таким значением не найден в деревьях.” |
| Выбор 1  Выбор 5  Размерность: 5  Макс. коллизии: 3 | Ровно 3 коллизии. |
| Выбор 1  Выбор 5  Размерность: 5  Макс. коллизии: 2 | Печать таблицы из прошлого пункта. Сообщение о превышении количества коллизий. Перестройка на новую хэш-функцию, печать:    Сообщение: “Превышено максимально допустимое количество коллизий. Использованы все имеющиеся хэш-функции. Попробуйте увеличить размерность таблицы.” |
| Выбор 1  Выбор 5  Размерность: 10  Макс. коллизии: 1 |  |
| Удаление из связного списка в хэш-таблице | Было:  Стало: |

1. **Оценка эффективности**

Временной анализ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во элементов | Операция | ДДП | Сбалансир.  ДДП | Хэш-таблица | Файл |
| **10** | Удаление | **75** тиков  3 сравн. | **38** тиков  2 ср. | **90** тиков  0 сравн. | **169360** тиков  10 сравн. |
| Добавление | **247** тиков  4 сравн. | **235** тиков  5 сравн. | **236** тиков  0 сравн. | **15756** тиков  0 сравн. |
| Создание | **20113** тиков  22 сравн. | **23103** тиков  21 сравн. | **47498** тиков  1 сравн. | **33802** тиков  0 сравн. |
| Поиск | **61** тиков  2 сравн. | **74** тиков  3 сравн. | **75** тиков  0 сравн. | **16880** тиков  9 сравн. |
| **100** | Удаление | **145** тиков  10 сравн. | **26** тиков  6 сравн. | **83** тиков  0 сравн. | **27569** тиков  100 сравн. |
| Добавление | **193** тиков  2 сравн. | **227** тиков  3 сравн. | **193** тиков  0 сравн. | **26201** тиков  0 сравн. |
| Создание | **47361** тиков  680 сравн. | **266274** тиков  537 сравн. | **61335** тиков  64 сравн. | **55740** тиков  0 сравн. |
| Поиск | **226** тиков  9 сравн. | **146** тиков  6 сравн. | **89** тиков  0 сравн. | **33226** тиков  100 сравн. |
| **1000** | Удаление | **212** тиков  14 сравн. | **26** тиков  13 сравн. | **81** тиков  0 сравн. | **1219182** тиков  833 сравн. |
| Добавление | **342** тиков  12 сравн. | **249** тиков  13 сравн. | **202** тиков  0 сравн. | **16063** тиков  0 сравн. |
| Создание | **484087** тиков  11061 сравн. | **20928508** тиков  8746 сравн. | **373371**тиков  935 сравн. | **242844** тиков  0 сравн. |
| Поиск | **299** тиков  13 сравн. | **180** тиков  8 сравн. | **398** тиков  0 сравн. | **119538** тиков  221 сравн. |

Для замеров удаления, поиска и добавления производилось 10.000 замеров, для создания - 1.000.

Объем занимаемой памяти:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | ДДП | Сбалансир. ДДП | Хэш-таблица | Файл |
| 2 | 148 | 148 | 124 | 16 |
| 4 | 296 | 296 | 248 | 32 |
| 8 | 592 | 592 | 496 | 64 |
| 16 | 1184 | 1184 | 992 | 128 |
| 32 | 2368 | 2368 | 1984 | 256 |
| 64 | 4736 | 4736 | 3968 | 512 |
| 128 | 9472 | 9472 | 7936 | 1024 |
| 256 | 18944 | 18944 | 15872 | 2048 |
| 512 | 37888 | 37888 | 41744 | 4096 |
| 1024 | 75776 | 75776 | 63488 | 8192 |

*Анализируя эти таблицы, можно прийти к следующему выводу:*

АВЛ-дерево создать сложнее (в среднем в 6-8 раз медленнее, чем ДДП), однако операции производятся быстрее (удаление эффективнее в среднем 5-7 раз, вставка в 1.5-2 раза, поиск аналогично 1.5-2 раза, количество сравнений также уменьшается примерно во столько же раз. Чем “глубже” находится необходимый узел, тем более эффективна АВЛ-структура, и преимущество будет только увеличиваться). АВЛ-дерево и ДДП требуют одинаковый объем памяти, т.к. способ хранения не меняется - меняется алгоритм.

Сравнивая дерево поиска и хэш-таблицу, приходим к выводу, что более эффективна хэш-таблица (Создается чуть медленнее ДДП, но существенно (6-9 раз) быстрее АВЛ-дерева. Базовые операции также производятся быстрее, хоть и ненамного). Хэш-таблица занимает меньше места, чем древесные структуры (74 байта против 62 на один узел).

Файл неэффективен всегда, особенно при выполнении базовых операций файл значительно проигрывает (несколько сотен раз). Единственный плюс - алгоритмы существенно проще и объем памяти меньше, т.к. хранятся только указатели на строки.

1. **Ответы на контрольные вопросы**

*1. Что такое дерево?*

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

Дерево с базовым типом Т определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел типа Т с конечным числом древовидных структур этого же типа, называемых поддеревьями.

*2. Как выделяется память под представление деревьев?*

В виде связного списка (информационная часть + 2 указателя на левого и правого потомка соответственно).

*3. Какие стандартные операции возможны над деревьями?*

Вставка ключа, удаление ключа, поиск ключа, 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, при этом у каждой вершины имеется не более двух потомков.

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

Дерево считается идеально сбалансированным, если для каждой вершины дерева число вершин в левом и правом поддеревьях отличается не более чем на 1, критерий для АВЛ-дерева менее жесткий - для каждой вершины дерева высота левого и правого поддеревьев отличается не более чем на 1.

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

Поиск в АВЛ-дереве происходит быстрее, чем в ДДП (трудоемкость поиска в АВЛ-дереве О(logn), в ДДП трудоемкость пропорциональна высоте дерева и может составлять от O(logn) до О(n)).

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

Хэш-таблица - таблица(массив), индексам которого ставится в соответствие (с помощью некоторой хэш-функции) значение ключа.

Хорошая хэш-функция должна распределять ключи равномерно по массиву, порождать минимальное число коллизий, также она должна быть простой с вычислительной точки зрения.

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

Коллизия - ситуация, когда разным значениям ключа соответствует один и тот же индекс хэш-таблицы (hash(key1) == hash(key2) && key1 != key2).

Способы устранения:

* Открытое (внешнее) хэширование - в случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список.
* Закрытое хэширование (открытая адресация) - 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.

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

Поиск в хэш-таблице становится неэффективным по мере возрастания количества коллизий.

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

В хэш-таблице минимальная трудоемкость поиска О(1).

В АВЛ-дереве минимальная трудоемкость поиска О(logn).

В ДДП минимальная трудоемкость поиска от О(logn) до O(n), в общем случае O(высота дерева).