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

Отчет по лабораторной работе №6 по предмету

“Типы и структуры данных”

Обработка деревьев и хеш-таблиц.

Выполнила:

Кузнецова Анастасия

Группа:

ИУ7-31Б

Вариант 1 (по списку 13)

Проверил(а):

**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.

**Описание условия задачи**

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из слов текстового файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного слова в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

**2. Описание ТЗ:**

Описание исходных данных и результатов

*Входные данные:*

Программа представляет собой меню, пользователь должен ввести свой выбор (целое число от 0 до 5)

Во входном файле хранятся слова. Если пользователь хочет выполнить поиск слова, то на вход подаётся слово – строка, если же пользователь выбрал реструктуризацию хэш-таблицы, то на вход подаётся максимальное количество сравнений – целое число.

Ограничения для входных данных:

* Максимальное количество сравнений должно быть целым числом
* Максимальное количество сравнений должно быть >0
* В файле все слова должны быть разными

*Выходные данные:*

Программа выводит меню и в зависимости от выбора пользователя выводит:

1. Вывод ДДП
2. Вывод АВЛ дерева
3. Вывод хэш-таблицы
4. Вывод результатов поиска в ДДП, АВЛ-дереве, хэш-таблице и файле и количество произведенных сравнений в этих структурах
5. Вывод текущего среднего количества сравнений в хэш-таблице, если количество сравнений в хэш-таблице больше введённого максимального, то вывод реструктуризированной хэш-таблицы.
6. Вывод результатов сравнения времени, памяти и количества сравнений при 4х реализациях

Описание задачи, реализуемой программой

Программа выводит меню и выполняет задачу, запрошенную пользователем. В программе реализовано меню с выборами действий:

1. Выход
2. Вывод ДДП
3. Вывод АВЛ дерева
4. Вывод хэш-таблицы
5. Вывод результатов поиска в ДДП, АВЛ-дереве, хэш-таблице и файле и количество произведенных сравнений в этих структурах.
6. Если количество сравнений в хэш-таблице больше введённого максимального, то вывод реструктуризированной хэш-таблицы
7. Вывод результатов сравнения времени, памяти и количества сравнений при 4х реализациях

Способ обращения к программе

Программу следует вызывать из командной строки с помощью команды:

./app.exe input.txt

Описание возможных аварийных ситуаций и ошибок пользователей

* Неверное количество аргументов командной строки
* Несуществующий входной файл
* Пустой входной файл
* Вместо максимального количества сравнений введена буква
* Максимальное количество сравнений <=0
* В файле есть одинаковые слова

**3. Описание внутренних СД**

ДДП:

typedef struct bin\_tree\_node bin\_tree\_node\_t;

struct bin\_tree\_node

{

char \*word;

bin\_tree\_node\_t \*left;

bin\_tree\_node\_t \*right;

};

АВЛ дерево:

typedef struct avl\_tree\_node avl\_tree\_node\_t;

struct avl\_tree\_node

{

char \*word;

int height;

avl\_tree\_node\_t \*left;

avl\_tree\_node\_t \*right;

};

Хэш-таблица:

typedef struct table\_node table\_node\_t;

struct table\_node {

char \*value;

table\_node\_t \*next;

};

typedef struct hash\_table hash\_table\_t;

struct hash\_table {

table\_node\_t \*\*table;

int size;

};

**4. Описание алгоритма**

1. Вывод меню

2. В зависимости от выбора пользователя:

2.0 Выход

2.1.1 Считывание из файла в ДДП слов

2.1.2 Вывод ДДП

2.2.1 Считывание из файла в АВЛ дерево слов

2.2.2 Балансировка АВЛ дерева

2.2.3 Вывод АВЛ дерева

2.3.1 Считывание из файла в хэш-таблицу слов

2.3.2 Вывод хэш-таблицы

2.4.1 Ввод искомого слова

2.4.2 Поиск слова в ДДП

2.4.3 Вывод результата поиска в ДДП

2.4.4 Поиск слова в АВЛ дереве

2.4.5 Вывод результата поиска в АВЛ дереве

2.4.6 Поиск слова в хэш-таблице

2.4.7 Вывод результата поиска в хэш-таблице

2.4.8 Поиск слова в файле

2.4.9 Вывод результата поиска в файле

2.5.1 Вывод текущего среднего количества сравнений в хэш-таблице

2.5.2 Ввод максимального среднего количества сравнений

2.5.3 Реструктуризация хэш-таблицы, если текущее среднее количество сравнений больше максимального

2.5.4 Вывод реструктуризированной хэш-таблицы

2.6.1 Вывод результатов сравнения времени, памяти и количества сравнений при 4х реализациях

**5. Набор тестов**

Позитивные тесты:

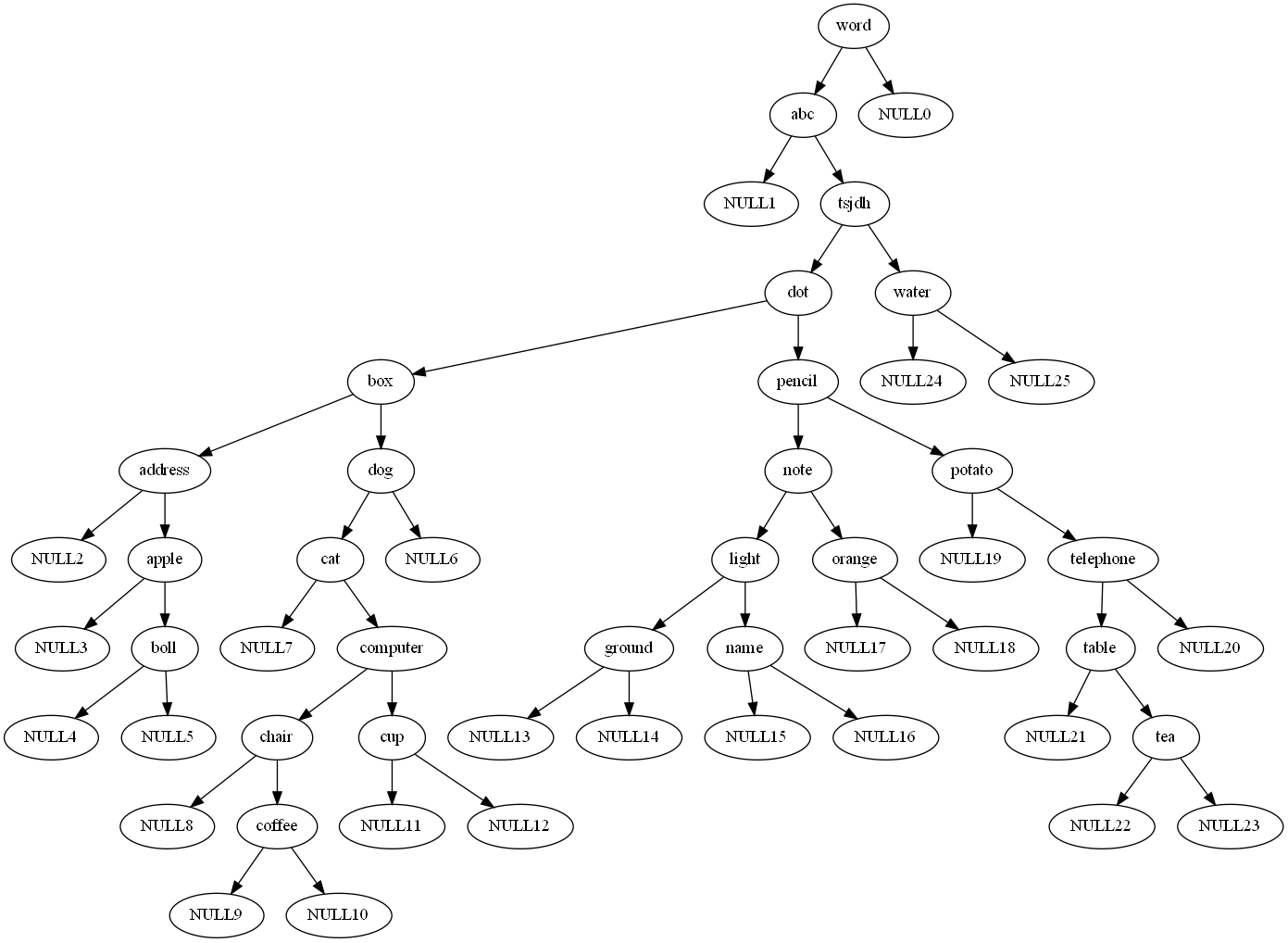
1. Вывод ДДП

Input:

Enter a choice:

1

Output:



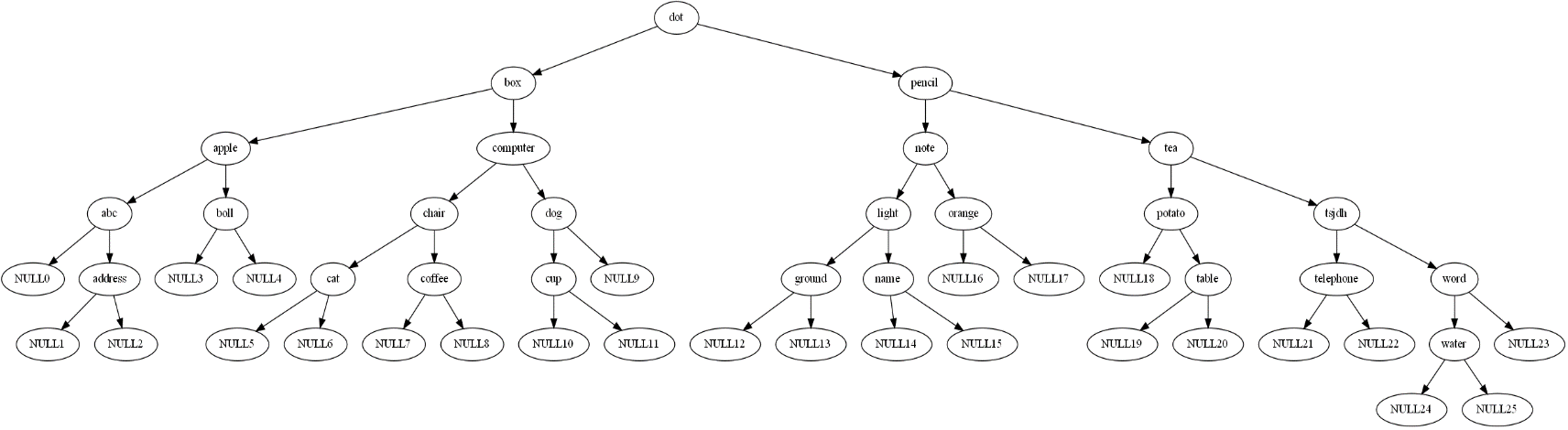
1. Вывод АВЛ дерева

Input:

Enter a choice:

2

Output:



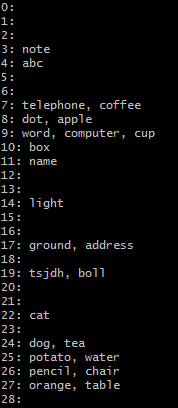
1. Вывод хэш-таблицы

Input:

Enter a choice:

3

Output:



1. Поиск существующего элемента

Input:

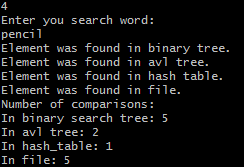
Enter a choice:

4

Enter you search word:

pencil

Output:



1. Поиск несуществующего элемента

Input:

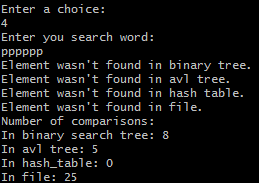
Enter a choice:

4

Enter you search word:

pppppp

Output:



1. Реструктуризация хэш-таблицы

Input:

Enter a choice:

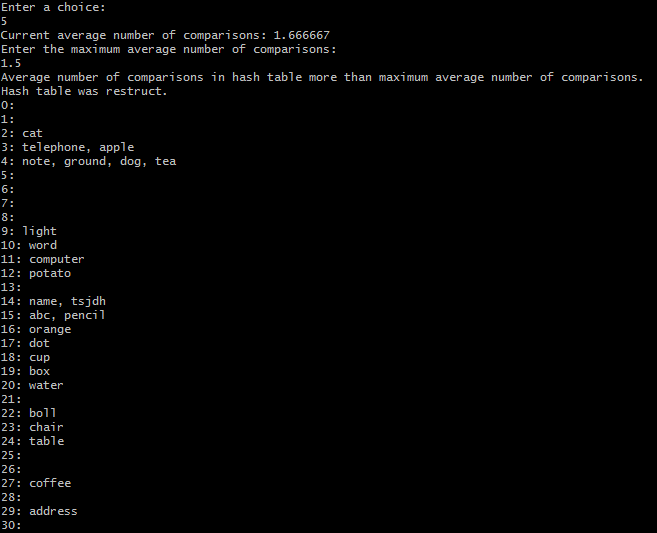
5

Current average number of comparisons: 1.666667

Enter the maximum average number of comparisons:

1.5

Output:



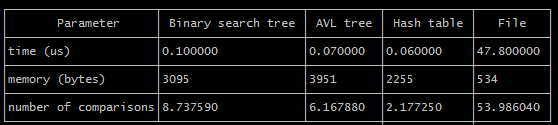
1. Вывод результатов эффективности 4х реализаций

Input:

Enter a choice:

6

Output:



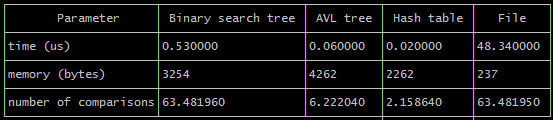
1. Проверка эффективности на правостороннем дереве

Input:

Enter a choice:

6

Output:

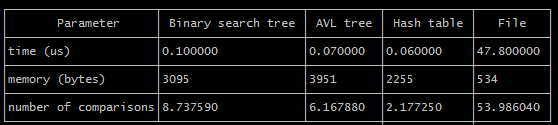


Негативные тесты:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Описание | Input | Output |
| 1 | Количество аргументов командной строки не равно 2 | Вызов программы в командной строке:  ./app.exe | Key error. |
| 2 | Входной файл не существует | Вызов программы в командной строке:  ./app.exe aaaaaa.txt | File error. |
| 3 | Входной файл пустой | Вызов программы в командной строке:  ./app.exe empty.txt | Input error. |
| 4 | Вместо максимального количества сравнений введена буква | Enter a choice:  5  Enter the maximum average number of comparisons:  as |  |
| 5 | Максимальное количество сравнений <= 0 | Enter a choice:  5  Enter the maximum average number of comparisons:  -1 |  |
| 6 | В файле есть одинаковые слова | Enter a choice:  1 | Input error. |

1. **Сравнение времени и памяти при двух реализациях.**

Я сравнивала время поиска элемента в ДДП, АВЛ дереве, хэш-таблице и файле. Замеры проводила на 100000 повторах брала среднее значение времени. Количество элементов 107. Результаты представлены ниже:

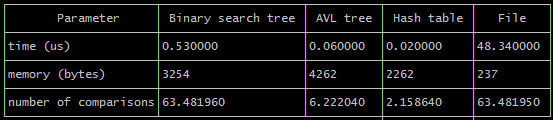


Хэш-таблица оказывается эффективнее по времени, чем остальные структуры (ДДП – на 40%, АВЛ дерево – на 14,3%, файл – в 796,7 раз). АВЛ дерево быстрее файла (в 682,9 раза) и ДДП (на 30%), но медленнее хэш-таблицы. ДДП быстрее файла (в 478 раз), но медленнее остальных структур данных. Файл по времени проигрывает всем.

По памяти наиболее эффективным оказывается файл (файл эффективнее ДДП в 5,8 раз, АВЛ дерева – в 7,4 раза, хэш-таблицы – в 4,2 раза), после него идёт хэш-таблица, которая эффективнее ДДП на 27% и АВЛ дерева на 42,9%, за ней ДДП, которое эффективнее АВЛ дерева на 21,7% и за ним АВЛ дерево.

По количеству сравнений опять же выигрывает хэш-таблица.

Если провести сравнение на правостороннем дереве, то результаты будут немного другими:



Время и количество сравнений в ДДП увеличилось.

**6. Выводы по проделанной работе**

В результате выполненной лабораторной работы я приобрела навыки применения двоичных деревьев, научилась реализовывать основные операции над деревьями, строить и обрабатывать хеш-таблицы, оценила эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска, в хеш-таблицах и в файле.

Можно делать вывод, что хеш-таблицы являются наиболее эффективной структурой данных для поиска необходимого элемента.

1. **Ответы на вопросы**
2. 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?

В памяти деревья можно представить в виде связей с предками (еще их называют родителями); связного списка потомков (сыновей) или структуры данных.

1. Какие стандартные операции возможны над деревьями?

Основные операции с деревьями: обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

1. Что такое дерево двоичного поиска?

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.

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Идеально сбалансированное дерево – это дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу.

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.

Таким образом, в идеально сбалансированном дереве все уровни, возможно за исключением последнего, полностью заполнены. А в АВЛ деревьях критерием сбалансированности является только то, что высота поддеревьев каждого узла должно отличаться не более чем на единицу, но при этом каждый уровень не обязательно должен быть заполнен полностью.

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

Поиск узла в двоичном дереве поиска можно осуществить, двигаясь от корня в левое или правое поддерево в зависимости от значения ключа поддерева. Также осуществляется поиск и в АВЛ дереве, но по сравнению с деревом двоичного поиска он будет быстрее, так как АВЛ дерево сбалансировано.

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

Хеш-функцией – это такая функция, которая ставит в соответствие каждому ключу ki индекс ячейки j, где расположен элемент с этим ключом, таким образом: h(ki) = j, если j=(1,m), где j принадлежит множеству от 1 до m, а m – размерность массива. Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей.

1. 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. В этом случае, очевидно, необходимо найти новое место для хранения ключей, претендующих на одну и ту же ячейку хеш-таблицы.

Существует несколько возможных вариантов разрешения коллизий.

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.

Другой путь решения проблемы, связанной с коллизиями – внутреннее (закрытое) хеширование (открытая адресация). Оно, 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.

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

Поиск в хеш-таблицах становится неэффективен, если при внешнем хешировании одному ключу соответствует большое количество элементов. В этом случае при поиске будет перебираться каждый элемент этого ключа.

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

Поиск эффективнее в хеш-таблицах при условии хорошо подобранной хеш-функции, которая распределяет элементы равномерно и с наименьшим количеством коллизий. Минимальная трудоемкость поиска в хеш-таблице равна О(1). Поиск в АВЛ дереве уступает по эффективности хеш-таблицам, но при этом в силу сбалансированности АВЛ дерева он эффективнее, чем в ДДП, так как высота дерева определяет длину пути поиска в нем.