Отчет по Лабораторной работе №6

Дисциплина «Типы и Структуры Данных»

# Автор: Маргарита Мищенко

**Группа ИУ7-35Б**

**Вариант 16**

Цель работы

построить ДДП, сбалансированное двоичное дерево и хеш-таблицу по указанным данным. Сравнить эффективность поиска в сбалансированном двоичном дереве, в ДДП и в хеш-таблице используя открытую адресацию. Вывести на экран деревья и хеш-таблицу. 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. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий.

Задание

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, то произвести реструктуризацию таблицы, выбрав другую функцию.

Входные данные

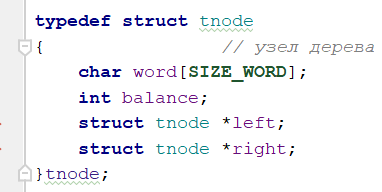
Текст в файле на английском языке

Выходные данные

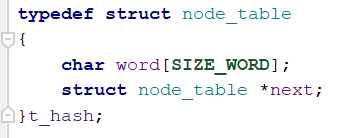
ДДП, АВЛ, хеш-таблица, оценка эффективности поиска элемента (для деревьев, хеш-таблицы и файла), среднее количество сравнений.

Внутренняя структура данных

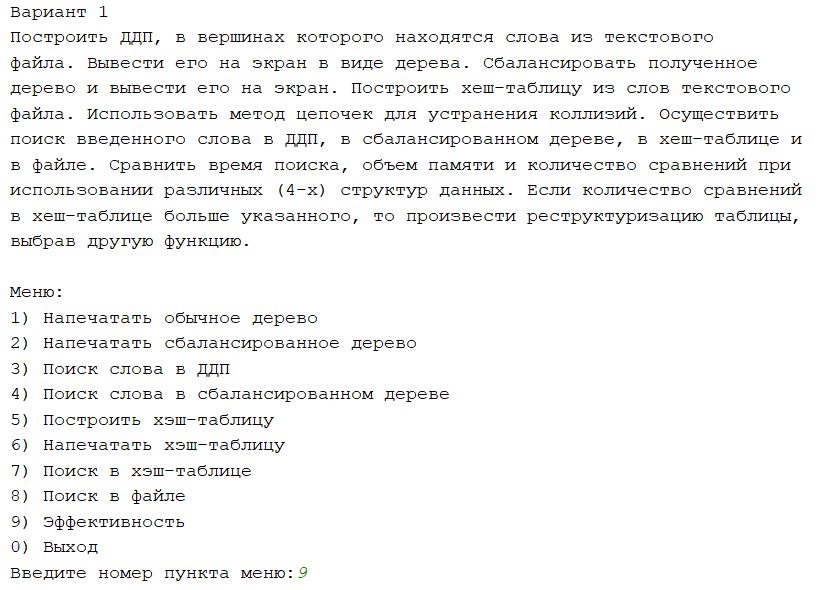
Дерево:



Хеш-таблица:



Меню

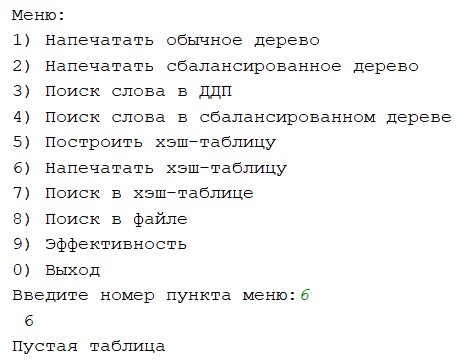


Аварийные ситуации

Аварийные ситуации в данной лабораторной связаны с неверным вводом пункта меню.

Также пользователь должен сначала заполнить хэш-таблицу, и только затем осуществлять в ней поиск, либо выводить ее.

Попытка вывести таблицу, не задав ее:



Функции для работы с ДДП

tnode \*create\_node(**char** \*val);

Создаем узел с словом в аргументе

**void** treeprint(**struct** tnode \*p);

Печатает дерево

**void** freemem(tnode \*tree);

Освобождает память

**void** count\_node(tnode \*tree, **int** \*sum);

Считает узлы

tnode \*ord\_add\_node(tnode \*tree, tnode \*node);

Добавляет узел в обычное дерево

Функции для работы с АВЛ

tnode\* rotateleft(tnode\* q);  
tnode\* rotateright(tnode\* p);

Поворачиваем дерево вправо и влево

**void** fixheight(tnode\* p);

Корректируем глубину

tnode\* bal\_add\_node(tnode \*root, tnode \*node);

Добавляем узел в сбалансированное дерево

tnode\* read\_file\_balanced(tnode \*root, FILE \*f);

Строим сбалансированное дерево из файла

Функции для работы с таблицей

**int** len\_list(t\_hash \*head);

Длина таблицы

t\_hash\* create\_hash\_node(**char** \*word);

Создаем узел хэш таблицы

**int** what\_index(**char** \*word, **int** len\_table);

Считает индекс строки

t\_hash \*\*insert\_in\_hash\_table(t\_hash \*\*table, **int** \*len\_table, t\_hash \*node, **int** max\_len);

Добавляет узел в строку

**void** print\_table(t\_hash \*\*table, **int** len\_table);

Печатает таблицу

**int** count\_node\_table(t\_hash \*\*table, **int** len\_table);

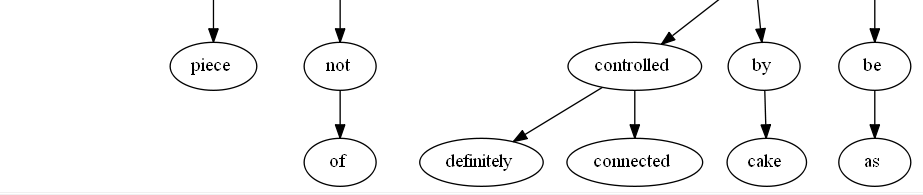
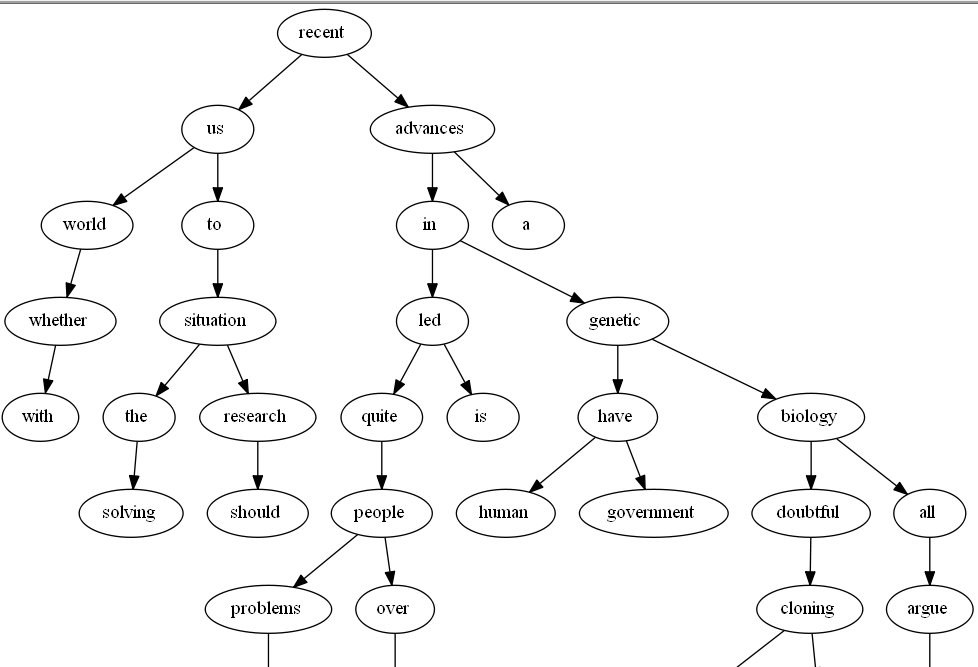
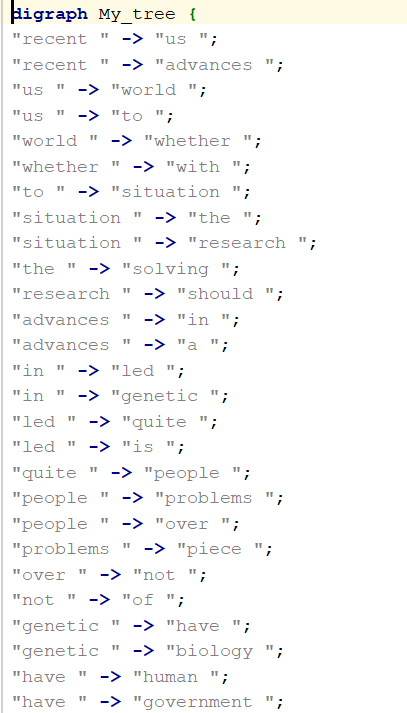
Считает узлы в таблице

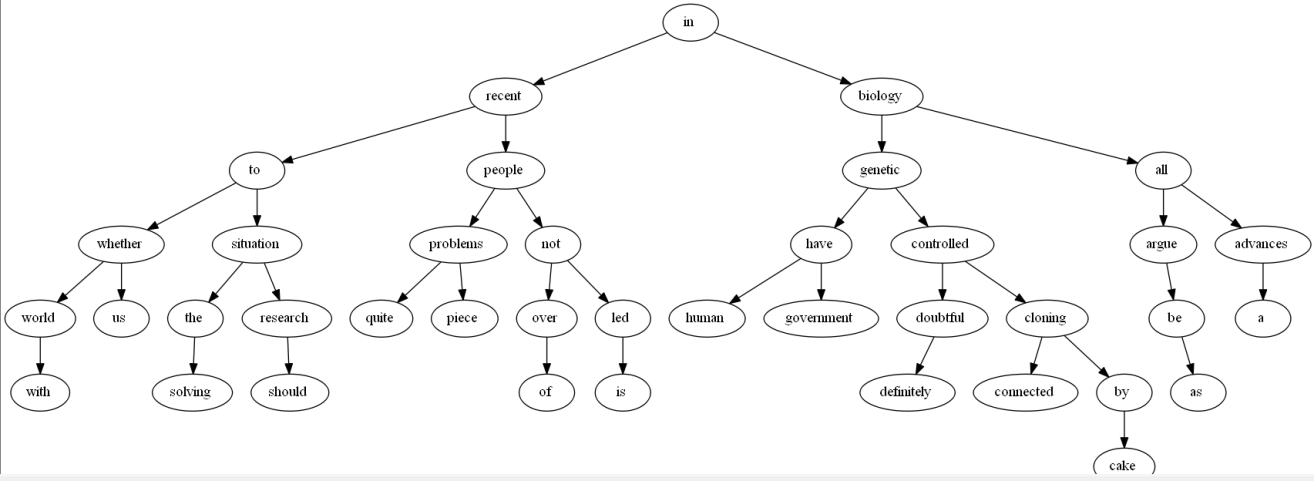
t\_hash \*\*create\_table(FILE \*f, **int** \*len\_table);

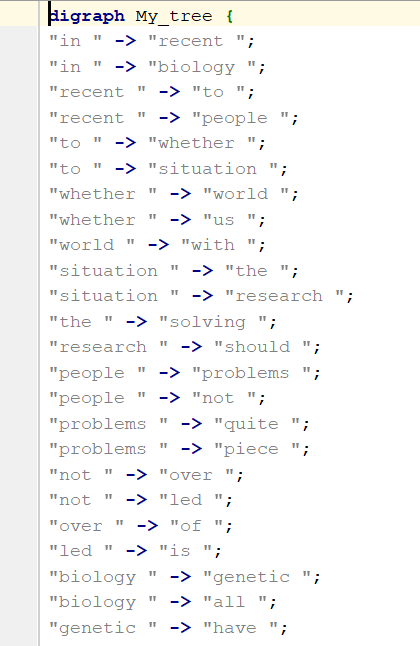
Читает таблицу из файла

Тесты

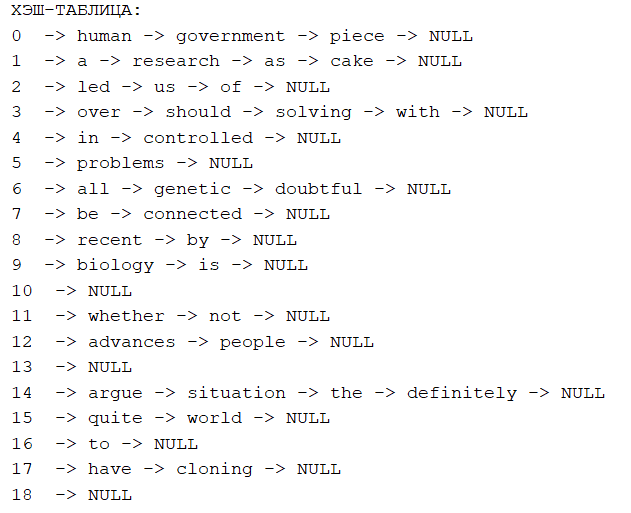
1. ДДП – вывод



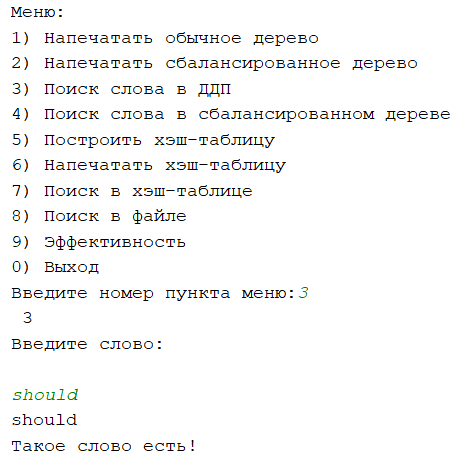
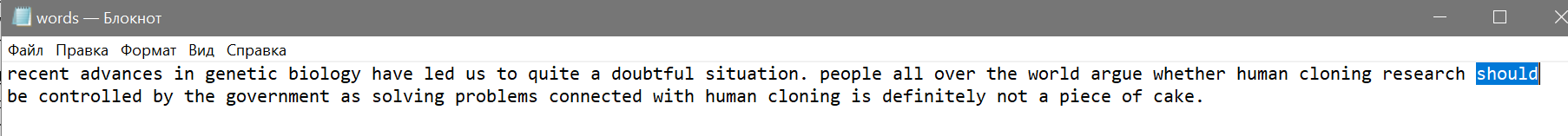
1. AVL



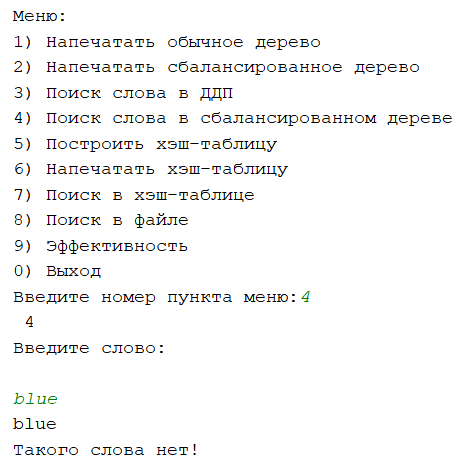
1. ХЭШ-таблица



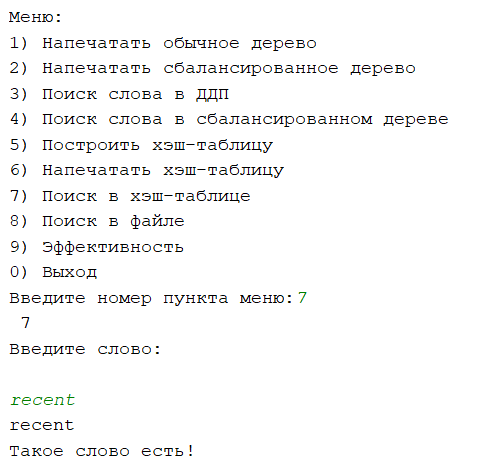
1. Поиск слова в ДДП

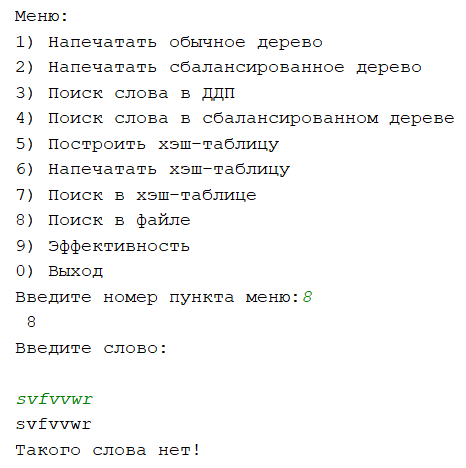
1. Поиск слова в АВЛ



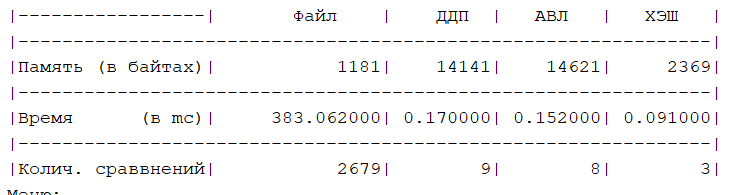
1. Поиск в таблице



1. Поиск в файле



1. Эффективность



Оценка эффективности

В каждой из четырех структур данных был произведен тысячу раз поиск несуществующего в ней слова, таким образом, был произведен самый длинный проход по каждой структуре. Для получения наиболее точных значений поиск производился 1000 раз.

В результате ХЭШ-таблица оказалась наиболее эффективной по всем параметрам (не считая эффективности по обьему по отношению к поиску в файле). Это обьясняется тем, что ХЭШ-таблица хранит только по одному указателю на каждое слово, ДДП хранит два указателя (на правого и левого потомка), АВЛ также хранит флаг баланса данного узла – дополнительно занятая память.

По памяти ДДП на 3% эффективнее АВЛ, а вот ХЭШ эффективнее на 84% ДДП.

По времени ХЭШ эффективнее АВЛ на 41%, ДДП на 47%, поиск в файле же примерно в 4000 раз медленнее (при оценке эффектиности на него тратится больше всего времени).

Среднее количество сравнений у файла равно количеству слов в нем, балансировкой мы выигрываем одно сравнение. Хэш-таблица имеет нормальный показатель – среднее количество сравнений 3. При таком показателе она считается эффективной.

Вывод

Вывод по эффективности представлен выше.

Для более быстрого поиска в хэш-таблцице должно быть как можно меньше коллизий. Количество коллизий зависит от того, сколько было выделено памяти, следовательно, для большей эффективности по времени придется выделить больше памяти.

Ответы на 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?

Выделение памяти под деревья определяется типом их представления. Это может быть таблица связей с предками или связный список сыновей. Оба представления можно реализовать в виде матрицы или списка. При реализации списком память выделяется динамически, при реализации матрицей статически.

3. Какие стандартные операции возможны над деревьями?

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

Дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу называется идеально сбалансированным. Двоичное дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу называется АВЛ-деревом.

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

Поиск в АВЛ дереве имеет сложность О(log2n), в то время как в обычном ДДП сложность О(n).

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

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш- таблицей.

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

8. Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам соответствует одно значение хеш-функции. Существует несколько возможных вариантов разрешения коллизий: внешнее (открытое) хеширование (метод цепочек) и внутреннее (закрытое) хеширование (открытая адресация).

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

Поиск в хэш-таблице становится неэффективным при большом числе коллизий – сложность поиска возрастает.

При открытом хэшировании 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.

При закрытом хэшировании в этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку, до тех пор, пока не будет найден ключ K или пустая позиция в таблице.

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

В хэш-таблице минимальное время поиска О(1). В АВЛ дереве О(log2n). В дереве двоичного поиска О(h), где h – высота дерева.