Отчёт по лабораторной работе № 7

Студент группы ИУ 7 - 34 Богунов Борис

Тема: Сбалансированные деревья, хеш–таблицы

**Цель работы:** построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем и внутреннем хешировании.

Техническое задание: Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла. Осуществить поиск введенного слова в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных. 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 и при различных методах их разрешения.

Входные данные:

1) При выборе пунктов меню – натуральные числа

2) Файл со словами

Выходные данные:

1) Двоичное дерево поиска

2) Сбалансированное дерево

3) Хеш-таблицы с открытой и закрытой адресацией, заполненные словами, сохраненными в текстовом файле.

Обращение к программе:

Запуск программы (main.exe) из каталога с проектом

Возможные реакции программы при аварийных ситуациях:

1) “Ошибка ввода” : Неверный выбор команды меню

2) “Слово не было найдено в файле”: поиск по слову не дал результатов

3) “Файл с информацией не был найден”: при открытии файла с данными возникла ошибка

4) “Файл с информацией пуст”: информация в файле отсутствует

Используемые структуры:

Узел двоичного дерева

struct tree\_node /// Структура двоичного дерева

{

char key[20]; /// Ключ

struct tree\_node \*left; /// Указатель на левого потомка

struct tree\_node \*right; /// Указатель на правого потомка

};

Узел сбалансированного дерева

struct tree\_balanced\_node

{

char key[20];

unsigned char height; /// Высота

struct tree\_balanced\_node\* left;

struct tree\_balanced\_node\* right;

};

Строка хеш-таблицы с внешним хешированием

typedef struct ht\_row\_open

{

char word[20]; /// Слово

struct ht\_row\_open\* next; /// Указатель на следующий элемент

} ht\_row\_open;

Строка хеш-таблицы с внутренним хешированием

typedef struct ht\_row\_close

{

char word[20];

} ht\_row\_close;

Алгоритмы:

**Поиск узла в двоичном дереве**

struct tree\_node\* search\_node(struct tree\_node \*tree, const char \*x)

Поиск происходит рекурсивно. В процессе поиска мы двигаемся от корня, сравнивая ключ с тем, что находится в вершине, в случае совпадения – поиск прекращается, иначе продолжается по правому или левому ребру

**Поиск узла в сбалансированном дереве**

struct tree\_balanced\_node\* search\_node\_b(struct tree\_balanced\_node \*tree, const char \*x);

Логично алгоритму поиска в обычном дереве.

**Поиск узла в хеш-таблице с открытым хешированием**

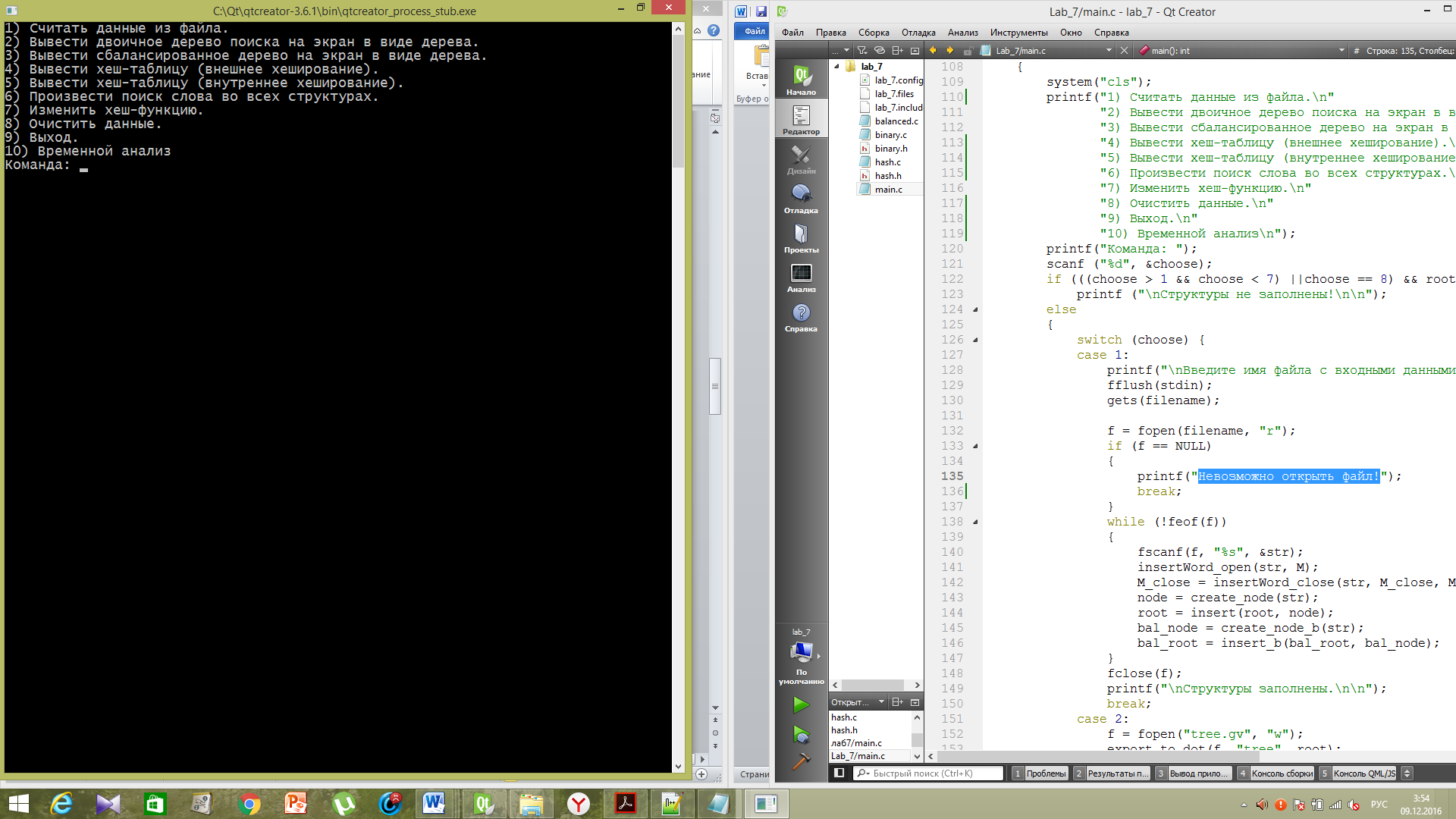
ht\_row\_open \*findWord\_open(char \*str, int M)

Хеш-функцией вычисляется индекс строки, в котором должно находиться искомое слово. В этой строке последовательно просматриваются все элементы связного списка, пока не будет найдено искомое слово или список не закончится. При этом так же ведется подсчет количества сравнений.

**Поиск узла в хеш-таблице с закрытым хешированием**

int findWord\_close(char \*str, int M\_close, int M)

Хеш-функцией вычисляется индекс строки, в которой должно находиться искомое слово. Если в этой строке слово не найдено, просматриваются все строки до конца таблицы, пока не будет найдено искомое слово или таблица не закончится. При этом так же ведется подсчет количества сравнений.



Набор тестов:

|  |  |
| --- | --- |
| Команда | Результат |
| Отсутствие файла или его неверный ввод | "Невозможно открыть файл!" |
| Считывание данных  “1” |  |
| Файл: Input.txt  Команды 1 и 2  Вывод двоичного дерева |  |
| Файл: Input.txt  Команды 1 и 3  Вывод сбалансированного дерева |  |
| Файл: Input.txt  Команды 1 и 4  Вывод хеш-таблицы (внешнее хеширование) |  |
| Файл: Input.txt  Команды 1 и 5  Вывод хеш-таблицы (внутреннее хеширование) |  |
| Файл: Input.txt  Команды 1 и 6  Поиск |  |
| Файл: Input.txt  Команды 1 и 7  Вывод сбалансированного дерева |  |
| Файл: Input.txt  Команды 1 и 8  Очистка данных |  |
| Файл: Input.txt  Команды 1 и 10  Временной анализ |  |

Вывод и сравнения:

Хеш-функция: h(i) = sum(i) mod m, где sum(i) – сумма кодов символов i-го слова, число m выбирается простым и пропорциональным числу слов.

Поиск осуществлялся в деревьях и хеш-таблицах из 1000 слов, m = 881

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время поиска (мкс) | Число сравнений | Объем памяти |
| Двоичное дерево | 667 | 30 | 28000 |
| Сбаланс. дерево | 150 | 10 | 32000 |
| Хеш-таблица внешнее хеширование | 420 | 3 | 24000 |
| Хеш-таблица  внутреннее хеширование | 445 | 170 | 20000 |

Поиск осуществлялся в деревьях и хеш-таблицах из 100 слов, m = 71

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время поиска (мкс) | Число сравнений | Объем памяти |
| Двоичное дерево | 300 | 14 | 28000 |
| Сбаланс. дерево | 50 | 6 | 32000 |
| Хеш-таблица внешнее хеширование | 400 | 1 | 24000 |
| Хеш-таблица  внутреннее хеширование | 378 | 2 | 20000 |

Поиск осуществлялся в деревьях и хеш-таблицах из 10 слов, m = 7

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время поиска (мкс) | Число сравнений | Объем памяти |
| Двоичное дерево | 220 | 10 | 28000 |
| Сбаланс. дерево | 40 | 3 | 32000 |
| Хеш-таблица внешнее хеширование | 400 | 1 | 24000 |
| Хеш-таблица  внутреннее хеширование | 425 | 1 | 20000 |

Из полученных результатов можно сделать вывод, что наименее эффективной по времени оказывается структура двоичного дерева. Наиболее эффективной оказывается структура сбалансированного дерева. Проигрыш хеш-таблиц сбалансированному дереву можно объяснить дополнительными затратами на вычисление хеш-функции (хеш-функция вычисляет сумму кодов символов и возвращает остаток от деления этой суммы на некоторое число). Также можно видеть, что число сравнений при поиске в хеш-таблице с внешней адресацией меняется в зависимости от числа слов в таблице. Это объясняется тем, что при большом объеме данных коллизии в одной ячейке приводят к коллизиям в других.

Проанализируем число сравнений при поиске в хеш-таблицах из 100 слов при различных хеш-функциях.

|  |  |  |
| --- | --- | --- |
| m | Внешнее хеширование | Внутреннее хеширование |
| 5 | 2 | 91 |
| 15 | 1.6 | 79,9 |
| 60 | 1.5 | 50 |

Вывод: реализация хеш-таблицы с внешним хешированием является более стабильной относительно реализации с внутренним хешированием. Хеш-функция при внутреннем хешировании дает лишь «начальное приближение» для поиска элемента, поэтому при возникновении коллизий мы вынуждены просматривать таблицу до конца, что приводит к увеличению числа сравнений. Поэтому при внутреннем хешировании грамотный выбор хеш-функции особенно важен.

Таким образом, выбор структуры напрямую зависит от вида и объема обрабатываемых данных. Каждая из структур имеет свои недостатки. Двоичное дерево поиска может выродиться в линейный односвязный список, если данные изначально упорядочены. При использовании сбалансированного дерева тратится значительное время на балансировку узлов. При использовании хеш-таблиц возникают проблемы правильного выбора хеш-функции и разрешения коллизий.

Ответы на контрольные вопросы:

1) Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Если при добавлении узлов в дерево располагать их равномерно 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. Идеальная балансировка дает наименьшую высоту дерева.В АВЛ дереве высота каждого узла двух поддеревьев отличается не более чем на единицу.

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

Поиск в АВЛ дереве имеет трудоёмкость О(log2n), в то время как в обычном ДДП может иметь O(n). АВЛ дерево никогда не вырождается в линейный список (исключение – дерево из двух элементов), в то время как «внешний вид» ДДП может зависеть от того, в каком порядке в него добавлялись элементы.

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

Хеш-таблицей называется массив, заполненный элементами в порядке, определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс. ХФ должна быть простой для вычисления, распределять клюи в таблице равномерно и давать минимум коллизий.

4) Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие одно и то же значение. Основные методы устранения коллизий: внешнее и внутреннее хеширование.

При внешнем хешировании, конфликтующие ключи просто добавляются в список, находящийся по их общему индексу. Поиск по ключу сводится к определению индекса, а затем к поиску ключа в списке перебором.

При внутреннем хешировании, конфликтующий ключ добавляется в первую свободную ячейку после «своего» индекса. Поиск по ключу сводится к определению начального приближения, а затем к поиску ключа методом перебора.

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

Поиск в хеш-таблицах становится неэффективен при большом числе коллизий – сложность поиска возрастает по сравнению с О(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

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

В хеш-таблице минимальная трудоёмкость поиска равна О(1). В АВЛ: О(log2n). В дереве двоичного поиска может достигать О(n), если дерево вырождается в линейный список.