Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: Информатика и системы управления
КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии
ДИСЦИПЛИНА: Типы и структуры данных
ТЕМА: Обработка деревьев и хеш-таблиц
ВАРИАНТ: 3

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Студент: _____ *Княжев А. В.*
(подпись, дата)

Преподаватель: _____ *Силантьева А. В.*
(подпись, дата)

2021 г.

Содержание

1 Условие 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. 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.

2 Техническое задание

Построить дерево в соответствии с заданным вариантом задания. Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты дерева и степени его ветвления. Построить хеш-таблицу по указанным данным. Вывести на экран деревья и хеш-таблицу. Сравнить эффективность поиска в двоичном дереве поиска, в сбалансированном дереве поиска и в хеш-таблице. Вывести на экран измененные структуры. 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.

Программа должна выводить меню с возможностью выбирать варианты. При вводе нуля на запрос варианта меню, происходит выход из программы.

2.1 Общие входные данные

* имя файла со словами;
* емкость хеш-таблицы;
* номер выбранного пункта меню.

2.2 Входные и выходные данные пунктов меню

2.2.1 Вывод ДДП

Выходные данные

* текстовый файл с описанием дерева;
* графическое изображение дерева.

2.2.2 Поиск элемента в ДДП

Входные данные

* строка — элемент, который ищется в ДДП.

Выходные данные

* сообщение о найденном элементе;
* время выполнения операции.

2.2.3 Удаление элемента из ДДП

Входные данные

* строка — удаляемый элемент.

Выходные данные

* сообщение об успешном удалении;
* время выполнения операции.

2.2.4 Вывод АВЛ-дерева

Выходные данные

* текстовый файл с описанием дерева;
* графическое изображение дерева.

2.2.5 Поиск элемента в АВЛ-дереве

Входные данные

* строка — элемент, который ищется в АВЛ-дереве.

Выходные данные

* сообщение о найденном элементе;
* время выполнения операции.

2.2.6 Удаление элемента из АВЛ-дерева

Входные данные

* строка — удаляемый элемент.

Выходные данные

* сообщение об успешном удалении;
* время выполнения операции.

2.2.7 Вывод хеш-таблицы

Входные данные

Выходные данные

* элементы хеш-таблицы в формате `<index> <key>`.

2.2.8 Поиск элемента в хеш-таблице

* строка — элемент, который ищется в хеш-таблице.

Выходные данные

* сообщение о найденном элементе;
* время выполнения операции.

2.2.9 Удаление элемента из хеш-таблицы

Входные данные

* строка — удаляемый элемент.

Выходные данные

* сообщение об успешном удалении;
* время выполнения операции.

2.2.10 Сравнение скорости

Входные данные

* количество элементов в сравниваемых структурах;
* емкость хеш-таблицы.

Выходные данные

* скорость поиска и удаления для:
    1. ДДП;
    2. АВЛ-дерева;
    3. хеш-таблицы;
    4. текстового файла.

2.2.11 Сравнение потребления памяти

Входные данные

* количество элементов в сравниваемых структурах;
* емкость хеш-таблицы.

Выходные данные

* объем памяти, занимаемый:
    1. ДДП;
    2. АВЛ-деревом;
    3. хеш-таблицей.

2.3 Действие программы

Программа осуществляет работу с двоичными деревьями поиска, сбалансированными деревьями и хеш-таблицами.

2.4 Обращение к программе

Программа может быть запущена из командных оболочек `sh/bash/zsh/fish`, а также от IDE, способных работать с языком Си. Программа не принимает никаких аргументов. Название исполняемого файла — app.exe, так что команда может быть вызвана из командной оболочки в корневой папке проекта как:

```
$ ./app.exe
```

2.5 Аварийные ситуации

2.5.1 Общие аварийные ситуации

1. файл со словами не существует;
2. файл со словами пуст;
3. строка, содержащая имя файла слишком длинная;
4. невалидая емкость хеш-таблицы;
5. некорректная емкость хеш-таблицы;
6. невалидный номер пункта меню (строка или пустая строка);
7. неверный номер пункта меню (такого пунта меню не существует).

2.5.2 Пункт меню №1

1. дерево пусто.

2.5.3 Пункт меню №2

1. введенная строка поиска слишком длинная.

2.5.4 Пункт меню №3

1. введенная удаляемая строка слишком длинная.

2.5.5 Пункт меню №4

1. дерево пусто.

2.5.6 Пункт меню №5

1. введенная строка поиска слишком длинная.

2.5.7 Пункт меню №6

1. введенная удаляемая строка слишком длинная.

2.5.8 Пункт меню №7

1. хеш-таблица пуста.

2.5.9 Пункт меню №8

1. введенная строка поиска слишком длинная.

2.5.10 Пункт меню №9

1. введенная удаляемая строка слишком длинная.

2.5.11 Пункт меню №10

1. невалидное количество элементов;
2. некорректное количество элементов;
3. невалидная емкость хеш-таблицы;
4. некорректная емкость хеш-таблицы.

2.5.12 Пункт меню №11

1. невалидное количество элементов;
2. некорректное количество элементов;
3. невалидная емкость хеш-таблицы;
4. некорректная емкость хеш-таблицы.

3 Структуры данных

В данной работе используется динамические массивы, записи, односвязные списки, деревья.

3.1 Модуль для работы двоичным деревом поиска

3.1.1 Типы данных

```
/**
 * Двоичное дерево поиска (ДДП).
 * key - ключ (слово);
 * left - указатель на левого потомка;
 * right - указатель на правого потомка.
 */
typedef struct bst_t bst_t;
struct bst_t
{
    mystring_t key;
    bst_t *left;
    bst_t *right;
};
```

3.1.2 Функции

```
/**
 * Вставка элемента в ДДП.
 * root - корень исходного дерева;
 * key - вставляемый ключ (слово).
 */
error_t bst_insert(bst_t **root, mystring_t key);
```

```
/**
 * Освобождение ДДП.
 * root - корень исходного дерева.
 */
void bst_free(bst_t **root);
```

```c
/**
 * Вывод ДДП на экран в виде изображения.
 * root - корень исходного дерева.
 */
error_t bst_show(bst_t **root);
```

```c
/**
 * Удаление элемента из ДДП.
 * root - корень исходного дерева;
 * key - удаляемый ключ (слово).
 */
error_t bst_remove(bst_t **root, mystring_t key);
```

```c
/**
 * Чтение ДДП из файла.
 * root - корень исходного дерева;
 * f - файл.
 */
error_t bst_from_file(bst_t **root, FILE *f);
```

```c
/**
 * Поиск элемента в ДДП.
 * root - корень исходного дерева;
 * key - искомый ключ (слово);
 * result - найденный узел.
 */
error_t bst_search(bst_t **root, mystring_t key, bst_t **result);
```

3.2 Модуль для работы с АВЛ-деревом

3.2.1 Типы данных

```
/**
 * АВЛ-дерево.
 * key - ключ (слово);
 * height - высота текущего узла;
 * left - указатель на левого потомка;
 * right - указатель на правого потомка.
 */
typedef struct avl_t avl_t;
struct avl_t
{
    mystring_t key;
    int height;
    avl_t *left;
    avl_t *right;
};
```

3.2.2 Функции

```
/**
 * Вставка элемента в АВЛ-дерево.
 * root - корень исходного дерева;
 * key - вставляемый ключ (слово).
 */
error_t avl_insert(avl_t **root, mystring_t key);
```

```
/**
 * Освобождение АВЛ-дерева.
 * root - корень исходного дерева.
 */
void avl_free(avl_t **root);
```

```c
/**
 * Вывод АВЛ-дерева на экран в виде изображения.
 * root - корень исходного дерева.
 */
error_t avl_show(avl_t **root);
```

```c
/**
 * Удаление элемента из АВЛ-дерева.
 * root - корень исходного дерева;
 * key - удаляемый ключ (слово).
 */
error_t avl_remove(avl_t **root, mystring_t key);
```

```c
/**
 * Чтение АВЛ-дерева из файла.
 * root - корень исходного дерева;
 * f - файл.
 */
error_t avl_from_file(avl_t **root, FILE *f);
```

```c
/**
 * Поиск элемента в АВЛ-дереве.
 * root - корень исходного дерева;
 * key - искомый ключ (слово);
 * result - найденный узел.
 */
error_t avl_search(avl_t **root, mystring_t key, avl_t **result);
```

3.3 Модуль для работы с хеш-таблицей

3.3.1 Типы данных

```c
/**
 * Список-элемент хеш-таблицы.
 * key - ключ (слово);
 * next - указатель на следующий элемент списка.
 */
typedef struct ht_element_t ht_element_t;
struct ht_element_t
{
    mystring_t key;
    ht_element_t *next;
};
```

```c
/**
 * Хеш-таблица.
 * table - элементы таблицы (списки);
 * size - емкость хеш-таблицы;
 * hash - функция хеширования.
 */
typedef struct ht_t
{
    ht_element_t **table;
    size_t size;
    size_t (*hash) (mystring_t, size_t);
} ht_t;
```

3.3.2 Функции

```c
/**
 * Создание новой хеш-таблицы.
 * size - емкость хеш-табицы;
 * hash - функция хеширования.
 */
ht_t *ht_new(size_t size, size_t (*hash) (mystring_t, size_t));
```

```
/**
 * Освобождение хеш-таблицы.
 * ht - исходная хеш-таблица.
 */
void ht_free(ht_t **ht);
```

```
/**
 * Вставка слова в хеш-таблицу.
 * ht - исходная хеш-таблица;
 * key - вставляемый ключ (слово).
 */
error_t ht_insert(ht_t **ht, mystring_t key);
```

```
/**
 * Удаление элемента из хеш-таблицы.
 * ht - исходная хеш-таблица;
 * key - удаляемый ключ (слово).
 */
error_t ht_remove(ht_t **ht, mystring_t key);
```

```
/**
 * Поиск элемента в хеш-таблице.
 * ht - исходная хеш-таблица;
 * key - искомый ключ;
 * found - найденный элемент.
 */
error_t ht_search(ht_t **ht, mystring_t key, ht_element_t **found);
```

```
/**
 * Вывод хеш-таблицы.
 * ht - исходная хеш-таблица.
 */
error_t ht_print(ht_t **ht);
```

```c
/**
 * Ввод хеш-таблицы из файла.
 * ht - исходная хеш-таблица;
 * f - файл.
 */
error_t ht_from_file(ht_t **ht, FILE *f);
```

```c
/**
 * Аддитивная хеш-функция.
 * str - преобразуемая строка;
 * size - емкость хеш-таблицы.
 */
size_t ht_additive_method(mystring_t str, size_t size);
```

3.4 Модуль для работы с текстовым файлом

3.4.1 Функции

```c
/**
 * Добавление слова в текстовый файл.
 * filename - имя файла;
 * key - добавляемое слово.
 */
error_t file_insert(char *filename, mystring_t key);
```

```c
/**
 * Удаление слова из текстового файла.
 * filename - имя файла;
 * key - удаляемое слово.
 */
error_t file_remove(char *filename, mystring_t key);
```

```c
/**
 * Поиск слова в текстовом файле.
 * filename - имя файла;
 * key - искомое слово.
 */
error_t file_search(char *filename, mystring_t key);
```

3.5 Модуль для работы с ошибками

3.5.1 Типы данных

```
/**
 * text - текст ошибки;
 * code - код ошибки;
 * func - функция, в которой случилась ошибка.
 */
typedef struct
{
    const char *text;
    int code;
    const char *func;
} error_t;
```

3.5.2 Функции

```
/**
 * Создание новой ошибки.
 * text - текст ошибки;
 * code - код ошибки;
 * func - функция, в которой случилась ошибка.
 */
error_t new_error(const char *text, int code, const char *func);
```

```
/**
 * Создание ошибки-маркера успеха.
 * func - функция, в которой был создан "успех".
 */
error_t new_success(const char *func);
```

```
/**
 * Проверка, отображает ли ошибка ошибочную ситуацию.
 * err - исходная ошибка.
 */
bool is_failure(error_t err);
```

3.6 Модуль для работы со строками

3.6.1 Основные константы

```
#define MYSTRING_SIZE 257
```

3.6.2 Типы данных

```
/**
 * Тип строки длиной 256 символов + терминальный нуль.
 */
typedef char mystring_t[MYSTRING_SIZE];
```

3.6.3 Функции

```
/**
 * Чтение строки из файла.
 * f - файл;
 * str - строка, в которую считываем.
 */
error_t ms_read_line(FILE *f, mystring_t str);
```

```
/**
 * Чтение целого числа из файла.
 * f - файл;
 * n - считываемое число.
 */
error_t ms_read_int_line(FILE *f, int *n);
```

```
/**
 * Чтение слова из файла.
 * f - файл;
 * str - считываемое слово.
 */
error_t ms_read_word(FILE *f, mystring_t str);
```

3.7 Модуль «ядра» программы

3.7.1 Функции

```
/**
 * Запуск основного диалога программы.
 */
error_t eng_work();
```

3.8 Модуль для тестирования скорости работы

3.8.1 Основные константы

```
/**
 * Количество запусков тестов производительности.
 */
#define BENCH_NUM_OF_RUNS 10
```

3.8.2 Функции

```
/**
 * Возвращает количество тактов процессора.
 */
int64_t bench_tick(void);
```

```
/**
 * Измерение скорости удаления из ДДП.
 * n - количество удаляемых элементов;
 * timer - среднее время удаления.
 */
error_t bench_bst_remove(size_t n, long long *timer);
```

```
/**
 * Измерение скорости поиска в ДДП.
 * n - количество элементов;
 * timer - среднее поиска.
 */
error_t bench_bst_search(size_t n, long long *timer);
```

```c
/**
 * Получение объема памяти, занимаемого ДДП.
 * n - количество элементов;
 * size - объем памяти.
 */
error_t bench_bst_size(size_t n, size_t *size);
```

```c
/**
 * Измерение скорости удаления из АВЛ-дерева.
 * n - количество удаляемых элементов;
 * timer - среднее время удаления.
 */
error_t bench_avl_remove(size_t n, long long *timer);
```

```c
/**
 * Измерение скорости поиска в АВЛ-дереве.
 * n - количество элементов;
 * timer - среднее поиска.
 */
error_t bench_avl_search(size_t n, long long *timer);
```

```c
/**
 * Получение объема памяти, занимаемого АВЛ-деревом.
 * n - количество элементов;
 * size - объем памяти.
 */
error_t bench_avl_size(size_t n, size_t *size);
```

```c
/**
 * Измерение скорости удаления из хеш-таблицы.
 * n - количество удаляемых элементов;
 * cap - емкость хеш-таблицы;
 * timer - среднее время удаления.
 */
error_t bench_ht_remove(size_t n, size_t cap, long long *timer);
```

```c
/**
 * Измерение скорости поиска в хеш-таблице.
 * n - количество элементов;
 * timer - среднее поиска.
 */
error_t bench_ht_search(size_t n, size_t cap, long long *timer);
```

```c
/**
 * Получение объема памяти, занимаемого хеш-таблицей.
 * n - количество элементов;
 * size - объем памяти.
 */
error_t bench_ht_size(size_t n, size_t cap, size_t *size);
```

```c
/**
 * Измерение скорости удаления из текстового файла.
 * n - количество удаляемых элементов;
 * timer - среднее время удаления.
 */
error_t bench_file_remove(size_t n, long long *timer);
```

```c
/**
 * Измерение скорости поиска в текстовом файле.
 * n - количество элементов;
 * timer - среднее поиска.
 */
error_t bench_file_search(size_t n, long long *timer);
```

4 Описания алгоритмов

1. получение у пользователя имени файла со словами;
2. получение у пользователя емкости хеш-таблицы;
3. открытие файла со словами, заполнение словами оттуда двоичного дерева поиска, АВЛ-дерева и хеш-таблицы;
4. вывод главного меню;
5. получение у пользователя номера пункта меню:

4.1 Пункт меню №1

1. создание текстового файла;
2. запись информации о связях в исходном ДДП в специальном формате *dot*;
3. запуск утилиты `dot` и преобразование файла в изображение;
4. вывод изображения с деревом на экран.

4.2 Пункт меню №2

1. получение от пользователя вставляемого слова-ключа;
2. если ключ равен ключу текущего элемента, возврат с ошибкой;
3. если ключ меньше ключа текущего элемента, повторять с (2) для левого поддерева;
4. если ключ больше ключа текущего элемента, повторять операции для правого поддерева;
5. если текущий элемент нулевой, добавляем ключ на текущую позицию.

4.3 Пункт меню №3

1. получение от пользователя удаляемого слова-ключа;
   item если ключ меньше ключа текущего элемента, повторять с (2) для левого поддерева;
2. если ключ больше ключа текущего элемента, повторять операции для правого поддерева;
3. если текущий элемент нулевой, возврат с ошибкой, так как ключа нет в дереве;

4. если ключ равен ключу текущего элемента, удаляем элемент:
   (a) если у элемента нет потомков, то он просто удаляется;
   (b) если у элемента только один потомок, то он замещается этим потомком;
   (c) иначе поиск самого левого узла правого поддерева;
   (d) вставка левого поддерева удаляемого элемента слева от найденного самого левого узла;
   (e) замещение удаляемого узла правым поддеревом.

4.4 Пункт меню №4

1. создание текстового файла;
2. запись информации о связях в исходном АВЛ-дереве в специальном формате *dot*;
3. запуск утилиты `dot` и преобразование файла в изображение;
4. вывод изображения с деревом на экран.

4.5 Пункт меню №5

1. получение от пользователя вставляемого слова-ключа;
2. если ключ равен ключу текущего элемента, возврат с ошибкой;
3. если ключ меньше ключа текущего элемента, повторять с (2) для левого поддерева;
4. если ключ больше ключа текущего элемента, повторять операции для правого поддерева;
5. после попытки вставки на левую или правую позицию балансировка узла:
   (a) заполнение высот для каждого узла;
   (b) если высота правого поддерева больше высоты левого на 2:
      i. если высота левого поддерева текущего правого поддерева больше высоты правого, то поворот вправо относительно правого узла;
      ii. поворот влево относительно текущего узла.
   (c) если высота левого поддерева больше высоты правого на 2:
      i. если высота правого поддерева текущего левого поддерева больше высоты левого, то поворот влево относительно левого узла;
      ii. поворот вправо относительно текущего узла.
6. если текущий элемент нулевой, добавляем ключ на текущую позицию.

4.6 Пункт меню №6

1. получение от пользователя удаляемого слова-ключа;
2. если ключ меньше ключа текущего элемента, повторять с (2) для левого поддерева;
3. если ключ больше ключа текущего элемента, повторять операции для правого поддерева;
4. если текущий элемент нулевой, возврат с ошибкой, так как ключа нет в дереве;
5. если ключ равен ключу текущего элемента, удаляем элемент:
   (a) если у элемента нет потомков, то он просто удаляется;
   (b) если у элемента только один потомок, то он замещается этим потомком;
   (c) иначе поиск самого левого узла правого поддерева;
   (d) вставка левого поддерева удаляемого элемента слева от найденного самого левого узла;
   (e) замещение удаляемого узла правым поддеревом.
6. балансировка нового замещенного узла.

4.7 Пункт меню №7

1. проход по каждому элемента массива-основы хеш-таблицы;
2. если список на данной позиции не пуст, то вывод элементов списка в формате `<index> <key>`.

4.8 Пункт меню №8

1. получение от пользователя вставляемого слова-ключа;
2. расчет хеш-функции от данного ключа;
3. проход по списку, соответствующему результату хеш-функции;
4. если элемент уже есть, возврат ошибки;
5. если нет, добавление элемента в конец списка.

4.9 Пункт меню №9

1. получение от пользователя удаляемого слова-ключа;
2. расчет хеш-функции от данного ключа;
3. проход по списку, соответствующему результату хеш-функции;
4. если элемент не найден, возврат ошибки;
5. если найден, то удаление его из списка.

4.10 Пункт меню №10

1. получение от пользователя количества слов;
2. получение от пользователя емкости хеш-таблицы;
3. измерение времени удаления и поиска для заданного количества элементов на ДДП, АВЛ-дереве, хеш-таблице и текстовом файле;
4. вывод результатов измерений на экран.

4.11 Пункт меню №11

1. получение от пользователя количества слов;
2. получение от пользователя емкости хеш-таблицы;
3. расчет объема памяти, занимаемого на данном количестве элементов ДДП, АВЛ-деревом и хеш-таблицей;
4. вывод результатов расчетов на экран.

4.12 Правый поворот относительно узла

1. установка левого узла в качестве нового «корня»;
2. установка правого поддерева старого «корня» левым поддеревом нового;
3. установка правого поддерева нового «корня» старым корнем.

4.13 Левый поворот относительно узла

1. установка правого узла в качестве нового «корня»;
2. установка левого поддерева старого «корня» правым поддеревом нового;
3. установка левого поддерева нового «корня» старым корнем.

5 Тестирование

Для проверок корректности работы программы было проведено функциональное тестирование. Таблица с тестовыми данными для «позитивных» и «негативных» случаев приведена ниже.

Так как входных/выходных данных может быть очень много, то в соответствующих полях могут быть указаны наиболее значимые части.

5.1 «Негативные» тесты

5.1.1 Ввод слишком длинного имени файла

Входные данные: *Слишком длинная строка...*
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.2 Файл не существует

Входные данные: fivuvnfivnidfnvidf $\rightarrow$ 4
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 164: ошибка открытия файла.

5.1.3 Файл пуст

Входные данные: empty.txt $\rightarrow$ 4
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 405: ошибка ввода дерева.

5.1.4 Невалидный пункт меню

Входные данные: ауоауолм
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 104: строка не является корректным числом.

5.1.5 Неверный пункт меню

Входные данные: 122
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 161: введен неверный пункт меню.

5.1.6 Вывод пустого ДДП

Входные данные: 1
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 403: дерево пусто.

5.1.7 Слишком длинная искомая строка при поиске в ДДП

Входные данные: 2 → *Слишком длинная строка...*
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.8 Слишком длинная удаляемая строка при удалении из ДДП

Входные данные: 3 → *Слишком длинная строка...*
Слова в ДДП: hello, world
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.9 Вывод пустого АВЛ-дерева

Входные данные: 4
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 503: дерево пусто.

5.1.10 Слишком длинная искомая строка при поиске в АВЛ-дереве

Входные данные: 5 → *Слишком длинная строка...*
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.11 Слишком длинная удаляемая строка при удалении из АВЛ-дерева

Входные данные: 6 → *Слишком длинная строка...*
Слова в ДДП: —
Слова в АВЛ-дереве: hello, all, have, nice, day
Слова в хеш-таблице: —
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.12 Вывод пустой хеш-таблицы

Входные данные: 7
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 603: хеш-таблица пуста.

5.1.13 Слишком длинная искомая строка при поиске в хеш-таблице

Входные данные: 8 → *Слишком длинная строка...*
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.14 Слишком длинная удаляемая строка при удалении из хеш-таблицы

Входные данные: 9 → *Слишком длинная строка...*
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: hello, all, have, nice, day
Результат: Ошибка 101: введенная строка слишком длинная.

5.1.15 Невалидное количество элементов в замерах скорости

Входные данные: 10 → abc

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 104: строка не является корректным числом.

5.1.16 Отрицательное количество элементов в замерах скорости

Входные данные: 10 → -10

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 161: размер не может быть меньше нуля.

5.1.17 Невалидная емкость хеш-таблицы в замерах скорости

Входные данные: 10 → 100 → abc

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 104: строка не является корректным числом.

5.1.18 Отрицательная емкость хеш-таблицы в замерах скорости

Входные данные: 10 → 100 → -10

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 161: размер хеш-табицы не может быть меньше нуля.

5.1.19 Невалидное количество элементов в замерах памяти

Входные данные: 11 → abc

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 104: строка не является корректным числом.

5.1.20 Отрицательное количество элементов в замерах памяти

Входные данные: 11 → -10

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 161: размер не может быть меньше нуля.

5.1.21 Невалидная емкость хеш-таблицы в замерах памяти

Входные данные: 11 → 100 → abc

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 104: строка не является корректным числом.

5.1.22 Отрицательная емкость хеш-таблицы в замерах памяти

Входные данные: 11 → 100 → -10

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат: Ошибка 161: размер хеш-табицы не может быть меньше нуля.

5.2 «Позитивные» тесты

5.2.1 Вывод ДДП

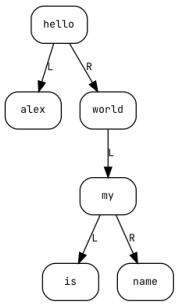
Входные данные: 1

Слова в ДДП: hello, world, my, name, is, alex

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат:



5.2.2 Поиск слова в ДДП: элемент найден

Входные данные: 2 → my
Слова в ДДП: hello, world, my, name, is, alex
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Слово "my"найдено в ДДП.

5.2.3 Поиск слова в ДДП: элемент не найден

Входные данные: 2 → myty
Слова в ДДП: hello, world, my, name, is, alex
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Элемент не найден.

5.2.4 Удаление слова из ДДП: элемент найден

Входные данные: 3 → my
Слова в ДДП: hello, world, my, name, is, alex
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Слово "my"успешно удалено из ДДП.

5.2.5 Удаление слова из ДДП: элемент не найден

Входные данные: 3 → myty
Слова в ДДП: hello, world, my, name, is, alex
Слова в АВЛ-дереве: —
Слова в хеш-таблице: —
Результат: Элемент не найден.

5.2.6 Вывод АВЛ-дерева

Входные данные: 4
Слова в ДДП: —
Слова в АВЛ-дереве: hello, world, my, name, is, alex
Слова в хеш-таблице: —
Результат:

5.2.7 Поиск слова в АВЛ-дереве: элемент найден

Входные данные: 5 → my
Слова в ДДП: —
Слова в АВЛ-дереве: hello, world, my, name, is, alex
Слова в хеш-таблице: —
Результат: Слово "my"найдено в АВЛ-дереве.

5.2.8 Поиск слова в АВЛ-дереве: элемент не найден

Входные данные: 5 → myty
Слова в ДДП: —
Слова в АВЛ-дереве: hello, world, my, name, is, alex
Слова в хеш-таблице: —
Результат: Элемент не найден.

5.2.9 Удаление слова из АВЛ-дерева: элемент найден

Входные данные: 6 → my
Слова в ДДП: —
Слова в АВЛ-дереве: hello, world, my, name, is, alex
Слова в хеш-таблице: —
Результат: Слово "my"успешно удалено из АВЛ-дерева.

5.2.10 Удаление слова из АВЛ-дерева: элемент не найден

Входные данные: 6 → myty
Слова в ДДП: —
Слова в АВЛ-дереве: hello, world, my, name, is, alex
Слова в хеш-таблице: —
Результат: Элемент не найден.

5.2.11 Вывод хеш-таблицы

Входные данные: 7
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: hello, world, my, name, is, alex
Результат:

```
0000: "hello"
0001: EMPTY
0002: EMPTY
0003: "is"
0004: "name"
0005: EMPTY
0006: "world"
0006: "my"
0006: "alex"
```

5.2.12 Поиск слова в хеш-таблице: элемент найден

Входные данные: 8 → my
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: hello, world, my, name, is, alex
Результат: Слово "my"найдено в хеш-таблице.

5.2.13 Поиск слова в хеш-таблице: элемент не найден

Входные данные: 8 → myty
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: hello, world, my, name, is, alex
Результат: Элемент не найден.

5.2.14 Удаление слова из хеш-таблицы: элемент найден

Входные данные: 9 → my
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: hello, world, my, name, is, alex
Результат: Слово "my"успешно удалено из хеш-таблицы.

5.2.15 Удаление слова из хеш-таблицы: элемент не найден

Входные данные: 9 → myty
Слова в ДДП: —
Слова в АВЛ-дереве: —
Слова в хеш-таблице: hello, world, my, name, is, alex
Результат: Элемент не найден.

5.2.16 Измерения скорости

Входные данные: $10 \to 100 \to 150$

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат:

```
ДДП:
  Поиск: 25, количество сравнений: 100
  Удаление: 23, количество сравнений: 100
АВЛ-дерево:
  Поиск: 1, количество сравнений: 7
  Удаление: 1, количество сравнений: 7
Хеш-таблица:
  Поиск: 0, количество сравнений: 1
  Удаление: 1, количество сравнений: 1
Файл:
  Поиск: 302, количество сравнений: 100
  Удаление: 42571, количество сравнений: 100
```

5.2.17 Измерения объема памяти

Входные данные: $11 \to 100 \to 150$

Слова в ДДП: —

Слова в АВЛ-дереве: —

Слова в хеш-таблице: —

Результат:

```
ДДП:
  Размер: 28000 Б
АВЛ-дерево:
  Размер: 28000 Б
Хеш-таблица:
  Размер: 28424 Б
Текстовый файл:
  Размер: 25700 Б
```

6 Оценка эффективности

6.1 Объем занимаемой памяти

Размер	Емкость	ДДП, Б	АВЛ-дерево, Б	Хеш-табл., Б	Файл, Б
20	10	5600	5600	5544	5140
20	20	5600	5600	5624	5140
20	40	5600	5600	5784	5140
200	100	56000	56000	55224	51400
200	200	56000	56000	56024	51400
200	400	56000	56000	57624	51400
1000	500	280000	280000	276024	257000
1000	1000	280000	280000	280024	257000
1000	2000	280000	280000	288024	257000

6.2 Скорость операций

6.2.1 Поиск

Размер	Емкость	ДДП	АВЛ	Хеш-таблица	Файл
20	10	1554	369	324	31656
20	20	1859	353	471	28781
20	40	802	640	332	30627
200	100	15278	379	572	128379
200	200	9105	371	317	111428
200	400	16723	519	199	128862
1000	500	42754	441	281	357513
1000	1000	40975	458	488	317712
1000	2000	41697	766	549	284568

6.2.2 Удаление

Размер	Емкость	ДДП	АВЛ	Хеш-таблица	Файл
20	10	1380	348	454	440309
20	20	1695	580	499	419928
20	40	1855	323	225	383716
200	100	15043	861	323	2671087
200	200	9323	1879	460	2741421
200	400	8301	857	482	2625421
1000	500	41334	672	463	12615204
1000	1000	40827	638	404	12806856
1000	2000	41045	541	374	13031966

6.3 Количество сравнений

6.3.1 Поиск

Размер	Емкость	ДДП	АВЛ	Хеш-таблица	Файл
20	10	20	5	7	20
20	20	20	5	2	20
20	40	20	5	1	20
200	100	200	8	5	200
200	200	200	8	2	200
200	400	200	8	1	200
1000	500	1000	10	4	1000
1000	1000	1000	10	3	1000
1000	2000	1000	10	6	1000

6.3.2 Удаление

Размер	Емкость	ДДП	АВЛ	Хеш-таблица	Файл
20	10	20	5	1	20
20	20	20	5	1	20
20	40	20	5	1	20
200	100	200	8	1	200
200	200	200	8	2	200
200	400	200	8	1	200
1000	500	1000	10	2	1000
1000	1000	1000	10	1	1000
1000	2000	1000	10	1	1000

7 Контрольные вопросы

7.1 Что такое дерево?

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

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

7.2 Как выделяется память под представление деревьев?

Память выделяется динамически под каждый узел дерева.

7.3 Какие стандартные операции возможны над деревьями?

Обход дерева, вставка элемента, удаление элемента.

7.4 Что такое дерево двоичного поиска?

Двоичное дерево — дерево, каждый узел которого имеет не более двух потомков.

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.

7.5 Чем отличается идеально сбалансированное дерево от АВЛ дерева?

У АВЛ-дерева для каждого узла высота двух его поддеревьев различается не более чем на 1, а у идеально сбалансированного дерева различается количество узлов в каждом поддереве не более чем на 1.

7.6 Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?

Поиск в АВЛ-дереве происходит быстрее из-за примерное одинаковой высоты каждой ветви.

7.7 Что такое хеш-таблица, каков принцип ее построения?

Хеш-таблица — массив, заполненный в порядке, определяемым хеш-функцией. Хеш-функция каждому ключу ставит в соответствие некоторый индекс в массиве.

7.8 Что такое коллизии? Каковы методы их устранения?

Коллизия — ситуация, когда разным ключам соответствует одно значение хеш-функции.

Методы устранений коллизий:

7.8.1 Метод цепочек (открытое хеширование)

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.

7.8.2 Открытая адресация (закрытое хеширование)

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. При этом, если индекс следующего просматриваемого элемента определяется добавлением какого-то постоянного шага (от 1 до n), то данный способ разрешения коллизий называется линейной адресацией.

7.9 В каком случае поиск в хеш-таблицах становится неэффективен?

При большом количестве коллизий.

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

В худшем случае, в ДДП сложность поиска составляет $O(n)$, в АВЛ-дереве — $O(\log_2 n)$, в хеш-таблице с маленьким количеством коллизий — $O(1)$. Таким образом, хеш-таблица заметно эффективнее деревьев для поиска, при условии небольшого количества коллизий.

8 Вывод

Как можно убедиться по результатам измерений, хеш-таблица позволяет осуществлять более быстрый поиск и удаление элементов, чем деревья. В то же время, в общем случае она менее эффективна по памяти.

Если сравнивать деревья, то поиск в АВЛ-дереве происходит быстрее. Однако, удаление в нем может быть медленнее, чем в обычном двоичном дереве поиска. Это обусловлено тем, что при удалении АВЛ-дерево нуждается в дополнительной балансировке.

Также операции в АВЛ-дереве могут быть медленнее в том случае, если искомый/удаляемый элемент находятся близко к корню в обычном ДДП, а в АВЛ-дереве он находится дальше.

По памяти АВЛ-дерево менее эффективно обычного ДДП, так как каждый его узел должен содержать информацию о высоте текущего поддерева. Но эта разница в потреблении памяти менее значительна, чем, например, разница в потреблении памяти деревьями и хеш-таблицей.

Относительно операций поиска и удаления наихудший результат показал текстовый файл. Такой результат обусловлен последовательным доступом к файлу, а так же накладными расходами на открытие файла.

Текстовый файл является наиболее эффективным в данном случае по памяти.

Список литературы

[1] Методические рекомендации по лабораторной работе №6(*http://wwwcdl.bmstu.ru/*)