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 №7 ПО ДИСЦИПЛИНЕ ТИПЫ И СТРУКТУРЫ ДАННЫХ

Студент **Простев Тимофей Александрович**

Группа **ИУ7-33Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Простев Т. А.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

Оценка _____

2023 г.

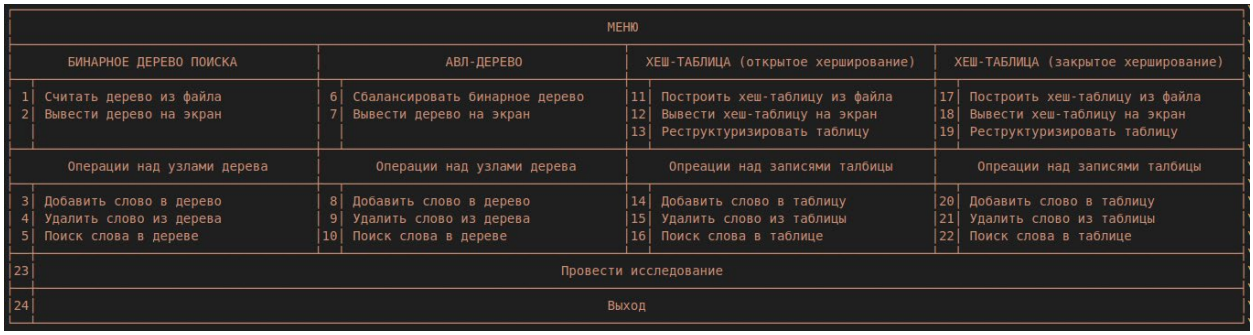
1. Описание условия задачи.

Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана. 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.

2. Техническое задание.

Исходные данные и правила их ввода:

- Номер пункта меню: целое число от 1 до 24:



- Имя файла: строка, не больше 255 символов.
- Слово для добавления, удаления, поиска: строка, не больше 255 символов.

Выходные данные:

- При выводе дерева: png-изображения на экран.
- При выводе таблицы – таблица на экране.
- При поиске слово – сообщение о том, было ли найдено слово.

Способ обращения к программе:

Запуск исполняемого файла. В рабочей директории выполнить команду ./app.exe.

Аварийные ситуации:

- Ввод некорректных данных.
- Ошибка выделения памяти.

В случае аварийной ситуации выводится сообщение об ошибке.

3. Описание внутренних структур данных.

Структура для хранения массива слов:

```
struct words {
   char **words;
   size_t size;
};
```

Структура содержит два поля:

- `words` - массив строк.
- `size` - количество слов в массиве - беззнаковое целое типа `size_t`

Структура для хранения узла дерева:

```
struct tree_node {
    char *data;
    tree_node_t *smaller;
    tree_node_t *bigger;
};
```

Структура содержит 3 поля:

- `data` - указатель на строку.
- `smaller` - указатель на меньший узел.
- `bigger` - указатель на больший узел.

Структура для хранения АВЛ-дерева:

```
struct avl_node {
    avl_tree_type_t data;
    avl_node_t *right;
    avl_node_t *left;
    unsigned int height;
};
```

Структура содержит 4 поля:

- `data` - указатель на строку.
- `smaller` - указатель на меньший узел.
- `bigger` - указатель на больший узел.
- `height` – высота узла – беззнаковое целое число типа `unsigned int`.

Структура для хранения хеш-таблицы с закрытым хешированием:

```
struct closed_hash_table {
   char **words;
   size_t size;
};
```

Структура содержит два поля:

- `words` - массив строк.
- `size` – размер таблицы - беззнаковое целое типа `size_t`

Хеш-таблица с открытым хешированием:

Структура для хранения узла таблицы:

```
struct record {
    char *word;
    record_t *next;
};
```

Структура содержит 2 поля:

- `word` – слово – строка.
- `next` – указатель на следующий узел.

Структура для хранения таблицы:

```
struct open_hash_table {
    record_t **records;
    size_t size;
};
```

Структура содержит 2 поля:

- `records` – массив узлов таблицы.
- `size` – размер таблицы - беззнаковое целое типа `size_t`.

Для работы со структурами используются следующие функции:

Бинарное дерево поиска:

`bs_tree_t *bs_tree_new();` - Создание дерева.
`void bs_tree_delete(bs_tree_t *tree);` - Удаление дерева с освобождением памяти.
`void bs_tree_clear(bs_tree_t *tree)` – Очищение всех узлов дерева.
`void bs_tree_for_each(bs_tree_t *tree, void (*function)(void *data));` - Применить функцию к каждому элементу дереве.
`int bs_tree_insert(bs_tree_t *tree, char *value);` - Вставить слово в дерево.
`int bs_tree_remove(bs_tree_t *tree, char *value);` - Удалить слово из дерева.
`int bs_tree_bs(bs_tree_t *tree, char *value,);` - Присутствует ли слово в дереве.
`void bs_tree_to_dot(FILE *stream, bs_tree_t *tree);` - Перевести дерево в dot формат в файл.

АВЛ-дерево:
`avl_tree_t *avl_tree_new(); - Создание дерева.`
`void avl_tree_delete(avl_tree_t *tree);` - Удаление дерева с освобождением памяти.
`void avl_tree_clear(avl_tree_t *tree);` - Очищение всех узлов дерева.
`int avl_tree_is_empty(avl_tree_t *tree);` - Пустое ли дерево.
`size_t avl_node_size();` - Получить размер узла дерева.
`size_t avl_height(avl_tree_t *tree);` - Получить высоту дерева.
`int avl_tree_insert(avl_tree_t *tree, avl_tree_type_t value, avl_comparator_f comparator);` - Вставить слово в дерево.
`int avl_tree_remove(avl_tree_t *tree, avl_tree_type_t value, avl_comparator_f comparator);` - Удалить слово из деева.
`avl_tree_type_t avl_tree_search(avl_tree_t *tree, avl_tree_type_t value, avl_comparator_f comparator);` - Поиск в дереве.

Хеш-талбицы:
Открытое хеширование:
`open_hash_table_t* open_hash_table_new(size_t size);` - Создание таблицы.
`void open_hash_table_delete(open_hash_table_t *table);` - Удаление таблицы с освобождением памяти.
`size_t open_hash_table_get_size(open_hash_table_t *table);` - Получение размера таблицы.
`size_t open_hash_table_record_size();` - Получение размера узла дерева.
`int open_hash_table_insert(open_hash_table_t *table, char *key, int *comparisons);` - Вставить слово в таблицу.
`char* open_hash_table_remove(open_hash_table_t *table, char *key, int *comparisons);` - Удалить слово из таблицы.
`char* open_hash_table_search(open_hash_table_t *table, char *key, int *comparisons);` - Поиск слова в таблице.

`int open_hash_table_restuct(open_hash_table_t **table, int *max_comparisons);` _- Реструктуризация таблицы.
`void open_hash_table_print(open_hash_table_t *table);` - Вывести таблицу на экран.

Закрытое хеширование:
`closed_hash_table_t* closed_hash_table_new(size_t size);` - Создание таблицы.
`void closed_hash_table_delete(closed_hash_table_t *table);` - Удаление таблицы с освобождением памяти.
`size_t closed_hash_table_get_size(closed_hash_table_t *table);` - Получение размера таблицы.
`int closed_hash_table_insert(closed_hash_table_t *table, char *key, int *comparisons);` - Вставить слово в таблицу.
`char* closed_hash_table_remove(closed_hash_table_t *table, char *key, int *comparisons);` - Удалить слово из таблицы.
`char* closed_hash_table_search(closed_hash_table_t *table, char *key, int *comparisons);` - Поиск слова в таблице.
`int closed_hash_table_restuct(closed_hash_table_t **table, int *max_comparisons);` _- Реструктуризация таблицы.
`void closed_hash_table_print(closed_hash_table_t *table);` - Вывести таблицу на экран.

4. Описание алгоритма.

Выводится меню программы. Пользователь вводит номер команды, после чего выполняется действие.

АВЛ-дерво:

При добавлении слова в АВЛ-дерево временный указатель проходит рекурсивно по узлам дерева, пока не дойдёт до листа. После вставки нового элемента, рекурсивно происходит возврат к корню дерева, при этом для каждого посещаемого узла происходит балансировка, если разница высот его сыновей превосходит по модулю 1.

При удалении слова из дерева, слово сравнивается со словом в вершине. Если слово оказалось больше, чем слово в вершине, алгоритм рекурсивно вызывается для левого потомка, если меньше - для правого. Если слово совпало со словом вершине, проверяется наличие всех потомков. Если один из потомков отсутствует, на место данной вершины встает второй, а данная вершина удаляется. Если существуют оба потомка, в правом поддереве производится поиск наименьшего элемента, слово которого встает на место слова в данной вершине, после чего удаляется найденный наименьший элемент правого поддерева. После удаления наименьшего элемента рекурсивно происходит возврат к корню дерева, при этом для каждого посещаемого узла происходит балансировка, если разница высот его сыновей превосходит по модулю 1.

При поиске слова в дереве слово для поиска сравнивается со словом в вершине дерева. Если слово больше, алгоритм вызывается рекурсивно для правого наследника, если меньше - для левого. Алгоритм работает до того, как слово будет найдено или пока не дойдёт до неподходящего листа дерева.

Хеш-таблицы:

Основой всех действий над таблицами является хеш-функция. Реализованная хеш-функция работает следующим образом: задается переменная хеша, равная нулю. Цикл проходит по символам строки, умножая

промежуточный результат на простое число, после чего прибавляет к нему числовое значение символа. При использовании хеш-функции в таблицах, от её результата берётся остаток деления на размер таблицы.

Открытое хеширование:

При вставке в таблицу для слова, которое нужно вставить в таблицу, вычисляется хеш. После чего проверяется элемент массива под индексом хеша. Если элемент пуст – на его место записывается строка. Если элемент занят, проверяются по порядку всё элементы таблицы. Если находится пустой элемент массива – на его место записывается строка. Если находится строка, равная строке для вставки, алгоритм завершается. Если был достигнут конец массива, происходит поиск с начала массива до значения хеша. Если свободное место в таблице не было найдено, алгоритм завершается с ошибкой.

При поиске в таблице вычисляется хеш слова для поиска. Проверяется элемент под индексом хеша. Если строка равно строке под индексом, она возвращается. Иначе алгоритм проходит по таблице, пока не найдет нужное слово или таблица не закончится.

При удалении слова из таблицы, производится поиск слова. Если слово было найдено, оно удаляется. Иначе алгоритм завершается с ошибкой.

Закрытое хеширование:

При вставке в таблицу для слова, которое нужно вставить в таблицу, вычисляется хеш. После чего проверяется элемент массива под индексом хеша. Если элемент пуст – на его место записывается узел, содержащий строку. Если элемент занят, в начало списка вставляется узел, содержащий строку.

При поиске в таблице вычисляется хеш слова для поиска. Проверяется элемент под индексом хеша. Если строка в узле совпадает, она возвращается. Иначе алгоритм проходит по списку, пока не дойдёт до нужного слова или до конца списка.

При удалении слова из таблицы, производится поиск слова. Если слово было найдено, оно удаляется. Иначе алгоритм завершается с ошибкой.

При реструктуризации всех таблиц создается новая таблица, размер которой равен ближайшему простому числу, превышающему нынешний размер таблицы. Все элементы старой таблицы вставляются в новую, после чего старая удаляется.

5. Оценка эффективности.

При проведении исследования производился поиск всех слов в каждом из наборов, размерами от 5000 до 100000 слов, после чего бралось среднее время поиска одного слова.

Таблица с открытым хешированием создавалась с размером, равному количеству слов в наборе. С закрытым — количество слов в наборе, умноженное на 1.2.

	Среднее время поиска слова, тики			
Количество слов	БДП	АВЛ-дерево	Закрытое хеш.	Открытое хеш.
5000	802	644	318	218
10000	1059	911	358	383
15000	1217	988	408	348
20000	1175	935	351	319
25000	1256	993	388	334
30000	2103	1628	641	517
35000	1383	1113	407	351
40000	1450	1181	419	367
45000	1532	1236	450	389
50000	1611	1324	484	403
55000	1680	1373	504	416
60000	1814	1464	525	432
65000	1906	1547	569	439
70000	1917	1563	539	442
75000	1905	1567	558	443
80000	1981	1619	563	453
85000	1994	1630	562	449
90000	2062	1679	574	454
95000	2145	1742	588	455
100000	2177	1779	600	463

	Средний размер, занимаемый структурой, байты			
Количество слов	БДП	АВЛ-дерево	Закрытое хеш.	Открытое хеш.
5000	120000	160000	40000	80000
10000	240000	320000	80000	160000
15000	360000	480000	120000	240000
20000	480000	640000	160000	320000
25000	600000	800000	200000	400000
30000	720000	960000	240000	480000
35000	840000	1120000	280000	560000
40000	960000	1280000	320000	640000
45000	1080000	1440000	360000	720000
50000	1200000	1600000	400000	800000
55000	1320000	1760000	440000	880000
60000	1440000	1920000	480000	960000
65000	1560000	2080000	520000	1040000
70000	1680000	2240000	560000	1120000
75000	1800000	2400000	600000	1200000
80000	1920000	2560000	640000	1280000
85000	2040000	2720000	680000	1360000
90000	2160000	2880000	720000	1440000
95000	2280000	3040000	760000	1520000
100000	2400000	3200000	800000	1600000

	Среднее количество сравнений для поиска			
Количество слов	БДП	АВЛ-дерево	Закрытое хеш.	Открытое хеш.
5000	15,5	11,6	2,85	1,49
10000	16,9	12,6	2,95	1,5
15000	17,7	13,2	2,95	1,5
20000	18,3	13,6	2,94	1,5
25000	18,7	13,9	3,24	1,5
30000	19,1	14,2	3,08	1,5
35000	19,4	14,4	3,06	1,49
40000	19,7	14,6	3,09	1,5
45000	19,9	14,8	3,1	1,5
50000	20,2	14,9	3,15	1,5
55000	20,3	15,1	3,29	1,5
60000	20,5	15,2	3,15	1,5
65000	20,7	15,3	3,24	1,5
70000	20,8	15,4	3,1	1,5
75000	21	15,5	3,22	1,5
80000	21,1	15,6	3,23	1,5
85000	21,2	15,7	3,16	1,5
90000	21,3	15,8	3,22	1,5
95000	21,4	15,9	3,18	1,49
100000	21,5	15,9	3,31	1,5

6. Выводы.

Меньший объем памяти занимает хеш-таблица с закрытым хешированием, однако хеш-таблице с открытым хешированием по скорости поиска и количеству сравнений.

АВЛ-дерево занимает больше памяти, чем бинарное, однако поиск в нём, в среднем, производится быстрее.

Поиск в таблице с открытым хешированием самый быстрый. При этом такая таблица занимает меньше памяти, чем все деревья, но уступает таблице с закрытым хешированием в 2 раза.

7. Контрольные вопросы.

1. Чем отличается идеально сбалансированное дерево от АВЛ-дерева?
   Идеально сбалансированное дерево – дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу. При этом идеально сбалансированное дерево не является двоичным деревом поиска.
   АВЛ-дерево – двоичное дерево поиска, если у каждого узла высота двух поддеревьев отличается не более, чем на единицу.
2. Чем отличается поиск в а АВЛ-дереве от поиска в дереве двоичного поиска?
   Алгоритмически поиск ничем не отличается. Однако структура АВЛ-дерева гарантирует поиск за O(logn), когда в бинарном дереве поиска возможен поиск за O(n).
3. Что такое хеш-таблица, каков принцип её построения?
   Хеш-таблица – структура данных, допуск к данным которых производится путём вычисления значения хеш-функции к ключу для поиска.
4. Что такое коллизии? Каковы методы их устранения?
   Коллизия – совпадение значений хеш-функции для двух разных элементов. Методы – открытое и закрытое хеширование. При открытом, при коллизии, элементу добавляется связный список. При закрытом происходит поиск свободного места в таблице.
5. В каком случае поиск в хеш-таблицах становится неэффективен?
   Если для поиска элемента необходимо произвести более 3-4 сравнений.
6. Эффективность поиска в АВЛ-деревьях, в ДДП, в хеш-таблице и в файле.
   В АВЛ-дереве время поиска – O(logn), в ДДП в среднем – O(logn), в худшем – O(n). В хеш-таблице с закрытым хешированием от O(1) до O(n), с открытым от O(1) до O(n).