Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

Отчет по дисциплине «Типы и структуры данных»
Лабораторная работа №6

Вариант 3

Студент Голикова С.М.
Группа ИУ7-35Б

Москва, 2021

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. Внешняя спецификация

a) Исходные данные и результаты

Входные данные:

- Целое число от 0 до 5 — номер пункта меню, который вызывает описанное в пункте действие
- Строка — слово для удаления

Допущения:

1) Максимальная длина слова — 20

Выходные данные:

```
Выберите действие:

1 - Прочитать данные из файла
2 - Вывести двоичное дерево поиска (ДДП)
3 - Вывести АВЛ-дерево
4 - Вывести хеш-таблицу
5 - Удалить слово при использовании ДДП, сбалансированного дерева, хеш-таблицы и файла
0 - Выйти из программы

Ваш выбор:
```

В зависимости от пункта меню (см.выше):

- Дерево слов.
- Хеш-таблица.
- Результаты анализа эффективности при работе с различными структурами — время, объем памяти, количество сравнений при удалении элемента.

b) Задачи, реализуемые программой

- Создание и вывод двоичного дерева поиска.
- Создание и вывод АВЛ-дерева (сбалансированного дерева).
- Формирование, вывод, реструктуризация (при необходимости) хеш-таблицы.
- Сравнение эффективности удаления элемента при использовании различных структур

c) Способ обращения к программе

Программа запускается из терминала в директории с проектом при помощи команды «./app.exe filename.txt» (filename - имя файла, из которого происходит считывание слов).

d) Возможные аварийные ситуации и ошибки пользователя

Аварийные ситуации и ошибки:

- некорректный выбор пункта меню
- некорректное имя файла
- пустой файл
- неверное число допустимых сравнений при удалении слова
- пустое дерево / хеш-таблица при удалении всех элементов

В случае аварийной ситуации пользователю выдается сообщение об ошибке и происходит возвращение в меню.

3. Внутренние структуры данных

```
// текущая вершина дерева
typedef struct tree_node
{
    char *data;             // значение текущей вершины (слово)
    unsigned int height;  // высота вершины отн. других вершин
    struct tree_node *left;  // левый потомок
    struct tree_node *right; // правый потомок
} tree_node_t;

// текущий элемент списка
typedef struct hash
{
    char data[MAX_WORD_LEN]; // значение текущего элемента
                             // списка
    struct hash *next;       // след. элемент списка (список
                             // содержит элементы с одинаковым
                             // хешем)
} hash_t;

// хеш-таблица
typedef struct hash_table
{
    int size;           // размер хеш-таблицы
    hash_t **list;      // массив списков элементов с
                        // одинаковым хешем
} hash_table_t;
```

4. Описание алгоритма

Двоичное дерево поиска:

Искомое слово сравнивается со словом, находящимся в текущей вершине. Если они совпадают — поиск завершен, и элемент удаляется. Если искомое слово меньше — поиск продолжается в левом поддереве вершины, иначе — в правом. Если у узла нет дочерних узлов, то у его родителя указатель просто заменяется на NULL. Если у узла есть только один дочерний узел, то создается новая связь между родителем удаляемого узла и его дочерним узлом. Наконец, если у узла два дочерних узла, то на место удаляемого элемента ставится самый левый (минимальный) элемент из правого поддерева.

АВЛ-дерево:

Алгоритм аналогичен ДДП.

Хеш-таблица:

Для каждого элемента таблицы: определяется хеш-значение, по хеш-значению добавляется в односвязный список (метод цепочек устранения коллизий). Хеш-значение по слову S определяется как сумма символов слова % размер хеш-таблицы. При поиске в хеш-таблице считается число сравнений для элемента. Если оно превышает введенное пользователем, происходит реструктуризация хеш-таблицы.
Реструктуризация хеш-таблицы: используется новая хеш-функция
hash = S[0] + S[1] * (1 * 10) + S[2] * (1 * 10 * 11) + S[3] * (1 * 10 * 11 * 12) + ...

5. Основные функции, используемые в программе

ДДП:

```
tree_node_t *create_node(char *value) // создание вершины дерева
tree_node_t *add_node_to_tree(tree_node_t *cur_node, char
*value) // добавление вершины в дерево
int fill_tree_from_file(FILE *file, tree_node_t **root)
// заполнение дерева словами из файла
tree_node_t *remove_bst_node(tree_node_t *cur_node, char
*target, int *cnt, int *found) // удаление вершины дерева
tree_node_t *get_min_node(tree_node_t *root) // минимальная
вершина
```

АВЛ-дерево:

```
// функции для балансировки
unsigned int height(tree_node_t *cur_node)
int balance_factor(tree_node_t *cur_node)
void fix_height(tree_node_t *cur_node)
tree_node_t *rotate_right(tree_node_t *cur_node)
tree_node_t *rotate_left(tree_node_t *cur_node)
tree_node_t *balance(tree_node_t *cur_node)
tree_node_t *add_node_to_avl_tree(tree_node_t *cur_node, char
*value) // добавление вершины в дерево
int fill_avl_tree_from_file(FILE *file, tree_node_t **root)
// заполнение дерева словами из файла
tree_node_t *remove_min(tree_node_t *cur_node // удаление
минимальной вершины дерева
tree_node_t *remove_avl_node(tree_node_t *cur_node, char
*value, int *cnt, int *found) // удаление вершины дерева
```

Хеш-таблица:

```
int hash_function(char *key, int size) // хеш-функция
int new_hash_function(char *key, int size) // новая хеш-функция
void hash_table_init(hash_table_t *table, int table_size)
void insert(hash_t **list, char *key)
void add_key_to_hash_table(hash_table_t *table, char *key, int
restruct)
```

`int fill_hash_table_from_file(FILE *file, hash_table_t *table, int restruct)` // заполнение хеш-таблицы словами из файла
`int remove_hash_table_value(hash_table_t *table, char *value, int *cnt, int restruct, int *words_num)` // удаление значения в хеш-таблице

6. Тесты

№ теста	Ввод	Действие	Вывод
1	6	Ввод неверного пункта меню	Ошибка ввода: необходимо ввести номер одного из предложенных вариантов
2	1	Неверное имя файла	Не удалось открыть файл
3	1	Пустой файл	Файл пуст
4	2-5	Данные не загружены	Дерево пусто (Хеш-таблица пуста)
5	5	Удаление всех элементов	Дерево пусто (Хеш-таблица пуста)
6	2-3	Дерево не пустое	Дерево слов
7	4	Хеш-таблица не пустая	Хеш-таблица
8	5	Удаление элемента	Сравнение эффективности

7. Анализ эффективности

Время удаления самого дальнего элемента (такты):

Количество элементов	ДДП	АВЛ	Хеш-таблица	Файл
50	6320	4431	2319	349125
100	7070	4500	2321	564098
500	7500	5426	2325	1022590

Объем памяти (байты) - размер хеш-таблицы равен количеству слов:

Количество элементов	ДДП	АВЛ	Хеш-таблица	Файл
50	1600	1600	2952	479
100	3200	3200	5412	962
500	6400	6400	10212	1921

8. Выводы

При сравнении удаления слова в четырех структурах данных (дерево двоичного поиска, сбалансированное, хеш-таблица и файл) получили следующие результаты:

Самой эффективной структурой данных по времени обработки является хеш-таблица. Выигрыш по времени объясняется числом сравнений при поиске (при отсутствии коллизий количество сравнений при поиске слова равно 1). Но при этом объем памяти хеш-таблицы меньше, чем объем памяти, выделенной под деревья.

Самой эффективной структурой данных по занимаемому объему памяти является файл. При этом файл является самой неэффективной структурой данных по времени, так как все слова в файле идут последовательно и необходимо пройти все слова, лежащие до искомого.

Сравнение двух реализаций деревьев (ДДП и сбалансированного) показало, что поиск слова в сбалансированном дереве происходит быстрее, чем в ДДП, что объясняется меньшей высотой сбалансированного дерева. Объем занимаемой памяти двух реализаций одинаков.

9. Контрольные 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?

Дерево реализуется при помощи односвязного списка, поэтому память выделяется для каждого узла отдельно.

3) Какие стандартные операции возможны над 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 (меньше, либо равны), а все правые – старше (больше, либо равны). Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень.

5) Чем отличается идеально сбалансированное дерево от АВЛ-дерева?

У идеально сбалансированного дерева число вершин в левом и правом поддеревьях отличается не более, чем на единицу. У каждого узла АВЛ-дерева высота двух поддеревьев отличается не более, чем на единицу.

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

Поиск в АВЛ-дереве происходит быстрее, чем поиск в дереве двоичного поиска и с меньшим числом сравнений.

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

Хеш-таблица - массив, заполненный в порядке, определенным хеш-функцией. Принцип построения: хеш-функция 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.

8) Что такое коллизии? Каковы методы их устранения?

Коллизии - ситуации, когда разным ключам соответствует одно значение хешфункции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2. Методы устранения: 1) 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. 2) Внутреннее (закрытое) хеширование (открытая адресация). 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. Для вычисления шага можно также применить формулу: h = h + a 2 , где a – это номер попытки поиска ключа. Этот вид адресации называется квадратичной или произвольной адресацией.

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

Если для поиска элемента необходимо более 3–4 сравнений, то эффективность использования хеш-таблицы пропадает.

10) Эффективность поиска в АВЛ-деревьях, в дереве двоичного поиска и в хеш-таблицах. АВЛ-деревья : O(log2(n)) ДДП : O(log2(n)) - O(n)
Хеш-таблица: O(1)