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 №6

По дисциплине «Типы и структуре данных»

Название: Обработка деревьев

Студент Дремин Кирилл

Группа ИУ7 – 36Б

Тип лабораторной работы: Учебная

Вариант №0

Преподаватель Барышникова Марина Юрьевна

2022 г.

Содержание

Описание условия задачи

Получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход дерева, включение, исключение, поиск узлов. Оценить эффективность применения деревьев в прикладных задачах.

Описание ТЗ

Описание исходных данных и результатов:

Программа реализует дерево, использующее в качестве ключевого значения код символа, а в качестве информационной частит - число его вхождений в строку.

Доступны операции построения двоичного дерева поиска по введённой строке, включения и исключения элементов в дерево, удаления повторяющихся символов из дерева и строки, поиск информации о числе вхождения символа в дереве.

Способ обращения к программе

Обращение происходит посредством вызова заранее скомпилированного файла и ввода через консоль.

Описание входных данных

Программа позволяет произвести анализ введённой строки, а также визуализировать используемое для работы дерево.

```
1. Построить дерево поиска по строке
2. Вывести дерево на экран (левый правый >1)
3. Удалить повторяющиеся буквы из строки и дерева
4. Вывести элементы дерева при постфиксном обходе
5. Сравнить время удаления повторяющихся букв из текущих д
ерева и строки
6. Сгенерировать случайную строку заданного размера и созд
ать дерево на её основе
7. Добавить символ в дерево
8. Удалить символ из дерева
9. Найти символ в дереве
10. Измерить время обработки узла в дереве
0. Выход из программы
```

Корректный ввод - цифра от 0 до 10.

Описание возможных аварийных ситуаций и ошибок пользователя:

1. Ввод некорректных данных.
2. Ошибки выделения динамической памяти.
3. Указание некорректных символов при вводе.
4. Запрос вывода дерева на экран при наличии в нём символов угловых скобок или амперсанда.

Описание внутренних СД:

Программа оперирует АТД, описывающим операции над двоичным деревом, хранящим информацию о частотах символов:

```
typedef struct tree_t tree_t;

/// Получить значение символа из узла
char get_value(const tree_t *node);

/// Получить число вхождений символа из узла
size_t get_count(const tree_t *node);

/// Создание дерева на основе строки
tree_t *create_tree(char *str);

/// Освобождение памяти, выделенной под дерево
void free_tree(tree_t *tree);

/// Постфиксный обход дерева
void traverse_post(tree_t *tree, void (*apply)(tree_t *, void *), v
oid *acc);

/// Префиксный обход дерева
void traverse_pre(tree_t *tree, void (*apply)(tree_t *, void *), vo
id *acc);

/// Инфиксный обход дерева
void traverse_in(tree_t *tree, void (*apply)(tree_t *, void *), voi
d *acc);

/// Вставка символа в дерево
tree_t *tree_insert(tree_t *tree, char ch);
```

```
включения
/// Удаление символа из дерева
tree_t *tree_remove(tree_t *tree, char ch);

/// Поиск узла в дереве
tree_t *tree_find(tree_t *tree, char ch);

/// Вывод дерева при помощи утилиты graphviz
void show_tree(tree_t *tree);

/// Копия дерева
tree_t *copy_tree(tree_t *tree);

/// Копия дерева без повторяющихся элементов
tree_t *tree_without_repeating(tree_t *tree);
```

Непосредственная реализация дерева:

```
struct tree_t
{
    char key;
    size_t count;
    tree_t *left;
    tree_t *right;
};
```

В данной реализации дерева символ является ключевым значением и образует вместе с количеством вхождений информационную часть узла дерева.

Замеры эффективности обработки строк

Сравним эффективность удаления повторяющихся символов в дереве и строке.

Для начала рассмотрим случай случайного заполнения строки большим числом символов (таким образом, что все символы повторяются), здесь и далее встречаются ~90 различных символов.

Используется 1000000 повторений для замеров:

Структура	Кол-во символов	Всего (мс)	В среднем (мс)
Строка	1000	8835	0.0088
Дерево	1000	7935	0.0079
Строка	500	4798	0.0048
Дерево	500	7395	0.0074

Строка	100	1910	0.0019
Дерево	100	4417	0.0044
Строка	50	1731	0.0017
Дерево	50	3139	0.0031
Строка	10	1353	0.0014
Дерево	10	1545	0.0015

Можно убедиться в том, что дерево поиска становится эффективным при отношении длины строки к количеству доступных символов в ~10 раз, это объясняется меньшей эффективностью операций обхода дерева по сравнению с обходом строки, до этого же эффективность дерева меньше, чем эффективность простой обработки строки, однако сравнив эффективность обработки строки из 500 и 1000 символов, можно убедиться в том, что начиная с некоторого порога, время обработки дерева не растёт, в то время как время обработки строки растёт с ростом длины строки.

Далее сравним эффективность удаления повторяющихся символов из строки, состоящей из одного и того же символа, повторённого заданное число раз.

Используется 1000000 повторений для замеров:

Структура	Кол-во символов	Всего (мс)	В среднем (мс)
Строка	1000	8260	0.0083
Дерево	1000	1221	0.0012
Строка	500	4390	0.0044
Дерево	500	1245	0.0012
Строка	100	1877	0.0019
Дерево	100	1248	0.0012
Строка	50	1551	0.0016
Дерево	50	1235	0.0012
Строка	10	1268	0.0013
Дерево	10	1248	0.0012
Строка	5	1245	0.0013
Дерево	5	1260	0.0013

Можно убедиться в том, что в силу однообразности строки и малой (по сути, нулевой) глубины дерева, скорость его обработки является постоянной и можно уверенно заявить, что обработка дерева эффективнее обработки строки для случаев малой глубины дерева.

Проведём также измерение эффективности для строк, состоящих из символов с монотонно возрастающими кодами, что должно привести к формированию правостороннего дерева при обработке. Используется 1000000 повторений для замеров.

Структура	Кол-во символов	Всего (мс)	В среднем (мс)
Строка	94	1745	0.0018
Дерево	94	4625	0.0046
Строка	50	1596	0.0016
Дерево	50	2876	0.0029
Строка	25	1386	0.0014
Дерево	25	1962	0.0020
Строка	10	1300	0.0013
Дерево	10	1550	0.0015
Строка	5	1271	0.0013
Дерево	5	1358	0.0014

Видно, что эффективность обработки дерева с большой глубиной гораздо ниже эффективности обработки строки, которой оно порождено.

Также стоит измерить эффективность обработки дерева с небольшой, но не нулевой глубиной, для этого используем дерево, состоящее из 7 узлов, глубиной 3, при 1000000 повторениях для замеров.

Структура	Кол-во символов	Всего (мс)	В среднем (мс)
Строка	500	4799	0.0049
Дерево	500	1489	0.0015
Строка	250	2978	0.0030
Дерево	250	1555	0.0016
Строка	100	1972	0.0020

Дерево	100	1479	0.0015
Строка	25	1308	0.0013
Дерево	25	1488	0.0015

Действительно, можно утверждать, что эффективность обработки строки с использованием дерева зависит исключительно от глубины полученного дерева, а следовательно, от набора символов и их порядка в строке. Зависимость от порядка может быть устранена путём модификации дерева в самобалансирующееся двоичное дерево поиска.

Проверим также эффективность поиска символа в правостороннем дереве. Для каждого случая проведено 1000000 замеров.

Символа нет, и он меньше корня: 18 мс. всего, 0.00001800 мс. в среднем, символ не найден!

Символа нет, и он больше листа: 504 мс. всего, 0.00050400 мс. в среднем, символ не найден!

Символ есть, и он равен корню: 9 мс. всего, 0.00000900 мс. в среднем.

Символ есть, и он находится в середине строки: 234 мс. всего, 0.00023400 мс. в среднем.

Символ есть, и он равен листу: 497 мс. всего, 0.00049700 мс. в среднем.

В целом можно сделать вывод, что поиск по вырожденному дереву является поиском по линейному связному списку, однако возможны также дополнительные эвристики, применимые для распознавания отсутствия элемента в случае, если он меньше корня для правостороннего дерева и в случае, если он больше корня для левостороннего дерева. (в таких случаях не происходит перебора элементов списка). Данная эвристика также может быть использована в случае, если искомый элемент принимает значение между двумя какими-либо узлами списка. В таком случае поиск будет происходить до ближайшего к нему по значению узла.

Вывод

Двоичное дерево поиска действительно является полезной структурой данных, позволяющей эффективно обрабатывать достаточно однородные данные любого вида.

Ответы на вопросы

1. Что такое дерево?

Дерево - нелинейная структура данных, используемая для представления иерархических связей в виде отношения “один ко многим”

2. Как выделяется память под представление деревьев?

В общем случае память под представление деревьев выделяется, например, с использованием связного списка потомков дерева, также для двоичных деревьев и иных деревьев с фиксированным максимальным числом потомков может применяться представление в виде хранения отдельных указателей на поддеревья (либо массива таковых указателей)

Непосредственно выделение динамической памяти происходит при необходимости вставки.

3. Какие бывают типы деревьев?

Деревья в целом могут быть классифицированы на большое количество видов в зависимости от налагаемых на их содержимое ограничений, так, можно рассмотреть: квадро- и окто- деревья, используемые в основном для разбиения двух- и трёх- мерного пространства, самобалансирующиеся двоичные деревья поиска (АВЛ-деревья и красно-чёрные деревья), префиксные деревья для работы со строками, одними из часто применяемых для оптимизации поиска являются двоичные деревья поиска и их разновидности.

4. Какие стандартные операции возможны над деревьями?

Стандартные операции над деревьями: включение и исключение элементов, поиск элемента и обход дерева (префиксный, постфиксный, инфиксный).

5. Что такое дерево двоичного поиска?

Дерево двоичного поиска - такое двоичное дерево (имеющее двух потомков, традиционно называемых левым и правым), в котором все левые потомки меньше своего предка, а все правые потомки больше своего предка, в соответствии с выбранным алгоритмом сравнения.