

Оценка

Описание условия задачи

Построить двоичное дерево поиска из букв вводимой строки. Вывести его на экран в виде дерева. Выделить цветом все буквы, встречающиеся более одного раза. Удалить из дерева эти буквы. Сбалансировать дерево после удаления повторяющихся букв. Вывести его на экран в виде дерева. Составить хеш-таблицу, содержащую буквы и количество их вхождений во введенной строке. Вывести таблицу на экран. Осуществить поиск введенной буквы в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных.

Техническое задание

Исходные данные:

Пункты меню, выраженные целыми числами 0-11, 99, -1. Внутри некоторых пунктов вводятся символы или строки.

- 1) Добавить строку в дерево
- 2) Добавить узел
- 3) Удалить узел
- 4) Вывести в строку
- 5) Вывести в виде дерева
- 6) Вывести узел
- 7) Удалить повторяющиеся буквы и сбалансировать
- 8) Ввести хэш-таблицу
- 9) Вывести хэш-таблицу
- 99) Вывести всю хэш-таблицу
- 10) Найти символ в таблице
- 11) Сравнить структуры
- 1) Очистить дерево
- 0) Выход

Результат:

Дерево (в виде дерева и в строку), поддереву, дерево без повторяющихся символов, сбалансированное дерево, хэш-таблица, результаты измерений

Описание задачи:

Преобразовать строку в дерево, добавить узел в дерево, удалить узел, вывести дерево в строку, вывести дерево в виде дерева, найти узел по символу и вывести поддерево, удалить повторяющиеся буквы, сбалансировать дерево, преобразовать строку в хэш-таблицу, вывести хэш-таблицу, найти символ в таблице, сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных

Способ обращения к программе:

Запуск с помощью ./app.exe

Аварийные ситуации и ошибки:

1. Введена пустая строка
2. Символ не введен
3. Пустое дерево
4. Узел не найден

Описание внутренних структур данных

```
struct Node {  
    char data;  
    int count;  
    int height;  
    struct Node* left;  
    struct Node* right;  
};
```

struct Node: узел дерева

1. data – Символ
2. count – Количество повторений символа в строке
3. height – Высота узла
4. left – Указатель на левый смежный узел
5. right – Указатель на правый смежный узел

```
#define TABLE_SIZE 128

struct HashNode
{
    char key;
    int count;
    struct HashNode* next;
};

struct HashTable
{
    struct HashNode* table[TABLE_SIZE];
};
```

struct HashTable: Структура для представления хеш-таблицы

struct HashNode* table[TABLE_SIZE]; - массив элементов

struct HashNode : Структура для представления элемента хеш-таблицы

key – ключ

count – значение

next – ссылка на следующий элемент (не равно NULL в случае коллизии)

Описание алгоритмов

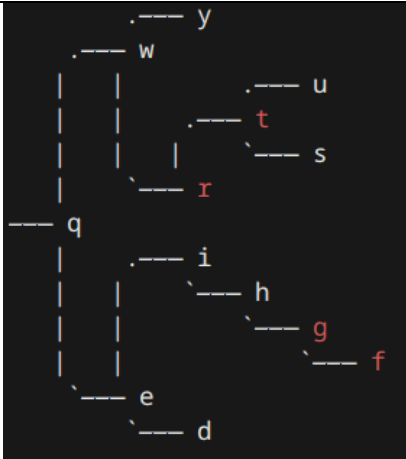
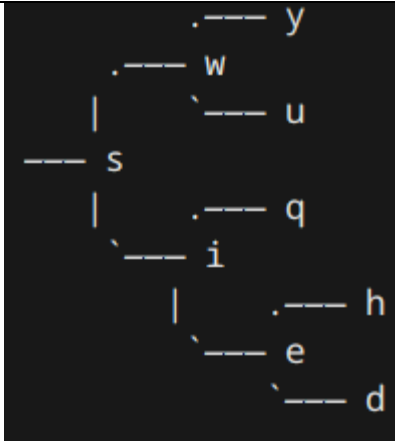
1. Отобразить пользователю список команд и дождаться ввода номера нужной команды.
2. Команда добавления символа в дерево считывает символ, затем постепенно спускается в глубь дерева, пока не находит позицию, в которую надо вставить символ, затем вставляет и балансирует дерево.
3. Команда преобразования строки в дерево, поочередно добавляет символы строки в дерево.
4. Команда удаления узла ищет узел, а затем заменяет его на минимальный из его потомков.
5. Команда вывода в строку постфиксно обходит дерево и выводит элементы в строку
6. Команда вывода в виде дерева проходится с правого края дерева до левого, выводя узлы с соответствующей им глубиной.

7. Команда вывода узла ищет с помощью бинарного поиска узел в дереве, а затем выводит соответствующее этому узлу поддерево.
8. Команда удаления повторяющихся символов удаляет из дерева все узлы, поле count которых больше 1 и балансирует дерево основываясь на высотах узлов и высчитывая факторы баланса и при необходимости поворачивая дерево.
9. Команда ввода хэш-таблицы считывает строку, высчитывает для каждого символа значение хэш-функции как код символа % 128 и добавляет символ в таблицу.
10. Команда вывода хэш-таблицы выводит все элементы таблицы с соответствующими ключами и значениями хэш-функции.
11. Найти символ в таблице: по значению хэш-функции выбирается элемент, при необходимости проходя по соответствующему списку.
12. Команда сравнения структур, генерирует случайные строки длиной от 10 до 510, а затем засекает время поиска символов. Все замеры производятся путем многочисленных запусков с ожиданием $RSE < 5$. Затем производится анализ затраченного времени и памяти.

Тестовые данные

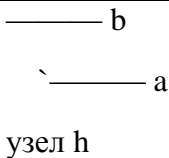
Позитивные тесты

| Тест | Входные данные | Результат |
|--|--|--|
| Добавление узла в дерево | Узел b Дерево <pre> ._____ n _____ c ._____ b _____ a _____ c _____ a </pre> | <pre> ._____ n _____ c ._____ b _____ a _____ c _____ a </pre> |
| Удаление узла из дерева | Удалить a из: <pre> ._____ n _____ c _____ a _____ c _____ a </pre> | <pre> ._____ n _____ c _____ c _____ a </pre> |
| Создание дерева из строки | Строка asp | <pre> ._____ n _____ c _____ a </pre> |
| Вывод дерева в строку (Постфиксный обход) | <pre> ._____ n _____ c _____ a </pre> | а с n |
| Вывод дерева в виде дерева | Существующее дерево из asp | <pre> ._____ n _____ c _____ a </pre> |
| Вывод дерева с повторяющимися буквами | Существующее дерево из bcacfgb | <pre> ._____ g ._____ f _____ c _____ b _____ a </pre> |
| Поиск узла по символу и вывод поддерев | Существующее дерево <pre> ._____ n _____ c _____ a </pre> символ a для поиска | Вывод поддерев, начинающегося с найденного узла. <pre> _____ a </pre> |

| | | |
|--|---|--|
| Удаление повторяющихся букв и балансировка |  |  |
| Создание хэш-таблицы | qqweerty | Символ: e, Количество: 2 Символ: q, Количество: 2 Символ: r, Количество: 1 Символ: t, Количество: 1 Символ: w, Количество: 1 Символ: y, Количество: 1 |
| Найти символ в таблице | Таблица qqweerty Символ e | Количество = 2 |
| Сравнить структуры | - | Результаты сравнения |

Негативные тесты

| | | |
|---|--|--|
| Удаление несуществующего узла из дерева | <pre> .——— g .——— f \——— c ——— b \——— a узел h </pre> | Сообщение об ошибке и отсутствие изменений в дереве. |
| Поиск несуществующего символа и вывод поддерева | <pre> .——— g .——— f \——— c </pre> | Сообщение об ошибке |

| | | |
|---|---|---------------------|
| |  | |
| Поиск несуществующего символа в таблице | Таблица qqweerty Символ o | Символ не найден |
| Пустая строка для преобразования | Пустая строка | Сообщение об ошибке |
| Попытка вывода пустого дерева | Пустое дерево | Сообщение об ошибке |

Замеры

Программа сравнивает время поиска символов в несбалансированном, сбалансированном деревьях и в хэш-таблице. Программа создает случайные строки длинами от 10 до 510 с шагом 100 и проводит замеры для каждого из размеров. Для каждого из тестов программа выполняет многочисленные замеры, пока их RSE не становится <5. Затем все тестовые случаи сравниваются и оценивается их эффективность.

Длина строки - 10

Реализация с помощью дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 95.98 | 60 | 4.60 |

Занимаемая память - 288 байт

Количество сравнений - 3

Реализация с помощью сбалансированного дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 112.70 | 20 | 4.46 |

Занимаемая память - 288 байт

Количество сравнений - 3

Реализация с помощью хэш-таблицы:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 74.73 | 30 | 4.60 |

Занимаемая память - 1168 байт

Количество сравнений - 1

Сравнение дерева и сбалансированного дерева:

Разность производительности = 16.72 нс

Реализация с помощью сбалансированного дольше на 17.416218 %

Разность занимаемой памяти = 0 байт

Разность количества сравнений = 0

Сравнение несб. дерева и хэш-таблицы:

Разность производительности = 21.25 нс

Реализация с помощью дерева дольше на 28.434434 %

Разность занимаемой памяти = 880 байт

Реализация с помощью хэш-таблицы больше на 305 %

Разность количества сравнений = 2

Реализация с помощью несбалансированного требует больше на 200 %

Сравнение сб. дерева и хэш-таблицы:

Разность производительности = 37.97 нс

Реализация с помощью дерева дольше на 50.802855 %

Разность занимаемой памяти = 880 байт

Реализация с помощью хэш-таблицы больше на 305 %

Разность количества сравнений = 2

Реализация с помощью сбалансированного требует больше на 200 %

Длина строки - 110

Реализация с помощью дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 168.47 | 60 | 4.82 |

Занимаемая память - 1280 байт

Количество сравнений - 6

Реализация с помощью сбалансированного дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 133.30 | 20 | 4.72 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью хэш-таблицы:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 69.10 | 10 | 1.47 |

Занимаемая память - 1664 байт

Количество сравнений - 1

Сравнение дерева и сбалансированного дерева:

Разность производительности = 35.17 нс

Реализация с помощью несбалансированного дольше на 26.381595 %

Разность занимаемой памяти = 0 байт

Разность количества сравнений = 1

Реализация с помощью несбалансированного требует больше на 20 %

Сравнение несб. дерева и хэш-таблицы:

Разность производительности = 99.37 нс

Реализация с помощью дерева дольше на 143.801254 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 5

Реализация с помощью несбалансированного требует больше на 500 %

Сравнение сб. дерева и хэш-таблицы:

Разность производительности = 64.20 нс

Реализация с помощью дерева дольше на 92.908828 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью сбалансированного требует больше на 400 %

Длина строки - 210

Реализация с помощью дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 141.60 | 30 | 4.13 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью сбалансированного дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 135.07 | 30 | 4.26 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью хэш-таблицы:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|-----|
|-----------|-----------------|-----|

| | | | | | |
|--|-------|--|----|--|------|
| | 72.87 | | 30 | | 3.89 |
|--|-------|--|----|--|------|

Занимаемая память - 1664 байт

Количество сравнений - 1

Сравнение дерева и сбалансированного дерева:

Разность производительности = 6.53 нс

Реализация с помощью несбалансированного дольше на 4.837117 %

Разность занимаемой памяти = 0 байт

Разность количества сравнений = 0

Сравнение несб. дерева и хэш-таблицы:

Разность производительности = 68.73 нс

Реализация с помощью дерева дольше на 94.327539 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью несбалансированного требует больше на 400 %

Сравнение сб. дерева и хэш-таблицы:

Разность производительности = 62.20 нс

Реализация с помощью дерева дольше на 85.361391 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью сбалансированного требует больше на 400 %

Длина строки - 310

Реализация с помощью дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 136.00 | 30 | 4.30 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью сбалансированного дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 129.15 | 20 | 3.94 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью хэш-таблицы:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 71.15 | 20 | 4.32 |

Занимаемая память - 1664 байт

Количество сравнений - 1

Сравнение дерева и сбалансированного дерева:

Разность производительности = 6.85 нс

Реализация с помощью несбалансированного дольше на 5.303910 %

Разность занимаемой памяти = 0 байт

Разность количества сравнений = 0

Сравнение несб. дерева и хэш-таблицы:

Разность производительности = 64.85 нс

Реализация с помощью дерева дольше на 91.145467 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью несбалансированного требует больше на 400 %

Сравнение сб. дерева и хэш-таблицы:

Разность производительности = 58.00 нс

Реализация с помощью дерева дольше на 81.517920 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью сбалансированного требует больше на 400 %

Длина строки - 410

Реализация с помощью дерева:

| | Время, нс | | Кол-во итераций | | RSE |
|--|-----------|--|-----------------|--|------|
| | 131.65 | | 20 | | 3.33 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью сбалансированного дерева:

| | Время, нс | | Кол-во итераций | | RSE |
|--|-----------|--|-----------------|--|------|
| | 133.87 | | 30 | | 3.69 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью хэш-таблицы:

| | Время, нс | | Кол-во итераций | | RSE |
|--|-----------|--|-----------------|--|-----|
|--|-----------|--|-----------------|--|-----|

| | | | | | |
|--|-------|--|----|--|------|
| | 69.87 | | 30 | | 4.72 |
|--|-------|--|----|--|------|

Занимаемая память - 1664 байт

Количество сравнений - 1

Сравнение дерева и сбалансированного дерева:

Разность производительности = 2.22 нс

Реализация с помощью сбалансированного дольше на 1.683757 %

Разность занимаемой памяти = 0 байт

Разность количества сравнений = 0

Сравнение несб. дерева и хэш-таблицы:

Разность производительности = 61.78 нс

Реализация с помощью дерева дольше на 88.430344 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью несбалансированного требует больше на 400 %

Сравнение сб. дерева и хэш-таблицы:

Разность производительности = 64.00 нс

Реализация с помощью дерева дольше на 91.603053 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью сбалансированного требует больше на 400 %

Длина строки - 510

Реализация с помощью дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 123.40 | 10 | 4.84 |

Занимаемая память - 1280 байт

Количество сравнений - 6

Реализация с помощью сбалансированного дерева:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 129.65 | 20 | 4.39 |

Занимаемая память - 1280 байт

Количество сравнений - 5

Реализация с помощью хэш-таблицы:

| Время, нс | Кол-во итераций | RSE |
|-----------|-----------------|------|
| 81.10 | 10 | 3.92 |

Занимаемая память - 1664 байт

Количество сравнений - 1

Сравнение дерева и сбалансированного дерева:

Разность производительности = 6.25 нс

Реализация с помощью сбалансированного дольше на 5.064830 %

Разность занимаемой памяти = 0 байт

Разность количества сравнений = 1

Реализация с помощью несбалансированного требует больше на 20 %

Сравнение несб. дерева и хэш-таблицы:

Разность производительности = 42.30 нс

Реализация с помощью дерева дольше на 52.157830 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 5

Реализация с помощью несбалансированного требует больше на 500 %

Сравнение сб. дерева и хэш-таблицы:

Разность производительности = 48.55 нс

Реализация с помощью дерева дольше на 59.864365 %

Разность занимаемой памяти = 384 байт

Реализация с помощью хэш-таблицы больше на 30 %

Разность количества сравнений = 4

Реализация с помощью сбалансированного требует больше на 400 %

На всех размерах самым быстрым способом поиска оказалась хэш-таблица, однако по памяти она всегда проигрывает. Также можно заметить, что сбалансированное дерево показало себя лучше по сравнению с несбалансированным.

Ответы на вопросы

1. Идеально сбалансированное дерево и AVL-дерево:

- Идеально сбалансированное дерево (Perfectly Balanced Tree): Каждый уровень дерева полностью заполнен узлами, и высота дерева минимальна. Такие деревья обеспечивают оптимальное время выполнения операций, но в практике редко встречаются из-за ограничений на количество элементов в дереве.

- AVL-дерево: Это форма сбалансированного дерева двоичного поиска, в котором разница в высоте между левым и правым поддеревьями для каждого узла ограничена (высота различается не более чем на 1). Это обеспечивает быстрое выполнение операций вставки, удаления и поиска.

2. Поиск в AVL-дереве и дереве двоичного поиска:

- В AVL-дереве поиск выполняется так же, как и в обычном дереве двоичного поиска. Разница заключается в том, что AVL-дерево поддерживает балансировку после каждой операции вставки или удаления, чтобы сохранять свою структуру сбалансированной.

3. Хеш-таблица и её принцип построения:

- Хеш-таблица это структура данных, позволяющая эффективно выполнять операции вставки, удаления и поиска. Она использует хеш-функцию для преобразования ключа в индекс массива, где хранятся значения. Принцип построения:

- Выбор хеш-функции.
- Выделение массива определенного размера.
- Разрешение коллизий (в случае, если два ключа хешируются в один и тот же индекс).

4. Коллизии и методы их устранения:

- Коллизии возникают, когда два различных ключа хешируются в один и тот же индекс. Методы разрешения коллизий включают:

- Цепочки: Каждый индекс массива представляет собой связанный список.
- Открытое хеширование: При коллизии производится поиск следующего свободного слота в массиве.
- Двойное хеширование: Используются две хеш-функции для определения следующего индекса при коллизии.

5. Неэффективность поиска в хеш-таблицах:

- Поиск в хеш-таблицах становится неэффективным при большом количестве коллизий, что может привести к увеличению длины цепочек или увеличению размера открытого адреса.

6. Эффективность поиска:

- В AVL-деревьях и деревьях двоичного поиска поиск выполняется за время, пропорциональное логарифму числа элементов в дереве.
- В хеш-таблицах, при эффективном хешировании, поиск может быть выполнен за постоянное время $O(1)$.
- В файлах эффективность поиска зависит от типа файла (например, последовательный доступ, индексированный доступ) и размера данных.

Вывод

В результате проделанной мною работы я убедился, что сбалансированные деревья и хэш-таблицы представляют собой эффективные средства для организации и управления данными. Проанализировав алгоритмы, я убедился, что использование балансировки деревьев позволяет значительно ускорить поиск необходимых данных, а при отсутствии большого количества коллизий хэш-таблицы показывают себя крайне эффективно.