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»

Лабораторная Работа №6 «Деревья»

Вариант №0

Студент **Поляков Андрей Игоревич**

Группа **ИУ7-32Б**

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

Студент **Поляков А.И.**

Проверяющий **Барышникова М.Ю.**

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. Вывести оставшиеся элементы дерева при постфиксном его обходе. Сравнить время удаления повторяющихся букв из дерева и из строки. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритма поиска в зависимости от высоты дерева.

Техническое задание

Исходные данные:

Пункты меню, выраженные целыми числами 0-10, -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
8) Сравнить время удаления повторяющихся букв из дерева и из строки
9) Сравнить эффективность алгоритма поиска
10) Вывести информацию о программе
-1) Очистить дерево
0) Выход

Результат:

Дерево (в виде дерева и в строку), поддерево, дерево без повторяющихся символов

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, сравнить время удаления повторяющихся букв из дерева и из строки, сравнить эффективность алгоритма поиска.

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

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.
9. Команда сравнения времени удаления повторяющихся букв из дерева и из строки, генерирует случайные строки длиной от 10 до 910, а затем засекает время удаления символов из строки (путем записи в ту же строку только нужных символов) и преобразует строку в дерево и засекает время удаления из дерева. Все замеры производятся путем многочисленных запусков с ожиданием RSE < 5. Затем производится анализ затраченного времени и памяти.
10. Команда сравнения эффективности алгоритма поиска, производит аналогичные замеры, но только для дерева и используя функцию поиска элемента вместо функции удаления повторяющихся символов.

Тестовые данные

Позитивные тесты

Тест	Входные данные	Результат
Добавление узла в дерево	Узел b Дерево .——— n ——— c `——— a	.——— n ——— c | .——— b `——— a
Удаление узла из дерева	Удалить a из: .——— n ——— c `——— a	.——— n ——— c
Создание дерева из строки	Строка acn	.——— n ——— c `——— a
Вывод дерева в строку (Постфиксный обход)	.——— n ——— c `——— a	a c n
Вывод дерева в виде дерева	Существующее дерево из acn	.——— n ——— c `——— a
Вывод дерева с повторяющимися буквами	Существующее дерево из bcacfgb	.——— g .——— f | `——— c ——— b `——— a
Поиск узла по символу и вывод поддерева	Существующее дерево .——— n ——— c `——— a символ a для поиска	Вывод поддерева, начинающегося с найденного узла. ——— a
Удаление	Строка с повторяющимися	Строка без повторяющихся

повторяющихся букв	буквами bcacfgb	букв. a f g
Сравнение времени удаления повторяющихся букв из дерева и строки:	-	Результаты сравнения времени выполнения операции удаления повторяющихся букв из дерева и строки.
Сравнение эффективности алгоритма поиска	-	Результаты сравнения времени выполнения алгоритма поиска в дереве.

Негативные тесты

Удаление несуществующего узла из дерева	.——— g .——— f | `——— c ——— b `——— a узел h	Сообщение об ошибке и отсутствие изменений в дереве.
Поиск несуществующего символа и вывод поддерева	.——— g .——— f | `——— c ——— b `——— a узел h	Сообщение об ошибке
Пустая строка для преобразования	Пустая строка	Сообщение об ошибке
Попытка вывода пустого дерева	Пустое дерево	Сообщение об ошибке

Замеры

Программа сравнивает время удаления повторяющихся букв из дерева и строки. Из строки символы удаляются путем записи в начало строки только нужных символов и отсечения лишнего. Программа создает случайные

строки длинами от 10 до 910 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 строки:

Время, нс	Кол-во итераций	RSE
210.45	130	4.96

Занимаемая память - 10 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
259.30	50	4.34

Занимаемая память - 256 байт

Разность производительности = 48.85 нс
Реализация с помощью дерева дольше на 23.214416 %

Разность занимаемой памяти = 246 байт
Реализация с помощью дерева больше на 2460 %

Длина строки - 110

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
8457.08	50	4.37

Занимаемая память - 110 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
2270.80	20	4.43

Занимаемая память - 1696 байт

Разность производительности = 6186.28 нс
Реализация с помощью строки дольше на 272.427338 %

Разность занимаемой памяти = 1586 байт
Реализация с помощью дерева больше на 1441 %

Длина строки - 210

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
17479.70	10	0.63

Занимаемая память - 210 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
1618.92	80	4.95

Занимаемая память - 1920 байт

Разность производительности = 15860.78 нс
Реализация с помощью строки дольше на 979.710302 %

Разность занимаемой памяти = 1710 байт
Реализация с помощью дерева больше на 814 %

Длина строки - 310

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
28132.60	10	0.49

Занимаемая память - 310 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
2570.70	10	3.84

Занимаемая память - 1984 байт

Разность производительности = 25561.90 нс
Реализация с помощью строки дольше на 994.355623 %

Разность занимаемой памяти = 1674 байт
Реализация с помощью дерева больше на 540 %

Длина строки - 410

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
39322.70	10	1.91

Занимаемая память - 410 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
2356.50	10	1.53

Занимаемая память - 1984 байт

Разность производительности = 36966.20 нс
Реализация с помощью строки дольше на 1568.690855 %

Разность занимаемой памяти = 1574 байт
Реализация с помощью дерева больше на 383 %

Длина строки - 510

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
20681.11	90	4.63

Занимаемая память - 510 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
872.90	10	1.91

Занимаемая память - 1984 байт

Разность производительности = 19808.21 нс
Реализация с помощью строки дольше на 2269.241736 %

Разность занимаемой памяти = 1474 байт
Реализация с помощью дерева больше на 289 %

Длина строки - 610

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE

21492.20	10	0.18

Занимаемая память - 610 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
861.50	10	1.76

Занимаемая память - 1984 байт

Разность производительности = 20630.70 нс
Реализация с помощью строки дольше на 2394.741730 %

Разность занимаемой памяти = 1374 байт
Реализация с помощью дерева больше на 225 %

Длина строки - 710

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
24893.70	10	0.20

Занимаемая память - 710 байт
Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
888.60	10	1.46

Занимаемая память - 1984 байт

Разность производительности = 24005.10 нс
Реализация с помощью строки дольше на 2701.451722 %

Разность занимаемой памяти = 1274 байт
Реализация с помощью дерева больше на 179 %

Длина строки - 810

Реализация с помощью строки:

Время, нс	Кол-во итераций	RSE
28603.60	10	0.18

Занимаемая память - 810 байт
Реализация с помощью дерева:

```
|  Время, нс   | Кол-во итераций | RSE
|      877.70 |              10 | 1.00
```

Занимаемая память - 1984 байт

Разность производительности = 27725.90 нс
Реализация с помощью строки дольше на 3158.926740 %

Разность занимаемой памяти = 1174 байт
Реализация с помощью дерева больше на 144 %

Длина строки - 910

Реализация с помощью строки:

```
|  Время, нс   | Кол-во итераций | RSE
|    32238.40 |              10 | 0.66
```

Занимаемая память - 910 байт
Реализация с помощью дерева:

```
|  Время, нс   | Кол-во итераций | RSE
|      858.70 |              10 | 2.64
```

Занимаемая память - 1984 байт

Разность производительности = 31379.70 нс
Реализация с помощью строки дольше на 3654.326307 %

Разность занимаемой памяти = 1074 байт
Реализация с помощью дерева больше на 118 %

По результатам замеров можно сделать вывод, что реализация с помощью строки по скорости выигрывает лишь на строках совсем небольшого размера, однако по памяти дерево в большинстве случаев проигрывает.

Также программа сравнивает время поиска элемента в дереве в зависимости от глубины структуры. Аналогично первому набору тестов программа создает случайные строки длинами от 10 до 910 с шагом 100 и проводит замеры для каждого из размеров и 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
40.17	100	2.51

Занимаемая память - 288 байт

Длина строки - 110

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
56.77	100	3.77

Занимаемая память - 1280 байт

Длина строки - 210

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
103.27	100	2.24

Занимаемая память - 1280 байт

Длина строки - 310

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
98.94	100	1.98

Занимаемая память - 1280 байт

Длина строки - 410

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
96.06	100	1.65

Занимаемая память - 1280 байт

Длина строки - 510

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
99.95	100	2.06

Занимаемая память - 1280 байт

Длина строки - 610

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
98.81	100	2.18

Занимаемая память - 1280 байт

Длина строки - 710

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
99.45	100	2.35

Занимаемая память - 1280 байт

Длина строки - 810

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE
100.41	100	2.05

Занимаемая память - 1280 байт

Длина строки - 910

Реализация с помощью дерева:

Время, нс	Кол-во итераций	RSE

99.79	100	1.77

Занимаемая память - 1280 байт

Можно заметить, что скорость и размер становятся стабильными после примерно 200-300 символов. Это связано с тем, что тип char ограничен 256 символами, а значит максимальное число узлов дерева – 256.

Ответы на вопросы

1. **Что такое дерево? Как выделяется память под представление деревьев?**

Дерево в информатике - это структура данных, состоящая из узлов, связанных между собой рёбрами. Один из узлов называется корнем, остальные разделяются на узлы и листья. Узлы, соединенные ребрами, образуют поддеревья. Память под представление деревьев обычно выделяется динамически. Каждый узел дерева содержит информацию и указатели на своих потомков (или нулевые указатели, если потомков нет). Для каждого узла память выделяется отдельно при добавлении новых узлов.

2. **Какие бывают типы деревьев?**

Существует множество типов деревьев, вот некоторые из них:

- Дерево двоичное: Каждый узел имеет не более двух потомков.
- Дерево двоичного поиска: Узлы упорядочены так, что для каждого узла все узлы в его левом поддереве меньше его, а в правом — больше.
- N-арное дерево: Каждый узел может иметь произвольное количество потомков.
- Распределенное дерево: Используется в распределенных вычислениях и сетевых структурах.
- AVL-дерево, красно-черное дерево: Сбалансированные бинарные деревья для эффективного поиска.

3. **Какие стандартные операции возможны над деревьями?**

Стандартные операции над деревьями включают:

- Добавление узла: Вставка нового узла в дерево.
- Удаление узла: Удаление существующего узла из дерева.
- Поиск узла: Нахождение узла с определенным значением.
- Обход дерева: Посещение всех узлов дерева в определенном порядке (прямой, обратный, симметричный).
- Вывод дерева в виде строки: Представление дерева в текстовой или графической форме.
- Изменение данных узла: Обновление значений в существующем узле.

4. **Что такое дерево двоичного поиска?**

Дерево двоичного поиска - это бинарное дерево, в котором каждый узел имеет не более двух потомков. При этом для каждого узла выполнено следующее свойство: все узлы в левом поддереве меньше текущего узла, а

все узлы в правом поддереве больше текущего узла. Это свойство делает дерево двоичного поиска эффективной структурой данных для поиска, вставки и удаления элементов, так как оно обеспечивает логарифмическую сложность этих операций в среднем случае.

Вывод

В результате проделанной мною работы я убедился, что деревья представляют собой эффективные средства для организации и управления данными. На примере бинарного дерева символов я изучил способы реализации деревьев. Проанализировав алгоритмы, я убедился, что использование деревьев позволяет значительно ускорить поиск необходимых данных. Так, использовать мой алгоритм поиска и удаления повторяющихся символов разумно с большими объемами данных, при условии что чуть большие затраты по памяти не критичны. Анализируя алгоритм поиска элементов, я убедился, что в случае дерева символов он работает достаточно быстро при любой глубине дерева.