



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 СТРУКТУРЫ ДАННЫХ
ОБРАБОТКА ДЕРЕВЬЕВ**

Студент **Попов Ю.А.**

Группа **ИУ7-32Б**

Вариант 0

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

Студент _____ **Попов Ю.А.**

Преподаватель _____ **Барышникова М.Ю.**

Оглавление

Оглавление	2
Описание условия задачи	3
Описание технического задания	3
Входные данные:	3
Выходные данные:	4
Действие программы:	4
Обращение к программе:	5
Аварийные ситуации:	5
Описание структуры данных	5
Описание алгоритма	6
Описание основных функций	6
Набор тестов	8
Оценка эффективности	11
Объем занимаемой памяти (байты)	11
Временные замеры (нс)	12
Ответы на контрольные вопросы	14
1. Что такое дерево?	14
2. Как выделяется память под представление деревьев	14
3. Какие бывают типы деревьев?	15
4. Какие стандартные операции возможны над деревьями?	15
5. Что такое дерево двоичного поиска?	15
Вывод	15

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.

Описание технического задания

Входные данные:

При работе с программой на экран выводится меню:

- 0 - Выход
- 1 - Тестирование дерева
- 2 - Ввести строку для обработки
- 3 - Построить дерево
- 4 - Вывести дерево (картинкой)
- 5 - Очистить дерево от повторяющихся символов
- 6 - Очистить строку от повторяющихся символов
- 7 - Префиксный обход
- 8 - Инфиксный обход
- 9 - Постфиксный обход

Пользователь вводит целое число-команду от 0 до 9. Программа выполняет операции, в зависимости от выбранной опции. Если пользователь выбрал первую или вторую операцию, то пользователь попадает в подпрограмму для тестирования, где должен ввести номер операции (от 1 до 7) и элемент, один символ, для вставки в случае, если выбрана операция “добавления элемента дерева” или “поиск элемента в дереве”. Выбрав вторую команду, пользователь должен ввести строку, с которой, в

дальнейшем будет работать программа. Остальные операции не требуют входных данных. Чтобы корректно завершить программу, необходимо воспользоваться соответствующей командой.

Выходные данные:

Если пользователь выбрал 3 или 4 операцию, то он получит изображение, построенного дерева. Если выбрана 5 операция, то на экран будет выведено изображение дерева только из элементов, которые не повторялись, а также время работы алгоритма. При выборе 6 операции, будет выведена строка, состоящая из элементов, которые не повторялись, а также время работы алгоритма. При выборе 7, 8 и 9 операции, на экран будут выведены элементы дерева, в порядке выбранного обхода.

При тестировании дерева, в зависимости от операции на экран выводится либо дерево, либо его удаленный элемент.

Действие программы:

Программа работает до тех пор, пока пользователь не введет число 0. Программное обеспечение позволяет ввести команды от 0 до 9, в случае если команда выходит за этот диапазон, или не подходит по типу данных, выводится соответствующее сообщение, и программа завершает работу.

Пользователю доступны следующие операции при тестировании дерева:

- 1) Вывод дерева на экран (картинкой)
- 2) Добавить элемент в дерево
- 3) Удалить элемент дерева
- 4) Поиск элемента в дереве
- 5) Префиксный обход дерева
- 6) Инфиксный обход дерева

7) Постфиксный обход дерева

Обращение к программе:

Программа запускается через терминал, в командной строке, с помощью команды `./app.exe`. После запуска, пользователю будет доступно меню программы.

Аварийные ситуации:

1. `ERR_STRING` – Ошибка при вводе строки;
2. `ERR_OPERATION` – Неверный выбор операции;
3. `ERR_MEMORY_ALLOCATION` - Ошибка при выделении памяти;
4. `ERR_DATA_INPUT` – Ошибка при вводе данных;
5. `ERR_FILE` - Неверный файл для построения графика;
6. `WARNING_NO_EL` - Нет нужного элемента для поиска или удаления

Во всех нештатных ситуациях, программа уведомляет о месте, где произошла ошибка.

Описание структуры данных

```
typedef char data_t;
```

В качестве данных, которые будет хранить дерево был выбран символ. Это обусловлено условием задачи.

Структура для хранения одного элемента дерева представлена на листинге 1.

data - данные

is_repeated - Флаг, который показывает повторяется элемент или нет

is_search - Флаг для того, чтобы выделить элемент цветом при поиске

left - Указатель на левого потомка

right - Указатель на правого потомка

```
typedef struct _tree_t_
{
    data_t data;
    int is_repeated; // 0 - not repeat; 1 - repeat
    int is_search;    // 1 - флаг активируется, после поиска
    struct _tree_t_ *left;
    struct _tree_t_ *right;
} tree_t;
```

листинг 1. Структура tree_t

Описание алгоритма

1. После запуска программы выводит приглашение к вводу и меню.
2. Программа принимает номер операции, и запускает соответствующие действия в соответствии с номером команды
3. При добавлении элемента в дерево рекурсивно ищется место, в которое следует вставить элемент.
4. При поиске элемента, рекурсивно ищется тот элемент, который совпадает по указанному свойству.
5. При удалении элемента из дерева, 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, после чего он удаляется.
6. Если пользователь вводит 0, то программа завершается.

Описание основных функций

```
tree_t *tree_create_node(data_t data);
```

Создание нового элемента дерева

Входные параметры: данные

Выходные параметры: указатель на новый элемент

```
int tree_insert(tree_t **root, data_t data);
```

Вставка элемента в дерево

Входные параметры: указатель на дерево, данные

Выходные параметры: измененное дерево

Возвращаемый результат: код возврата.

```
tree_t *tree_search(tree_t *root, data_t data);
```

Поиск элемента в дереве

Входные параметры: указатель на дерево, данные для поиска

Выходные параметры: найденный элемент дерева

Возвращаемый результат: найденный элемент дерева

```
int tree_remove(tree_t **root, data_t data);
```

Удаление элемента из дерева

Входные параметры: указатель на дерево, данные для поиска нужного элемента

Выходные параметры: дерево без удаленного элемента

Возвращаемый результат: код возврата.

```
int tree_in_picture(tree_t *tree);
```

Функция реализует вывод на экран дерева, с помощью утилиты graphviz

Входные параметры: указатель на дерево

Выходные параметры: изображение дерева

Возвращаемый результат: код возврата.

```
operations_t input_operation(void);
```

Функция обеспечивает получение операции от пользователя

Выходные параметры: операция, введенная пользователем

Возвращаемый результат: операция, введенная пользователем

```
void tree_delete_repeat(tree_t **tree);
```

Функция удаляет повторы из дерева

Входные параметры: указатель на дерево

Выходные параметры: дерево, состоящее из элементов, которые входили в исходную строку только 1 раз.

```
void string_remove_duplicates(char *string);
```

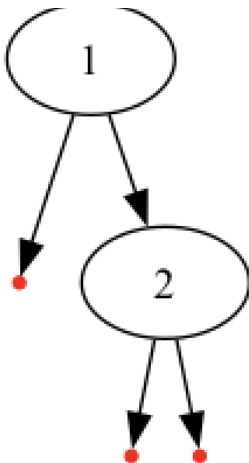
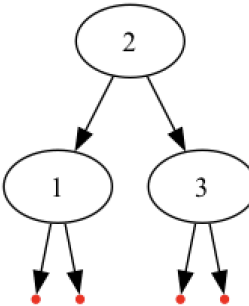
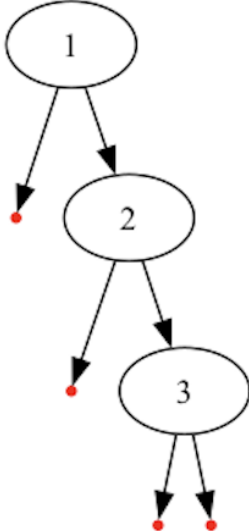
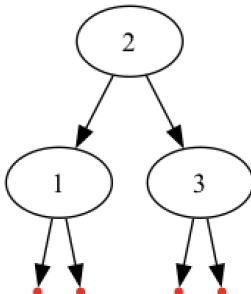
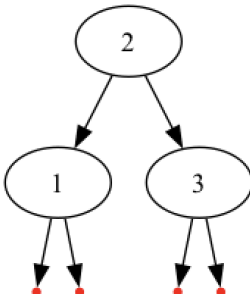
Функция оставляет в строке символы, которые входили в нее только 1 раз

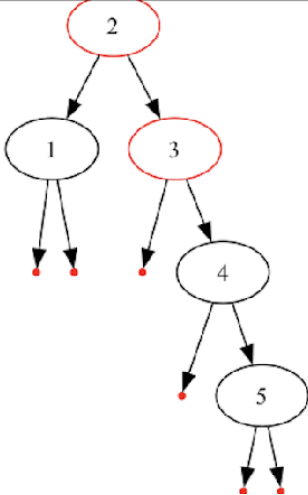
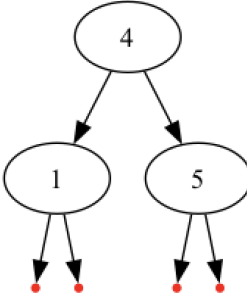
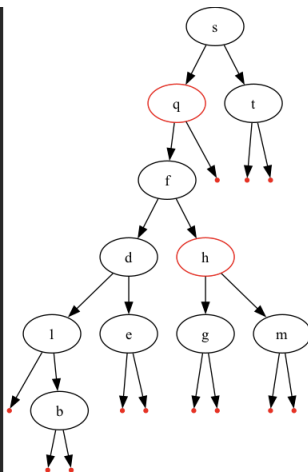
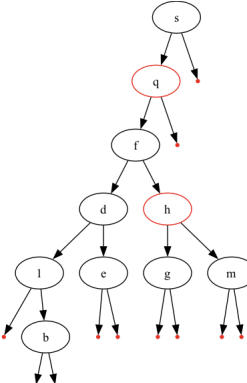
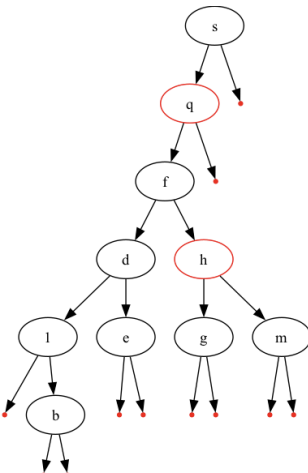
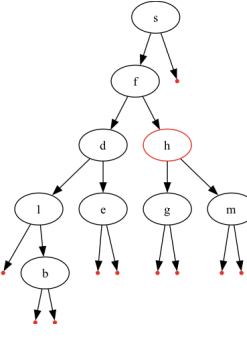
Входные параметры: указатель на массив чаров

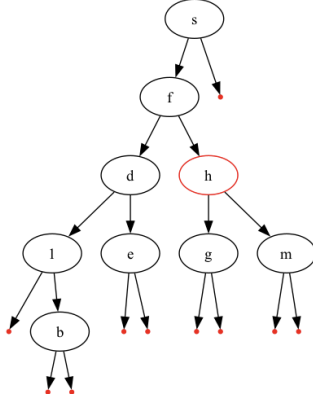
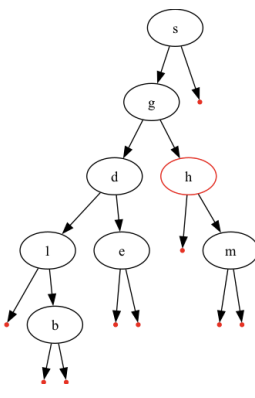
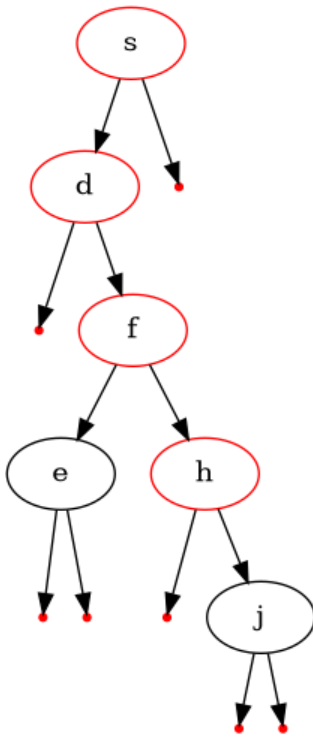
Выходные параметры: измененная строка

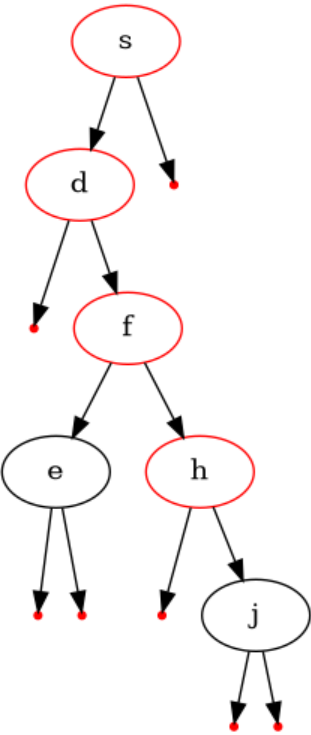
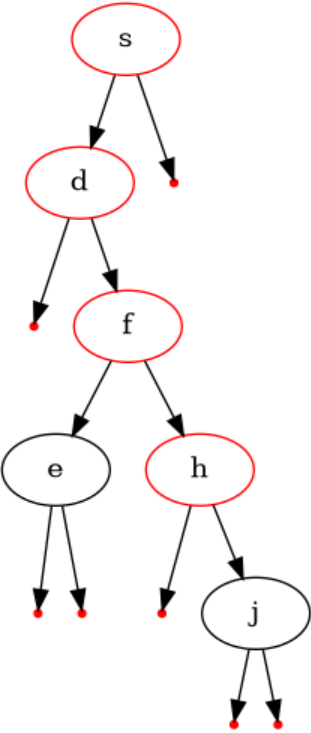
Набор тестов

№	Описание теста	Ввод	Вывод
1	Некорректный ввод операции	abc	Ошибка при выборе операции.
2	Некорректный ввод операции. Выход за диапазон	10000	Ошибка при выборе операции.
3	Некорректный ввод операции. Отрицательное число	-100	Ошибка при выборе операции
6	Пустая строка		Ошибка при вводе строки
7	Неверный ввод данных для дерева. Ввод строки	aaa	Ошибка, неверный ввод элемента
8	Добавление одного элемента в дерево	1	

9	Добавление одного потомка	1 2	
10	Добавление двух потомков (в виде дерева)	2 1 3	
11	Добавление двух потомков (в виде списка)	1 2 3	
12	Удаление дубликатов (нет дубликатов)	 	

13	Удаление дубликатов (символы дубликаты из исходной строки)		
14	Удаление элемента без потомков		
15	Удаление элемента с одним потомком		

16	Удаление элемента с двумя потомками		
17	Префиксный обход		s d f e h j

18	Инфиксный обход		d e f h j s
19	Постфиксный обход		e j h f d s

Оценка эффективности

Объем занимаемой памяти (байты)

Объем, занимаемой памяти деревом в зависимости от его размера

Количество элементов	Память, занимаемая деревом (байт)	Память, занимаемая строкой из уникальных элементов (байт)
1	32	2
10	320	11
50	1600	51
100	3200	101
1000	32000	1001

Таблица 1. Оценка занимаемой памяти, относительно размера.

Один элемент дерева занимает 32 байта, следовательно можно легко рассчитать размер всего дерева. В таблице 1 указан размер строки, если в нее входят только уникальные элементы. Однако, если в строке символы будут повторяться, то объем занимаемой памяти будет увеличиваться, так как он линейно зависит от количества символов в строке. Таким образом строка более эффективна, чем дерево, пока количество повторений каждого элемента не превосходит 32.

Временные замеры (нс)

Временные замеры осуществлялись с помощью библиотеки `time.h`. Ниже представлены таблицы времени работы алгоритма удаления элементов, которые встретились в строке более 1 раза. Из-за специфичности задачи нельзя провести сравнение производительности на большом объеме данных, потому что диапазон ограничен таблицей ASCII.

Таблица 2: удаление из сбалансированного дерева. По таблице видна корреляция. Чем больше размер, тем выгоднее становится использовать дерево.

Количество элементов	Удаление элементов из бинарного дерева (мкс)	Удаление элементов из строки (мкс)	Отношение времени удаления из бинарного дерева к строке
10	1.58	0.71	2.22
50	4.83	1.81	2.66
100	11.2	3.96	2.82
500	13.46	7.61	1.76
1000	9.79	10.53	0.92
1500	11.50	19.96	0.57

Таблица 2. Время выполнения алгоритма по удалению дубликатов из строки и дерева.

№	Среднее время добавления элемента в бинарное дерево (мкс)	Среднее время удаления элемента из бинарного дерева (мкс)	Среднее время поиска элемента в бинарном дереве (мкс)
1	1.01	0.94	0.69
2	0.96	0.85	0.58
3	0.94	0.79	0.70
4	1.05	0.81	0.59
5	0.92	0.70	0.51

Среднее	0.98	0.81	0.61
----------------	------	------	------

Таблица 4. Сравнение времени выполнения операций добавления и удаления элемента в дерево.

В таблице 5 представлено сравнение времени обхода дерева, в зависимости от высоты и ветвистости

Количество элементов	Сбалансированное дерево (мкс)	Дерево как линейный список (мкс)
7	3.47	4.75
15	4.57	4.94
40	9.30	9.47

Таблица 5. Сравнение времени обхода дерева, в зависимости от высоты и ветвистости.

Время поиска в сбалансированном дереве можно увидеть в таблице 6. Методика выбрана следующая: замеряется время поиска каждого элемента дерева, а после делится на количество элементов. Таким образом, получается среднее время поиска элементов в дереве.

Количество элементов	Сбалансированное дерево (мкс)	Дерево как линейный список (мкс)
7	0.05	0.07
15	0.06	0.10
22	0.08	0.22

Таблица 6. Сравнение времени поиска элемента дерева, в зависимости от высоты и ветвистости.

Использование бинарного дерева для хранения данных выгодно, когда в наборе данных нужно производить много операций поиска, так как поиск в бинарном дереве поиска происходит в среднем за $O(\log N)$. При этом вставка и удаление из дерева производятся также за $O(\log N)$, но в худшем случае все эти операции могут производиться за $O(N)$. Время выполнения сильно зависит от порядка вставки элементов в дерево. Если совмещать в одном алгоритме поиск и удаление элементов, как в задаче из варианта, то в худшем случае сложность будет $O(N^2)$, в то время как эффективный алгоритм для работы со строкой будет сложностью $O(M)$, где M - длина строки.

Ответы на 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

Память выделяется под каждый узел. В каждом узле есть указатель на 2 потомка. Один узел занимает 32 байта, следовательно можно рассчитать сколько занимает в памяти все дерево: $32 * N$, где N - число элементов дерева.

3. Какие бывают типы деревьев?

Бинарные деревья, бинарные деревья поиска, АВЛ-деревья, красно-черные деревья

4. Какие стандартные операции возможны над деревьями?

Поиск в дереве, добавление в дерево, удаление из дерева, обход дерева.

5. Что такое дерево двоичного поиска?

Дерево двоичного поиска - дерево, в котором все левые потомки меньше предка, а все правые - больше вершины.

Вывод

Двоичное дерево поиска выгодно использовать, когда требуется осуществлять много операция поиска. Поиск в дереве является очень эффективной операцией, но в первую очередь это зависит от степени сбалансированности дерева. В сбалансированном дереве поиск занимает $O(\log(N))$, что является одним из самых эффективных видов поиска.