



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» (ИУ)

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

«ОБРАБОТКА ДЕРЕВЬЕВ»

по курсу:

«ТИПЫ И СТРУКТУРЫ ДАННЫХ»

Вариант: 1

Студент:

Авдейкина Валерия Павловна, группа ИУ7-33Б

(подпись, дата)

Руководители:

Преподаватель ИУ7

Силантьева Александра Васильевна

(подпись, дата)

Преподаватель ИУ7

Барышникова Марина Юрьевна

(подпись, дата)

Проверяющий:

Барышникова Марина Юрьевна

(подпись, дата)

Оценка: _____

2022 г.

Оглавление

Описание условий задачи.....	3
Техническое задание.....	4
1 Входные данные.....	4
2 Выходные данные.....	5
3 Задача, реализуемая программой.....	6
4 Способ обращения к программе.....	6
5 Возможные аварийные ситуации и ошибки со стороны пользователя.....	6
Описание внутренних структур данных.....	7
Алгоритм.....	8
Тестирование.....	10
Оценка эффективности.....	16
Контрольные вопросы.....	18
1 Что такое дерево?.....	18
2 Как выделяется память под представление деревьев?.....	18
3 Какие бывают типы деревьев?.....	18
4 Какие стандартные операции возможны над деревьями?.....	18
5 Что такое дерево двоичного поиска?.....	18
Выводы.....	19

Описание условий задачи

Построить дерево в соответствии со своим вариантом задания. Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

Вариантное задание: В текстовом файле содержатся целые числа. Построить двоичное дерево из чисел файла. Вывести его на экран в виде дерева. Используя процедуру, определить количество узлов дерева на каждом уровне. Добавить число в дерево и в файл. Сравнить время добавления чисел в указанные структуры.

Техническое задание

1 Входные данные

- Входные данные при вводе номера выбранной опции меню:
 - Целые беззнаковые числа [0; 14]
- Входные данные для различных опций меню:

Номер опции	Название опции	Входные данные
0	Выход	-
1	Загрузить дерево из файла	Имя файла
2	Экспортировать дерево в изображение (dot)	-
3	Добавить узел в дерево / файл, сравнение времени	Метка узла, имя файла
4	Удалить узел из дерева	Метка узла
5	Выполнить поиск узла	Метка узла
6	Выполнить обход дерева	-
7	Отсортировать числа из файла с помощью дерева	Имя файла
8	Отсортировать числа из файла с помощью метода qsort	Имя файла
9	Вывести время разных вариантов обхода	-
10	Очистить дерево	-
11	Вывести меню	-
12	Вывести правила ввода	-
13	Определить кол-во узлов на каждом уровне дерева	-
14	Перевести текущий .dot файл в .png и открыть его	-

Таблица 1: Входные данные опций меню

- Имя файла: символы, не более 255
- Метка узла: целое число из [-2147483648; 2147483647]

2 Выходные данные

- Выходные данные при вводе номера выбранной опции меню:
 - Выходные данные команды в случае корректного ввода
- Выходные данные для различных опций меню:

Номер	Название опции	Выходные данные
0	Выход	Сообщение об успешном выполнении / сообщение об ошибке
1	Загрузить дерево из файла	Сообщение об успешном выполнении / сообщение об ошибке
2	Экспортировать дерево в изображение (dot)	Сообщение об успешном выполнении / сообщение об ошибке
3	Добавить узел в дерево / файл, сравнение времени	Сообщение об успешном выполнении, время выполнения / сообщение об ошибке
4	Удалить узел из дерева	Сообщение об успешном выполнении / сообщение об ошибке
5	Выполнить поиск узла	Информация об узле / сообщение о ее отсутствии
6	Выполнить обход дерева	Информация о узлах в порядке выполнения каждого из обходов / сообщение об ошибке
7	Отсортировать числа из файла с помощью дерева	Время выполнения сортировки + результат сортировки при кол-ве узлов < 100 / сообщение об ошибке
8	Отсортировать числа из файла с помощью метода qsort	Время выполнения сортировки + результат сортировки при кол-ве узлов < 100 / сообщение об ошибке
9	Вывести время разных вариантов обхода	Время выполнения каждого из обходов / сообщение об ошибке
10	Очистить дерево	Сообщение об успешном выполнении / сообщение об ошибке
11	Вывести меню	Меню

12	Вывести правила ввода	Правила ввода
13	Определить кол-во узлов на каждом уровне дерева	Информация о кол-ве узлов на каждом уровне дерева / сообщение об ошибке
14	Перевести текущий .dot файл в .png и открыть его	Системные сообщения
15	Получить высоту загруженного дерева	Высота дерева

Таблица 2: Выходные данные опций меню

- Формат вывода информации об узле (метка потомка равна метке узла, если потомок отсутствует):
 - метка_узла: метка_левого_потомка, метка_правого_потомка

3 Задача, реализуемая программой

Программа реализует обработку набора целых чисел с помощью составления двоичного дерева поиска и оценивает эффективность разных способов обработки.

4 Способ обращения к программе

Программа вызывается в командной строке без каких-либо аргументов. Ввод данных производится с клавиатуры после соответствующего приглашения к вводу («Команда . . .» / «Введите . . .»).

5 Возможные аварийные ситуации и ошибки со стороны пользователя

1. Некорректный ввод имени файла:

1.1. Пустой ввод

1.2. Недопустимые символы

1.3. Превышение допустимого количества символов (*поведение программы не определено*)

1.4. Имя несуществующего файла или директории

2. Некорректные данные в обрабатываемом файле / вводе с клавиатуры

3. Некорректный ввод номера опции:

3.1. Пустой ввод

3.2. Недопустимые символы (*поведение программы не определено*)

3.3. Недопустимый номер

Описание внутренних структур данных

В ходе работы была составлена следующая структура:

1. Дерево двоичного поиска — АДД, тип которого — tree_t, указатель на структуру tree_node:
 - struct tree_node:
 - key (метка узла) — поле типа int (key_t)
 - left (левый потомок) — поле типа tree_t (struct tree_node*)
 - right (правый потомок) — поле типа tree_t (struct tree_node*)

```
typedef struct tree_node* tree_t;
typedef int key_t;

struct tree_node
{
    key_t key;
    tree_t left;
    tree_t right;
};
```

Итоговый размер структуры: 20 байт.

Алгоритм

1. Сравнение различных вариантов обхода, сортировок

В ходе работы будут сравниваться: разные варианты обхода двоичного дерева поиска (*префиксный, инфиксный, постфиксный*) и сортировка исходного набора чисел с помощью двоичного дерева поиска и сортировки qsort (stdlib.h).

При сортировке с помощью двоичного дерева поиска набор чисел записывается в заново созданное дерево, а затем при инфиксном обходе происходит запись каждой метки узла дерева в одномерный массив для последующей обработки.

2. Работа с деревом:

- Создание узла:

Динамическое выделение памяти под новую структуру и ее обнуление

- Включение узла в дерево:

- Если очередной узел пуст: замещение места добавляемым узлом
- Если очередной узел не пуст: рекурсивное добавление нового узла к левому / правому потомку рассматриваемого

- Исключение узла из дерева (по метке):

- Если очередной узел пуст: возвращаем его
- Если очередной узел не пуст и метка не равна удаляемой:
 - ищем узел по дереву далее
- Если метка равна удаляемой:
 - при отсутствии потомков весь узел удаляется
 - при наличии только левого (правого) потомка «переставляем» его к родителю удаляемого узла
 - при наличии обоих потомков находим узел с наименьшей меткой в правых потомках и заменяем текущий узел на него

- Поиск узла (по метке):

Рекурсивно ищем узел сначала по левой ветке от корня, если не был найден — продолжаем по правой.

В реализации решения задачи были использованы следующие основные функции:

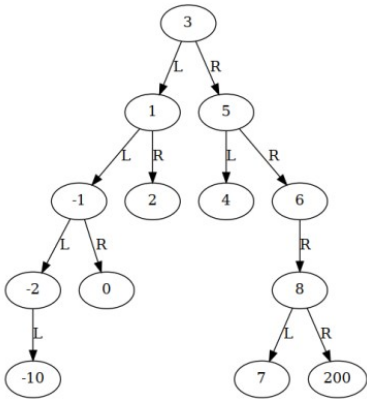
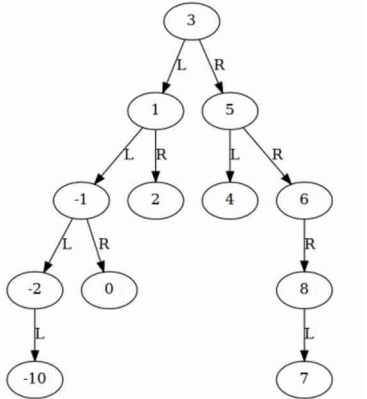
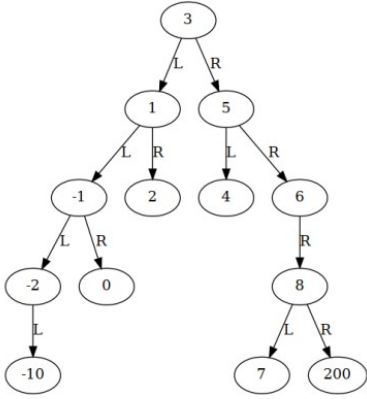
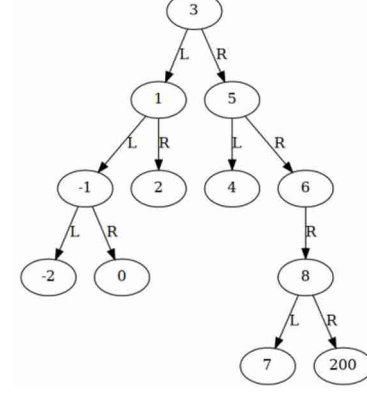
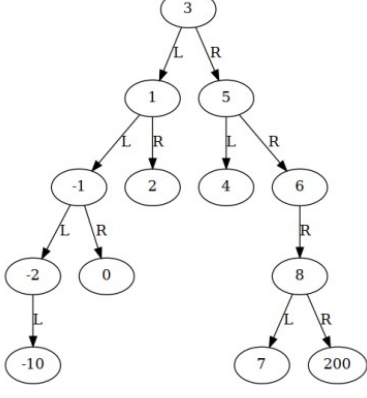
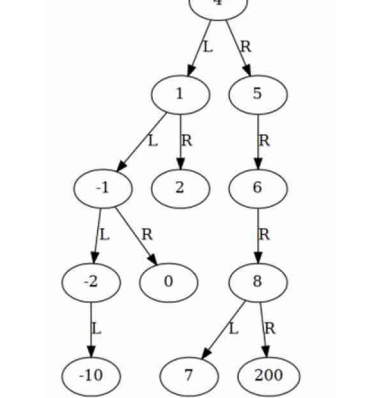
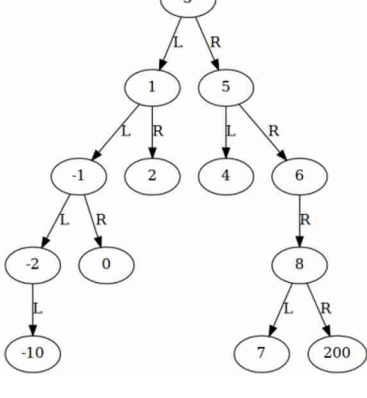
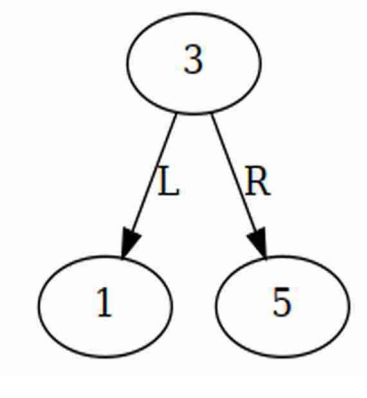
Функция	Описание
<code>tree_t create_node(const key_t key)</code>	Создает и возвращает новый узел дерева с соответствующим значением key В случае неудачи возвращает NULL
<code>tree_t insert_node(tree_t tree, tree_t node)</code>	Вставляет созданный узел node в дерево tree, возвращает указатель на новый корень
<code>tree_t delete_node(tree_t tree, const key_t key)</code>	Удаляет узел со значением key из дерева tree, возвращает указатель на новый корень
<code>void delete_tree(tree_t *tree)</code>	Удаляет дерево (освобождает память)
<code>tree_t search_node(tree_t tree, const key_t key)</code>	Выполняет поиск узла со значением key в дереве tree и возвращает его В случае неудачи возвращает NULL
<code>void apply_tree(const int code, tree_t tree, void (*f)(tree_t, void*), void *arg)</code>	Выполняет обход дерева в зависимости от кода code (PRE, IN, POST), применяя функцию f к узлам дерева tree в порядке обхода
<code>void node_to_dot(tree_t node, void *param)</code>	Выводит информацию об узле node в файловую переменную param в формате .dot
<code>void store_sorted(tree_t tree, int *data, int *const i)</code>	Рекурсивно переносит данные из дерева tree в массив data с помощью инфиксного обхода
<code>size_t get_nodes_count(tree_t tree, int level)</code>	Выполняет подсчет количества узлов дерева tree на уровне level
<code>void get_all_count(tree_t tree, size_t *const count)</code>	Выполняет подсчет общего количества узлов дерева tree
<code>int get_height(tree_t tree)</code>	Поиск высоты дерева tree
<code>void print_node_lookup(tree_t node, void *param)</code>	Выводит информацию об узле node в формате команды обхода

Таблица 3. Описание основных функций

Тестирование

- Позитивные тесты

Номер	Описание теста	Исходное дерево	Дерево после выполнения команды / Результат вып.
1.	Заполнение дерева из файла (команда 1)	Пусто	
2.	Добавление узла в правое поддерево (команда 3)		
3.	Добавление узла в левое поддерево (команда 3)		
4.	Добавление узла в пустое дерево (команда 3)	Пусто	

5.	Удаление листа в правом поддереве (команда 4)		
6.	Удаление листа в левом поддереве (команда 4)		
7.	Удаление корня (команда 4)		
8.	Поиск корня (команда 5)		
9.	Поиск листа в правом поддереве (команда 5)		

10.	Поиск листа в левом поддереве (команда 5)		
11.	Поиск узла в середине (команда 5)		
12.	Обход дерева из одного узла (команда 6)		<p>Высота дерева: 0 Префиксный обход: 100 Инфиксный обход: 100 Постфиксный обход: 100</p>
13.	Обход дерева с одной веткой (команда 6)		<p>Высота дерева: 9 Префиксный обход: 1 2 3 4 5 6 7 8 9 10 Инфиксный обход: 1 2 3 4 5 6 7 8 9 10 Постфиксный обход: 10 9 8 7 6 5 4 3 2 1</p>
14.	Обход дерева с		

	двумя ветвями (команда 6)		Высота дерева: 4 Префиксный обход: 3 1 -1 -2 -10 0 2 5 4 6 8 7 200 Инфиксный обход: -10 -2 -1 0 1 2 3 4 5 6 7 8 200 Постфиксный обход: -10 -2 0 -1 2 1 4 7 200 8 6 5 3
15.	Определить кол-во узлов на каждом уровне дерева из одного узла (команда 13)	1 узел	Команда (11 - меню, 12 - правила): 13 Уровень 0: 1 узл.
16.	Определить кол-во узлов на каждом уровне дерева с одной веткой (команда 13)	10 узлов	Уровень 0: 1 узл. Уровень 1: 1 узл. Уровень 2: 1 узл. Уровень 3: 1 узл. Уровень 4: 1 узл. Уровень 5: 1 узл. Уровень 6: 1 узл. Уровень 7: 1 узл. Уровень 8: 1 узл. Уровень 9: 1 узл.
17.	Определить кол-во узлов на каждом уровне дерева с двумя ветвями (команда 13)	10 узлов	Уровень 0: 1 узл. Уровень 1: 2 узл. Уровень 2: 2 узл. Уровень 3: 2 узл. Уровень 4: 1 узл. Уровень 5: 2 узл.
18.	Определить высоту дерева из одного узла (команда 15)	1 узел	0
19.	Определить высоту дерева с одной ветвью	1000 узлов	999
20.	Определить высоту дерева с		4

	двумя ветвями	<pre> graph TD 3((3)) -- L --> 1((1)) 3 -- R --> 5((5)) 1 -- L --> m1((-1)) 1 -- R --> 2((2)) m1 -- L --> m2((-2)) m1 -- R --> 0((0)) m2 -- L --> m3((-10)) 5 -- L --> 4((4)) 5 -- R --> 6((6)) 6 -- R --> 8((8)) 8 -- L --> 7((7)) 8 -- R --> 200((200)) </pre>	
--	---------------	--	--

Таблица 4. Позитивные тесты

- Негативные тесты (вызов при пустом дереве / некорректный ввод)

- Ввод команды

Команда (11 – меню, 12 – правила): вфыыв
 Ошибка: Неверный формат данных

Команда (11 – меню, 12 – правила): 123
 Ошибка: Неверный формат данных

- Команда 1

Команда (11 – меню, 12 – правила): 1
 Введите имя файла: аввыа
 Ошибка: Невозможно открыть файл

Команда (11 – меню, 12 – правила): 1
 Введите имя файла: 000000
 Ошибка: Невозможно открыть файл

- Команда 2

Команда (11 – меню, 12 – правила): 10
 Дерево было успешно очищено

Команда (11 – меню, 12 – правила): 2
 Ошибка: Дерево пусто

- Команда 3

Команда (11 – меню, 12 – правила): 3
 Введите узел: авав
 Ошибка: Неверный формат данных

Команда (11 – меню, 12 – правила): 3
 Введите имя файла: 20.txt
 Введите узел: 3
 Узел уже присутствует в дереве

- Команда 4

Команда (11 – меню, 12 – правила): 10
 Дерево было успешно очищено

Команда (11 – меню, 12 – правила): 4
 Ошибка: Дерево пусто

- Команда 5

Команда (11 – меню, 12 – правила): 5
 Ошибка: Дерево пусто

Команда (11 – меню, 12 – правила): 1
 Введите имя файла: 20.txt
 Дерево было успешно заполнено

Команда (11 – меню, 12 – правила): 5
 Введите узел: sdfsd
 Ошибка: Неверный формат данных

○ Команда 6

Команда (11 - меню, 12 - правила): 10
Дерево было успешно очищено

Команда (11 - меню, 12 - правила): 6
Ошибка: Дерево пусто

○ Команда 7

Команда (11 - меню, 12 - правила): 7
Введите имя файла: fafaf
Ошибка: Невозможно открыть файл

○ Команда 8

Команда (11 - меню, 12 - правила): 8
Введите имя файла: sddsfdsfsd
Ошибка: Невозможно открыть файл

○ Команда 9

Команда (11 - меню, 12 - правила): 9
Ошибка: Дерево пусто

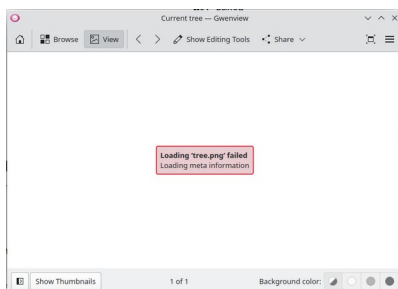
○ Команда 10

Команда (11 - меню, 12 - правила): 10
Ошибка: Дерево пусто

○ Команда 13

Команда (11 - меню, 12 - правила): 13
Ошибка: Дерево пусто

○ Команда 14



Оценка эффективности

Время замерялось в двух плоскостях: во время выполнения обхода дерева (в зависимости от метода) и во время сортировки набора целых чисел (сортировка деревом и сортировка qsort). Количество итераций для всех плоскостей: 200.

Метод	Кол-во чисел	Высота дерева		Среднее время, мкс		Выигрыш относительно сортировки qsort	
		Sorted	Unsorted	Sorted	Unsorted	Sorted	Unsorted
Сортировка двоичным деревом поиска	10	9	5	3	3	-50,0%	-50,0%
	500	499	17	406	83	-612,3%	-9,2%
	1000	999	20	1618	174	-1294,8%	0,5%
	2000	1999	21	6483	375	-2546,1%	0,8%
qsort (stdlib.h)	10	-		2	2	-	
	500	-		57	76	-	
	1000	-		116	175	-	
	2000	-		245	378	-	

Таблица 5. Оценка эффективности различных методов сортировки

Исходя из таблицы 5, можно сделать следующие выводы:

- Сортировка с помощью двоичного дерева поиска является менее эффективной при отсортированных данных, то есть, когда высота дерева становится достаточно большой
- Сортировка с помощью двоичного дерева поиска более эффективна при «разбросанных» данных, то есть, когда высота дерева намного меньше, чем общее количество чисел
- В силу особенностей работы процессора и малых единиц измерения появляются большие погрешности, ввиду которых эффективность сортировки деревом относительно быстрой сортировки (qsort) не является явной

Количество чисел	Время префиксного обхода, мкс		Время инфиксного обхода, мкс		Время постфиксного обхода, мкс	
	Sorted	Unsorted	Sorted	Unsorted	Sorted	Unsorted
10	6	6	6	9	6	6
500	319	354	320	337	315	334
1000	631	661	631	659	630	657
2000	1292	1193	1300	1993	1288	1163

Таблица 6. Сравнение времени различных методов обхода

Исходя из таблицы 6 можно сделать два вывода:

- Среднее время обхода не зависит от его метода
- Время обхода в среднем уменьшается, если данные заполняются в дерево в отсортированном виде

Затраты по памяти:

- элемент дерева — 20 байт
- элемент массива — 4 байта
- память, занимаемая деревом, больше, чем память, занимаемая одномерным массивом, на **400%**

Контрольные вопросы

1 Что такое дерево?

Дерево — нелинейная структура данных, используемая при представлении иерархических связей, имеющих отношения «один ко многим»;

2 Как выделяется память под представление деревьев?

Память под представление деревьев может выделяться несколькими способами: $P(n) = 3 \cdot n$ — для списочного представления (структуры с указателями на левого и правого потомков), $P(n) = 2 \cdot n$ — для представления в виде массива (массив структур с информацией об узле и индексах потомков), $P(n) = 2 \cdot n$ или $P(n) = n$ — для польской записи, где n — количество узлов дерева.

3 Какие бывают типы деревьев?

Существует много типов деревьев, перечислим основные: двоичное, красно-черное, АВЛ-дерево, расширяющееся (косое дерево) и т. д.

4 Какие стандартные операции возможны над деревьями?

Над деревьями возможны следующие операции: вставка узла, удаление узла, поиск узла, обход узлов.

5 Что такое дерево двоичного поиска?

Дерево двоичного поиска — двоичное дерево, каждая вершина которого имеет значение, которое больше, чем содержание любой из вершин его левого поддерева и меньше, чем содержание любой из вершин его правого поддерева.

Выводы

В ходе работы было выяснено, что использование дерева двоичного поиска является более эффективным по времени для обработки неотсортированных данных по сравнению с хранением данных в одномерном массиве, но в то же время используется больший объем памяти.