Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

Отчёт

по лабораторной работе №4
по курсу “Логика и основы алгоритмизации в инженерных задачах”
на тему “Бинарное дерево поиска”

Выполнили:
Студенты группы 24BBB4
Чердаков В.С.
Аверьянов Д.С.

Приняли:
Юрова О.В.
Акифьев И.В.

Пенза 2025

Название:

Бинарное дерево поиска

Цель работы:

Разработать программу, где реализуется поиск того или иного значения с клавиатуры

Лабораторное задание:

*Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном

дереве.

*Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

* Изменить функцию добавления элементов для исключения добавления одинаковых символов.

* Оценить сложность процедуры поиска по значению в бинарном дереве.

Листинг

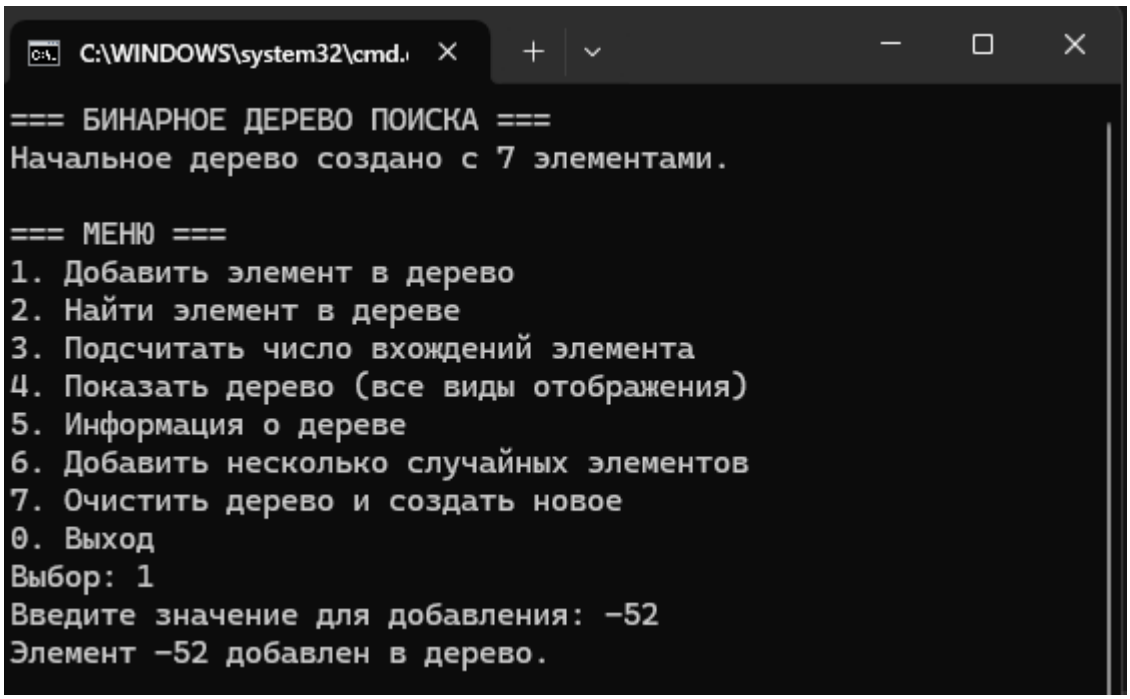
```
import random

# Узел дерева
class TreeNode:
    def __init__(self, value):
        self.data = value
        self.left = None
        self.right = None

# Создание нового узла
def create_node(value):
    return TreeNode(value)

# Вставка элемента в дерево (без дубликатов)
def insert(root, value):
    if root is None:
        return create_node(value)
    if value < root.data:
        root.left = insert(root.left, value)
    elif value > root.data:  # дубликаты запрещены
        root.right = insert(root.right, value)
    else:
        print(f"⚠ Элемент {value} уже есть в дереве, дубликат не добавлен.")
    return root

# Поиск элемента + количество шагов
def search_with_steps(root, value):
    steps = 0
    current = root
    while current is not None:
        steps += 1
        if current.data == value:
            print(f"Элемент {value} найден за {steps} шаг(а/ов).")
            return current
        elif value < current.data:
            current = current.left
        else:
```

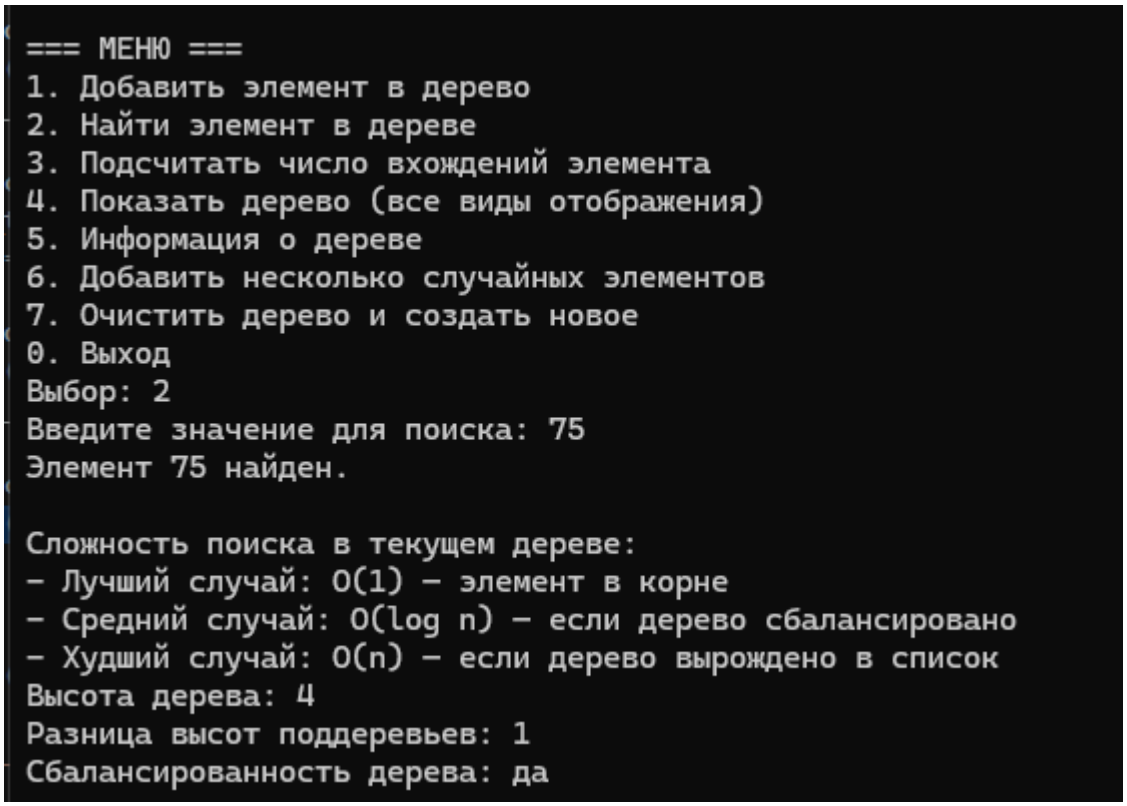
```
            current = current.right
    print(f"Элемент {value} не найден (прошли {steps} шагов).")
    return None

# Подсчёт вхождений (полный обход)
def count_occurrences(root, value):
    if root is None:
        return 0
    count = 1 if root.data == value else 0
    count += count_occurrences(root.left, value)
    count += count_occurrences(root.right, value)
    return count

# Высота дерева
def get_height(root):
    if root is None:
        return 0
    return 1 + max(get_height(root.left), get_height(root.right))

# Количество узлов
def count_nodes(root):
    if root is None:
        return 0
    return 1 + count_nodes(root.left) + count_nodes(root.right)

# Безопасный ввод числа
def safe_input_int(prompt):
    while True:
        try:
            return int(input(prompt))
        except ValueError:
            print("Ошибка: введите целое число!")

# Освобождение дерева
def free_tree(root):
    if root is not None:
        free_tree(root.left)
        free_tree(root.right)
        root.left = None
        root.right = None

# Упрощённый вывод
def print_tree_simple(root, level=0):
    if root is None:
        return
    print_tree_simple(root.right, level + 1)
    print("    " * level, end="")
    if level > 0:
        print("L-- ", end="")
    print(root.data)
    print_tree_simple(root.left, level + 1)

# Скобочная структура
def print_tree_bracket(root):
    if root is None:
        print("-", end="")
        return
    print(root.data, end="")
    if root.left is not None or root.right is not None:
        print("(", end="")
        print_tree_bracket(root.left)
        print(",", end="")
        print_tree_bracket(root.right)
        print(")", end="")
```

```python
# Демонстрация
def main():
    print("=== БИНАРНОЕ ДЕРЕВО ПОИСКА ===")
    root = None
    initial_values = []
    for v in initial_values:
        root = insert(root, v)
    print(f"Начальное дерево создано с {len(initial_values)} элементами.")

    while True:
        print("\n=== МЕНЮ ===")
        print("1. Добавить элемент в дерево")
        print("2. Найти элемент в дереве")
        print("3. Подсчитать число вхождений элемента")
        print("4. Показать дерево (упрощённый вид)")
        print("5. Информация о дереве")
        print("6. Добавить несколько случайных элементов")
        print("7. Очистить дерево и создать новое")
        print("0. Выход")

        choice = safe_input_int("Выбор: ")

        if choice == 1:
            value = safe_input_int("Введите значение для добавления: ")
            root = insert(root, value)
            print(f"Попытка добавления {value} завершена.")

        elif choice == 2:
            value = safe_input_int("Введите значение для поиска: ")
            result = search_with_steps(root, value)

            # Анализ сложности поиска
            print("\nСложность поиска в текущем дереве:")
            print("- Лучший случай: O(1) — элемент в корне")
            print("- Средний случай: O(log n) — если дерево сбалансировано")
            print("- Худший случай: O(n) — если дерево вырождено в список")


            print(f"Высота дерева: {get_height(root)}")
            left_height = get_height(root.left)
            right_height = get_height(root.right)
            balance = left_height - right_height
            print(f"Разница высот поддеревьев: {abs(balance)}")
            print("Сбалансированность дерева:", "да" if abs(balance) <= 1 else "нет")

        elif choice == 3:
            value = safe_input_int("Введите значение для подсчета: ")
            count = count_occurrences(root, value)
            print(f"Элемент {value} встречается {count} раз(а).")

        elif choice == 4:
            print("\n=== ДЕРЕВО (УПРОЩЁННЫЙ ВИД) ===")
            print_tree_simple(root, 0)

        elif choice == 5:
            print("\n=== ИНФОРМАЦИЯ О ДЕРЕВЕ ===")
            print(f"Количество узлов: {count_nodes(root)}")
            print(f"Высота дерева: {get_height(root)}")
            left_height = get_height(root.left)
            right_height = get_height(root.right)
            balance = left_height - right_height
            print(f"Высота левого поддеревья: {left_height}")
            print(f"Высота правого поддеревья: {right_height}")
            print(f"Разница высот: {abs(balance)}")
            print("Сбалансированность:", "да" if abs(balance) <= 1 else "нет")
```

```
            print("\nСтруктура дерева (упрощённый вид):")
            print_tree_simple(root, 0)

        elif choice == 6:
            count = safe_input_int("Сколько случайных элементов добавить? ")
            max_value = safe_input_int("Максимальное значение? ")
            for _ in range(count):
                val = random.randint(1, max_value)
                root = insert(root, val)
            print("Элементы добавлены. Новое дерево:")
            print_tree_simple(root, 0)

        elif choice == 7:
            free_tree(root)
            root = None
            for v in initial_values:
                root = insert(root, v)
            print("Дерево сброшено. Новое дерево создано:")
            print_tree_simple(root, 0)

        elif choice == 0:
            print("Выход из программы.")
            break

        else:
            print("Неверный выбор!")

if __name__ == "__main__":
main()


# Высота дерева
def get_height(root):
    if root is None:
        return 0
return 1 + max(get_height(root.left), get_height(root.right))
```

Результат работы программы:


C:\WINDOWS\system32\cmd.
=== БИНАРНОЕ ДЕРЕВО ПОИСКА ===
Начальное дерево создано с 7 элементами.
=== МЕНЮ ===
1. Добавить элемент в дерево
2. Найти элемент в дереве
3. Подсчитать число вхождений элемента
4. Показать дерево (все виды отображения)
5. Информация о дереве
6. Добавить несколько случайных элементов
7. Очистить дерево и создать новое
0. Выход
Выбор: 1
Введите значение для добавления: -52
Элемент -52 добавлен в дерево.

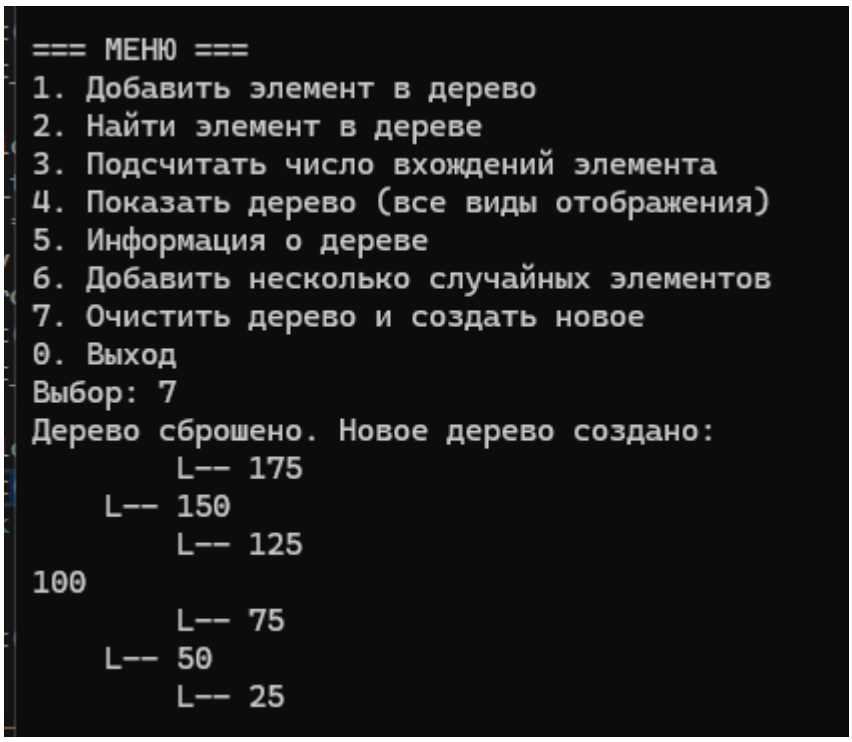

=== МЕНЮ ===
1. Добавить элемент в дерево
2. Найти элемент в дереве
3. Подсчитать число вхождений элемента
4. Показать дерево (все виды отображения)
5. Информация о дереве
6. Добавить несколько случайных элементов
7. Очистить дерево и создать новое
0. Выход
Выбор: 2
Введите значение для поиска: 75
Элемент 75 найден.
Сложность поиска в текущем дереве:
- Лучший случай: O(1) - элемент в корне
- Средний случай: O(log n) - если дерево сбалансировано
- Худший случай: O(n) - если дерево вырождено в список
Высота дерева: 4
Разница высот поддеревьев: 1
Сбалансированность дерева: да

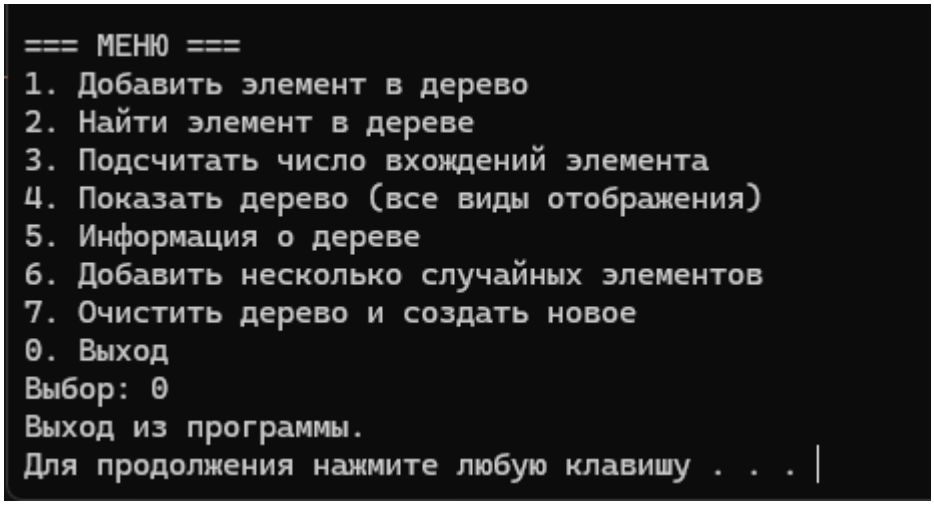
```
=== МЕНЮ ===
1. Добавить элемент в дерево
2. Найти элемент в дереве
3. Подсчитать число вхождений элемента
4. Показать дерево (все виды отображения)
5. Информация о дереве
6. Добавить несколько случайных элементов
7. Очистить дерево и создать новое
0. Выход
Выбор: 3
Введите значение для подсчета: 100
Элемент 100 встречается 1 раз(а).
```

```
=== МЕНЮ ===
1. Добавить элемент в дерево
2. Найти элемент в дереве
3. Подсчитать число вхождений элемента
4. Показать дерево (все виды отображения)
5. Информация о дереве
6. Добавить несколько случайных элементов
7. Очистить дерево и создать новое
0. Выход
Выбор: 4

=== ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ДЕРЕВА ===

Фото-стиль:
  !   Г---175
 Г---150
  !   L---125
100 (корень)
  !   Г---75
 L---50
  !   L---25
  !    !   L----52

Упрощённый вид:
        L-- 175
    L-- 150
        L-- 125
100
        L-- 75
    L-- 50
        L-- 25
            L-- -52
```

```
=== МЕНЮ ===
1. Добавить элемент в дерево
2. Найти элемент в дереве
3. Подсчитать число вхождений элемента
4. Показать дерево (все виды отображения)
5. Информация о дереве
6. Добавить несколько случайных элементов
7. Очистить дерево и создать новое
0. Выход
Выбор: 5

=== ИНФОРМАЦИЯ О ДЕРЕВЕ ===
Количество узлов: 8
Высота дерева: 4
Высота левого поддеревья: 3
Высота правого поддеревья: 2
Разница высот: 1
Сбалансированность: да

Структура дерева (упрощённый вид):
        L-- 175
    L-- 150
        L-- 125
100
        L-- 75
    L-- 50
        L-- 25
            L-- -52
```

```
=== МЕНЮ ===
1. Добавить элемент в дерево
2. Найти элемент в дереве
3. Подсчитать число вхождений элемента
4. Показать дерево (все виды отображения)
5. Информация о дереве
6. Добавить несколько случайных элементов
7. Очистить дерево и создать новое
0. Выход
Выбор: 6
Сколько случайных элементов добавить? 2
Максимальное значение? 75
Элементы добавлены. Новое дерево:
        L-- 175
    L-- 150
        L-- 125
100
        L-- 75
            L-- 56
    L-- 50
        L-- 25
                L-- 19
            L-- -52
```





Выводы:

В ходе выполнения лабораторной работы были разработаны программы, выполняющие указанные в лабораторной работе задачи. Результаты работ программ совпали с результатами трассировок, следовательно, программы работают без ошибок. Получили опыт в создании проектов в среде MicrosoftVisualStudio.