**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра** **ВТ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Двоичные деревья»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0322 |  | Шаронин А.Д. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2022

**Постановка задачи**

Реализовать структуру данных двоичное дерево поиска и следующие методы:

* нахождение минимума
* нахождение максимум
* прямой (preorder)
* центрированный (inorder) и
* обратный обход (postorder) по дереву
* поиск элемента
* нахождение следующего и
* предыдущего элемента
* удаление элемента
* обход в ширину
* Реализовать визуализацию дерева.

Указать теоретическую временную сложность для всех операций.

С помощью реализованной структуры данных написать программу, позволяющую преобразовать запись из префиксной/инфиксной/постфиксной нотации в префиксную/инфиксную/постфиксную нотацию.

**Описание реализуемых функций**

Двоичное дерево поиска:

* insert — Вставляет элемент в дерево. Сложность: в среднем O(log(n)), в худшем O(n)
* search — Ищет элемент с заданным значением в дереве, и если находит, то возвращает его адрес в памяти. Сложность: в среднем O(log(n)), в худшем O(n)
* remove — Удаляет элемент с заданным значением в дереве. Сложность: в среднем O(log(n)), в худшем O(n)
* min\_elem — Выводит самый маленький элемент дерева. Сложность: в среднем O(log(n)), в худшем O(n)
* max\_elem — Выводит самый большой элемент дерева. Сложность: в среднем O(log(n)), в худшем O(n)
* Preorder — Выводит значения дерева в префиксной форме. Сложность O(n)
* Inorder — Выводит значения дерева в инфиксной форме (отсортированный вид). Сложность O(n)
* Postorder — Выводит значения дерева в постфиксной форме. Сложность O(n)
* Widthorder — Выводит значения дерева по уровням, начиная с корня дерева. Сложность O(n)
* prev\_elem — Выводит элемент предшествующий введённому элементу. Сложность: в среднем O(log(n)), в худшем O(n)
* next\_elem — Выводит потомков введённого элемента. Сложность: в среднем O(log(n)), в худшем O(n)
* Print\_tree\_as\_tree — Выводит дерево в виде дерева. Сложность O(n)
* del\_all — удаляет все элементы дерева. Сложность O(n)

**Дерево арифметических выражений:**

* Preorder — Выводит значения дерева в префиксной форме. Сложность O(n)
* Inorder — Выводит значения дерева в инфиксной форме (отсортированный вид). Сложность O(n)
* Postorder — Выводит значения дерева в постфиксной форме. Сложность O(n)
* inf\_to\_post — Переводит выражение из инфиксной формы в постфиксную. Сложность O(n), где n — длина вводимого выражения
* pre\_to\_post — Переводит выражение из префиксной формы в постфиксную. Сложность O(n), где n — длина вводимого выражения
* post\_to\_post — Переводит выражение из постфиксной формы в постфиксную. Сложность O(1), просто присваивает переменной post\_notation введённое выражение
* create\_tree — Разбивает введённое выражение на операции и операнды, которые потом вставляются в дерево. Сложность: в среднем O(log(m)\*n), в худшем O(n\*m), где n — длина вводимого выражения, m — количество элементов выражения (например, (5 + 3) / 7, n = 11(с пробелами), m = 5)
* is\_operation — проверяет, является ли символ операцией (+, -, \*, /, ^)
* Push\_to\_tree — Вставляет элемент в дерево. Сложность: в среднем O(log(n)), в худшем O(n)
* Del\_all — Удаляет все элементы дерева. Сложность O(n)

**Пример работы программы**

На первом рисунке было создано двоичное дерево поиска из элементов: 25 61 59 81 26 50 77 19 60 80 2 93 6 70 27. Найден минимальный элемент, максимальный элемент. Выведено дерево в префиксной, инфиксной и постфиксной формах. Найден элемент 77. Выведены “потомки” элемента 80, “родитель” элемента 59. Удалено число 2. Сделан вывод дерева в ширину, а также вывод дерева в виде дерева.

Также было создано арифметическое дерево, для преобразования записи из префиксной/инфиксной/постфиксной нотации в префиксную/инфиксную/постфиксную нотацию. Была введена инфиксная нотация (((98 \* 654) / 5) / (7 ^ (1 + 1))) и преобразована во все три посредством вывода дерева разными путями.

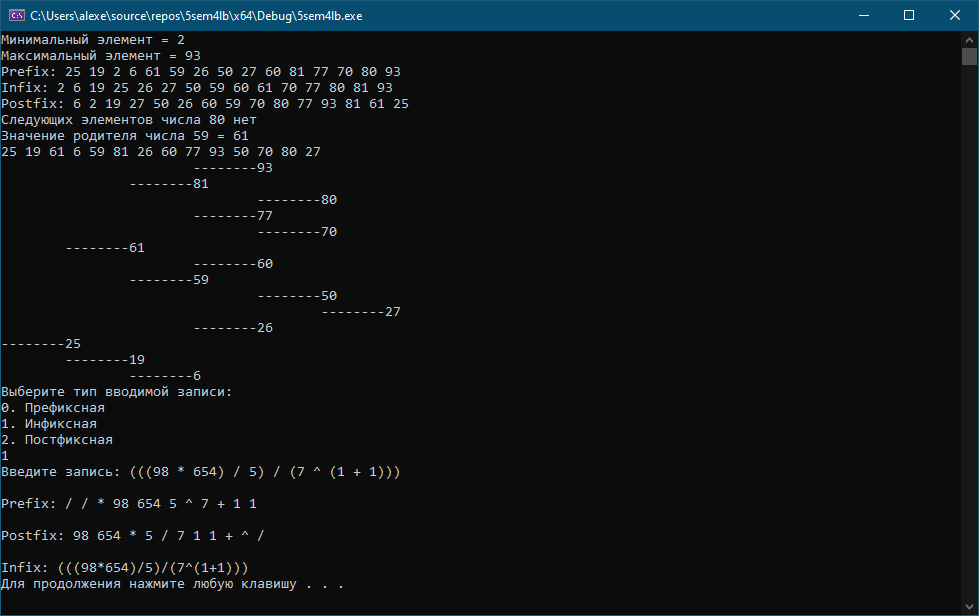
****

Рисунок 1 — инфиксная запись

На втором рисунке была введена префиксная запись того же выражения / / \* 98 654 5 ^ 7 + 1 1

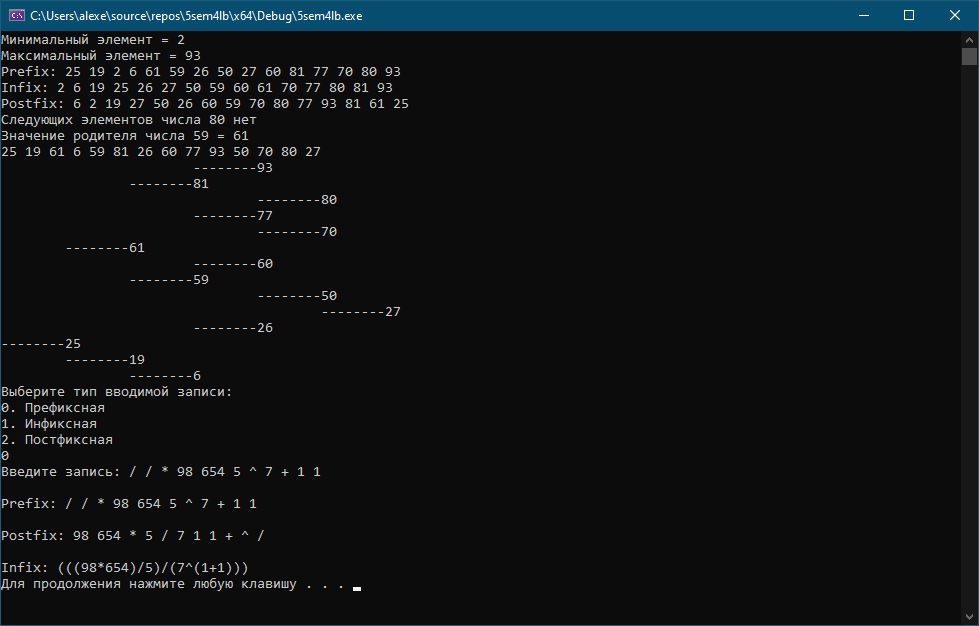


Рисунок 2 — префиксная запись

**Выводы**

В ходе выполнения данной лабораторной работы было реализовано бинарное дерево поиска и дерево арифметических выражений. В бинарном дереве поиска были реализованы следующие методы: нахождение минимума, нахождение максимума, прямой (preorder), центрированный (inorder) и обратный обход (postorder) по дереву, поиск элемента, нахождение следующего и предыдущего элемента, удаление элемента, обход в ширину, вывод дерева в виде дерева. С помощью дерева арифметических выражений и разных способов обхода дерева мы научились преобразовывать записи из префиксной/инфиксной/постфиксной нотации в префиксную/инфиксную/постфиксную нотацию.

Ссылка на гитхаб с кодом работы: https://github.com/FOOZBY/5sem4lb.git