МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и струтуры данных»

Тема: Двоичные деревья

Студент гр. 1301 _____ Устинов Г.А.

Преподаватель _____ Родионова Е.А.

Санкт-Петербург

2022

Цель работы

Изучить и реализовать структуру данных двоичное дерево поиска и заданные операции на нём. Использовать написанную структуру для считывания и преобразования арифметических выражений, записанных в инфиксной, префиксной и постфиксной формах.

Задание

Реализовать структуру данных двоичное дерево поиска и 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. 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.

Выполнение работы

Для описания обобщённого типа данных был написан шаблонный класс *BST — Binary Search Tree —* который реализует структуру данных и операции на ней. Каждый узел дерева представлен структурой *Node*, которая хранит данные, ключ, а также указатели на левого и правого потомка и на родителя. Указатель на корневой элемент хранится в классе объекте *BST*.

Операции встаавки элемента, поиска и удаления по ключу, благодаря древовидной структуре, работают за логарифмическое время, однако из-за того, что дерево не сбалансировано, в худшем случае асимптотика может стать линейной.

Методы *min* и *max* — поиск наибольшего и наименьшего элемента, хранимого деревом. Поскольку все элементы в дереве отсортированы, поиск наименьшего сводится к получению самого левого и нижнего элемента, а поиск наибольшего — к получению самого правого и нижнего элемента.

Методы для поиска следующего и предыдущего элементов — *findNext* и *findPrev* — реализованы путём обхода дерева итератором в глубину. Когда очередь доходит искомого элемента, благодаря тому, что итераторы поддерживают обход в обратную сторону, для нахождения предыдущего элемента производится «шаг назад», а для нахождения следующего - «шаг вперёд».

Метод *visualize* — создаёт поуровневое строковое представление дерева.

Методы *infixBegin*, *prefixBegin*, *postfixBegin* и *breadthBegin* возвращают итераторы для обхода в глубину одним их способов или для обхода в ширину.

Для представления и работы с арифметическими выражениями, были реализованы два класса — *Component* — оператор или операнд, составляющие части выражения, и *Expression* — дерево, кодирующее выражение, и методы его строкового представления в трёх формах.

Считывание и интерпретация выражений произовдится тремя функциями, которые обеспечивают построение дерева по выражениям — *parseInfixExpression*, *parsePrefixExpression* и *parsePostfixExpression.*

Пример работы

```
m41nfr4m3% src/converter
Enter mode: infix
infix > ((1 + 2) * (3 + 4)) / 5 + 6/7
Prefix form: + / * + 1 2 + 3 4 5 / 6 7
Infix form: (1 + 2) * (3 + 4) / 5 + 6 / 7
Postfix form: 1 2 + 3 4 + * 5 / 6 7 / +
              +
         /         /
     *     5     6     7
  +  +
1234
```

Выводы

Была изучена и реализована структура данных двоичное дерево поиска. Также, реализованный класс был использован для решения задачи считывания, представления и преобразования арифметических выражений.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

https://github.com/CmmSheparD/ads/pull/5