Лабораторная работа 7

Построение бинарного дерева поиска. Подсчет количества элементов в дереве

Компьютерная математика II, ММФ, БГУ

Лаврова О.А., май 2025

Навыки

1. **Python**: ООП: композиция, перегрузка операций; чтение числовых данных из текстового файла
2. **математика**: бинарное дерево поиска

Задание 7.1 Определение классов

Бинарное дерево поиска будем описывать с помощью трех классов: `BinaryTree`, `BinaryNode`, `EmptyNode`, связанных друг с другом на основе композиции. Проектирование классов представлено в лекции к теме *Бинарное дерево поиска. Реализация на основе ООП*.

Напишите базовое определение классов `BinaryTree`, `BinaryNode`, `EmptyNode` на основе лекционных материалов.

Базовое определение класса `BinaryTree` содержит метод инициализации `__init__(self)`, метод строкового представления `__repr__(self)`, метод вставки элемента в дерево `insert(self, value)`.

Базовое определение класса `BinaryNode` содержит метод инициализации `__init__(self)` и метод строкового представления `__repr__(self)`.

Базовое определение класса `EmptyNode` содержит метод строкового представления `__repr__(self)` и метод вставки элемента в дерево `insert(self, value)`.

Напишите комментарии для каждой строки кода в определении классов `BinaryTree`, `BinaryNode`, `EmptyNode`.

Задание 7.2 Метод вставки элемента в бинарное дерево поиска

Переопределите класс `BinaryNode` за счет определения метода вставки `insert(self, value)`. Рекомендации по выполнению представлены в лекции к теме *Бинарное дерево поиска. Реализация на основе ООП*.

Протестируйте корректность построения бинарного дерева поиска на основе вставки в дерево элементов некоторого итерируемого объекта. При этом важно, чтобы для элементов итерируемого объекта были определены операции сравнения. Приведите три примера построения бинарного дерева поиска на основе итерируемых объектов различных типов (например, `str`, `range`, `list`).

Задание 7.3 Построение бинарного дерева поиска

Прочитайте числовые данные, записанные в файлы, и постройте на основании этих данных бинарные деревья поиска.

Рассмотрите четыре варианта хранения данных в файлах:

1. числовые данные хранятся в текстовом файле и записаны в столбец;
2. числовые данные хранятся в текстовом файле, записаны в строки, разделены пробелами, в каждой строке расположено одинаковое количество числовых значений;
3. числовые данные хранятся в текстовом файле, записаны в строки, разделены пробелами, в каждой строке расположено различное количество числовых значений;
4. числовые данные хранятся в файле формата `json`.

Рекомендации по выполнению:

- строковый метод `split` разбирает строку на список подстрок по разделителю;
- строковые объекты нужно преобразовывать в числовые объекты перед их записью в бинарное дерево поиска;
- функция `loadtxt` из расширения `numpy` читает числовые данные из текстового файла без предварительного создания файлового объекта; в каждой строке файла должно быть расположено одинаковое количество числовых значений;
- для работы с файлами в формате `json` используйте, например, функции `load` и `values` из модуля `json`. Для используемых функций модуля сформулируйте спецификации.

Задание 7.4 Перегрузка операции принадлежности in

Переопределите классы `BinaryTree`, `BinaryNode`, `EmptyNode` за счет определения нового метода `__contains__(self, value)` для перегрузки операции принадлежности `in`. Рекомендации по выполнению представлены в лекции к теме *Бинарное дерево поиска. Реализация на основе ООП.*

Протестируйте корректность выполнения операции `in` на трех примерах.

In [4]:
```
source_data = [5,1,10,3,4]
tree = BinaryTree()
for i in source_data:
    tree.insert(i)
    print(tree)

for i in range(10):
    print((i, i in tree), end = ' ')
```

```
(*, 5, *)
((*, 1, *), 5, *)
((*, 1, *), 5, (*, 10, *))
((*, 1, (*, 3, *)), 5, (*, 10, *))
((*, 1, (*, 3, (*, 4, *))), 5, (*, 10, *))
(0, False) (1, True) (2, False) (3, True) (4, True) (5, True) (6, False) (7, Fals
e) (8, False) (9, False)
```

Задание 7.5 Перегрузка встроенной функции len

Переопределите классы `BinaryTree`, `BinaryNode`, `EmptyNode` за счет определения нового метода `__len__(self)` для перегрузки встроенной функции `len`, которая возвращает количество вершин в бинарном дереве поиска. Рекомендации по выполнению представлены в лекции к теме *Бинарное дерево поиска. Реализация на основе ООП.*

Протестируйте корректность выполнения функции `len` на трех примерах.

In [6]:
```
tree = BinaryTree()
for i in source_data:
    tree.insert(i)

len(tree)
```

Out[6]: 5