Отчёт по практическому заданию

Флексер Александр

26 ноября 2023 г.

1 Цели работы

Данная практическая работы позволит мне изучить работу лексемезации и вычисления арифметического (и алгебраического) выражения и очередной раз убедиться в существовании Б-га.

2 Постановка задачи

Нужно реализовать три способа подсчета выражения: Польская запись, дерево выражения и БНФ. Каждый из них будет в себе иметь основную функцию calc для подсчета значения выражения.

3 Методы

3.1 Польская форма записи

Польская запись, также известна как префиксная запись - это форма записи арифметических и алгебраических выражений. Ее отличие от, нам привычной, инфиксной записи состоит в том что сначаоа идут два опереанда (в том же порядке, что они и идут в инфиксной записи), и только потом идет знак операции между ними. Благодря такой записи можно избавиться от скобок, так как расположение знака всегда будет зависеть от его приоритетности. Для реализации я использовал такую стркутура данных, как стек. Она позовляет сделать процесс работы исключительно с верхним элементом списка более удобным, благодя тому что все обращения к элементам списка спрятаны под красимы названиясми push() и top(). Ну и когда процесс переводы в постфисную запись проведен, уже и без труда можно посчитать результат функции. Нужно просто иди по новому выражению слева направа, пока не встретим операцаю. Когда встретили, применим ее к двум последним элементам списка и заменим их на результат математической операции. Будем так идти по дереву пока его длина не станет равна одному. Когда стала равна одному- выводим тот самый единственный оставшийся элемент.

3.2 Дерево выражений

Для реализации дерева выражений понадобилось сделать класс ExprNode, кадный экземпляр которого в себе хранит три поля: left, right и node-value. Первые два отвечают за два поддерава, если же вершина является числом или переменной, то она не имеет поддеревеьв и эти же поля становяться None. Для реализации метода calc понадобиться использовать рекурсию. Каждый calc будет вызывать самого себя пока не "успрется"в вершину без детей. Таким образом кадный вершина будет вызывать эту функцию пдсчета от правого и левого своего поддерева, а дальше будет производить соответствующие операции над полученными результатми. Так же в таком дереве можно проводить множесто предпроцессов для оптимизации дерва с приводом его к делению одного полинома на другой.

3.3 Бэкусовы нормальные формы

БНФ - метод, построенный по принципу разделения приоритетов дву слагаемых/множителей. Сначала нам следует найти наименее приоритетную операцию в выражении и по ней разделить всё выражение. Права часть станет слагаемым (если делили по знаку сложения / вычитания), а левая часть станет выражением, котрое мы будем точно таким же образом обрабатывать. А затем уже получнное ранее слагаемое будет разделено по умножению (делению), если оно есть в выражении (по той же схеме: слева выражение, а справа- множитель), а в пративном случае, если так и не найдтся знака для разделения (*, /), то оно будет передано в третий класс отвечающий за множителей, полностью. А уже в третьем классе оно будет в случае если передано одно число или переменная, то будет возращено оно же, а иначе переводим множитель в выражение и все то же самое продеываем по кругу.

4 Тесты на исправность работы программ

Для проверки работоспособности всех трех программ, я использовал следующие 9 тестов, причем кадный сложней предыдущего.

```
from random import randint


INPUT_DATA = ['1 + 2',
              '1 + 2 - 3',
              '1 * 2 - (x - 1) * 9',
              '((((1 + 2))))',
              '(1 + 2)^3 - ((1 - 1) * (2 + 2) - (3^3)^3)',
              '(12 - 3) * ((1 + 2 + 3) - (12 - 4) / 2 + 10^3) / 10',
              '(2*3*x+5*7*x*2*x)*(x-1)/2',
              '(1/x + x/3) / ((1+2) / (x+4)) - (x/(x - 1))',
              'x - 35 + 2^3 - 2 * (5 - 2) + 1 * 4 + 3^(4/2)']
```

А для проверки своих ответов, я использую встроенную функцию eval(). Единственное, приходиться заменять знак возведения в степень.

5 Тесты на скорость

Здесь я сгенерировал 98 тестов, причем длина кадого выражения большь длины предыдущего на 10 чисел. Сами чила задаются через функцию randint модулю random. Прорамма берет число от 1 до 10000 и делит на 100, таким образом мы получаем любое число с плавающей запятой от 0.01 до 99.99 (включительно). А так же кадые пять чисел вставляется вместо чиста некоторая переменная x, которая только при вызове функции calc примет конкретное значение.

```
from random import randint, choice


oparatioans = ('-', '+', '/', '*')

INPUT_DATA = [''.join([(str(randint(1, 10000) / 100) if ind % 5 != 0 else 'x') +
                                        choice(oparatioans)
                      for ind in range(length * 10)])[:-1] for length in range(2, 100)
                                                            ]
```

А в этой части кода я с помощью декоратора замеряю скорость каждого метода. Общаяя функция для "прогонки"функция- tester, которая как аргумент принимает метод подсчет выражения и запускает его уже изнутри, передаваяя нужные входные тестовые данные.

```
from SpeedTests import INPUT_DATA
from BNF import Expression
from PolishCalculator import calc as polish_calc
from LexemeTree import ExprNode
from time import time as now
from random import randint
from matplotlib import pyplot as plt
from sys import setrecursionlimit
setrecursionlimit(2000)


results: dict[str, list] = {'bnf_test': [],
                            'polish_test': [],
                            'lexeme_tree_test': []}


def test_time(func):
    def wraper(*args):
        start_time = now()
        func(*args)
        finish_time = now()
        results.get(func.__name__).append(round(finish_time - start_time, 10))
    return wraper


def tester(function):
    for data_test in INPUT_DATA:
        function(data_test, randint(100, 10000) / 100)


@test_time
def polish_test(expr: str, x):
    polish_calc(expr, x)


@test_time
def lexeme_tree_test(expr: str, x):
    expr_node: ExprNode = ExprNode(expr)
    expr_node.calc(x)


@test_time
def bnf_test(expr: str, x):
    Expression(expr).calc(x)
```

6 Вывод

А теперь главный результат анализа - график зависимоит длины выражения и скорости работы с зависимостью времени (в секундах) для каждого метода.

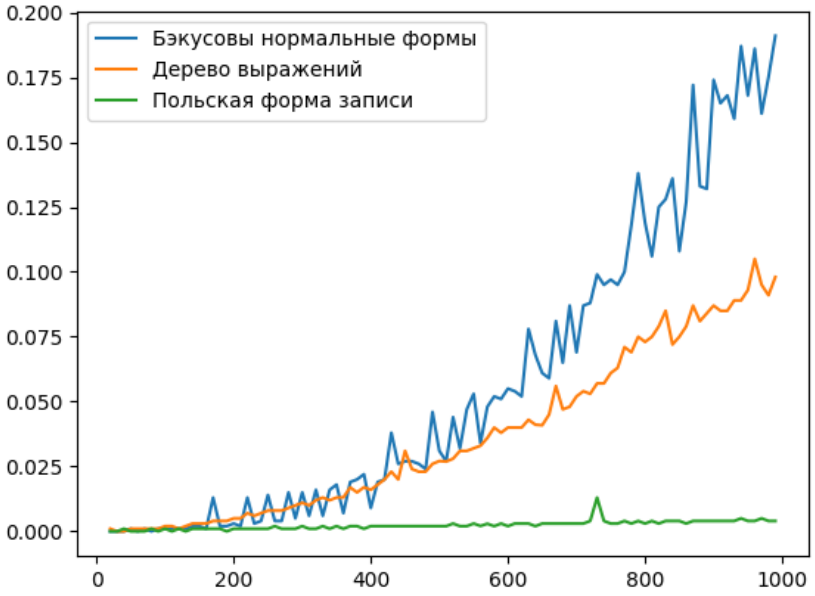

Рис. 1: Диаграмма моментов на участке выбора момента прокатки

На графике можно заметить, что БНФ показывает максимальное время, при длине выражения равной 1000 чисел, его время работы стало больше времени работы дерева выражений более чем в два раза, но лучше всего себя показала польская запись, время работы которой крайне близко к нулю секундам. (Для создания графика использовал библиотеку matplotlib).