Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по рубежному контролю № 1

По курсу "Анализ Алгоритмов"

Параллельная реализация вычислений ответа по ОПЗ

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-56Б

Преподователи:

Волкова Лилия Леонидовна
Строганов Юрий Владимирович

Москва, 2020 г.

Содержание

Введение

Целью данной лабораторной работы является изучение параллельного вычисления обратной польской записи представленной в виде дерева.

В ходе лабораторной предстоит выполнить следующие задачи:

1. рассмотреть и изучить алгоритм ОПЗ;
2. сравнить временные характеристики ОПЗ реализованного параллельно в виде дерева и стека;
3. на основе проделанной работы сделать выводы.

Постановка задачи: Параллельная реализация вычисления ответа по ОПЗ (ОПЗ представлено деревом, сначала два потока находят высоты левого и правого поддеревьев, затем в зависимости от Высоты Вы выделяете от 2 до 4 потоков на расчёт для поддеревьев (например, если дерево несбалансированное, то на маленькое поддерево выделить 1 поток, а на большое поддерево - 2).

1 Аналитическая часть

В данной части будут представлены теоретические сведения о рассматриваемых алгоритмах.

1.1 Алгоритм обратной польской записи

Обратная польская запись или же ОПЗ - форма записи математических и логических выражений, в которой операнды расположены перед знаками операций.[1]

Обратная польская нотация (ОПН) была разработана австралийским философом и специалистом в области теории вычислительных машин Чарльзом Хэмблином в середине 1950-х на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем. Работа Хэмблина была представлена на конференции в июне 1957, и издана в 1957 и 1962. Первыми компьютерами, поддерживающими обратную польскую нотацию, были KDF9 от English Electric Company, который был анонсирован в 1960 и выпущен (появился в продаже) в 1963, и американский Burroughs B5000, анонсирован в 1961, выпущен в том же 1963.

Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

1. запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами;

2. Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу;

3. Результатом вычисления выражения становится результат последней вычисленной операции.

2 Конструкторская часть

В данном разделе представлены схемы алгоритма.

2.1 Схема решения задачи

На рисунке 1 показана схема решения поставленной задачи.

Рис. 1: Схема параллельного вычисления обратной польской записи представленной в виде дерева.

На рисунке 2 показана схема алгоритма ОПЗ с использованием стека.

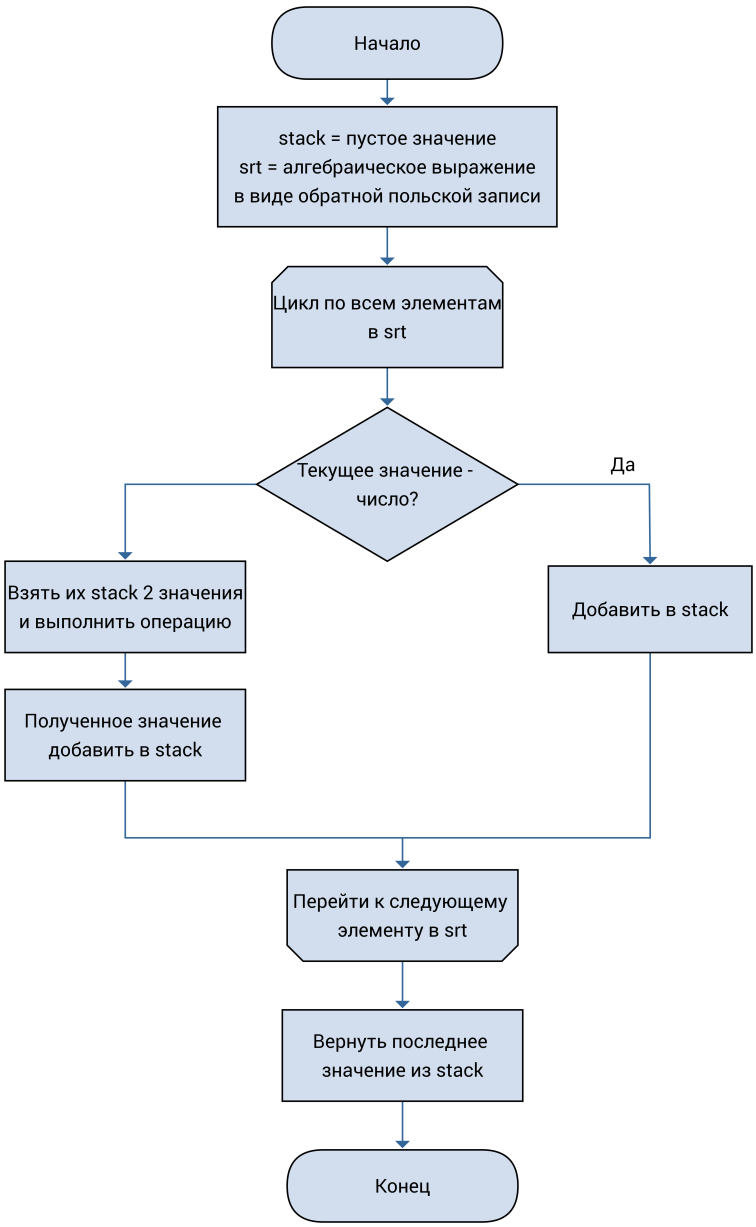

Рис. 2: схема алгоритма ОПЗ с использованием стека.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

1. на вход подается арифметическое выражение представленное в виде ОПЗ;
2. на выходе - результат полученного решения;
3. допущение - вводимое выражение должно быть представлено в корректном виде ОПЗ.

3.2 Средства реализации

В данной работе используется язык программирования Python, за высокую скорость выполнения программ и широкий выбор библиотек.[2] Проект выполнен в среде разработки Visual Studio Code.

3.3 Листинг кода

На листинге 1 представлена реализация параллельного вычисления обратной польской записи представленной в виде дерева.

```
def isOperator(c):
    if (c == "+" or c == "-" or c == "*" or c == "/"):
        return True
    else:
        return False

OPERATORS = {"+": operator.add, "-": operator.sub,
             "*": operator.mul, "/": operator.truediv}
@dataclass
class Node:
    data: str
    left: Optional[Node] = None
    right: Optional[Node] = None

```

```
res = []

@dataclass
class Tree:
    root: Node

    def post_order(self, node: Optional[Node]) -> None:
        if node:
            self.post_order(node.left)
            self.post_order(node.right)
            if isOperator(node.data) == False:
                res.append(int(node.data))
            else:
                tmp = OPERATORS[node.data](res[-2], res[-1])
                del res[-1]
                del res[-1]
                res.insert(1, tmp)

    def height(self,node):
        if node==None:
            return 0
        else:
            lheight = self.height(node.left)
            rheight = self.height(node.right)
            if lheight > rheight:
                return(lheight+1)
            else:
                return(rheight+1)
if __name__ == "__main__":

    j = Node("6")
    k = Node("2")
    h = Node("+", j, k)
    i = Node("3")
    d = Node("*", h, i)
    e = Node("4")
    b = Node("-", d, e)
    f = Node("7")
    g = Node("3")
```

```
    c = Node("+", f, g)
    a = Node("/", b, c)

    tree = Tree(a)
    h1 = tree.height(a.left)
    h2 = tree.height(a.right)
    w1 = 4 if h1 > h2 else 1
    w2 = 2 if h1 < h2 else 1
    for i in range(w1):
        th = Thread(target = tree.post_order, args=(a.left, ))
        th.start()
    for i in range(w2):
        th = Thread(target = tree.post_order, args=(a.right, ))
        th.start()
    print("Result = ", OPERATORS[a.data](res[0], res[1]))
```

Листинг 1: Параллельное вычисление обратной польской записи представленной в виде дерева.

На листинге 2 представлена реализация алгоритма ОПЗ с использованием стека.

```
def polska(srt):
    if len(srt) == 0:
        return None
    stack = []
    lst = list(srt)
    for i in srt:
        if i.isdigit():
            stack.append(i)
            lst.remove(i)
        else:
            cnt1, cnt2 = stack.pop(), stack.pop()
            stack.append(OPERATORS[i](int(cnt2), int(cnt1)))
            lst.remove(i)
    return stack.pop()

```

Листинг 2: ОПЗ с использованием стека.

3.4 Тестирование функций

В таблице 1 приведены функциональные тесты для функций, реализующих алгоритмы ОПЗ. Все тесты прошли успешно.

Таблица 1: Результаты тестирования.

Входной параметр	Ожидаемый результат	Полученный результат
Пустая строка	None	None
6 2 + 3 * - 3 7 + /	2	2

4 Исследовательская часть

В данном разделе приведены примеры работы и анализ характеристик разработанного программного обеспечения.

4.1 Системные характеристики

Характеристики компьютера на котором проводился замер времени сортировки массива:

1. операционная система - Windows 10;
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz;
3. оперативная память - 16 ГБ;
4. количество ядер - 4;
5. количество логических процессов - 8.

4.2 Пример работы

Демонстрация работы программы приведена на рисунке 3 и 4.

```
PS C:\Jasur\projects\bmstu_aa\rk> python exp.py
Введети выражение в виде ОПЗ: 6 2 + 3 * 4 - 3 7 + /
Полученный результат:  2.0
```

Рис. 3: Пример работы программы.

```
PS C:\Jasur\projects\bmstu_aa\rk> python exp.py
Введети выражение в виде ОПЗ: 3 4 +
Полученный результат:  7.0
```

Рис. 4: Пример работы программы.

4.3 Сравнительный анализ на основе замеров времени работы программы

Был проведен замер времени работы параллельной реализации ОПЗ.

В таблице 2 показаны результаты эксперимента, суть которого заключается в анализе зависимости количества потоков от времени выполнения алгоритма. Под случаем один подразумевается, что высота левого поддерева больше чем высота правого. В случае 2 - высота правого поддерева больше левого.

Таблица 2: Результаты эксперимента.

Случай	Левая часть | Правая часть	Время
1	4 | 2	0.00327019
1	2 | 4	0.00486018
1	1 | 1	0.00120833
1	Нет потоков	0.01741666
2	4 | 2	0.00368871
2	2 | 4	0.00295467
2	1 | 1	0.00160833
2	Нет потоков	0.02173285

4.4 Вывод

По проведенному анализу из эксперимента можно сделать вывод, что в случае когда высота одной из поддеревьев больше другой, то лучше использовать большее количество потоков. Но это не самое быстрое время выполнения. Наилучшее время было достигнуто в случае, когда количество потоков было равно 1 для обеих поддеревьев. Так как в дереве значение родителя зависит от значения ребенка, соответственно когда у нас большое количество потоков, то возникает время ожидания. За счет этого в случае когда количество работников было равно 1, мы получили лучший результат. Так же был проведен замер времени для классического алгоритма ОПЗ с реализацией в виде стека без использования параллельного программирования и получен следующий результата: 0,02155, что в раза дольше чем самое худшее время полученное при параллельном вычислении.

Соответственно можно сделать вывод о том, что лучше использовать алгоритм ОПЗ с древовидной реализацией.

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

1. рассмотреть и изучить алгоритмы алгоритмы ОПЗ;
2. сравнить временные характеристики каждого из рассмотренных алгоритмов;
3. на основании проделанной работы сделать выводы.

При сравнении этих алгоритмов был сделан вывод, что параллельная реализация ОПЗ в виде дерева более быстродействена чем простая реализация ОПЗ в виде стека.

Список литературы

[1] Wikipedia. *Обратная польская запись[ЭЛ. РЕСУРС] Режим доступа:* URL: https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C. (дата обращения: 27.12.2020).

[2] Гвидо ван Россум. *Python documentation[ЭЛ. РЕСУРС] Режим доступа:* URL: https://docs.python.org/3/library/concurrent.futures.html. (дата обращения: 22.11.2020).