САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4
по курсу «Алгоритмы и структуры данных»
Тема: Стек, очередь, связанный список.
Вариант 14

Выполнил:
Мурашов Н.А.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Задачи по варианту

Задача 1: Стек

Текст задачи.

1 задача. Стек

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ N”, либо “–”. Команда “+ N”означает добавление в стек числа N, по модулю не превышающего 10^9. Команда “–”означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 10^6 элементов.

Листинг кода.

```
#Task1/src/StackProcessor.py

"""Модуль для обработки операций со стеком."""

from lab4.utils.IOHandler import IOHandler
from lab4.utils.consts import *
import os


class StackProcessor:
    """Класс для обработки команд работы со стеком."""

    def __init__(self):
        """Инициализация стека и результатов."""
        self.stack = []
        self.results = []

    def process_commands(self, commands):
        """
        Обрабатывает список команд и обновляет результаты.

        :param commands: Список команд.
        """
        for command in commands:
            if command.startswith("+"):
                _, number = command.split()
                self.stack.append(int(number))
            elif command == "-":
                self.results.append(self.stack.pop())

    @staticmethod
    def read_commands(input_path):
        """
        Считывает команды из файла.

        :param input_path: Путь к входному файлу.
```

```
        :return: Список команд.
        """
        lines = IOHandler.read_file(input_path)
        commands = [cmd.strip() for cmd in lines[1:]]  # Убираем первую строку и \n
        return commands

    @staticmethod
    def validate_commands(commands):
        """
        Валидирует список команд.

        :param commands: Список команд.
        :return: True если команды валидны, иначе False.
        """
        if not (1 <= len(commands) <= 10 ** 6):
            return False
        for cmd in commands:
            if cmd.startswith("+"):
                parts = cmd.split()
                if len(parts) != 2:
                    return False
                try:
                    number = int(parts[1])
                    if abs(number) > 10 ** 9:
                        return False
                except ValueError:
                    return False
            elif cmd != "-":
                return False
        return True

    def get_results(self):
        """
        Возвращает результаты обработки команд.

        :return: Список результатов.
        """
        return self.results

    @staticmethod
    def write_results(output_path, results):
        """
        Записывает результаты в файл.

        :param output_path: Путь к выходному файлу.
        :param results: Список результатов.
        """
        data = "\n".join(map(str, results))
        IOHandler.write_file(output_path, data)
```

Текстовое объяснение решения.

Решение задачи реализовано с использованием структуры данных "стек". Для выполнения операций добавления и удаления используются методы списка Python: *append* и *pop*.

Каждая команда обрабатывается в зависимости от ее типа:

- При + x число добавляется на вершину стека.
- При - элемент удаляется с вершины стека, а его значение сохраняется в результирующем списке.

Алгоритм эффективен, так как каждая операция выполняется за O(1). Входные данные проверяются на соответствие ограничениям задачи.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.004003 сек	0.30МБ
Пример из задачи	0.05832 сек	0.52 МБ
Пример из задачи	0.09345 сек	1.45 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.42455 сек	1.99 МБ

Вывод по задаче.

Реализован стек с поддержкой операций добавления и удаления элементов. Вывод результатов удаления корректно записывается в выходной файл.

Задача 2: Очередь

Текст задачи.

2 задача. Очередь

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «–». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «–» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит 10^6 элементов.

Листинг кода.

```
# Task2/src/QueueProcessor.py

"""Модуль для обработки операций с очередью."""

from lab4.utils.IOHandler import IOHandler
```

```python
class QueueProcessor:
    """Класс для обработки команд работы с очередью."""

    def __init__(self):
        """Инициализация очереди и результатов."""
        self.queue = []
        self.front_index = 0
        self.results = []

    def process_commands(self, commands):
        """
        Обрабатывает список команд и обновляет результаты.

        :param commands: Список команд.
        """
        for command in commands:
            if command.startswith("+"):
                _, number = command.split()
                self.queue.append(int(number))
            elif command == "-":
                self.results.append(self.queue[self.front_index])
                self.front_index += 1

    @staticmethod
    def read_commands(input_path):
        """
        Считывает команды из файла.

        :param input_path: Путь к входному файлу.
        :return: Список команд.
        """
        lines = IOHandler.read_file(input_path)
        commands = [cmd.strip() for cmd in lines[1:]]  # Убираем первую строку и \n
        return commands

    @staticmethod
    def validate_commands(commands):
        """
        Валидирует список команд.

        :param commands: Список команд.
        :return: True если команды валидны, иначе False.
        """
        if not (1 <= len(commands) <= 10 ** 6):
            return False
        for cmd in commands:
            if cmd.startswith("+"):
                parts = cmd.split()
                if len(parts) != 2:
                    return False
                try:
                    number = int(parts[1])
                    if abs(number) > 10 ** 9:
                        return False
                except ValueError:
```

```
                return False
            elif cmd != "-":
                return False
        return True

    def get_results(self):
        """
        Возвращает результаты обработки команд.

        :return: Список результатов.
        """
        return self.results

    @staticmethod
    def write_results(output_path, results):
        """
        Записывает результаты в файл.

        :param output_path: Путь к выходному файлу.
        :param results: Список результатов.
        """
        data = "\n".join(map(str, results))
        IOHandler.write_file(output_path, data)
```

Текстовое объяснение решения.

Для реализации очереди используется список, где элементы добавляются методом *append*. Удаление элементов выполняется без их физического удаления из памяти, а с помощью увеличения индекса начала очереди.

Алгоритм работы следующий:

При + x число добавляется в конец списка.

При - элемент на текущем начальном индексе списка сохраняется в результирующем списке. Индекс начала очереди увеличивается.

Метод эффективен, так как операции добавления и удаления выполняются за O(1).

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.003465 сек	0.35 МБ
Пример из задачи	0.03958 сек	0.56 МБ
Пример из задачи	0.10345 сек	1.59 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.4124 сек	1.89 МБ

Вывод по задаче.
Реализована очередь с поддержкой операций добавления и удаления. Результаты операций удаления записываются в выходной файл.

Задача 3: Скобочная последовательность. Версия 1

Текст задачи.

3 задача. Скобочная последовательность. Версия 1

Последовательность A, состоящую из символов из множества «(», «)», «[» и «]», назовем ***правильной скобочной последовательностью***, если выполняется одно из следующих утверждений:

- A – пустая последовательность;
- первый символ последовательности A – это «(», и в этой последовательности существует такой символ «)», что последовательность можно представить как $A = (B)C$, где B и C – правильные скобочные последовательности;
- первый символ последовательности A – это «[», и в этой последовательности существует такой символ «]», что последовательность можно представить как $A = (B)C$, где B и C – правильные скобочные последовательности.

Так, например, последовательности «(())» и «()[]» являются правильными скобочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

Листинг кода.

```
"""Модуль для проверки корректности скобочных последовательностей."""


from lab4.utils.IOHandler import IOHandler

class BracketSequenceChecker:
    """Класс для проверки правильности скобочных последовательностей."""

    def __init__(self):
        """Инициализация сопоставления скобок."""
        self.matching_brackets = {')': '(', ']': '['}

    def is_valid_sequence(self, sequence):
        """
        Проверяет, является ли скобочная последовательность правильной.

        :param sequence: Строка со скобочной последовательностью.
        :return: True, если последовательность правильная, иначе False.
        """
        stack = []

        for char in sequence:
```

```
            if char in "([":  # Открывающая скобка
                stack.append(char)
            elif char in ")]":  # Закрывающая скобка
                if stack and stack[-1] == self.matching_brackets[char]:
                    stack.pop()
                else:
                    return False
        return not stack  # Если стек пустой, последовательность правильная

    def process_sequences(self, sequences):
        """
        Проверяет список скобочных последовательностей.

        :param sequences: Список строк с последовательностями.
        :return: Список результатов ("YES" или "NO").
        """
        results = []
        for seq in sequences:
            seq = seq.strip()
            if self.is_valid_sequence(seq):
                results.append("YES")
            else:
                results.append("NO")
        return results

    @staticmethod
    def read_sequences(input_path):
        """
        Считывает последовательности из входного файла.

        :param input_path: Путь к входному файлу.
        :return: Список последовательностей.
        """
        lines = IOHandler.read_file(input_path)
        n = int(lines[0])
        sequences = lines[1:]
        return n, sequences

    @staticmethod
    def validate_input(n, sequences):
        """
        Валидирует входные данные.

        :param n: Количество последовательностей.
        :param sequences: Список последовательностей.
        :return: True, если данные валидны, иначе False.
        """
        if not (1 <= n <= 500):
            return False
        if len(sequences) != n:
            return False
        for seq in sequences:
            seq = seq.strip()
            if not (1 <= len(seq) <= 10**4):
                return False
        return True
```

```python
@staticmethod
def write_results(output_path, results):
    """
    Записывает результаты в выходной файл.

    :param output_path: Путь к выходному файлу.
    :param results: Список результатов.
    """
    data = "\n".join(results)
    IOHandler.write_file(output_path, data)
```

Текстовое объяснение решения.
Для проверки используется стек. Алгоритм работает следующим образом:
Открывающие скобки добавляются в стек.
Закрывающие скобки проверяются на соответствие последней открывающей. Если пара корректна, открывающая скобка удаляется из стека.
После обработки всех символов стек должен быть пустым.
Каждая операция добавления и удаления выполняется за O(1). Проверка всей последовательности имеет сложность O(N), где N — длина строки. Валидация входных данных проверяет, что длина каждой строки не превышает 10^4.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.00178 сек	0.27 МБ
входных данных из текста задачи		
Пример из задачи	0.02354 сек	0.61 МБ
Пример из задачи	0.12789 сек	1.32 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.34218 сек	2.67 МБ

Вывод по задаче.
Реализован алгоритм проверки корректности скобочных последовательностей.
Результаты проверки записываются в выходной файл.

Дополнительные задачи

Задача 4: Скобочная последовательность. Версия 2

Текст задачи.

4 задача. Скобочная последовательность. Версия 2

Определение правильной скобочной последовательности такое же, как и в задаче 3, но теперь у нас больше набор скобок: []{}().

Нужно написать функцию для проверки наличия ошибок при использовании разных типов скобок в текстовом редакторе типа LaTeX.

Для удобства, текстовый редактор должен не только информировать о наличии ошибки в использовании скобок, но также указать точное место в коде (тексте) с ошибочной скобочкой.

Листинг кода.

```python
# Task4/src/BracketChecker.py

"""Модуль для проверки скобок в строке."""

from lab4.utils.IOHandler import IOHandler


class BracketChecker:
    """Класс для проверки правильности скобок в строке."""

    def __init__(self):
        """Инициализация сопоставления скобок."""
        self.bracket_pairs = {')': '(', ']': '[', '}': '{'}

    def check_brackets(self, data):
        """
        Проверяет строку на корректность расстановки скобок.

        :param data: Строка для проверки.
        :return: "Success" или индекс первой ошибки.
        """
        stack = []
        for index, char in enumerate(data, start=1):
            if char in "([{":
                stack.append((char, index))
            elif char in ")]}":
                if not stack or stack[-1][0] != self.bracket_pairs[char]:
                    return str(index)
                stack.pop()

        if stack:
            # Возвращаем позицию первой незакрытой открывающей скобки
            return str(stack[0][1])

        return "Success"
```

```python
    @staticmethod
    def read_data(input_path):
        """
        Считывает данные из входного файла.

        :param input_path: Путь к входному файлу.
        :return: Строка с данными.
        """
        lines = IOHandler.read_file(input_path)
        return lines[0].strip()

    @staticmethod
    def write_result(output_path, result):
        """
        Записывает результат в выходной файл.

        :param output_path: Путь к выходному файлу.
        :param result: Результат проверки.
        """
        IOHandler.write_file(output_path, result)
```

Текстовое объяснение решения.
Для проверки используется стек, в который записываются открывающие скобки и их индексы. Если стек пуст или скобка не соответствует паре, возвращается индекс ошибки. Если после проверки стек не пуст, возвращается индекс первой незакрытой скобки.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.01134 сек	1.67 МБ
Пример из задачи	0.02689 сек	1.77 МБ
Пример из задачи	0.07581 сек	1.87 МБ
Верхняя граница диапазона значений входных данных из текста задачи	1.02345 сек	1.90 МБ

Вывод по задаче.
Реализована проверка корректности расстановки скобок. Результат проверки выводится корректно.

Задача 8: Постфиксная запись

Текст задачи.

8 задача. Постфиксная запись

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как A B +. Запись B C + D * обозначает привычное нам (B + C) * D, а запись A B C + D * + означает A + (B + C) * D. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

Дано выражение в обратной польской записи. Определите его значение.

Листинг кода.

```
# Task8/src/PostfixEvaluator.py

"""Модуль для вычисления выражений в постфиксной записи."""

from lab4.utils.IOHandler import IOHandler
from lab4.utils.consts import *


class PostfixEvaluator:
    """Класс для вычисления выражений в постфиксной записи."""

    def __init__(self):
        """Инициализация класса."""
        self.bracket_limit = 2 ** 31

    def evaluate_postfix(self, expression):
        """
        Вычисляет значение выражения в постфиксной записи.

        :param expression: Список строк, представляющих постфиксное выражение.
        :return: Результат вычисления.
        :raises ValueError: Если найдено число, выходящее за пределы |2^31|,
                            или если обнаружен неизвестный оператор.
        :raises IndexError: Если выражение некорректно (например, недостаточно операндов).
        """
        stack = []  # Инициализируем стек для промежуточных значений

        for token in expression:
            if token.isdigit() or (token.startswith('-') and token[1:].isdigit()):
                # Если токен - число, помещаем его в стек
                num = int(token)
                if abs(num) >= self.bracket_limit:
                    raise ValueError("Найдено число, выходящее за пределы |2^31|")
                stack.append(num)
            else:
                # Если токен - оператор, выполняем соответствующую операцию
                if len(stack) < 2:
                    raise IndexError("Недостаточно операндов для выполнения операции")
                b = stack.pop()  # Второй операнд
                a = stack.pop()  # Первый операнд

                # Выполняем операцию в зависимости от оператора
                if token == '+':
```

```
                result = a + b
            elif token == '-':
                result = a - b
            elif token == '*':
                result = a * b
            else:
                raise ValueError(f"Неизвестный оператор: {token}")

            # Проверяем, не превышает ли промежуточный результат предел
            if abs(result) >= self.bracket_limit:
                raise ValueError("Промежуточный результат превышает предел |2^31|")
            stack.append(result)

    if len(stack) != 1:
        raise IndexError("Некорректное выражение")

    return stack.pop()

@staticmethod
def read_expression(input_path):
    """
    Считывает выражение из входного файла.

    :param input_path: Путь к входному файлу.
    :return: Кортеж из числа элементов и списка элементов выражения.
    """
    lines = IOHandler.read_file(input_path)
    if not lines:
        raise ValueError("Входной файл пуст")
    n = int(lines[0].strip())
    expr = lines[1].strip().split()
    return n, expr

@staticmethod
def validate_input(n, expr):
    """
    Валидирует входные данные.

    :param n: Количество элементов в выражении.
    :param expr: Список элементов выражения.
    :return: True, если данные валидны, иначе False.
    """
    if not (1 <= n <= 10 ** 6):
        return False
    if len(expr) != n:
        return False
    for token in expr:
        if token.isdigit() or (token.startswith('-') and token[1:].isdigit()):
            num = int(token)
            if abs(num) >= 2 ** 31:
                return False
        elif token not in ('+', '-', '*'):
            return False
    return True

@staticmethod
```

```
def write_result(output_path, result):
    """
    Записывает результат в выходной файл.

    :param output_path: Путь к выходному файлу.
    :param result: Результат вычисления.
    """
    IOHandler.write_file(output_path, str(result))
```

Текстовое объяснение решения.
Сортировка выполняется с использованием дополнительного стека. Алгоритм перекладывает элементы из одного стека в другой, обеспечивая упорядоченность элементов. Сложность алгоритма составляет O(N^2).

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.00782 сек	0.48 МБ
входных данных из текста задачи		
Пример из задачи	0.06548 сек	1.48 МБ
Пример из задачи	0.25642 сек	1.60 МБ
Верхняя граница диапазона значений входных данных из текста задачи	1.10294 сек	1.91 МБ

Вывод по задаче.
Реализован алгоритм сортировки стека с использованием дополнительного стека. Сортировка выполняется корректно.

Вывод

В рамках лабораторной работы реализованы и протестированы алгоритмы работы со структурами данных: стек и очередь. Были выполнены задачи на их основные операции, проверку скобочных последовательностей, реализацию стека с поддержкой *max* за *O(1)*, а также сортировку стека. Все алгоритмы продемонстрировали корректность и эффективность при обработке больших объемов данных.