Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский институт ИТМО»

Факультет МР и П

Алгоритмы и структуры данных

Лабораторная работа №4.

«Стек, очередь, связанный список»

Вариант «9»

Выполнил студент:

Розметов Джалолиддин

Группа № D3210

Преподаватель: Артамонова Валерия Евгеньевна

г. Санкт-Петербург

2024

Оглавление

Задание 1

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.

• Формат входного файла (input.txt). В первой строке входного файла содержится M ($1 \le M \le 10^6$) – число команд. Каждая последующая строка исходного файла содержит ровно одну команду.

• Формат выходного файла (output.txt). Выведите числа, которые удаляются из стека с помощью команды “–”, по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

- Пример:

input.txt	output.txt
6 + 1 + 10 - + 2 + 1234 -	10 1234

Код

```
def main():
    with open("input.txt", "r") as file:
        commands = file.read().strip().split("\n")

    stack = []
    output = []

    for command in commands[1:]:
        if command == "-":
            removed_element = stack.pop()
            output.append(removed_element)
        else:
            _, number = command.split()
            stack.append(int(number))

    with open("output.txt", "w") as file:
        for element in output:
            file.write(str(element) + "\n")

if __name__ == "__main__":
    main()
```

Вводимые данные:

6
+ 1
+ 10
-
+ 2
+ 1234
-

Вывод кода:

10
1234

Описание кода:

1. Считывание команд из файла: Читает команды из файла input.txt и сохраняет их в список commands.
2. Обработка команд: Инициализирует пустой стек stack и список output для результатов.
3. Обрабатывает каждую команду: Если команда -, удаляет верхний элемент из стека и добавляет его в список output. Если команда + x, добавляет число x в стек.
4. Запись результатов в файл: Записывает все удаленные элементы из стека в файл output.txt.

Описание проведенных тестов:

Проведенные тесты проверяют правильность обработки команд для стека. Включены сценарии с добавлением и удалением элементов, последовательными удалениями, добавлением без удалений, пустым входным файлом и большим количеством команд. Эти тесты подтверждают, что программа корректно выполняет операции добавления и удаления, и правильно записывает результаты в файл.

Вывод работы кода:
Проведенные тесты подтвердили, что код корректно обрабатывает команды для стека и правильно записывает результаты в выходной файл.

Задание 2

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.

• Формат входного файла (input.txt). Первая строка входного файла содержит число N ($1 \le N \le 500$) – число скобочных последовательностей, которые необходимо проверить. Каждая из следующих N строк содержит скобочную последовательность длиной от 1 до 10^4 включительно. В каждой из последовательностей присутствуют только скобки указанных выше видов.

• Формат выходного файла (output.txt). Для каждой строки входного файла (кроме первой, в которой записано число таких строк) выведите в выходной файл «YES», если соответствующая последовательность является правильной скобочной последовательностью, или «NO», если не является.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

- Пример:

input.txt	output.txt
5	YES
()()	YES
([])	NO
([)]	NO
(([]	NO
)(	

Код

```
def is_valid_sequence(s):
    stack = []
    for char in s:
        if char in '([':
            stack.append(char)
        else:
            if not stack:
                return False
            if char == ')' and stack[-1] != '(':
                return False
            if char == ']' and stack[-1] != '[':
                return False
            stack.pop()
    return len(stack) == 0

def main():
    with open("input.txt", "r") as file:
        n = int(file.readline().strip())
        sequences = [file.readline().strip() for _ in range(n)]

    results = []
    for sequence in sequences:
        if is_valid_sequence(sequence):
            results.append("YES")
```

```
        else:
            results.append("NO")

    with open("output.txt", "w") as file:
        file.write("\n".join(results))

if __name__ == "__main__":
    main()
```

Вводимые данные:

5
()()
[()]
([)]
(()]
)(

Вывод кода:
YES
YES
NO
NO
NO

Описание кода:

1. Функция is_valid_sequence(s):

- Проверяет корректность последовательности скобок в строке s.
- Использует стек для отслеживания открывающих скобок.
- Возвращает True для корректных последовательностей и False для некорректных.

2. Функция main():

- Считывает входные данные из файла input.txt.
- Для каждой строки вызывает is_valid_sequence(s) и сохраняет результаты ("YES" или "NO") в список results.
- Записывает результаты в файл output.txt.

Описание проведенных тестов:
Проведенные тесты оценивают работу функции is_valid_sequence, проверяя её на правильные и неправильные последовательности скобок, включая пустые строки и

строки без скобок. Это позволяет убедиться в корректности работы функции при различных условиях ввода данных.

Вывод по работе кода:
Код корректно проверяет правильность последовательностей скобок и записывает результаты в выходной файл, обрабатывая различные сценарии, включая правильные и неправильные последовательности.

Задание 3

Реализуйте работу очереди. В дополнение к стандартным операциям очереди,

необходимо также отвечать на запрос о минимальном элементе из тех, которые сейчас находится в очереди. Для каждой операции запроса минимального элемента выведите ее результат. На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда – это либо «+ N», либо «–», либо «?». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «–» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

• Формат входного файла (input.txt). В первой строке содержится M ($1 \le M \le 10^6$) – число команд. В последующих строках содержатся команды, по одной в каждой строке.

• Формат выходного файла (output.txt). Для каждой операции поиска минимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Гарантируется, что операций извлечения или поиска минимума для пустой очереди не производится.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

input.txt	output.txt
7 + 1 ? + 10 ? - ? -	1 1 10

Код

```
class MinQueue:
    def __init__(self):
        self.queue = []
        self.min_stack = []

    def enqueue(self, value):
        self.queue.append(value)
        while self.min_stack and self.min_stack[-1] > value:
            self.min_stack.pop()
        self.min_stack.append(value)

    def dequeue(self):
        if not self.queue:
            return None
        value = self.queue.pop(0)
        if value == self.min_stack[0]:
            self.min_stack.pop(0)
        return value

    def get_min(self):
        return self.min_stack[0] if self.min_stack else None

with open('input.txt', 'r') as f:
    commands = f.readlines()

min_queue = MinQueue()

with open('output.txt', 'w') as f:
    for command in commands[1:]:
        command = command.strip()
        if command.startswith('+'):
            value = int(command[2:])
            min_queue.enqueue(value)
        elif command == '-':
            min_queue.dequeue()
        elif command == '?':
            min_value = min_queue.get_min()
            f.write(str(min_value) + '\n')
```

Вводимые данные:

7
+ 1
?
+ 10
?
-
?
-

Вывод кода:

1
1
10

Описание кода:

1. Класс MinQueue содержит методы для добавления элемента в очередь (enqueue), извлечения элемента из очереди (dequeue) и получения минимального элемента (get_min).
2. При добавлении нового элемента в очередь (enqueue), элемент также проверяется на минимальность и добавляется в стек минимальных элементов только если он меньше или равен текущему минимальному элементу.
3. При извлечении элемента из очереди (dequeue), если извлекаемый элемент совпадает с минимальным элементом, он также удаляется из стека минимальных элементов.
4. При запросе минимального элемента (get_min) возвращается верхний элемент стека минимальных элементов, если он существует, иначе возвращается None.

Описание проведенных тестов:
Проверка включала тесты на добавление и удаление элементов, запрос минимального элемента, а также обработку пустой очереди и проверку работы на различных входных данных.

Вывод по работе кода:

Код успешно реализует функциональность "очереди с минимальным элементом" и проходит проверку на различных тестовых сценариях. Он обеспечивает эффективное добавление, извлечение и получение минимального элемента из очереди. Реализация без использования сторонних библиотек делает код более простым и переносимым.

Задание 4

Очередь в поликлинике работает по сложным правилам. Обычные пациенты при посещении должны вставать в конец очереди. Пациенты, которым "только справку забрать встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром. Напишите программу, которая отслеживает порядок пациентов в очереди.

• Формат входного файла (input.txt). В первой строке записано одно целое

число n ($1 \le n \le 10^5$) - число запросов к вашей программе. В следующих n строках заданы описания запросов в следующем формате: – «+ i» – к очереди присоединяется пациент i ($1 \le i \le N$) и встает в ее конец; – «* i» – пациент i встает в середину очереди ($1 \le i \le N$); – «-» – первый пациент в очереди заходит к врачу. Гарантируется, что на момент каждого такого запроса очередь будет не пуста.

• Формат выходного файла (output.txt). Для каждого запроса третьего типа в отдельной строке выведите номер пациента, который должен зайти к шаманам.

• Ограничение по времени. Оцените время работы и используемую память при заданных максимальных значениях.

• Пример:

input.txt	output.txt	input.txt	output.txt
7	1	10	1
+ 1	2	+ 1	3
+ 2	3	+ 2	2
-		* 3	5
+ 3		-	4
+ 4		+ 4	
-		* 5	
-		-	
		-	
		-	
		-	

Код

```
class Queue:
    def __init__(self):
        self.queue = []

    def add_patient(self, patient):
        self.queue.append(patient)

    def add_patient_middle(self, patient):
        mid = len(self.queue) // 2
        if len(self.queue) % 2 == 0:
            self.queue.insert(mid, patient)
        else:
            self.queue.insert(mid + 1, patient)

    def remove_patient(self):
        return self.queue.pop(0)


def process_queue(queries):
    queue = Queue()
    result = []

    for query in queries:
        if query[0] == '+':
            queue.add_patient(int(query[1]))
        elif query[0] == '*':
            queue.add_patient_middle(int(query[1]))
        elif query[0] == '-':
            result.append(queue.remove_patient())

    return result


def main():
    with open('input.txt', 'r') as f:
        n = int(f.readline())
        queries = [f.readline().strip().split() for _ in range(n)]  #
```

```
Splitting the query string

    result = process_queue(queries)

    with open('output.txt', 'w') as f:
        for item in result:
            f.write(str(item) + '\n')



if __name__ == '__main__':
    main()
```

Вводные данные:

10
+ 1
+ 2
* 3
-
+ 4
* 5
-
-
-
-

Вывод данных:

1
3
2
5
4

Описание кода:

1. Функциональность: Код успешно реализует указанный функционал очереди с добавлением, удалением и добавлением в середину.
2. Читаемость: Код хорошо структурирован и читаем, что облегчает его понимание и поддержку.
3. Эффективность: Для операций добавления и удаления из очереди используется список, что обеспечивает достаточно эффективное выполнение операций.
4. Тестирование: Для убедительности стоит провести дополнительное тестирование на разнообразных входных данных, чтобы убедиться в корректности работы кода в различных ситуациях.

Описание проведенных тестов:
Тестирование включало проверку корректности работы программы на различных сценариях использования, таких как добавление элементов в конец очереди, удаление элементов из начала очереди, добавление элементов в середину очереди,

обработка пустой очереди, а также смешанные операции для проверки работы кода в различных сценариях.

Вывод по работе кода:
Код успешно прошел тестирование и демонстрирует корректное функционирование в различных сценариях использования. Он обеспечивает работу с очередью, позволяя добавлять, извлекать и добавлять элементы в середину, что делает его универсальным инструментом для управления данными в программе.

Вывод

Стек (Stack), очередь (Queue) и связанный список (Linked List) - важные структуры данных в программировании. Стек - LIFO (последний вошел, первый вышел), используется, например, в обратной польской записи и управлении вызовами функций. Очередь - FIFO (первый вошел, первый вышел), полезна для управления задачами в порядке поступления. Связанный список - структура данных, где каждый элемент содержит данные и ссылку на следующий элемент, что обеспечивает гибкость при вставке и удалении элементов в середине списка. Каждая из этих структур имеет свои уникальные характеристики и применения, и понимание их работы помогает программистам эффективно решать задачи.