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 12

Выполнила:
Мкртчян.К.Г.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Оглавление

Задачи по варианту

Задача №2. Очередь. [N баллов]

Текст задачи:

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.

Решение:

```
def queue_actions(actions: list[str]) -> tp.List[str]:
    queue = []
    head = 0
    deleted_elements = []
    for a in actions:
        if a == "-":
            deleted_elements += [queue[head]]
            head += 1
        else:
            elem = a[2:]
            queue += [elem]
    return deleted_elements
```

Объяснение решения:

Основные шаги функции

Инициализация переменных:

queue: Это список, который используется для хранения элементов, добавленных в очередь.

head: Это указатель на начало очереди; он изначально установлен в 0. Он помогает отслеживать, сколько элементов уже было удалено из очереди.

deleted_elements: Это список, который будет хранить элементы, удаленные из очереди.

Обработка каждого действия в actions:

Если действие равно "-":

Это означает, что необходимо удалить элемент из очереди. Элемент, который будет удален, берется по индексу head, и он добавляется в список deleted_elements. Затем head увеличивается на 1, чтобы указать на следующий элемент в очереди.

Если действие не равно "-":

Предполагается, что это действие связано с добавлением нового элемента в очередь. Элемент берется из строки, начиная с третьего символа (так как строка a имеет формат, где два первых символа можно игнорировать,

например "A1", "A2" и т.п.). Добавленный элемент помещается в конец списка queue.
Возврат результата:
После обработки всех действий функция возвращает список deleted_elements, который содержит все элементы, удаленные из очереди.

Время выполнения программы, затраты памяти и результат:

Task #2 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	5	0.0	27.0	2
	+ 2			5
	-			
	+ 5			
	+ 9			
	-			
Значения из примера	4	0.0	27.0	1
	+ 1			10
	+ 10			
	-			
	-			
Максимальные значения	1000000	0.1	95.68	-458298725
	-			106946144
	+ 868685273			804248650
	+ 351447563			211656737
	-			-493136891
	-			

• Ограничение по времени. 2 сек. • Ограничение по памяти. 256 мб.
Вывод по задаче: мы поняли принцип работы очереди и реализовали ее.

Задача №4. Скобочная последовательность. Версия 2. [N баллов]
Текст задачи:
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. В первую очередь объявляется ошибка при наличии первой несовпадающей закрывающей скобки, перед которой отсутствует открывающая скобка, или которая не соответствует открывающей, например, ()[} - здесь ошибка укажет на }. Во вторую очередь, если описанной выше ошибки не было найдено, нужно указать на первую несовпадающую открывающую скобку, у которой отсутствует закрывающая, например, (в ([]. Если не найдено ни одной из указанный выше ошибок, нужно сообщить, что

использование скобок корректно. Помимо скобок, код может содержать большие и маленькие латинские буквы, цифры и знаки препинания. Формально, все скобки в коде (тексте) должны быть разделены на пары совпадающих скобок, так что в каждой паре открывающая скобка идет перед закрывающей скобкой, а для любых двух пар скобок одна из них вложена внутри другой, как в (foo[bar]) или они разделены, как в f(a,b)-g[c]. Скобка [соответствует скобке], соответствует и (соответствует).

Решение:

```
def bracket_sequence(brackets: str):
    bracket_pairs = {')': '(',
                     '}': '{',
                     ']': '['}
    if len(brackets) == 1:
        symbol = brackets[0]
        if symbol in bracket_pairs.keys() or symbol in
bracket_pairs.values():
            return "1"
        else:
            return "Success"
    else:
        stack = ""
        err_ind = 1
        for b in brackets:
            if b in bracket_pairs.values() or len(stack) == 0:
                stack += b
            elif b in bracket_pairs.keys():
                if bracket_pairs[b] == stack[-1]:
                    stack = stack[:-1]
                else:
                    return str(err_ind)
            else:
                pass
            err_ind += 1
    return "Success"
```

Объяснение решения:

Функция def bracket_sequence(brackets: str) проверяет правильность последовательности скобок в строке brackets. Она использует словарь bracket_pairs для сопоставления открывающих и закрывающих скобок.

Если длина строки равна 1, функция проверяет, является ли первый символ допустимой скобкой. Если да, возвращается значение «1». В противном случае возвращается «Success».

В противном случае создаётся пустой стек и начинается цикл по всем символам строки. Если текущий символ находится в значениях словаря или если стек пуст, добавляется текущий символ в стек. Если текущий символ является открывающей скобкой, то добавляется соответствующая закрывающая скобка в стек. Если текущая скобка не совпадает с

предыдущей, возвращается ошибка с номером индекса текущего символа. Если стек пуст, возвращается «Success».

Время выполнения, затраты памяти и результат:

Task #4 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	[]	0.0	27.71	Success
Значения из примера	{}[]	0.0	27.71	Success
Значения из примера	[()]	0.0	27.71	Success
Значения из примера	(())	0.0	27.71	Success
Значения из примера	{	0.0	27.71	1
Значения из примера	{[}	0.0	27.71	3
Значения из примера	foo(bar);	0.0	27.71	Success
Значения из примера	foo(bar[index);	0.0	27.71	14
Значения из примера	{{[(	0.01	27.87	50091

• Ограничение по времени. 5 сек. • Ограничение по памяти. 256 мб.

Вывод по задаче: мы научились анализировать скобочную последовательность с помощью стека.

Задача №6. Очередь с минимумом. [N баллов]

Текст задачи:

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.

Решение:

```
def queue_actions(actions: tp.List[str]) -> tp.List[int]:
    queue = collections.deque()
    minim = collections.deque()
    answers = []

    for a in actions:
        if a == "?":
            answers.append(minim[0])
        elif a == "-":
            deleted_elem = queue.popleft()
            if deleted_elem == minim[0]:
                minim.popleft()
        else:
            elem = int(a[2:])
            queue.append(elem)

            while minim and minim[-1] > elem:
                minim.pop()
            minim.append(elem)
    return answers
```

Объяснение решения:

Функция queue_actions реализует логику обработки операций с очередью, а именно добавление элементов, удаление элементов и запрос на вывод минимального элемента в очереди. Давайте разберем её работу по шагам:

Основные компоненты функции

Импорт необходимых модулей:

Предполагается, что в начале кода импортирован модуль collections, чтобы использовать deque (двунаправленная очередь).

Инициализация структур данных:

queue: основная очередь, в которой хранятся добавленные элементы.

minim: вспомогательная очередь для отслеживания текущего минимального элемента. Это позволяет эффективно находить минимальный элемент за константное время.

answers: список для хранения результатов запросов минимальных элементов.

Обработка каждого действия в actions

Цикл по элементам actions:

Обработка действий:

Если элемент a равен "?":

Добавляем текущий минимальный элемент (спереди вспомогательной очереди minim) в список answers.

Если элемент a равен "-" (действие удаления):

Удаляем элемент из начала queue с помощью popleft().

Если удалённый элемент равен текущему минимальному элементу (первый элемент из minim), то также удаляем его из minim.

Добавляем элемент в конце queue.
Затем обновляем очередь minim, удаляя элементы с конца, которые больше добавленного элемента. Это гарантирует, что minim всегда содержит элементы в неубывающем порядке.
Возврат результатов

В конце функции возвращается список, который содержит все результаты запросов на минимальный элемент.

Время выполнения, затраты памяти и результат:

Task #6 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	7	0.0	27.88	1
	+ 1			1
	?			10
	+ 10			
	?			
	-			
	?			
	-			
Значения из примера	9	0.0	27.88	10
	+ 21			4
	+ 10			
	+ 13			
	?			
	-			
	-			
	+ 4			
	?			
	-			
Значения из примера	1000000	0.17	75.12	-999989132
	+ -557946537			-999989132
	+ -731552878			-999989132
	?			-999989132
	+ 785811837			-999989132
	-			

• Ограничение по времени. 2 сек. • Ограничение по памяти. 256 мб.
Вывод по задаче: мы научились реализовывать очередь с минимумом.

Задача №11. Бюрократия. [N баллов]

Текст задачи:

В министерстве бюрократии одно окно для приема граждан. Утром в очередь встают n человек, i-й посетитель хочет получить ai справок. За один прием можно получить только одну справку, поэтому если после приема посетителю нужны еще справки, он встает в конец очереди. За время приема министерство успевает выдать m справок. Остальным придется ждать следующего приемного дня. Ваша задача - сказать, сколько еще справок хочет получить каждый из оставшихся в очереди посетитель в тот момент, когда прием закончится. Если все к этому моменту разойдутся, выведите -1.

Решение:

```
def bureaucracy(visitors_count: int, documents_count: int, queue: list[int])
-> tp.Union[

int, tp.Tuple[int, tp.List[int]]]:
    deque = queue.copy()
    head = 0
    end = visitors_count
    while documents_count > 0:
        deque[head] -= 1
        documents_count -= 1
        if deque[head] > 0:
            deque += [deque[head]]
            end += 1
        head += 1
    remainder = end - head
    return (remainder, deque[head:end]) if remainder else (-1, [])
```

Объяснение решения:

Функция bureaucracy моделирует процесс обработки документов для посетителей в бюрократическом контексте. Она работает следующим образом:

Параметры функции

visitors_count (int): Общее количество посетителей, стоящих в очереди.

documents_count (int): Количество документов, которые нужно обработать.

queue (list[int]): Список, представляющий количество документов, которые каждый посетитель в очереди изначально должен обработать.

Описание работы функции

Копирование очереди:deque = queue.copy()

Здесь создается копия исходного списка queue, чтобы не изменять его напрямую.

Инициализация указателей:

head = 0

end = visitors_count

head: Указатель на текущего посетителя, который обрабатывается.

end: Указывает на количество посетителей, оставшихся в очереди (инициализируется количеством посетителей).

Обработка документов:

while documents_count > 0:

deque[head] -= 1

documents_count -= 1

В этом цикле:

Уменьшается количество оставшихся документов у текущего посетителя (с индексом head) на 1.
Количество оставшихся документов также уменьшается на 1.
Проверка состояния текущего посетителя:

```
if deque[head] > 0:
    deque += [deque[head]]
    end += 1
```

Если текущий посетитель все еще имеет документы для обработки (т. е. его значение после уменьшения всё еще больше 0):
Этот посетитель возвращается в конец очереди, так как он не завершил обработку своих документов.
Увеличивается счетчик end, указывающий на новое количество посетителей в очереди.
Обработка завершена:

```
head += 1
```

Увеличивается указатель head, чтобы перейти к следующему посетителю.
Рассчитывается остаток:

```
remainder = end - head
```

Здесь remainder указывает на количество посетителей, которые еще находятся в очереди после завершения обработки документов.
Возврат результата:

```
return (remainder, deque[head:end]) if remainder else (-1, [])
```

Если остались незавершенные посетители (т. е. remainder больше 0), функция возвращает кортеж, содержащий:
количество оставшихся в очереди посетителей (remainder),
список документов, которые они должны обработать.
Если очередь пуста (т. е. все документы были обработаны), возвращается (-1, []).
Время выполнения, затраты памяти и результат:

Task #11 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	3 2 1 2 3	0.0	22.37	2 3 1
Значения из примера	4 5 2 5 2 3	0.0	22.37	3 4 1 2
Значения из примера	8 16 2 1 1 3 2 4 2 1	0.0	22.37	-1
Значения из примера	100000 10000000 166368 52467 299122	1.44	32.25	99986 947757 294319 185699

Вывод по задаче: мы научились использовать дек для решения нестандартной задачи.

Дополнительные задачи

Задача №8. Постфиксная запись. [N баллов]

Текст задачи:

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.

Решение:

```
def calc(a: int, b: int, action: str) -> int:
    if action == "+":
        return a + b
    elif action == "*":
        return a * b
    elif action == "-":
        return a - b
    elif action == "/":
        return int(a//b)
    else:
        return 0


def equal(expression: str) -> tp.Union[int, str]:
    expression = expression.split(" ")
    if 1 < len(expression) < 3:
        return int(expression[0])
    stack = []
    for e in expression:
        if e.isnumeric():
            stack += [int(e)]
        elif len(e) > 1 and e[1:].isnumeric():
            stack += [int(e)]
        else:
            stack[-2] = calc(stack[-2], stack[-1], e)
            stack.pop()
    if len(stack) == 1:
        return stack[0]
    else:
        return "error"
```

Объяснение решения:

Описание функции clac ():

Эта функция выполняет арифметические операции над двумя целыми числами a и b, в зависимости от заданной строки action, которая указывает на операцию.

Если action:
+: возвращает сумму a и b.
*: возвращает произведение a и b.
-: возвращает разность a и b.
/: возвращает целочисленное деление a на b (целую часть результата).
В любом другом случае возвращает 0.
Функция equal ():
Эта функция принимает строку expression, представляющую выражение, и обрабатывает его.
Она сначала разбивает строку на отдельные элементы (числа и операции) с помощью метода split().
Если в выражении только одно число, оно возвращается как целое число.
В противном случае используется стек для обработки арифметических операций по принципу обратной польской записи:
Если элемент является числом, он добавляется в стек.
Если элемент — это операция (+, -, *, /), он применяется к двум верхним элементам стека. Результат помещается обратно в стек, а верхний элемент (ранее использованный) убирается.
В конце:
Если в стеке остался только один элемент, он возвращается как результат.
Если в стеке больше одного элемента, возвращается строка "error", указывающая на ошибку в вычислении.

Время выполнения, затраты памяти и результат:

Task #8 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	7 8 9 + 1 7 - *	0.0	29.22	-102
Значения из примера	9 5 15 + 4 7 + 1 - /	0.0	29.22	2
Значения из примера	8 5 15 + 4 7 + 1 -	0.0	29.22	error
Значения из примера	1000000 3 7 / 3	0.15	35.36	3000000

• Ограничение по времени. 2 сек. • Ограничение по памяти. 256 мб

Вывод по задаче: с помощью стека мы смогли решить задачу расчета выражения, записанного в виде постфиксной записи.

Задача №13.1. Реализация стека, очереди и связанных списков. [N баллов]

Текст задачи: Реализуйте стек на основе связного списка с функциями isEmpty, push, pop и вывода данных.

Решение:

```
class Node:
    def __init__(self, val: int, next=None) -> None:
        self.val = val
        self.next = next


class LinkedList:
    def __init__(self, root) -> None:
        self.root = root

    def push(self, val: int) -> None:
        tmp = self.root
        while tmp.next:
            tmp = tmp.next
        tmp.next = Node(val=val)

    def output(self) -> list:
        ll = []
        tmp = self.root
        while tmp:
            ll.append(tmp.val)
            tmp = tmp.next
        return ll

    def is_empty(self) -> bool:
        tmp = self.root
        if tmp is None:
            return True
        else:
            return False

    def pop(self):
        tmp = self.root
        if tmp.next:
            while tmp.next.next:
                tmp = tmp.next
            tmp.next = None
        else:
            self.root = None
```

Объяснение решения:

Описание классов Node и LinkedList

Класс Node и класс LinkedList реализуют структуру данных "связный список" (Linked List). Давайте разберем их работу по частям.

Класс Node

Атрибуты:
val: хранит значение узла (целое число).
next: указывает на следующий узел в списке (по умолчанию None).
Конструктор: При создании нового узла инициализирует его значение и задает указатель на следующий узел.
Класс LinkedList
Атрибуты:
root: представляет корень (первый узел) связного списка.
Методы:
push(val: int) -> None:
Добавляет новый узел со значением val в конец списка.
Использует цикл для traversing (поиска) конца списка, а затем создает новый узел и связывает его с последним узлом.
output() -> list:
Возвращает список всех значений узлов в связном списке.
Использует цикл для обхода и добавления значений узлов в список.
is_empty() -> bool:
Проверяет, пуст ли связный список.
Если корень (self.root) равен None, список пуст и возвращает True, иначе — False.
pop():
Удаляет последний элемент (узел) из списка.
Если в списке есть более одного узла, используется цикл для traversing до предпоследнего узла, который устанавливается следующим в None.
Если в списке только один узел, устанавливает self.root в None.

Время выполнения, затраты памяти и результат:

Task #13.1 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	[] is_empty()	0.0	26.95	True
Значения из примера	[10, 13, 12] output()	0.0	26.95	[10, 13, 12]
Значения из примера	[9, 2, 25, 17, 12] pop()	0.0	26.95	[9, 2, 25, 17, 12]
Значения из примера	[] is_empty()	0.0	26.95	True
Значения из примера	[789, 313, 98, 189] push()	0.0	26.95	[789, 313, 98, 189]

Вывод по задаче: Классы Node и LinkedList обеспечивают базовую функциональность для работы со связным списком, включая добавление, удаление узлов и проверку состояния списка.

Задача №13.3. Реализация стека, очереди и связанных списков. [N баллов]

Текст задачи: Реализуйте односвязный список с функциями вывода содержимого списка, добавления элемента в начало списка, удаления элемента с начала списка, добавления и удаления элемента после заданного элемента (key); поиска элемента в списке.

Решение:

```
class Node:
    def __init__(self, val: int, next=None) -> None:
        self.val = val
        self.next = next


class LinkedList:
    def __init__(self, root) -> None:
        self.root = root

    def prepend(self, val: any) -> None:
        new_node = Node(val)
        tmp = self.root
        new_node.next = tmp
        self.root = new_node

    def popleft(self) -> None:
        tmp = self.root
        tmp = tmp.next
        self.root = tmp

    def output(self) -> list:
        ll = []
        tmp = self.root
        while tmp:
            ll.append(tmp.val)
            tmp = tmp.next
        return ll

    def search(self, key: any) -> bool:
        tmp = self.root
        while tmp:
            if tmp.val == key:
                return True
            tmp = tmp.next
        return False

    def insert_after(self, key: any, value: any) -> None:
        tmp = self.root
        while tmp:
            if tmp.val == key:
                new_node = Node(value)
                new_node.next = tmp.next
                tmp.next = new_node
```

```
                break
            tmp = tmp.next

    def delete_after(self, key):
        tmp = self.root
        while tmp:
            if tmp.val == key:
                if tmp.next:
                    tmp.next = tmp.next.next
                else:
                    break
            tmp = tmp.next
```

Объяснение решения:

Описание классов Node и LinkedList

Классы Node и LinkedList реализуют структуру данных "связный список". Давайте рассмотрим их работу.

Класс Node

Атрибуты:

val: хранит значение узла (целое число).

next: указывает на следующий узел в списке (по умолчанию устанавливается в None).

Конструктор: При создании нового узла инициализирует его значение и задает указатель на следующий узел.

Класс LinkedList

Атрибуты:

root: представляет корень (первый узел) связного списка.

Методы:

prepend(val: any) -> None: Добавляет новый узел со значением val в начало списка.

Создает новый узел и перенаправляет указатель next к текущему корню, после чего обновляет корень.

popleft() -> None: Удаляет первый элемент (узел) из списка, обновляя корень на следующий узел.

output() -> list:Возвращает список всех значений узлов в связном списке.

Обходит все узлы, добавляя их значения в список.

search(key: any) -> bool: Ищет узел со значением key в списке. Возвращает True, если узел найден; иначе False.

insert_after(key: any, value: any) -> None: Вставляет новый узел со значением value после узла со значением key.

Проходит по списку, пока не найдет узел с key, а затем обновляет указатели.

delete_after(key) -> None: Удаляет узел, следующий за узлом со значением key. Если узел с key найден и у него есть следующий узел, указатель на следующий узел перенаправляется, чтобы исключить его из списка.

Время выполнения, затраты памяти и результат:

Task #13.3 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	[] is_empty()	0.0	26.97	[]
Значения из примера	[19, 13, 10] prepend(), popleft()	0.0	26.97	[19, 13, 10]
Значения из примера	[19, 18, 17, 9, 8, 7] insert_after()	0.0	26.97	[19, 18, 17, 9, 8, 7]
Значения из примера	[19] delete_after()	0.0	26.97	[19]
Значения из примера	[16] delete_after()	0.0	26.97	[16]
Значения из примера	[21, 19, 18, 16] delete_after()	0.0	26.97	[21, 19, 18, 16]
Значения из примера	[21, 20, 19, 18, 17, 16] search(18)	0.0	26.97	True
Значения из примера	[21, 20, 19, 18, 17, 16] search(Hello)	0.0	26.97	False

Вывод по задаче: Классы Node и LinkedList предоставляют основные операции для работы со связным списком, включая добавление в начало, удаление первого элемента, поиск, вставку и удаление узлов.

Вывод по лабораторной работе

В ходе лабораторной работы мы научились реализовывать стек, очередь, дек, а также использовать их при решении различных задач. Также с помощью классов мы реализовали связанный список с различными функциями.