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 10

Выполнил:

Абакар Иссака Малли

К3141

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

**Содержание отчета**

**Оглавление**

[Содержание отчета 2](#_heading=h.30j0zll)

[2 задача. Очередь 3](#_heading=h.3znysh7)

[3 задача. Скобочная последовательность. Версия 1 5](#_heading=h.tyjcwt)

[6 задача. Очередь с минимумом 7](#_heading=h.3dy6vkm)

[7 задача. Максимум в движущейся последовательности 9](#_heading=h.1t3h5sf)

[10 задача. Очередь в пекарню 11](#_heading=h.4d34og8)

[13 задача⋆. Реализация стека, очереди и связанных списков 13](#_heading=h.17dp8vu)

[Вывод 16](#_heading=h.3rdcrjn)

**Задачи по варианту**

**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 109. Команда «–» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит 106

элементов.

• Формат входного файла (input.txt). В первой строке содержится M (1 ≤

M ≤ 106) – число команд. В последующих строках содержатся команды, по

одной в каждой строке.

• Формат выходного файла (output.txt). Выведите числа, которые удаляются из очереди с помощью команды «–», по одному в каждой строке. Числа

нужно выводить в том порядке, в котором они были извлечены из очереди.

Гарантируется, что извлечения из пустой очереди не производится.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

Листинг кода

class Queue:  
 def \_\_init\_\_(self, n: int):  
 self.queue = [None] \* n  
 self.head = 0  
 self.tail = 0  
 self.count\_el = 0  
 self.n = n  
  
 def append(self, item):  
 if self.count\_el < self.n:  
 self.queue[self.tail] = item  
 self.tail = (self.tail + 1) % self.n  
 self.count\_el += 1  
 else:  
 raise "Queue overflow"  
  
 def pop(self):  
 if self.count\_el > 0:  
 temp\_per = self.queue[self.head]  
 self.head = (self.head + 1) % self.n  
 self.count\_el -= 1  
 return temp\_per  
 else:  
 raise "Queue empty"  
  
  
 def print(self):  
 if self.head > self.tail:  
 print(self.queue[self.head:] + self.queue[:self.tail])  
 else:  
 print(self.queue[self.head:self.tail])

Текстовое объяснение решения.

Создаем класс, который представляет собой очередь, добавляем и убираем элементы с помощью перемещения указателей.

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях: Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00046 sec | 14.70 Mb |
| Пример из задачи | 0.00099 sec | 14.89 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.75439 sec | 34.30 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**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.

• Формат входного файла (input.txt). Первая строка входного файла содержит число N (1 ≤ N ≤ 500) – число скобочных последовательностей,

которые необходимо проверить. Каждая из следующих N строк содержит

скобочную последовательность длиной от 1 до 104

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.

Листинг кода

def valid\_brackets(s):  
 stack = []  
  
 for ch in s:  
 if ch in "([":  
 stack.append(ch)  
 elif ch in ")]":  
 if len(stack) == 0 or ("(" != stack[-1] if ch == ")" else "[" != stack[-1]):  
 return False  
 else:  
 stack.pop()  
  
 return len(stack) == 0

Текстовое объяснение решения.

Если скобка открывающая – добавляем, иначе проверяем совпадает ли с последней в стэке.

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00575 sec | 14.86 Mb |
| Пример из задачи | 0.0061 sec | 14.89 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.67889 sec | 34.50 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**6 задача. Очередь с минимумом**

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 109.

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 ≤

M ≤ 106) – 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

Листинг кода

class QueueMin:  
 def \_\_init\_\_(self, n: int):  
 self.queue = [None] \* n  
 self.min\_deque = deque(maxlen=n)  
 self.head = 0  
 self.tail = 0  
 self.count\_el = 0  
 self.n = n  
  
 def append(self, item):  
 if self.count\_el < self.n:  
 self.queue[self.tail] = item  
 self.tail = (self.tail + 1) % self.n  
 self.count\_el += 1  
  
 while self.min\_deque and self.min\_deque[-1] > item:  
 self.min\_deque.pop()  
 self.min\_deque.append(item)  
  
 else:  
 raise IndexError("Queue overflow")  
  
 def pop(self):  
 if self.count\_el > 0:  
 temp\_per = self.queue[self.head]  
 self.head = (self.head + 1) % self.n  
 self.count\_el -= 1  
  
 if self.min\_deque and temp\_per == self.min\_deque[0]:  
 self.min\_deque.popleft()  
  
 return temp\_per  
 else:  
 raise IndexError("Queue empty")  
  
 def min(self):  
 if self.min\_deque:  
 return self.min\_deque[0]  
 raise IndexError("Queue empty")  
  
  
 def print(self):  
 if self.head > self.tail:  
 print(self.queue[self.head:] + self.queue[:self.tail])  
 else:  
 print(self.queue[self.head:self.tail])

Текстовое объяснение решения.

Делаем аналогично обычной очереди через класс, только при добавлении и удалении обновляем дек минимума

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях: Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00584 sec | 15.2 Mb |
| Пример из задачи | 0.00627 sec | 15.23 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.65656 sec | 35.43 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы O(n), а также используемую память.

**7 задача. Максимум в движущейся последовательности**

Задан массив из n целых чисел - a1, ..., an и число m < n, нужно найти максимум среди последовательности ("окна") {ai, ..., ai+m−1} для каждого значения1 ≤ i ≤ n − m + 1. Простой алгоритм решения этой задачи за O(nm) сканирует каждое "окно"отдельно.

Ваша цель - алгоритм за O(n).

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

число n (1 ≤ n ≤ 105) – количество чисел в исходном массиве, вторая строка содержит n целых чисел a1, ..., an этого массива, разделенных пробелом (0 ≤ ai ≤ 105). В третьей строке - целое число m - ширина "окна"(1 ≤ m ≤ n).

• Формат выходного файла (output.txt). Нужно вывести max ai, ..., ai+m−1 для каждого 1 ≤ i ≤ n − m + 1.

• Ограничение по времени. 5 сек.

• Ограничение по памяти. 512 мб

Листинг кода

def max\_slide\_window(arr, w\_len):  
  
 ans\_arr = []  
 deq = deque()  
  
 for ind, item in enumerate(arr):  
 while deq and arr[deq[-1]] <= item:  
 deq.pop()  
  
 deq.append(ind)  
  
 if deq[0] == ind - w\_len:  
 deq.popleft()  
  
 if ind + 1 >= w\_len:  
 ans\_arr.append(arr[deq[0]])  
  
 return ans\_arr

Текстовое объяснение решения.

Постепенно удаляя и добавляя в дек элементы поддерживаем максимум в окне

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00524 sec | 14.98 Mb |
| Пример из задачи | 0.00643 sec | 14.80 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.70423 sec | 32.34 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**10 задача. Очередь в пекарню**

Формат входного файла (input.txt). В первой строке вводится натуральное число N, не превышающее 100 - количество покупателей. В следующих N строках вводятся времена прихода покупателей - по два числа, обозначающие часы и минуты (часы - от 0 до 23, минуты - от 0 до 59) и степень его нетерпения (неотрицательное целое число не большее 100) - максимальное количество человек, которое он готов ждать впереди себя в очереди. Времена указаны в порядке возрастания (все времена различны). Гарантируется, что всех покупателей успеют обслужить до полуночи. Если для каких-то покупателей время окончания обслуживания одного покупателя и время прихода другого совпадают, то можно считать, что в начале заканчивается обслуживание первого покупателя, а потом приходит второй покупатель. 9 • Формат выходного файла (output.txt). В выходной файл выведите N пар чисел: времена выхода из пекарни 1-го, 2-го, ..., N-го покупателя (часы и минуты). Если на момент прихода покупателя человек в очереди больше, чем степень его нетерпения, то можно считать, что время его ухода равно времени прихода.

Листинг кода:

def bakery\_queue(n,arr\_data):  
 deq = deque()  
  
 arr\_ans = [""]\*n  
 ind\_now = 0  
 time\_end = arr\_data[0][0]+10  
 deq.append(arr\_data[0])  
 smth\_add = True  
 ind\_now += 1  
 while ind\_now < n:  
 if arr\_data[ind\_now][0] > time\_end and not deq:  
 deq.append(arr\_data[ind\_now])  
 time\_end = arr\_data[ind\_now][0]  
 smth\_add = True  
  
 if not smth\_add:  
 while deq and arr\_data[ind\_now][0] < time\_end:  
 temp\_arr = deq.popleft()  
 arr\_ans[temp\_arr[2]] = convert\_to\_hours\_min(time\_end)  
 time\_end += 10  
 if ind\_now < n and deq and deq[-1] != arr\_data[ind\_now]:  
 deq.append(arr\_data[ind\_now])  
 smth\_add = False  
 while ind\_now < n and arr\_data[ind\_now][0] < time\_end:  
 if arr\_data[ind\_now][1] >= len(deq):  
 deq.append(arr\_data[ind\_now])  
 smth\_add = True  
 else:  
 arr\_ans[arr\_data[ind\_now][2]] = convert\_to\_hours\_min(arr\_data[ind\_now][0])  
 ind\_now += 1  
  
 temp\_arr = deq.popleft()  
  
 arr\_ans[temp\_arr[2]] = convert\_to\_hours\_min(time\_end)  
 time\_end += 10  
  
 while deq:  
 temp\_arr = deq.popleft()  
 arr\_ans[temp\_arr[2]] = convert\_to\_hours\_min(time\_end)  
 time\_end += 10  
  
 return arr\_ans  
  
  
def convert\_to\_hours\_min(num):  
 return f"{num//60} {num%60}"

Текстовое объяснение решения.

Анализируя очередь и время прихода, терпение покупателей, будем моделировать ситуацию относительно покупателей

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00772 sec | 15.12 Mb |
| Пример из задачи | 0.0087 sec | 15.19 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.69952 sec | 31.48 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**13 задача⋆. Реализация стека, очереди и связанных списков**

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

2. Реализуйте очередь на основе связного списка функциями Enqueue, Dequeue c проверкой на переполнение и опустошения очереди.

Листинг кода

class Node:  
 def \_\_init\_\_(self, data=None):  
 self.next = None  
 self.data = data  
  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.last = None  
  
 def is\_empty(self):  
 return self.last is None  
  
 def pop(self):  
 if not self.is\_empty():  
 temp\_data = self.last.data  
 self.last = self.last.next  
 return temp\_data  
 else:  
 raise IndexError("Stack is empty")  
  
 def push(self, data):  
 temp\_node = Node(data)  
 temp\_node.next = self.last  
 self.last = temp\_node  
  
 def print(self):  
 if not self.is\_empty():  
 last = self.last  
 temp\_arr = []  
 while last:  
 temp\_arr.append(str(last.data))  
 last = last.next  
 return " ".join(reversed(temp\_arr))

class Node:  
 def \_\_init\_\_(self, data=None):  
 self.next = None  
 self.data = data  
  
  
class Queue:  
 def \_\_init\_\_(self, max\_length=10\*\*3):  
 self.head = None  
 self.tail = None  
 self.max\_length = max\_length  
 self.length = 0  
  
  
 def enqueue(self, data):  
 if self.length == self.max\_length:  
 raise OverflowError("Queue is overflow")  
  
 temp\_node = Node(data)  
 if self.head is None:  
 self.head = temp\_node  
 self.tail = temp\_node  
 else:  
 self.tail.next = temp\_node  
 self.tail = temp\_node  
  
 self.length += 1  
  
  
 def dequeue(self):  
 if self.length == 0:  
 raise IndexError("Queue is empty")  
  
 temp\_data = self.head.data  
 self.head = self.head.next  
 self.length -= 1  
 return temp\_data  
  
  
 def is\_full(self):  
 return self.length == self.max\_length  
  
  
 def is\_empty(self):  
 return self.length == 0  
  
  
 def print(self):  
 if not self.is\_empty():  
 now\_node = self.head  
 temp\_arr = []  
 while now\_node:  
 temp\_arr.append(str(now\_node.data))  
 now\_node = now\_node.next  
 return " ".join(temp\_arr)

Текстовое объяснение решения.

По типу графов присоединяем ребенка к последнему звену структуры данных.

Результат работы на примере:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00198 sec | 14.85 Mb |
| Пример из задачи | 0.00275 sec | 14.91 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.55322 sec | 30.11 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**Вывод**

1. В этой лабораторной мы научились делать сами stack, queue различными способами (через перемещение указателей, связный список), а также узнали некоторые алгоритмы для них.
2. Алгоритм O(n) и O(nlogn) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
3. С помощью методов *time.perf\_counter()* и *psutil.Process().memory\_info().rss* можно отслеживать ресурсозатратность алгоритмов.