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 21

Выполнила:

Савченко А.С.

Санкт-Петербург 2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Стек	3
Задача №3. Скобочная последовательность. Версия 1	5
Задача №5.Стек с максимумом	9
Задача №9. Поликлиника	12
Вывод (по всей лабораторной)	16

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.

- 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	10
+ 1	1234
+ 10	
-	
+ 2	
+ 1234	
-	

Листинг кода:

```
def stackk(file):
   stack = []
   results = []
```

```
for line in file:
    c = line.strip()
    if c == '-':
        results.append(stack.pop())
    else:
        stack.append(int(c.split()[1]))

return results

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        m = int(file.readline().strip())
        results = stackk(file)

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

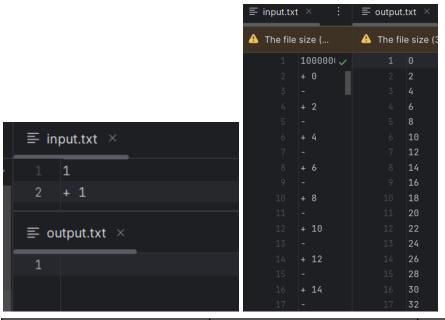
Текстовое объяснение решения:

Создаем пустой стек stack для хранения чисел и результирующий в котором будем хранить извлеченные. В цикле в зависимости от считанной команды - или + либо удаляем верхний элемент из стека и добавляем его в results, либо добавляем элемент в стек

Результат работы кода на примерах из текста задачи:(скрины input output файлов):

≡ in	put.txt ~		≡ 0ι	utput.txt
1	6	~	1	10
2	+ 1			1234
3	+ 10			
4				
5	+ 2			
6	+ 1234			
7				

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0003594 c	14.984375 Мб
Пример из задачи	0.0040779 с	14.8984375 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.3893281 c	16.47265625 Мб

Вывод по задаче:

Результаты тестирования доказывают эффективность структуры данных стек. Алгоритм справляется с обработкой большого объема данных , не превышая заданные ограничения по времени и памяти.

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:

- А 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
00	YES
([])	NO
(D)	NO
((]]	NO
)(

Листинг кода:

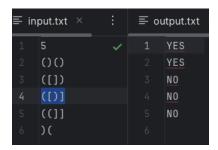
```
def brackets(s):
    stack = []
```

```
for char in s:
      if char in '([':
           stack.append(char)
       elif char == ')':
           if not stack or stack.pop() != '(':
               return "NO"
      elif char == ']':
          if not stack or stack.pop() != '[':
               return "NO"
   return "YES" if not stack else "NO"
if name == " main ":
  with open('input.txt', 'r') as file:
      n = int(file.readline().strip())
      results = []
      for line in file:
           results.append(brackets(line.strip()))
  with open('output.txt', 'w') as file:
       for result in results:
           file.write(result + '\n')
```

Текстовое объяснение решения:

Для каждой строки он поочередно добавляет открывающие скобки в стек и проверяет соответствие закрывающих скобок с вершиной стека. Если находим несовпадение сразу выводим NO. Дойдя до конца проверяем что стек пуст и в этом случае выводим YES

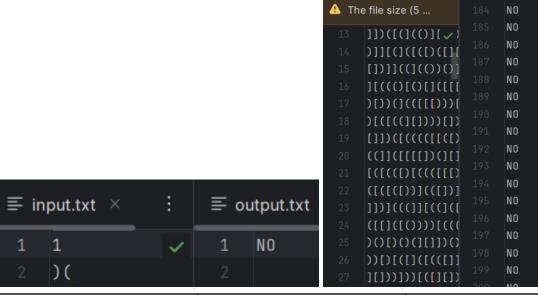
Результат работы кода на примерах из текста задачи:(скрины input output файлов):



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):

≡ input.txt ×

≡ output.txt



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004895 c	14.7265625 Мб
Пример из задачи	0.0005337 c	14.765625 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.011947 c	15.21875 Мб

Вывод по задаче:

Использование структуры данных стек позволяет эффективно проверять правильность скобочных последовательностей. Он быстро проверяет открывающиеся и закрывающиеся скобки и справляется со со всеми тестовыми входными данными.

Задача №5.Стек с максимумом

Текст задачи:

Стек - это абстрактный тип данных, поддерживающий операции Push() и Pop(). Нетрудно реализовать его таким образом, чтобы обе эти операции работали за константное время. В этой задаче ваша цель - реализовать стек, который также поддерживает поиск максимального значения и гарантирует, что все операции по-прежнему работают за константное время.

Реализуйте стек, поддерживающий операции Push(), Pop() и Max().

- Формат входного файла (input.txt). В первой строке входного файла содержится n ($1 \le n \le 400000$) число команд. Последющие n строк исходного файла содержит ровно одну команду: push V, pop или max. $0 \le V \le 10^5$.
- Формат выходного файла (output.txt). Для каждого запроса max выведите (в отдельной строке) максимальное значение стека.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
5	2	5	2	3	
push 2	2	push 1	1	push 1	
push 1		push 2		push 7	
max		max		pop	
pop		pop			
max		max			

Листинг кода:

```
class stack_w_max:
   def __init__(self):
     self.stack = []
   self.max_stack = []
```

```
def push(self, value):
       self.stack.append(value)
               if not self.max stack or value >=
self.max stack[-1]:
          self.max stack.append(value)
      if self.stack:
          value = self.stack.pop()
           if value == self.max stack[-1]:
               self.max stack.pop()
           return value
  def max(self):
      if self.max stack:
           return self.max stack[-1]
if name == " main ":
  with open('input.txt', 'r') as file:
      n = int(file.readline().strip())
      stack = stack w max()
      results = []
      for in range(n):
           command = file.readline().strip().split()
           if command[0] == "push":
               stack.push(int(command[1]))
          elif command[0] == "pop":
               stack.pop()
           elif command[0] == "max":
               results.append(stack.max())
  with open('output.txt', 'w') as file:
       for result in results:
```

file.write(str(result) + '\n')

Текстовое объяснение решения:

Напишем класс "максималоьного" стека. В нем заводим два списка stack со значениями значений, max stack для хранения максимальных значений.

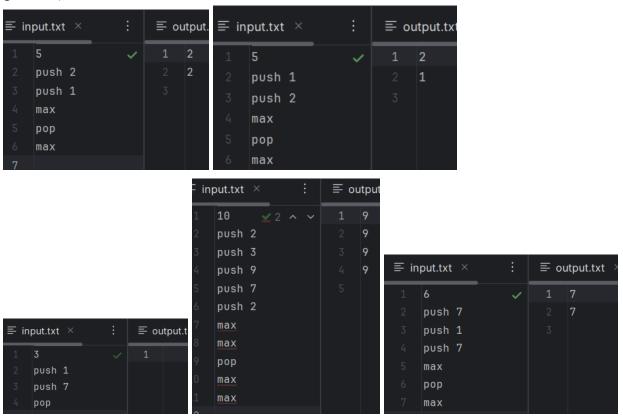
Метод push-loбавляет значение в стек stack, в нем, если значение больше или равно текущему максимуму, оно также добавляется в max stack.

Метод рор-удаляет значение из стека stack, в нем, удаляемое значение равно текущему максимуму, оно также удаляется из max stack.

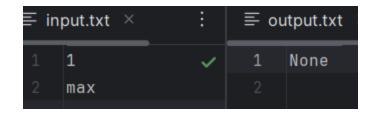
Метод тах-возвращает текущее максимальное значение из стека тах_stack.

Получение максимального значения из "максимального" стека выполняется за константное время, тк это = возврат последнего элемента из списка

Результат работы кода на примерах из текста задачи:(скрины input output файлов):



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005362 c	14.953125 Мб
Пример из задачи	0.0005048 c	14.8046875 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.1773777 c	15.265625 Мб

Вывод по задаче:

Мне удалось реализовать "максимальный" стек, который также поддерживает поиск максимального значения и гарантирует, что все операции по-прежнему работают за константное время.

Тк все операции выполняются за константное время видно что алгоритм работает очень быстро.

В этой задаче мне понравилось что приведено много тестов для проверки.

Задача №9. Поликлиника

Текст задачи:

9 задача. Поликлиника

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;
 - «* і» 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.

Листинг кола:

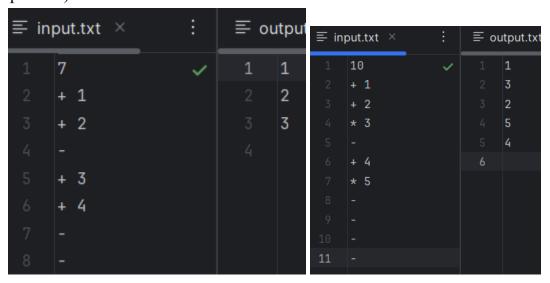
```
from time import perf counter ns
from psutil import Process
from os import getpid
start time = perf counter ns()
def vvod():
   from collections import deque
   def medecine(commands):
       q = deque()
       results = []
       for c in commands:
           if c.startswith('+'):
               _, i = c.split()
               q.append(i)
           elif c.startswith('*'):
               , i = c.split()
               mid = (len(q) + 1) // 2
```

```
q.insert(mid, i)
           elif c.startswith('-'):
               if q:
                   results.append(q.popleft())
       return results
   if name == " main ":
       with open('input.txt', 'r') as file:
           n = int(file.readline().strip())
              comm = [file.readline().strip() for in
range(n)]
       results = medecine(comm)
       with open('output.txt', 'w') as file:
           for result in results:
               file.write(result + \sqrt{n'})
vvod()
print('Время выполнения:', (perf counter ns()
start time) / 10 ** 9, 'c')
print('Затраты
                                              памяти: ',
Process(getpid()).memory info().rss / 1024 ** 2, 'M6')
```

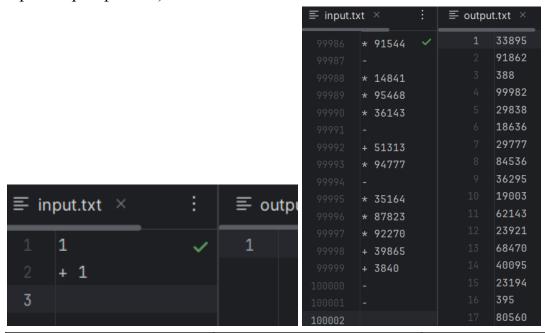
Текстовое объяснение решения:

Создаем двустороннюю очередь в которой будем сохранять пациентов, создаем результирующий список. В зависимости от команды будем выполнять действия: "+" - добавляем пациента в конец очереди .append; "*" - вычисляем индекс середины очереди, выполняем вставку пациента в середину очереди .insert; "-" удаляем первого пациента .popleft и одновременно возвращаем его номер в результирующий список. Выводим ответ

Результат работы кода на примерах из текста задачи:(скрины input output файлов):



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004883 c	14.87890625 Мб

Пример из задачи	0.0008506 c	15.00390625 M6
Верхняя граница диапазона значений входных данных из текста задачи	0.3141589 c	16.328125 Мб

Вывод по задаче:

"Пациенты, которым "только справку забрать встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром." - ВОТ ИМЕННО ИЗ ЗА ТАКИХ "придумок" возникает беспорядок в больницах!

Ну а вообщем оценивая задачу классическая очередь работает достаточно эффективно, но не все ее операции выполняются за константное время.

Чтобы ускорить алгоритм можно использовать встроенную в питон двустороннюю очередь deque. Так время улучшилось с 1.2750351 до 0.3141589 с, на памяти изменения не отразились ,ну это ожидаемо.

Вывод (по всей лабораторной)

Выполняя эту работу я несколько раз использовала такие структуры данных, как стек и очередь, а также их улучшенные версии стек с максимумом и двустороннюю очередь. Я убедилась в эффективности использования этих структур данных при работе с большими входными данными, ведь многие операции с ними имеют линейную сложность.