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 19

Выполнил:
Полегкий А.С.
К3142

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Задачи по варианту

Задача №1. Стек

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 ($1 \leq M \leq 10^6$) – 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

Листинг кода

```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = []
for i in range(n):
```

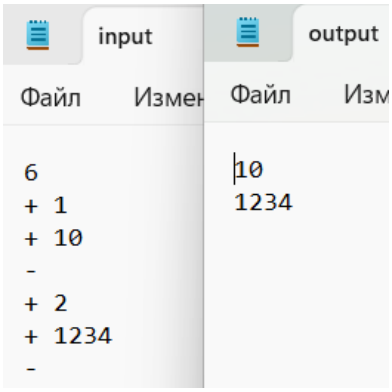
```
    s = list(f1.readline().split())
    if len(s) == 1:
        f2.write(a[-1] + "\n")
        a = a[:-1]
    else:
        a.append(s[-1])
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
```

Объяснение

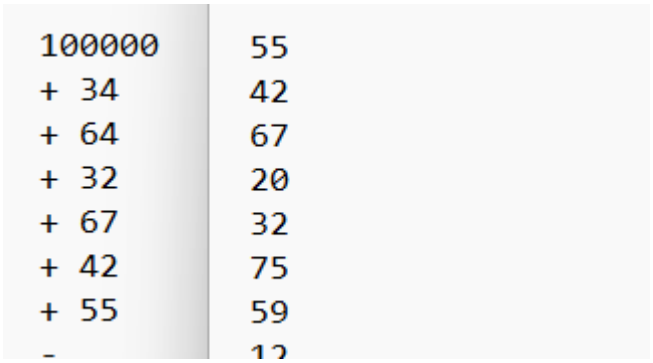
Написал небольшой стек, взяв за основу список, чтобы определить команды. Также добавил генератор, который помог в решении задачи.

```
import random

f1 = open("input.txt", "w")
n = int(input())
f1.write(str(n) + "\n")
k = 0
coms = ["+ ", "- "]
for i in range(n):
    if k <= 1:
        com = "+ "
    else:
        com = random.choice(coms)
    if com == "- ":
        f1.write(com + "\n")
        k -= 1
    else:
        k += 1
        f1.write(com + str(random.randint(1, 100)) +
"\n")
```

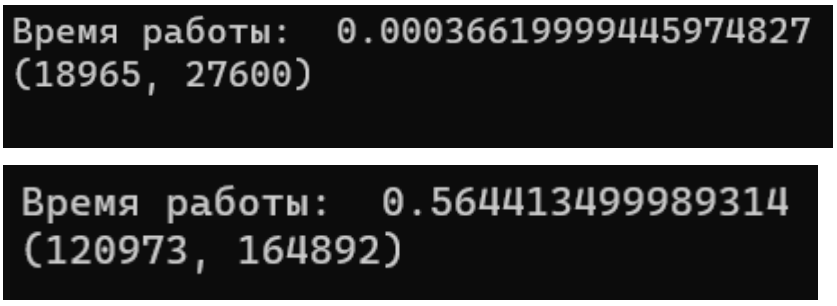
Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



Проверка задачи на acmp.



(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00036 сек	27600 bytes ≈ 27 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈0.56сек	164892 bytes ≈ 161 Kb

Вывод по задаче

Узнал, как работает стек, научился решать с ним задачи.

Задача №5. Стек с максимумом

5 задача. Стек с максимумом

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. Последующие 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
5 push 2 push 1 max pop max	2 2

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

input.txt	output.txt
3 push 1 push 7 pop	

Листинг кода

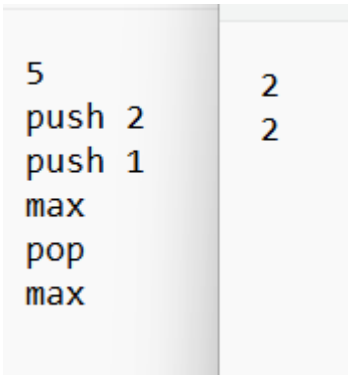
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
m = 0
a = []
for i in range(n):
```

```
    s = list(f1.readline().split())
    if len(s) == 2:
        a.append(int(s[-1]))
        if int(s[-1]) > m:
            m = int(s[-1])
    else:
        if s[0] == "pop":
            x = a.pop(-1)
            if x == m:
                if len(a) != 0:
                    m = max(a)
                else:
                    m = 0
        else:
            f2.write(str(m) + "\n")
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

Объяснение

Использовал pop, решил задачу методом, чем-то схожим в предыдущей.

Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях

```
100000
push 28
push 43
pop
push 63
max
push 10
pop
```
```
63
63
63
63
69
69
93
```

Проверка задачи на (acmp и тд при наличии в задаче).

```
Время работы:  0.00042749999556690454
(18928, 27566)
```

```
C:\WINDOWS\SYSTEM32\cmd
Время работы:  0.6240964999888092
(263195, 344374)
```

(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00042 сек	27566 bytes ≈ 27 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈0.62сек	344374 bytes ≈ 336.3Kb

Вывод по задаче

Реализовал стек, использовал pop, понял, как решать подобные задачи.

Задача №6. Очередь с минимумом

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 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 ($1 \leq M \leq 10^6$) – 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.
- Пример:

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

Листинг кода

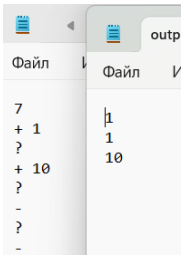
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
m = 999999999
```

```
a = []
for i in range(n):
    s = list(f1.readline().split())
    if len(s) == 2:
        a.append(int(s[-1]))
        if int(s[-1]) < m:
            m = int(s[-1])
    else:
        if s[0] == "-":
            x = a.pop(0)
            if x == m:
                if len(a) != 0:
                    m = min(a)
                else:
                    m = 999999999
        else:
            f2.write(str(m) + "\n")
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

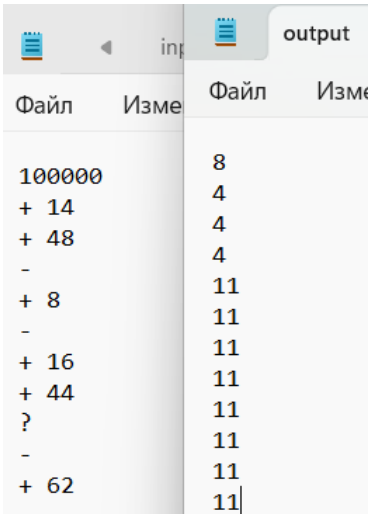
Объяснение

Решение задачи схожее с 5-ой, добавил pop.

Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).

```
Время работы:  0.0003880001493655145
(19041, 27706)
```

```
Время работы:  1.1046233999950346
(266911, 386084)
```

(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00038 сек	27706 bytes ≈ 27 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈1.1 сек	386084 bytes ≈ 377 Kb

Вывод по задаче

Реализовал работу очереди доступными мне способами.

Задача №9. Поликлиника

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 ($1 \leq n \leq 10^5$) - число запросов к вашей программе. В следующих n строках заданы описания запросов в следующем формате:
  - «+ i» – к очереди присоединяется пациент i ($1 \leq i \leq N$) и встает в ее конец;
  - «* i» – пациент i встает в середину очереди ($1 \leq i \leq N$);
  - «-» – 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 + 2 - + 3 + 4 - -	1 2 3	10 + 1 + 2 * 3 - + 4 * 5 - - - -	1 3 2 5 4

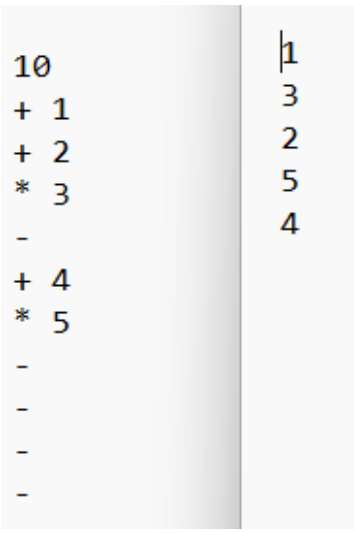
Листинг кода

```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = []
for i in range(n):
    s = list(f1.readline().split())
    if len(s) == 2:
        if s[0] == "+":
            a.append(s[-1])
        else:
            b = list(s[-1])
            a = a[:len(a) // 2 + len(a) % 2] + b +
a[len(a)//2 + len(a) % 2:]
    else:
        f2.write(str(a.pop(0)) + "\n")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
```

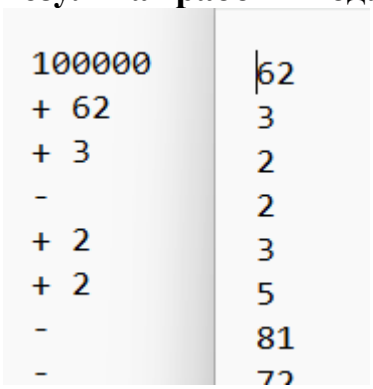
Объяснение

Через файл создаю очередь. Считываю этот файл. Потом через функцию определяю, с каким пациентом мы имеем дело. В другом файле вывожу результат.

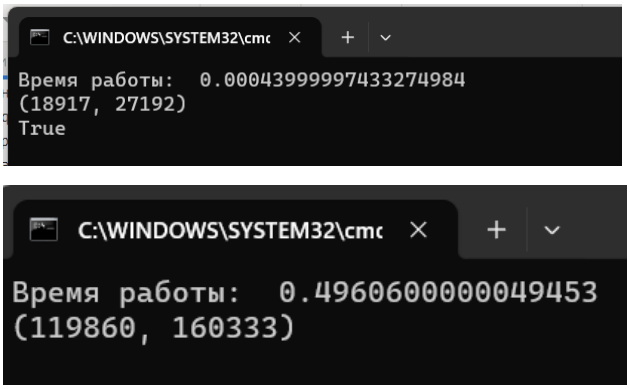
Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных и минимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).



(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00044сек	27192 bytes ≈ 26,55Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈0.49сек	160333bytes ≈ 156Kb

Вывод по задаче

Больше не пойду в поликлинику…

Задача №2. Очередь

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.

- **Формат входного файла (input.txt).** В первой строке содержится M ($1 \leq M \leq 10^6$) – 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.
- Пример:

input.txt	output.txt
4 + 1 + 10 - -	1 10

Листинг кода

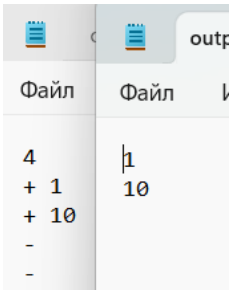
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = []
for i in range(n):
    s = list(f1.readline().split())
    if len(s) == 1:
        f2.write(a.pop(0) + "\n")
    else:
```

```
        a.append(s[-1])
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

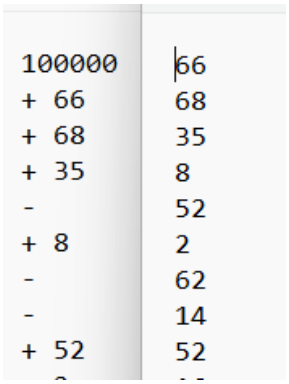
Объяснение

Решал задачу с помощью «очереди».

Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных и минимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).

(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00037сек	27025 bytes ≈ 26.56Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈0.46сек	158223bytes ≈ 154.5Kb

Вывод по задаче

Реализовал работу очереди.

Задача №8. Постфиксная запись

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.

- **Формат входного файла (input.txt).** В первой строке входного файла дано число N ($1 \le n \le 10^6$) – число элементов выражения. Во второй строке содержится выражение в постфиксной записи, состоящее из N элементов. В выражении могут содержаться неотрицательные однозначные числа и операции +, -, *. Каждые два соседних элемента выражения разделены ровно одним пробелом.
- **Формат выходного файла (output.txt).** Необходимо вывести значение записанного выражения. Гарантируется, что результат выражения, а также результаты всех промежуточных вычислений, по модулю будут меньше, чем 2^{31}.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
7 8 9 + 1 7 - *	-102

Листинг кода

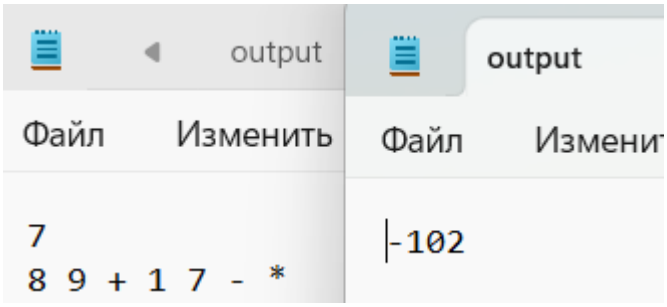
```
import time, tracemalloc, random, sys

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = []
s = list(f1.readline().split())
for x in s:
    if x.isdigit():
        a.append(int(x))
    else:
        n, m = a.pop(), a.pop()
```

```
        if x == "+":
            res = m + n
        elif x == "*":
            res = m * n
        elif x == "-":
            res = m - n
        a.append(res)
f2.write(str(a[0]))
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

Объяснение

Отправлял числа в стек, если было не число, то используем 2 из стека и делаем ту же операцию.

Результат работы кода на примерах из текста задачи



Проверка задачи на (acmp и тд при наличии в задаче).

```
Время работы:  0.0004247999968356453
(19593, 27232)
```

(пример вывода консоли для примера из задачи)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00042сек	27232 bytes ≈ 26.7Kb

Вывод по задаче

Простая задача, но всё равно интересная. Правда простите, не написал рандомизированный тест, здесь тоже его писать будет мне кажется сложнее чем саму задачу(чтобы всегда работало).

Вывод

В ходе выполнения лабораторной работы вспомнил как работают стеки и очереди.