САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5
по курсу «Алгоритмы и структуры данных»
Тема: Стек, очередь, связанный список
Вариант 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №1. Стек

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая стро ка содержит одну команду. Команда — это либо “+ *N*”, либо “–”. Команда “+ *N*”означает добавление в стек числа *N*, по модулю не превышающего 109. Ко манда “–”означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выпол нения команд не превысит 106элементов.

- **Формат входного файла (input.txt).** В первой строке входного файла со держится M ($1 \le M \le 106$) – число команд. Каждая последующая строка исходного файла содержит ровно одну команду.
- **Формат выходного файла (output.txt).** Выведите числа, которые удаля ются 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:

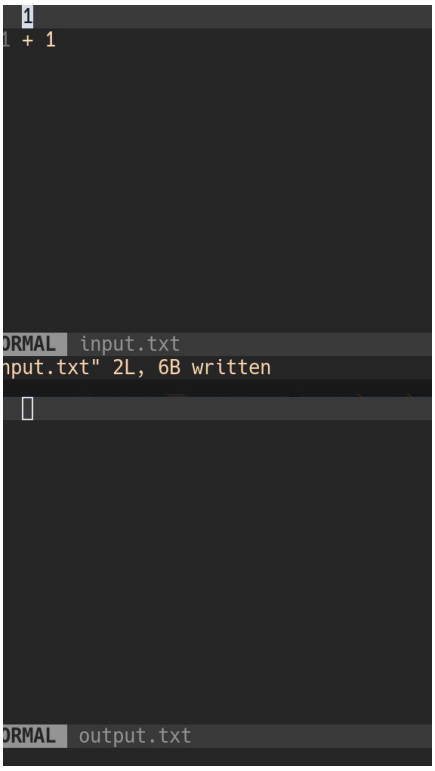
input.txt	output.txt
6 + 1 + 10 - + 2 + 1234 -	10 1234

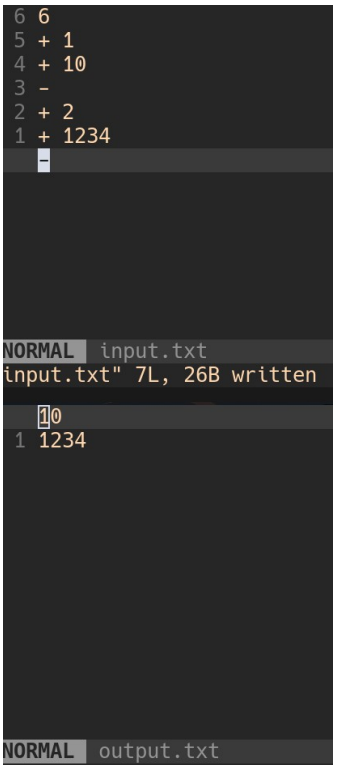
Код:

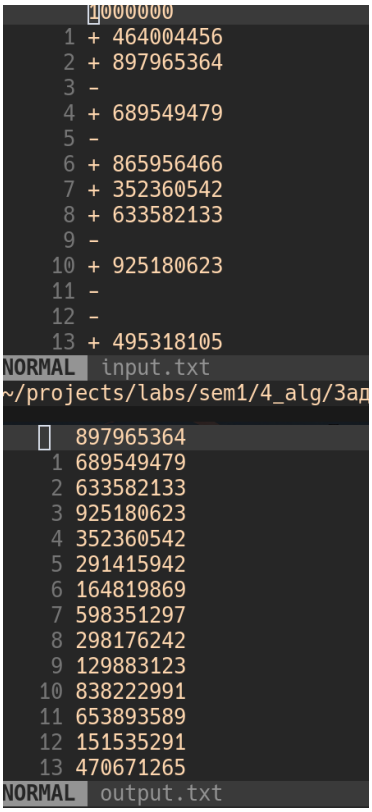
```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()


class Stack:
    memory = []

    def add(self, el):
        self.memory.append(el)

    def pop(self):
        del self.memory[-1]

    @property
    def top(self):
        try:
            return self.memory[-1]
        except:
            raise Exception("No elements in stack memory")


s = Stack()

n = int(lines[0])

out = []

for i in range(n):
    inputs = lines[i+1]
    if inputs[0] == "+":
        el = inputs.split()[1]
        s.add(el)
    else:
        out.append(s.top)
        s.pop()
open("output.txt", 'w').write("\n".join(map(str, out)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Для реализации структуры данных Stack в питоне был использован объектно-ориентированный подход. Стек реализован как класс для множества объектов.

Результат работы кода:







	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0003017129993168055	14920
Пример из задачи	0.0003048449998459546	15377
Верхняя граница диапазона значений входных данных из текста задачи	1.378559086999303	107460285

Вывод по задаче:
Стек — одна из структур реализующих базовый функционал массивов. Можно утверждать, что стеки устарели, поскольку современные компьютеры могут эффективно и удобно работать с памятью.

Задача №4. Скобочная последовательность. Версия 2

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 ко торая 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 совпа дающих скобок, так что в каждой паре открывающая скобка идет перед закрыва ющей 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) .

- **Формат входного файла (input.txt).** Входные данные содержат одну стро ку S, состоящую из больших и маленьких латинских букв, цифр, знаков препинания и скобок из набора [] (). Длина строки $S - 1 \leq S$ *leq*105.

- **Формат выходного файла (output.txt).** Если в строке *S* скобки исполь зуются правильно, выведите «Success» (без кавычек). В противном случае выведите отсчитываемый от 1 индекс первой несовпадающей закрывающей скобки, а если нет несовпадающих закрывающих скобок, выведите отсчиты ваемый от 1 индекс первой открывающей скобки, не имеющей закрывающей.

  • Ограничение по времени. 5 сек.

  • Ограничение по памяти. 256 мб.

  • Примеры:

input.txt	output.txt
[]	Success
{}[]	Success
[()]	Success
(())	Success
{	1
{[}	3
foo(bar);	Success
foo(bar[i);	10

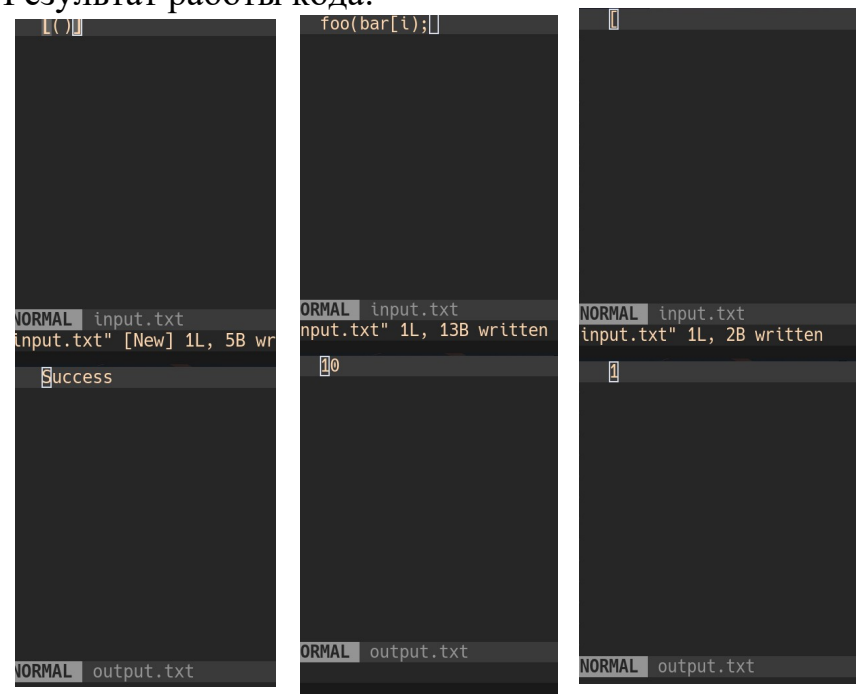
Код:

```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()


class Stack:
    memory = []

    def add(self, el):
        self.memory.append(el)
```

```
    def pop(self):
        self.memory.pop()

    @property
    def top(self):
        try:
            return self.memory[-1]
        except:
            return (None, None)

    @property
    def len(self):
        return len(self.memory)


s = Stack()

seq = lines[0]
n = len(seq)

allowed = "({[]})"
closing = "]})"
open_for_close = {
    "]": "[",
    "}": "{",
    ")": "(",
}

out = open("output.txt", 'w')

for i in range(n):
    el = seq[i]
    if el in allowed:
        if el not in closing:
            s.add((seq[i], i+1))
        else:
            if open_for_close[el] == s.top[0]:
                s.pop()
            else:
                out.write(str(i+1))
                print("Время выполнения: " +
                      str(time.perf_counter() - t_start) + " секунд")
                print("Использование памяти: " +
                      str(tracemalloc.get_traced_memory()[1]) + " байт")
                tracemalloc.stop()
                exit()
if s.len > 0:
    out.write(str(s.top[1]))
     print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
    print("Использование памяти: " +
          str(tracemalloc.get_traced_memory()[1]) + " байт")
```

```
    tracemalloc.stop()
    exit()
else:
    out.write("Success")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Для решения данной задачи использовалась структура данных стек. Стек позволяет удобно брать последний элемент из данных.
Результат работы кода:



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001823839993448928	14923
Пример из задачи	0.0001552920002723112 7	14846
Пример из задачи	0.0001731159991322783 8	14985
Верхняя граница диапазона значений входных данных из текста задачи	0.0003226219996577129	206086

Вывод по задаче:
Для решения задач можно обойтись без использования стека. Однако данная структура удобна в использовании в некоторых случаях.

Задача №6. 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 элемен та 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 ми нимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Га рантируется, 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

- Вам может помочь идея, изложенная во второй части вот этой страницы.

Код:

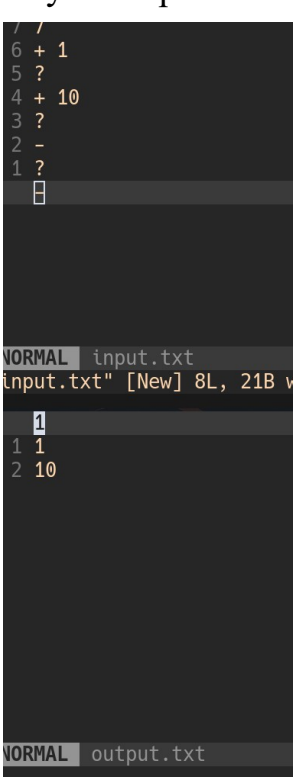
```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()


class Queue:
    memory = []
    head = 0
    tail = 0
    length = -1
```

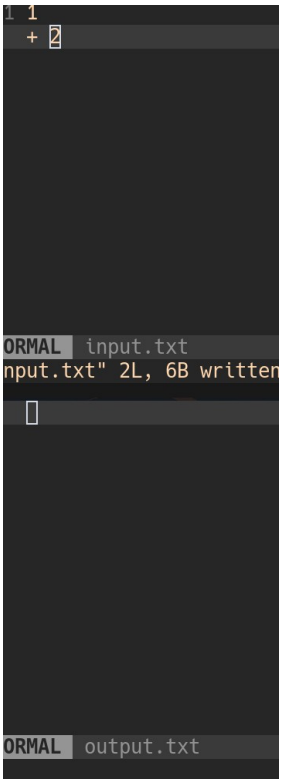
```
    def __init__(self, n):
        self.length = n+1
        self.memory = [None] * (n+1)

    def enqueue(self, el):
        self.memory[self.tail] = el
        if self.tail == self.length - 1:
            self.tail = 0
        else:
            self.tail += 1
        if self.tail == self.head:
            raise Exception("Nicht genug Speicher! Scheiße!")

    def dequeue(self):
        if self.head == self.tail:
                raise Exception("Scheßie! Es gibt keine Elementen im Queue
mehr!")
        self.memory[self.head] = None

        if self.head == self.length - 1:
            self.head = 0
        else:
            self.head += 1

    @property
    def min(self):
        if self.head == self.tail:
            return None

        head = self.head
        mini = self.memory[head]
        tail = self.length - 1 if self.tail == 0 else self.tail - 1
        while head != tail:
            if head == self.length - 1:
                head = 0
            else:
                head += 1
            if self.memory[head] < mini:
                mini = self.memory[head]
        return mini


n = int(lines[0])
s = Queue(n)
out = open("output.txt", "w")

for i in range(n):
    inputs = lines[i+1]
    if inputs[0] == "+":
        el = inputs.split()[1]
        s.enqueue(el)
    elif inputs[0] == "-":
```

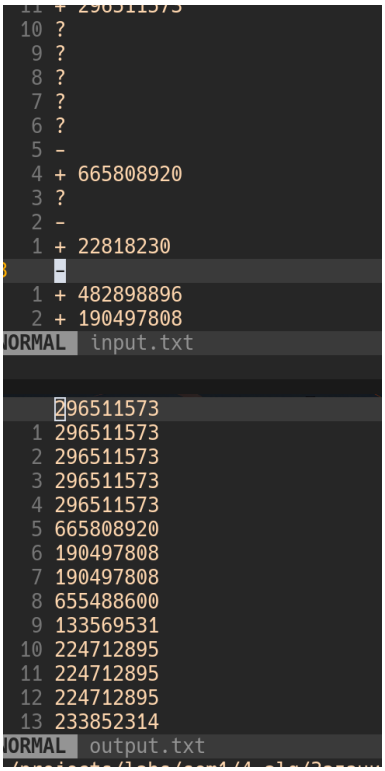
```
        s.dequeue()
    else:
        out.write(str(s.min) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Для реализации поиска минимального элемента использовались переменные head и tail. Их использование обосновано целью избежать обхода циклом всей доступной памяти, чтобы повысить эффективность алгоритма.

Результат работы кода:







	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001588350005476968 4	13941
Пример из задачи	0.0001851879987952998	14366
Верхняя граница диапазона значений входных данных из текста задачи	0.6100967649999802	3645964

Вывод по задаче:
Очередь — структура данных, которая в отличии от стека действительно улучшает эффективность кода. С помощью очереди можно эффективно удалять элементы не с конца, а с начала

Задача №10. Очередь в пекарню

В пекарне работает один продавец. Он тратит на одного покупателя ровно 10 минут, а затем сразу переходит к следующему, если в очереди кто-то есть, либо ожидает, когда придет следующий покупатель. Даны времена прихода покупате лей в пекарню (в том порядке, в котором они приходили). Также у каждого поку пателя есть характеристика, называемая *степенью нетерпения*. Она показывает, сколько человек может максимально находиться в очереди перед покупателем, чтобы он дождался своей очереди и не ушел раньше. Если в момент прихода по купателя в очереди находится больше людей, чем степень его нетерпения, то он решает не ждать своей очереди и уходит. Покупатель, который обслуживается в данный момент, также считается находящимся в очереди. Требуется для каждого покупателя указать время его выхода из пекарни.

- **Формат входного файла (input.txt).** В первой строке вводится натураль ное число *N*, не превышающее 100 - количество покупателей. В следующих *N* строках вводятся времена прихода покупателей - по два числа, обозна чающие 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. Време на 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.
- Ограничение по времени. Оцените время работы и используемую память при заданных максимальных значениях.
  - Пример:

input1.txt	output1.txt	input2.txt	output2.txt
3 10 0 0 10 1 1 10 2 1	10 10 10 20 10 2	5 13 0 100 14 0 0 14 1 0 14 2 3 14 3 0	13 10 14 10 14 1 14 20 14 3

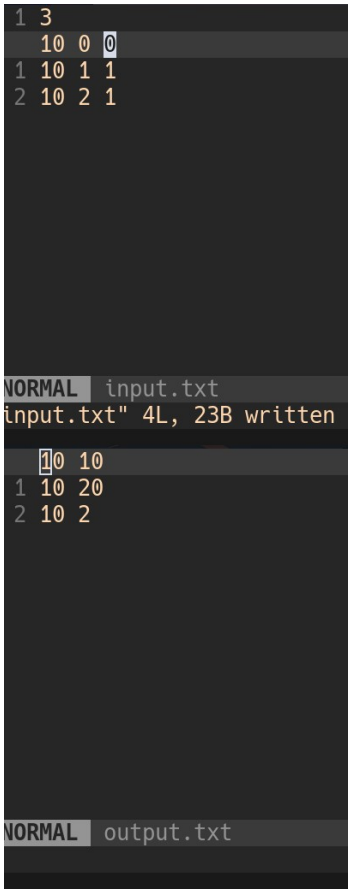
Код:

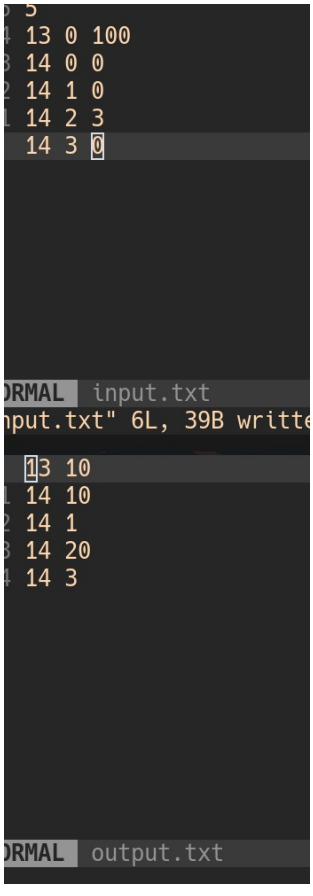
```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()


class Queue:
```

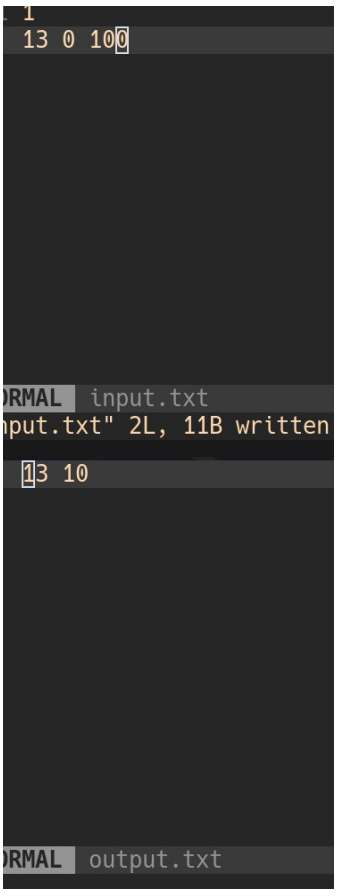
```
    memory = []
    head = 0
    tail = 0
    length = -1
    count = 0

    def __init__(self, n):
        self.length = (n+1)
        self.memory = [None] * (n+1)

    def enqueue(self, el):
        self.memory[self.tail] = el
        if self.tail == self.length - 1:
            self.tail = 0
        else:
            self.tail += 1
        if self.tail == self.head:
            raise Exception("Nicht genug Speicher! Scheiße!")
        self.count += 1

    def dequeue(self):
        self.memory[self.head] = None

        if self.head == self.length - 1:
            self.head = 0
        else:
            self.head += 1
        self.count -= 1

    @property
    def top(self):
        return self.memory[self.head]

    @property
    def len(self):
        return self.count


n = int(lines[0])
s = Queue(n)
h, m, tolerance = map(int, lines[1].split())

out = open("output.txt", "w")

unixtime = h * 60 + m + 10
s.enqueue((unixtime//60, unixtime % 60))
out.write(f"{unixtime//60} {unixtime % 60}\n")

for i in range(1, n):
    h, m, tolerance = map(int, lines[i+1].split())
    h1, m1 = s.top

    unixtime1 = h*60+m
```

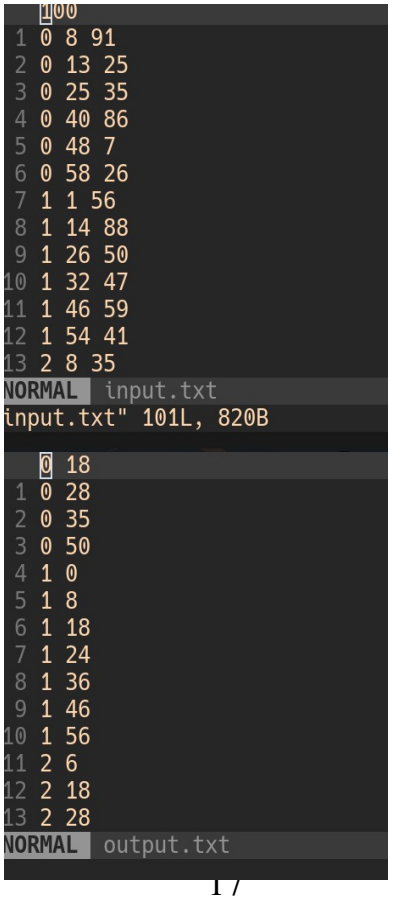
```
        unixtime2 = h1*60+m1

        while unixtime1 > unixtime2:
            s.dequeue()
            if s.len == 0:
                unixtime2 = unixtime1
                break
            h1, m1 = s.top
            unixtime1 = h*60+m

        if tolerance >= s.len:
            unixtime2 += 10
            h = unixtime2 // 60
            m = unixtime2 % 60
            s.enqueue((h, m))
            out.write(f"{h} {m}\n")
        else:
            out.write(f"{h} {m}\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Для решения данной задачи была использована очередь. Была добавлена функция для вывода первого элемента без его удаления
Результат работы кода:









	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00020964699979231227	15328
Пример из задачи	0.00023108899949875195	15683
Пример из задачи	0.00023071000032359734	16314
Верхняя граница диапазона значений входных данных из текста задачи	0.0011567099991225405	29249

Вывод по задаче:
Очередь можно использовать для эффективного решения задач. Эта задача показала, что в конечном итоге если решать задачу без очереди, то с помощью стеков и массивов будет реализован алгоритм похожий на данную структуру.

Дополнительные задачи

Задача №2. 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 106элементов.

- **Формат входного файла (input.txt).** В первой строке содержится M ($1 \leq M \leq 106$) – число команд. В последующих строках содержатся команды, по одной в каждой строке.
- **Формат выходного файла (output.txt).** Выведите числа, которые удаляют ся 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

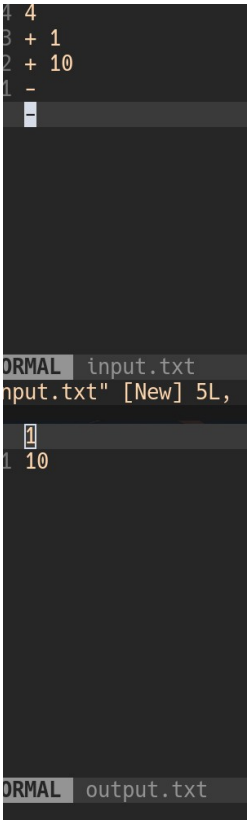
Код:

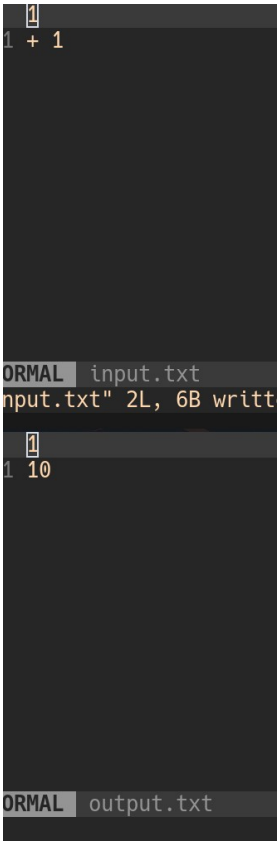
```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()


class Queue:
    memory = []
    head = 0
    tail = 0
    length = -1

    def __init__(self, n):
        self.length = n+1
        self.memory = [None] * (n+1)
```

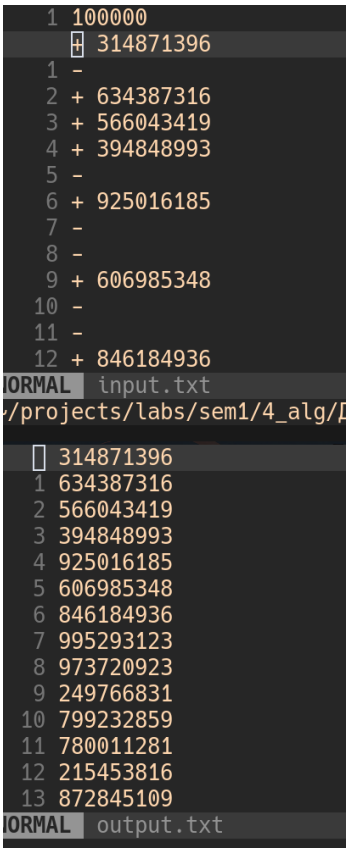
```
    def enqueue(self, el):
        self.memory[self.tail] = el
        if self.tail == self.length - 1:
            self.tail = 0
        else:
            self.tail += 1
        if self.tail == self.head:
            raise Exception("Nicht genug Speicher! Scheiße!")

    def dequeue(self):
        if self.head == self.tail:
                raise Exception("Scheßie! Es gibt keine Elementen im Queue
mehr!")
        el = self.memory[self.head]
        self.memory[self.head] = None

        if self.head == self.length - 1:
            self.head = 0
        else:
            self.head += 1
        return el


n = int(lines[0])
s = Queue(n)
out = open("output.txt", "w")

for i in range(n):
    inputs = lines[i+1]
    if inputs[0] == "+":
        el = inputs.split()[1]
        s.enqueue(el)
    else:
        out.write(f"{s.dequeue()}\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Куча была реализована с помощью объектно-ориентированного подхода. Класс может иметь множество объектов

Результат работы кода:







	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000190336999366991 22	13941
Пример из задачи	0.000235712999710813 16	14097
Верхняя граница диапазона значений входных данных из текста задачи	0.20998188699923048	7290305

Вывод по задаче: Очередь можно реализовать на любом языке программирования. Это одна из основных базовых структур данных.

Задача №3. 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 ско бочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

Входной файл содержит несколько строк, каждая из которых содержит после довательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

• **Формат входного файла (input.txt).** Первая строка входного файла со держит 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 104включительно. В каж дой 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 правиль ной 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

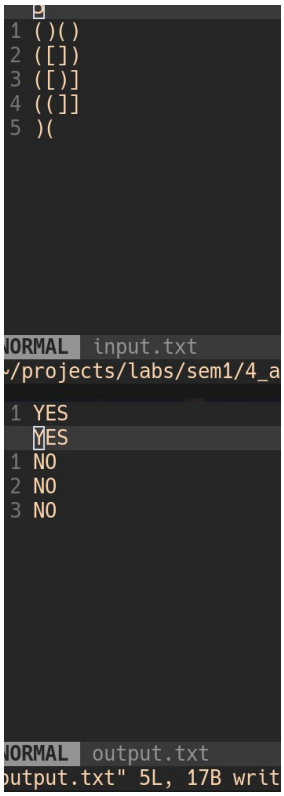
Код:

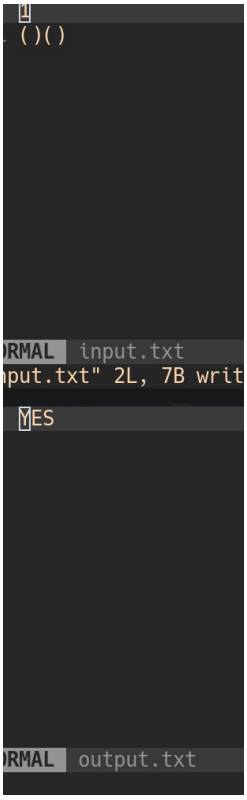
```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()

out = open("output.txt", 'w')


class Stack:
    memory = []

    def add(self, el):
        self.memory.append(el)

    def pop(self):
        self.memory.pop()

    @property
    def top(self):
        try:
            return self.memory[-1]
        except:
            return (None, None)

    @property
    def len(self):
        return len(self.memory)


n = int(lines[0])
```

```
allowed = "([])"
closing = "])"
open_for_close = {
    "]": "[",
    ")": "("
}
for _i in range(n):
    seq = lines[_i+1]
    m = len(seq)
    s = Stack()
    fail = False
    for i in range(m):
        el = seq[i]
        if el in allowed:
            if el not in closing:
                s.add((seq[i], i+1))
            else:
                if open_for_close[el] == s.top[0]:
                    s.pop()
                else:
                    out.write("NO\n")
                    fail = True
                    break;
    if not fail:
        if s.len > 0:
            out.write("NO\n")
        else:
            out.write("YES\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Для решения данной задачи была использована структура очередь. Был составлен алгоритм классификации символов на закрывающие и открывающие.

Результат работы кода:


1 ()()
2 ([])
3 ([)]
4 ((]]
5)(
NORMAL input.txt
~/projects/labs/sem1/4_a
1 YES
YES
1 NO
2 NO
3 NO
NORMAL output.txt
output.txt" 5L, 17B writ


()()
NORMAL input.txt
put.txt" 2L, 7B writ
YES
NORMAL output.txt


NORMAL input.txt text utf-8[unix] 501 words 0% ln :1/501 :1
put.txt" 501L, 500504B
NO
NO
NO
NO
NO
NO
NO
NO
NO
NO
NO
NO
NO
NORMAL output.txt[+] text utf-8[unix] 500 words 0% ln :1/499 :1

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00020803400184377097	15303
Пример из задачи	0.0002121279976563528 2	15303
Верхняя граница диапазона значений входных данных из текста задачи	0.003976686999521917	580101

Вывод по задаче:
Структуру данных куча можно использовать в прикладных задачах. Одним из примеров прикладных задач может послужить проверка синтаксиса в LaTeX.

Вывод

Стек — структура данных предназначенная упростить код разработчику, иными словами сделать его читабельным. Очередь же, помимо того что делает код более интуитивно понятным, ускоряет его. Связанные же списки ускоряют удаление элементов.