Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Выполнил студент группы.....КС-38..... Прилепский Артем Сергеевич
Ссылка на репозиторий: .....github.com/news1d/Algorithms_and_structures

Приняли: .....Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: 17.03.2023

Оглавление

Описание задачи.

1. В данной лабораторной работе необходимо было реализовать коллекцию очередь. Для этого использовал python.

   Основные требования к коллекции:

   - Использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными.
   - Реализовывать свой итератор с реализацией операторов ++ и !=
   - Обеспечивать работу стандартных библиотек и конструкции for each если она есть в языке.
   - Проверку на пустоту и подсчет количества элементов.
   - Операцию сортировки с использованием стандартной библиотеки.
   - Добавление в конец
   - Взятие с начала

2. Помимо реализации еще нужно было провести тесты ее работы:
   - Заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.
   - Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.
   - Заполнение контейнера 100 структур содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов не подходящих под условие в новых структурах.
   - Заполнить структуру 1000 элементов и отсортировать ее, проверить правильность использую структуру из стандартной библиотеки и сравнив результат.
   - (Стек и Очередь) Инверсировать содержимое контейнера заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

Описание метода/модели.

Очередь (англ. *queue*) — способ хранения данных. Данные хранятся последовательно, основная особенность очередь заключается в том, что кто первый зашел, тот первый вышел (FIFO – first-in, first-out).

Выполнение задачи.

Для реализации программы был выбран язык программирования Python. В моем случае очередью является класс, который в совей основе использует встроенный в Python контейнер list(), но с особенностями присущими очереди.

Код класса:

```
class Queue:

    def __init__(self, item=None):
        try:
            self.__queue = list(item)
        except Exception:
            self.  queue = list()

    # Добавляем элемент в конец
    def add(self, item) -> None:
        self.__queue.append(item)

    # Удаляем + возвращаем первый элемент
    def remove(self):
        if len(self.__queue) == 0:
            pass
        else:
            return self.__queue.pop(0)

    # Возвращаем длину
    def size(self):
        return len(self.__queue)

    # Проверка на пустоту
    def isempty(self) -> bool:
        return len(self.__queue) == 0

    # Возвращаем все элементы
    def get_queue(self):
        return self.__queue

    # Возвращаем объект итератора
    def __iter__(self):
        self.current = 0
        return self

    # Возвращаем следующий элемент очереди
    def __next__(self):
        if self.current < len(self.__queue):
            result = self.__queue[self.current]
            self.current += 1
            return result
        else:
            raise StopIteration
```

В классе также реализован механизм итератора, что позволяет нам работать с контейнером используя цикл. Представлен метод изъятия из начала и вставки в конец, проверки на пустоту и получение размера очереди. Т.к. Python является динамически типизированным языком программирования, в шаблонизации классов он не нуждается.

Тест №1. Заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.

1) Создаем объект очереди и заполняем его случайными числами из диапазона.
2) С помощью перебора очереди вычисляем нужные нам значения.

Код функции:

```
def test_1():
    container = Queue()
    for i in range(0, 1000):
        container.add(random.randint(-1000, 1000))

    max = -2000
    min = 2000
    sum = 0
    for item in container:
        sum += item
        if item > max:
            max = item
        elif item < min:
            min = item

    avg = sum // container.size()
    print(f"TEST 1:\nMAX = {max}\nAVG = {avg}\nMIN = {min}\nSUM = {sum}")
```

```
TEST 1:
MAX = 999
AVG = -15
MIN = -999
SUM = -14754
```

Результат работы программы

Тест №2. Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.

1) Создаем объект очереди и заполняем её данными типа string.
2) Добавляем новый элемент в конец очереди.
3) Удаляем элемент из начала очереди.

Код функции:

```
def test_2():
    container = Queue()

    for i in range(0, 10):
        container.add(f"item_{i}")

    print("\nTEST 2:")
    print("Очередь до добавления элементов: ", end="")
    for item in container:
        print(item, end=" ")

    container.add("new")

    print("\nОчередь после добавления элементов: ", end="")
    for item in container:
        print(item, end=" ")
```

```
    container.remove()

    print("\nОчередь после удаления элементов: ", end="")
    for item in container:
        print(item, end=" ")
```

```
Очередь до добавления элементов: item_0 item_1 item_2 item_3 item_4 item_5 item_6 item_7 item_8 item_9
Очередь после добавления элементов: item_0 item_1 item_2 item_3 item_4 item_5 item_6 item_7 item_8 item_9 new
Очередь после удаления элементов: item_1 item_2 item_3 item_4 item_5 item_6 item_7 item_8 item_9 new
```

Результат работы программы

Тест №3. Заполнение контейнера 100 структур содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов не подходящих под условие в новых структурах.

1) Создаем объект очереди.
2) Создаем 3 списка, хранящих в себе имя, фамилию и отчество.
3) Создаем 100 структур людей и заполняем их случайными данными из списков, созданных ранее.
4) С помощью перебора очереди сортируем структуры по новым очередям, согласно условию задания.

Код функции:

```
def test_3():
    container = Queue()
    name = ["Вадим", "Максим", "Егор", "Георгий", "Михаил", "Сергей", "Глеб",
"Александр", "Анатолий", "Андрей"]
    surname = ["Дмитриев", "Смирнов", "Киселев", "Степанов", "Волков", "Петров",
"Леонов", "Шаповалов", "Зубов", "Сидоров"]
    patronymic = ["Александрович", "Глебович", "Вадимович", "Федорович",
"Борисович", "Игоревич", "Кириллович", "Максимович", "Павлович", "Николаевич"]
    start_date = datetime.date(1980, 1, 1)
    end_date = datetime.date(2020, 1, 1)
    delta = end_date - start_date


    for i in range(0, 100):
        person = []

        random_days = random.randint(0, delta.days)
        random_date = start_date + datetime.timedelta(days=random_days)

        person.append(random.choice(name))
        person.append(random.choice(surname))
        person.append(random.choice(patronymic))
        person.append(random_date)

        container.add(person)

    less_than_20 = Queue()
    more_than_30 = Queue()
    other_people = Queue()
    for item in container:
        if ((datetime.date.today() - item[3]).days / 365) < 20:
```

```
            less_than_20.add(item)
        elif ((datetime.date.today() - item[3]).days / 365) > 30:
            more_than_30.add(item)
        else:
            other_people.add(item)
    print("\n\nTEST 3:")
    print(f"Количество людей старше 30 лет: {more_than_30.size()}")
    print(f"Количество людей младше 20 лет: {less_than_20.size()}")
    print(f"Количество остальных людей: {other_people.size()}")
```

```
TEST 3:
Количество людей старше 30 лет: 39
Количество людей младше 20 лет: 44
Количество остальных людей: 17
```

Результат работы программы

Тест №4. Заполнить структуру 1000 элементов и отсортировать ее, проверить правильность используя структуру из стандартной библиотеки и сравнив результат.

1) Создаем объект очереди и заполняем его случайными элементами.
2) Сортируем очередь используя стандартную функцию.

Код функции:

```
def test_4():
    container = Queue()

    for i in range(0, 1000):
        container.add(random.randint(-1000, 1000))

    print("\nTEST 4:")
    print("Очередь до сортировки:")
    for item in container:
        print(item, end=" ")

    container = Queue(sorted(container.get_queue()))

    print("\nОчередь после сортировки:")
    for item in container:
        print(item, end=" ")
```

```
TEST 4:
Очередь до сортировки:
188 174 -892 408 695 -897 -444 -614 150 674 848 992 723 893 952 288 -11 448 -623 -972
Очередь после сортировки:
-1000 -998 -997 -993 -992 -990 -986 -986 -986 -983 -981 -979 -979 -977 -972 -968 -967
```

Результат работы программы

Тест №5. Инверсировать содержимое контейнера заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

1) Создаем объект очереди и заполняем его случайными числами.
2) Сортируем очередь используя стандартную функцию.

3) Извлекаем элементы из очереди в новый список до тех пор, пока очередь не опустеет.

4) Извлекаем элементы из списка и добавляем их в очередь до тех пор, пока список не опустеет.

Код функции:

```
def test_5():
    container = Queue()

    for i in range(0, 1000):
        container.add(random.randint(-1000, 1000))

    print("\n\nTEST 5:")
    print("Исходная очередь:")
    for item in container:
        print(item, end=" ")

    container = Queue(sorted(container.get_queue()))

    print("\nОтсортированная очередь:")
    for item in container:
        print(item, end=" ")

    list_tmp = []
    while not container.isempty():
        list_tmp.append(container.remove())

    while len(list_tmp) > 0:
        container.add(list_tmp.pop())

    print("\nИнвертированная очередь:")
    for item in container:
        print(item, end=" ")
```

```
TEST 5:
Исходная очередь:
185 510 -178 135 49 -243 -747 -813 494 -781 -811 -382 -507 734 -345 576 -718 -801 851 -5
Отсортированная очередь:
-999 -997 -997 -991 -988 -985 -979 -979 -978 -978 -974 -970 -968 -963 -961 -961 -960 -95
Инвертированная очередь:
997 991 984 979 979 977 976 976 973 972 972 966 964 962 960 958 956 956 955 954 952 948
```

Результат работы программы

Заключение.

Благодаря динамической типизации языка Python, реализация очереди является не таким уж и сложным занятием. Все тесты, указанные в задании, пройдены положительно.