МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Информатика»
Тема: Алгоритмы и структуры данных в Python

Студент гр. 8381 _____ Почаев Н.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

Цель работы

Изучить основы работы с такой структурой данных Python, как словари, научиться применять их методы. Освоить самые распространённые алгоритмы сортировки и изучить их применение относительно разных типов данных языка Python.

Задание

Вы - колонизатор новой неизведанной планеты. Вокруг Вас огромное количество полезных ископаемых и других ресурсов; каждый день появляются новые виды. Вы понятия не имеете зачем они нужны, но привязанный за ногу к столу ученый настаивает на необходимости бережно хранить ресурсы и вести их учет. Он также уговаривал Вас ввести классификацию, основанную на различном регистре букв, но Вы не стали слушать чокнутого и для простоты учета решили хранить все названия в строго нижнем регистре. Ваши бестолковые подчиненные, инопланетные крысы и желание покушать приводят к тому, что некоторые ресурсы могут убывать.

На вход программе подается произвольное количество пар, разделенных пробелом: <ресурс> <количество>, где ресурс - название ресурса, а количество - либо положительно либо отрицательное число. В зависимости от получения или траты ресурса. Перечень пар заканчивается словом Total. После него программа должна распечатать список ресурсов и их текущее количество (ресурсов может быть отрицательное число. Мы сами не знаем как - считайте, что живете в кредит). Вывод должен быть лексикографически упорядочен по названию ресурса.

Выполнение работы

Написание работы производилось на базе операционной системы Linux Arch в интегрированной среде разработки PyCharm.

Ход выполнения работы:

1. В цикле while происходит считывание временной строки с консоли и проверка его на слово “Total”, в случае неравенства происходит

преобразование временной строки к списку из двух элементов методом str.split(): название ресурса и его кол-во. Далее методом str.lower() название ресурса приводится к нижнему регистру.

2. Оператором not in происходит проверка наличия ключа (названия вида) в словаре. В случае отсутсвия: добавляется новый элемент словаря со значением равным переданному кол-ву ресурсов. Если же данный ключ присутсвует, то происходит сложение его значения с новым изменением ресурса.
3. Так как словари – тип данных, который невозможно отсортировать напрямую, то создаём переменную list_sort, в которую записываем объект типа dict_keys, содержащий ключи словаря (при помощи метода dict.keys()). Далее приводим данную переменную к типу “список” и передаём его в функцию сортировки.
4. Функция sort_it сортирует переданный список, применяя алгоритм сортировки вставками. Сложность данного алгоритма равна: $o(n^2)$. Среднее время сортировки также равно $o(n^2)$. (см. Приложение А)

   Общее описание алгоритма сортировки: проходим по массиву слева направо и обрабатываем по очереди каждый элемент. Слева от очередного элемента наращиваем отсортированную часть массива, справа по мере процесса исчезает неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер (переменная key), в результате чего в массиве появляется свободная ячейка — это позволяет сдвинуть элементы и освободить точку вставки.
5. В результате в цикле for проходим по отсортированному списку ключей словаря и выводим его и сопоставленное ему значение из словаря.

Выводы

В ходе лабораторной работы были изучены основы использования такой структура данных языка Python, как словарь. Были разобраны основные алгоритмы сортировки данных и получены знания по правильной их

реализации, а также разобрана корректная реализаци бинарного поиска. В результате была создана программа (см. Приложение Б), использующая основные методы словаря (такие как: dict.keys() и обращение по ключу), алгоритм сортировки вставками и некоторые методы строк, например, str.lower().

ПРИЛОЖЕНИЕ А. РАСЧЁТ СЛОЖНОСТИ СОРТИРОВКИ ВСТАВКАМИ.

1. Если переданный в функцию массив уже упорядочен, то все элементы останутся на своем месте и вложенный цикл while не будет выполнен ни разу. В этом случае сложность алгоритма сортировки вставками — линейная, т. е. $o(n)$.

2. Аналогично, если массив «почти упорядочен», то есть для превращения его в упорядоченный нужно поменять местами несколько соседних или близких элементов, то сложность также будет линейной.

3.Если же массив упорядочен в обратном порядке, например, каждый элемент больше следующего, а необходимо добиться обратного порядка, то каждый элемент будет перемещаться самой крайней левой позиции. В этом случае количество выполняемых перемещений будет равно: $o(n^2)$.

4. Если же упорядоченность входного массива «средняя», то количество перемещений элементов будет равно половине от числа перемещений в худшем случае, т.е.

$$\frac{n(n+1)}{4} = o(n^2).$$

То есть в среднем этот алгоритм также имеет квадратичную сложность.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

```
def sort_it(x):
    for i in range(1, len(x)):
        j = i - 1
        key = x[i]
        while x[j] > key and j >= 0:
            x[j + 1] = x[j]
            j -= 1
        x[j + 1] = key
    return x


kinds = {}
temp = ''
while temp != 'Total':
    temp = input()
    if temp == 'Total':
        break
    else:
        temp = temp.split()
        temp[0] = temp[0].lower()
        if temp[0] not in kinds:
            kinds[temp[0]] = int(temp[1])
        else:
            kinds[temp[0]] += int(temp[1])
list_sort = list(kinds.keys())  # получаем ключи словаря в виде объекта
dict_keys и преобразуем к обычному списку
sort_it(list_sort)  # сортируем ключи
for i, list_sort[i] in enumerate(list_sort):
    print(list_sort[i], kinds[list_sort[i]])
```