Множества

Множество в языке Python — это структура данных, эквивалентная множествам в математике. Множество может состоять из различных элементов, порядок элементов в множестве неопределен. В множество можно добавлять и удалять элементы, можно перебирать элементы множества, можно выполнять операции над множествами (объединение, пересечение, разность). Можно проверять принадлежность элементу множества.

В отличие от массивов, где элементы хранятся в виде последовательного списка, в множествах порядок хранения элементов неопределён (более того, элементы множества хранятся не подряд, как в списке, а при помощи хитрых алгоритмов). Это позволяет выполнять операции типа "проверить принадлежность элемента множеству" быстрее, чем просто перебирая все элементы множества.

Элементами множества может быть любой неизменяемый тип данных: числа, строки, кортежи. Изменяемые типы данных не могут быть элементами множества, в частности, нельзя сделать элементом множества список (но можно сделать кортеж) или другое множество. Требование неизменяемости элементов множества обусловлено особенностями представления множества в памяти компьютера.

Задание множеств

Множество задается перечислением всех его элементов в фигурных скобках. Например:

`A = {1, 2, 3}`

Исключением является пустое множество, которое можно создать при помощи функции `set()`. Если функции `set` передать в качестве параметра список, строку или кортеж, то она вернет множество, составленное из элементов списка, строки, кортежа. Например:

```
A = set('qwerty')
print(A)
```

может вывести, например, такой результат: `{'e', 'q', 'r', 't', 'w', 'y'}`.

Каждый элемент может входить в множество только один раз, порядок задания элементов не важен. Например, программа выведет `True`, так как `A` и `B` — равные множества.

```
A = {1, 2, 3}
B = {3, 2, 3, 1}
print(A == B)
```

Каждый элемент может входить в множество только один раз. `set('Hello')` вернет множество из четырех элементов: `{'H', 'e', 'l', 'o'}`.

Работа с элементами множеств

Узнать число элементов в множестве можно при помощи функции `len()`.

Перебрать все элементы множества (в неопределённом порядке!) можно при помощи цикла `for`:

```
primes = {2, 3, 5, 7, 11}

for num in primes:
    print(num)
```

Проверить, принадлежит ли значение `elem` множеству можно при помощи операции `in`, возвращающей значение типа `bool`:

`elem in A`.

Проверка того, что элемент не принадлежит множеству записывается так:

`elem not in A`.

Для добавления элемента в множество есть метод `add`:

`A.add(x)`.

Для удаления элемента `x` из множества есть два метода: `discard` и `remove`. Их поведение различается только в случае, когда удаляемый элемент отсутствует в множестве. В этом случае метод `discard` не делает ничего, а метод `remove` генерирует исключение `KeyError`.

Наконец, метод `pop` удаляет из множества один случайный элемент и возвращает его значение. Если же множество пусто, то генерируется исключение `KeyError`.

Из множества можно сделать список при помощи функции `list`.

Несмотря на то, что для множеств не определён метод `sort()`, можно сортировать элементы множества при помощи функции `sorted`. Функция сначала преобразует множество в список, затем сортирует его и возвращает результат.

```
letters = ‘‘WINDOW’’
print(sorted(set(letters))) # ['D', 'I', 'N', 'O', 'W']
```

Операции с множествами

С множествами в Python можно выполнять обычные для математики операции над множествами (пересечение, объединение, разность и т.п.). Все такие операции приведены на сайте с документацией (ссылка).

Упомянем лишь разницу между операциями обновления множества (на примере пересечения).

Если вы создаёте новое множество, равное перечению двух (или более) других, то способ сделать это такой:

```
C = A & B.
```

Если жы требуется *обновить* множество `A` так, чтобы в нём остались только те элементы, которые есть в `B`, то вариантов несколько:

```
# slow
A = A & B

# fast
A &= B
A.intersection_update(B)
```

Используйте второй или третий вариант.

Во всех задачах можно считывать входные данные из стандартного ввода (`input()`) и из файла (`'input.txt'`).

Задачи

A. *Количество различных чисел*
Дан список чисел, который может содержать до 100000 чисел. Определите, сколько в нём встречается различных чисел.

Input	Output
1 2 3 2 1	3

B. *Количество совпадающих*
Даны два списка чисел, которые могут содержать до 100000 чисел каждый. Посчитайте, сколько различных чисел содержится одновременно как в первом списке, так и во втором.

Input	Output
1 2 3 4 3 2	2

C. *Пересечение списков*
Даны два списка чисел, которые могут содержать до 10000 чисел каждый. Выведите в порядке возрастания все различные числа, которые входят как в первый, так и во второй список.

Input	Output
1 2 3 4 3 2	2 3

D. *Встречалось ли число раньше*
Во входной строке записана последовательность целых чисел через пробел. Для каждого числа выведите слово `YES` (в отдельной строке), если это число ранее встречалось в последовательности или `NO`, если не встречалось.

Input	Output
1 2 3 2 3 4	NO NO NO YES YES NO

E. *Кубики*
Аня и Боря любят играть в разноцветные кубики, причем у каждого из них свой набор и в каждом наборе все кубики различны по цвету. Однажды дети заинтересовались, сколько существуют цветов таких, что кубики каждого цвета присутствуют в обоих наборах. Для этого они занумеровали все цвета случайными числами. На этом их энтузиазм иссяк, поэтому вам предлагается помочь им в оставшейся части.
Номер любого цвета — это целое число в пределах от 0 до 10^9. В первой строке входного файла записаны числа N и M — количество кубиков у Ани и Бори соответственно. В следующих N строках заданы номера цветов кубиков Ани. В последних M строках номера цветов кубиков Бори. Выведите сначала количество, а затем отсортированные по возрастанию номера цветов таких, что кубики каждого цвета есть в обоих наборах, затем количество и отсортированные по возрастанию номера остальных цветов у Ани, потом количество и отсортированные по возрастанию номера остальных цветов у Бори.

Input	Output
4 3 0 1 10 9 1 3 0	2 0 1 2 9 10 1 3

F. *Количество различных слов в тексте*

Во входном файле записан текст. Словом считается последовательность непробельных символов идущих подряд, слова разделены одним или бóльшим числом пробелов или символами конца строки. Можно считать, что непробельные символы это всё, кроме символа “пробел” и символа переноса строки.

Определите, сколько различных слов содержится в этом тексте.

Input	Output
She sells sea shells on the sea shore; The shells that she sells are sea shells I’m sure. So if she sells sea shells on the sea shore, I’m sure that the shells are sea shore shells.	19

G. *Угадай число*

Август и Беатриса играют в игру. Август загадал натуральное число от 1 до n. Беатриса пытается угадать это число, для этого она называет некоторые множества натуральных чисел. Август отвечает Беатрисе `YES`, если среди названных ей чисел есть задуманное или `NO` в противном случае. После нескольких заданных вопросов Беатриса запуталась в том, какие вопросы она задавала и какие ответы получила и просит вас помочь ей определить, какие числа мог задумать Август.

Первая строка входных данных содержит число n — наибольшее число, которое мог загадать Август. Далее идут строки, содержащие вопросы Беатрисы. Каждая строка представляет собой набор чисел, разделенных пробелами. После каждой строки с вопросом идет ответ Августа: `YES` или `NO`.

Наконец, последняя строка входных данных содержит одно слово `HELP`.

Вы должны вывести (через пробел, в порядке возрастания) все числа, которые мог задумать Август.

Input	Output
10 1 2 3 4 5 YES 2 4 6 8 10 NO HELP	1 3 5

H* *Угадай число - 2*

Август и Беатриса продолжают играть в игру, но Август начал жульничать. На каждый из вопросов Беатрисы он выбирает такой вариант ответа `YES` или `NO`, чтобы множество возможных задуманных чисел оставалось как можно больше. Например, если Август задумал число от 1 до 5, а Беатриса спросила про числа 1 и 2, то Август ответит `NO`, а если Беатриса спросит про $1, 2, 3$, то Август ответит `YES`.

Если же Беатриса в своем вопросе перечисляет ровно половину из задуманных чисел, то Август из вредности всегда отвечает `NO`.

Наконец, Август при ответе учитывает все предыдущие вопросы Беатрисы и свои ответы на них, то есть множество возможных задуманных чисел уменьшается.

Вам дана последовательность вопросов Беатрисы. Приведите ответы Августа на них.

Первая строка входных данных содержит число n — наибольшее число, которое мог загадать Август. Далее идут строки, содержащие вопросы Беатрисы. Каждая строка представляет собой набор чисел, разделенных пробелами. Последняя строка входных данных содержит одно слово `HELP`.

Для каждого вопроса Беатрисы выведите ответ Августа на этот вопрос. После этого выведите (через пробел, в порядке возрастания) все числа, которые мог загадать Август после ответа на все вопросы Беатрисы.

Input	Output
10 1 2 3 4 5 2 4 6 8 10 HELP	NO YES 6 8 10

I. *Полиглоты*

Каждый из N школьников некоторой школы знает M_i языков. Определите, какие языки знают все школьники и языки, которые знает хотя бы один из школьников.

Первая строка входных данных содержит количество школьников N. Далее идет N чисел M_i, после каждого из чисел идет M_i строк, содержащих названия языков, которые знает i-й школьник. Длина названий языков не превышает 1000 символов, количество различных языков не более 1000, $1 \leqslant N \leqslant 1000, 1 \leqslant M_i \leqslant 500$.

В первой строке выведите количество языков, которые знают все школьники. Начиная со второй строки — список таких языков. Затем — количество языков, которые знает хотя бы один школьник, на следующих строках — список таких языков.

Input	Output
3 3 Russian English Japanese 2 Russian English 1 English	1 English 3 Russian Japanese English

J* *Забастовки*

В стране действует K политических партий, каждая из которых регулярно объявляет национальную забастовку. Дни, когда хотя бы одна из партий объявляет забастовку, при условии, что это не суббота или воскресенье (когда и так никто не работает), наносят большой ущерб экономике страны.

i-я партия объявляет забастовки строго каждые b_i дней, начиная с дня с номером a_i. То есть i-я партия объявляет забастовки в дни $a_i, a_i + b_i, a_i + 2b_i$ и т.д. Если в какой-то день несколько партий объявляет забастовку, то это считается одной забастовкой.

В календаре страны N дней, пронумерованных от 1 до N. Первый день года является понедельником, шестой и седьмой дни недели — выходные, неделя состоит из семи дней.

Программа получает на вход число дней в году N ($1 \leqslant N \leqslant 10^6$) и число политических партий K ($1 \leqslant K \leqslant 100$). Далее идет K строк, описывающие графики проведения забастовок. i-я строка содержит числа a_i и b_i ($1 \leqslant a_i, b_i \leqslant N$).

Выведите единственное число: количество забастовок, произошедших в течение года.

Input	Output
19 3 2 3 3 5 9 8	8

Примечание. Первая партия объявляет забастовки в дни 2, 5, 8, 11, 14, 17. Вторая партия объявляет забастовки в дни 3, 8, 13, 18. Третья партия — в дни 9 и 17. Дни номер 6, 7, 13, 14 являются выходными. Таким образом, общенациональные забастовки пройдут в дни 2, 3, 5, 8, 9, 11, 17, 18.

Словари (ассоциативные массивы)

Обычные списки (массивы) представляют собой набор пронумерованных элементов, то есть для обращения к какому-либо элементу списка необходимо указать его номер. Номер элемента в списке однозначно идентифицирует сам элемент. Но идентифицировать данные по числовым номерам не всегда оказывается удобно. Например, маршруты поездов в России идентифицируются численно-буквенным кодом (число и одна буква), также численно-буквенным кодом идентифицируются авиарейсы, то есть для хранения информации о рейсах поездов или самолетов в качестве идентификатора удобно было бы использовать не число, а текстовую строку.
Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется словарем или ассоциативным массивом. Соответствующая структура данных в языке python называется `dict`.
Рассмотрим простой пример использования словаря. Заведем словарь Capitals, где индексом является название страны, а значением — название столицы этой страны. Это позволит легко определять по строке с названием страны ее столицу.

```
# Создадим пустой словать Capitals
Capitals = dict()

# Заполним его несколькими значениями
Capitals['Russia'] = 'Moscow'
Capitals['Ukraine'] = 'Kiev'
Capitals['USA'] = 'Washington'

# Считаем название страны
print('В какой стране вы живете?')
country = input()

# Проверим, есть ли такая страна в словаре Capitals
if country in Capitals:
    # Если есть - выведем ее столицу
    print('Столица вашей страны', Capitals[country])
else:
    # Запросим название столицы и добавив его в словарь
    print('Как называется столица вашей страны?')
    city = input()
    Capitals[country] = city
```

Каждый элемент словаря это пара: ключ и соответствующее ему значение. В нашем примере ключом является название страны, значением является название столицы. Ключ идентифицирует элемент словаря, значение является данными, которые соответствуют данному ключу. Значения ключей — уникальны, двух одинаковых ключей в словаре быть не может.
В жизни широко распространены словари, например, привычные бумажные словари (толковые, орфографические). В них ключом является слово-заголовок статьи, а значением — сама статья. Для того, чтобы получить доступ к статье, необходимо указать слово-ключ.
Другой пример словаря как структуры данных — телефонный справочник. В нём ключом является имя, а значением — номер телефона. И словарь, и телефонный справочник хранятся так, что легко найти элемент словаря по известному ключу (например, если записи хранятся в алфавитном порядке ключей, то легко можно найти известный ключ, например, бинарным поиском). Но если ключ неизвестен, а известно лишь значение, то поиск элемента с данным значением может потребовать последовательного просмотра всех элементов словаря.
В словарь можно добавлять новые элементы с произвольными ключами и удалять уже существующие элементы. При этом размер используемой памяти пропорционален количеству хранимых данных. Доступ к значению по ключу делается за $O(1)$.
В языке Python ключом словаря может быть значение любого неизменяемого типа данных: целым числом, строкой, кортежем. Использовать действительные числа в качестве ключей не стоит из-за проблем с округлением и сравнением на точное равенство.

Ключом в словаре не может быть множество, но может быть элемент типа `frozenset`: специальный тип данных, являющийся аналогом типа `set`, который нельзя изменять после создания. Значением элемента словаря может быть любой тип данных, в том числе и изменяемый.

Когда нужно использовать словари

Словари нужно использовать в следующих случаях:
Подсчёт числа каких-то объектов. В этом случае нужно создать словарь, в котором ключами являются объекты, а значениями — их количество.
Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря `Num['January'] = 1; Num['February'] = 2; ....`
Установка соответствия между объектами (например, "родитель—потомок"). Ключ — объект, значение — соответствующий ему объект.
Если нужен обычный массив, но при этом масимальное значение индекса элемента очень велико и при этом будут использоваться не все возможные индексы (так называемый "разреженный массив"), то можно использовать ассоциативный массив для экономии памяти.

Создание словаря

Пустой словарь можно создать при помощи функции `dict()` или пустой пары фигурных скобок `{}` (вот почему фигурные скобки нельзя использовать для создания пустого множества). Для создания словаря с некоторым набором начальных значений можно использовать следующие конструкции:

```
Capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
Capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')
Capitals = dict([("Russia", "Moscow"), ("Ukraine", "Kiev"), ("USA", "Washington")])
Capitals = dict(zip(["Russia", "Ukraine", "USA"], ["Moscow", "Kiev", "Washington"]))
```

Первые два способа можно использовать только для создания небольших словарей, перечисляя все их элементы. Кроме того, во втором способе ключи передаются как именованные параметры функции `dict`, поэтому в этом случае ключи могут быть только строками, причем являющимися корректными идентификаторами. В третьем и четвёртом случае можно создавать большие словари, если в качестве аргументов передавать уже готовые списки, которые могут быть получены не обязательно перечислением всех элементов, а любым другим способом построены по ходу исполнения программы. В третьем способе функции `dict` нужно передать список, каждый элемент которого является кортежем из двух элементов: ключа и значения. В четвертом способе используется функция `zip`, которой передаётся два списка одинаковой длины: список ключей и список значений.

Работа с элементами словаря

Основная операция: получение значения элемента по ключу, записывается так же, как и для списков: `A[key]`. Если элемента с заданным ключом не существует в словаре, то возникает исключение `KeyError`.
Другой способ определения значения по ключу — метод `get`:
`A.get(key)`
Если элемента с ключом `key` нет в словаре, то возвращается значение `None`. В форме записи с двумя аргументами `A.get(key, val)` метод возвращает значение `val`, если элемент с ключом `key` отсутствует в словаре.
Проверить принадлежность элемента словарю можно операциями in и not in, как и для множеств.
Для добавления нового элемента в словарь нужно просто присвоить ему какое-то значение:
`A[key] = value`.
Для удаления элемента из словаря можно использовать операцию `del A[key]` (операция возбуждает исключение `KeyError`, если такого ключа в словаре нет. Вот два безопасных способа удаления элемента из словаря. Способ с предварительной проверкой наличия элемента:

```
if key in A:
    del A[key]
```

Способ с перехватыванием и обработкой исключения:

```
try:
    del A[key]
except KeyError:
    pass
```

Еще один способ удалить элемент из словаря: использование метода pop: `A.pop(key)`. Этот метод возвращает значение удаляемого элемента, если элемент с данным ключом отсутствует в словаре, то возбуждается исключение. Если методу pop передать второй параметр, то если элемент в словаре отсутствует, то метод pop возвратит значение этого параметра. Это позволяет проще всего организовать безопасное удаление элемента из словаря: A.pop(key, None).

Перебор элементов словаря

Можно легко организовать перебор ключей всех элементов в словаре:

```
for key in A:
    print(key, A[key])
```

Следующие методы возвращают представления элементов словаря. Представления во многом похожи на множества, но они изменяются, если менять значения элементов словаря. Метод `keys` возвращает представление ключей всех элементов, метод `values` возвращает представление всех значений, а метод `items` возвращает представление всех пар (кортежей) из ключей и значений. Соответственно, быстро проверить, если ли значение val среди всех значений элементов словаря `A` можно так: `val in A.values()`, а организовать цикл так, чтобы в переменной `key` был ключ элемента, а в переменной `val` было его значение можно так:

```
for key, val in A.items():
    print(key, val)
```

Во всех задачах можно считывать входные данные из стандартного ввода (`input()`) и из файла (`'input.txt'`).

Задачи

K. *Номер появления слова*

Во входном файле записан текст. Словом считается последовательность непробельных символов идущих подряд, слова разделены одним или большим числом пробелов или символами конца строки.

Для каждого слова из этого текста подсчитайте, сколько раз оно встречалось в этом тексте ранее.

Input	Output
one two one two three	0 0 1 1 0

Input
She sells sea shells on the sea shore; The shells that she sells are sea shells I'm sure. So if she sells sea shells on the sea shore, I'm sure that the shells are sea shore shells.
Output
0 0 0 0 0 0 1 0 0 1 0 0 1 0 2 2 0 0 0 0 1 2 3 3 1 1 4 0 1 0 1 2 4 1 5 0 0

L. *Словарь синонимов*

Вам дан словарь, состоящий из пар слов. Каждое слово является синонимом к парному ему слову. Все слова в словаре различны. Для каждого слова из данной последовательности определите его синоним.

Программа получает на вход словарь синонимов. Каждая строка словаря содержит два слова, разделенных пробелами. Далее до конца файла идет последовательность слов (по одному слову в строке).

Программа должна для каждого слова вывести его синоним.

Эту задачу можно решить и без словарей (сохранив все входные данные в списке), но решение со словарем будет более простым.

Input	Output
Hello Hi Bye Goodbye List Array Goodbye List	Bye Array

M. *Выборы в США*

Как известно, в США президент выбирается не прямым голосованием, а путём двухуровневого голосования. Сначала проводятся выборы в каждом штате и определяется победитель выборов в данном штате. Затем проводятся государственные выборы: на этих выборах каждый штат имеет определённое число голосов — число выборщиков от этого штата. На практике все выборщики от штата голосуют в соответствии с результатами голосования внутри штата, то есть на заключительной стадии выборов в голосовании участвуют штаты, имеющие различное число голосов.

Вам известно за кого проголосовал каждый штат и сколько голосов было отдано данным штатом. Подведите итоги выборов: для каждого из участника голосования определите число отданных за него голосов.

Каждая строка входного файла содержит фамилию кандидата, за которого отдают голоса выборщики этого штата, затем через пробел идет количество выборщиков, отдавших голоса за этого кандидата.

Выведите фамилии всех кандидатов в лексикографическом порядке, затем, через пробел, количество отданных за них голосов (см. пример).

Input	Output
McCain 10 McCain 5 Obama 9 Obama 8 McCain 1	McCain 16 Obama 17
Ivanov 1	Ivanov 1

N. *Самое частое слово*

Дан текст, состоящий из нескольких строк. Выведите слово, которое в этом тексте встречается чаще всего. Если таких слов несколько, выведите то, которое меньше в лексикографическом порядке. Слово — это последовательность латинских букв, ограниченная пробелами или переносами строк.

Input	Output
apple orange banana banana orange	banana

O. *Права доступа*

Для каждого файла N_i файловой системы известно, с какими действиями можно к нему обращаться:

- запись W
- чтение R
- запуск X

Вам требуется определить права доступа к файлам. Ваша программа для каждого запроса должна будет печатать `OK`, если над файлом выполняется допустимая операция или `Access denied`, если операция недопустима.

В первой строке входного файла содержится число N ($1 \leqslant N \leqslant 10000$) — количество файлов содержащихся в данной файловой системе.

В следующих N строчках содержатся имена файлов и допустимых с ними операций, разделённые пробелами. Длина имени файла не превышает 15 символов.

Далее указано число M ($1 \leqslant M \leqslant 50000$) — количество запросов к файлам.

В последних M строках указан запрос вида `Операция Файл`. К одному и тому же файлу может быть применено любое количество запросов.

Для каждого из M запросов нужно вывести в отдельной строке `Access denied` или `OK`.

Input	Output
4 helloworld.exe R X pinglog W R nya R goodluck X W R 5 read nya write helloworld.exe execute nya read pinglog write pinglog	OK Access denied Access denied OK OK

P. *Частотный анализ*

Дан текст. Выведите все слова, встречающиеся в тексте, по одному на каждую строку. Слова должны быть отсортированы по убыванию их количества появления в тексте, а при одинаковой частоте появления — в лексикографическом порядке.

Input	Output
hi hi what is your name my name is bond james bond my name is damme van damme claude van damme jean claude van damme	damme is name van bond claude hi my james jean what your

Указание После того, как вы создадите словарь всех слов, вам захочется отсортировать его по частоте встречаемости слова. Желаемого можно добиться, если создать список, элементами которого будут кортежи из двух элементов: частота встречаемости слова и само слово. Например, `[(2, 'hi'), (1, 'what'), (3, 'is')]`. Тогда стандартная сортировка будет сортировать список кортежей, при этом кортежи сравниваются по первому элементу, а если они равны — то по второму. Это почти то, что требуется в задаче.

Q. *Страны и города*

Дан список стран и городов каждой страны. Затем даны названия городов. Для каждого города укажите, в какой стране он находится.

Программа получает на вход количество стран N. Далее идет N строк, каждая строка начинается с названия страны, затем идут названия городов этой страны. В следующей строке записано число M, далее идут M запросов — названия каких-то M городов, перечисленных выше.

Для каждого из запроса выведите название страны, в котором находится данный город.

Input	Output
2 Russia Moscow Saint-Petersburg Novgorod Kaluga Ukraine Kiev Lvov Odessa 3 Odessa Moscow Novgorod	Ukraine Russia Russia

R. *Банковские счета*

Некоторый банк хочет внедрить систему управления счетами клиентов, поддерживающую следующие операции:

- Пополнение счета клиента
- Снятие денег со счета
- Запрос остатка средств на счете
- Перевод денег между счетами клиентов
- Начисление процентов всем клиентам

Вам необходимо реализовать такую систему. Клиенты банка идентифицируются именами (уникальная строка, не содержащая пробелов). Первоначально у банка нет ни одного клиента. Как только для клиента проводится операция пололнения, снятия или перевода денег, ему заводится счет с нулевым балансом. Все дальнейшие операции проводятся только с этим счетом. Сумма на счету может быть как положительной, так и отрицательной, при этом всегда является целым числом.

Входной файл содержит последовательность операций. Возможны следующие операции:

- `DEPOSIT name sum` — зачислить сумму `sum` на счёт клиента `name`. Если у клиента нет счёта, то счёт создаётся.
- `WITHDRAW name sum` — снять сумму `sum` со счёта клиента `name`. Если у клиента нет счёта, то счёт создается.
- `BALANCE name` — узнать остаток средств на счету клиента `name`.
- `TRANSFER name1 name2 sum` — перевести сумму `sum` со счёта клиента `name1` на счёт клиента `name2`. Если у какого-либо клиента нет счёта, то ему создается счёт.
- `INCOME p` — начислить всем клиентам, у которых открыты счета, p% от суммы счёта. Проценты начисляются только клиентам с положительным остатком на счету, если у клиента остаток отрицательный, то его счёт не меняется. После начисления процентов сумма на счету остается целой, то есть начисляется только целое число денежных единиц. Дробная часть начисленных процентов отбрасывается.

Для каждого запроса `BALANCE` программа должна вывести остаток на счету данного клиента. Если же у клиента с запрашиваемым именем не открыт счёт в банке, выведите `ERROR`.

Input	Output
DEPOSIT Ivanov 100 INCOME 5 BALANCE Ivanov TRANSFER Ivanov Petrov 50 WITHDRAW Petrov 100 BALANCE Petrov BALANCE Sidorov	105 -50 ERROR

S. *Англо-латинский словарь*

Однажды, разбирая старые книги на чердаке, школьник Вася нашёл англо-латинский словарь. Английский он к тому времени знал в совершенстве, и его мечтой было изучить латынь. Поэтому попавшийся словарь был как раз кстати.

К сожалению, для полноценного изучения языка недостаточно только одного словаря: кроме англо-латинского необходим латинско-английский. За неимением лучшего он решил сделать второй словарь из первого.

Как известно, словарь состоит из переводимых слов, к каждому из которых приводится несколько слов-переводов. Для каждого латинского слова, встречающегося где-либо в словаре, Вася предлагает найти все его переводы (то есть все английские слова, для которых наше латинское встречалось в его списке переводов), и считать их и только их переводами этого латинского слова.

Помогите Васе выполнить работу по созданию латинско-английского словаря из англо-латинского.

В первой строке содержится единственное целое число N — количество английских слов в словаре. Далее следует N описаний. Каждое описание содержится в отдельной строке, в которой записано сначала английское слово, затем отделённый пробелами дефис (символ номер 45), затем разделённые запятыми с пробелами переводы этого английского слова на латинский. Переводы отсортированы в лексикографическом порядке. Порядок следования английских слов в словаре также лексикографический.

Все слова состоят только из маленьких латинских букв, длина каждого слова не превосходит 15 символов. Общее количество слов на входе не превышает 100000.

Выведите соответствующий данному латинско-английский словарь, в точности соблюдая формат входных данных. В частности, первым должен идти перевод лексикографически минимального латинского слова, далее — второго в этом порядке и т.д. Внутри перевода английские слова должны быть также отсортированы лексикографически.

Input	Output
3 apple - malum, pomum, popula fruit - baca, bacca, popum punishment - malum, multa	7 baca - fruit bacca - fruit malum - apple, punishment multa - punishment pomum - apple popula - apple popum - fruit

T. *Контрольная по ударениям*

Учительница задала Пете домашнее задание — в заданном тексте расставить ударения в словах, после чего поручила Васе проверить это домашнее задание. Вася очень плохо знаком с данной темой, поэтому он нашел словарь, в котором указано, как ставятся ударения в словах. К сожалению, в этом словаре присутствуют не все слова. Вася решил, что в словах, которых нет в словаре, он будет считать, что Петя поставил ударения правильно, если в этом слове Петей поставлено ровно одно ударение.

Оказалось, что в некоторых словах ударение может быть поставлено больше, чем одним способом. Вася решил, что в этом случае если то, как Петя поставил ударение, соответствует одному из приведенных в словаре вариантов, он будет засчитывать это как правильную расстановку ударения, а если не соответствует, то как ошибку.

Вам дан словарь, которым пользовался Вася и домашнее задание, сданное Петей. Ваша задача — определить количество ошибок, которое в этом задании насчитает Вася.

Вводится сначала число N — количество слов в словаре ($0 \leqslant N \leqslant 20000$).

Далее идет N строк со словами из словаря. Каждое слово состоит не более чем из 30 символов. Все слова состоят из маленьких и заглавных латинских букв. В каждом слове заглавная ровно одна буква — та, на которую попадает ударение. Слова в словаре расположены в алфавитном порядке. Если есть несколько возможностей расстановки ударения в одном и том же слове, то эти варианты в словаре идут в произвольном порядке.

Далее идет упражнение, выполненное Петей. Упражнение представляет собой строку текста, суммарным объемом не более 300000 символов. Строка состоит из слов, которые разделяются

между собой ровно одним пробелом. Длина каждого слова не превышает 30 символов. Все слова состоят из маленьких и заглавных латинских букв (заглавными обозначены те буквы, над которыми Петя поставил ударение). Петя мог по ошибке в каком-то слове поставить более одного ударения (тогда это ошибка вне зависимости от того, соответствуют его ударения словарю или нет) или не поставить ударения вовсе.

Выведите количество ошибок в Петином тексте, которые найдет Вася.

Input	Output
4 cAnnot cannOt fOund pAge thE pAge cAnnot be fouNd	2

Комментарии к тесту:

В слове `cannot`, согласно словарю возможно два варианта расстановки ударения. Эти варианты в словаре могут быть перечислены в любом порядке (т.е. как сначала `cAnnot`, а потом `cannOt`, так и наоборот).

Две ошибки, совершенные Петей — это слова `be` (ударение вообще не поставлено) и `fouNd` (ударение поставлено неверно). Слово `thE` отсутствует в словаре, но поскольку в нём Петя поставил ровно одно ударение, признается верным.

Input	Output
4 cAnnot cannOt fOund pAge The PAGE cannot be found	4

Комментарии к тесту:

Неверно расставлены ударения во всех словах, кроме `The` (оно отсутствует в словаре, в нём поставлено ровно одно ударение). В остальных словах либо ударные все буквы (в слове `PAGE`), либо не поставлено ни одного ударения.

U. *Продажи*

Дана база данных о продажах некоторого интернет-магазина. Каждая строка входного файла представляет собой запись вида Покупатель товар количество, где Покупатель — имя покупателя (строка без пробелов), товар — название товара (строка без пробелов), количество — количество приобретённых единиц товара.

Создайте список всех покупателей, а для каждого покупателя подсчитайте количество приобретённых им единиц каждого вида товаров.

Выведите список всех покупателей в лексикографическом порядке, после имени каждого покупателя выведите двоеточие, затем выведите список названий всех приобретенных данным покупателем товаров в лексикографическом порядке, после названия каждого товара выведите количество единиц товара, приобретенных данным покупателем. Информация о каждом товаре выводится в отдельной строке.

Input	Output
Ivanov paper 10 Petrov pen 5 Ivanov marker 3 Ivanov paper 7 Petrov envelope 20 Ivanov envelope 5	Ivanov: envelope 5 marker 3 paper 17 Petrov: envelope 20 pen 5

V. *Выборы в США - 2*

На этот раз вам известно число выборщиков от каждого штата США и результаты голосования каждого гражданина США (а также в каком штате проживает данный гражданин).

Вам необходимо подвести результаты голосования: сначала определить результаты голосования в каждом штате и определить, за какого из кандидатов отданы голоса выборщиков данного штата. Далее необходимо подвести результаты голосования выборщиков по всем штатам.

Первая строка входных данных содержит число N — количество штатов в США. Далее идет N строк, описывающих штаты США, каждая строка состоит из названия штата и числа выборщиков от этого штата. Далее до конца файла идут записи результатов голосования по каждому из участников голосования. Одна строка соответствует одному избирателю. Записи имеют вид: название штата, имя кандидата, за которого проголосовал данный избиратель. Названия штатов и имена кандидатов не содержат пробелов.

Выведите список кандидатов, упорядоченный по убыванию числа голосов выборщиков, полученных за данного кандидата, а при равенстве числа голосов выборщиков: в лексикографическом порядке. После имени кандидата выведите число набранных им голосов.

Если в каком-либо штате два или более число кандидатов набрали одинаковое число голосов, то все голоса выборщиков этого штата получает наименьший в лексикографическом порядке кандидат из числа победителей в этом штате.

Гарантируется, что в каждом штате проголосовал хотя бы один избиратель.

Input	Output
2 Florida 25 Pennsylvania 23 Florida Gore Pennsylvania Gore Florida Bush Pennsylvania Gore Pennsylvania Bush Florida Gore Pennsylvania Gore Florida Bush Pennsylvania Gore Florida Bush Pennsylvania Gore	Bush 25 Gore 23

Комментарий к тесту: В штате `Florida` 2 избирателя голосует за `Gore` и три избирателя за `Bush`, поэтому 25 голосов выборщиков от `Florida` получает `Bush`. В `Pennsylvania` побеждает `Gore` (5 голосов против 1), поэтому `Gore` получает 23 голоса выборщиков от `Pennsylvania`.

Input	Output
3 Florida 5 Pennsylvania 4 Alaska 3 Florida Gore Pennsylvania Obama Pennsylvania Clinton Alaska Bush	Gore 5 Clinton 4 Bush 3 Obama 0

Комментарий к тесту: В штате `Florida` побеждает `Gore` (5 голосов выборщиков), в `Alaska` — `Bush` (2 голоса выборщика). В `Pennsylvania` два кандидата набрали наибольшее число голосов (по 1), поэтому 4 голоса выборщиков от этого штата получает `Clinton`, т.к. он идёт раньше в лексикографическом порядке.

W. *Родословная: подсчет высоты*

В генеалогическом древе у каждого человека, кроме родоначальника, есть ровно один родитель. На рисунке приведена часть древа рода Романовых, начиная с Петра I Великого.

Рис. 1: Генеалогическое древо Романовых

Каждом элементу дерева сопоставляется целое неотрицательное число, называемое высотой. У родоначальника высота равна 0, у любого другого элемента высота на 1 больше, чем у его родителя.

Вам дано генеалогическое древо, определите высоту всех его элементов.

Программа получает на вход число элементов в генеалогическом древе N. Далее следует $N - 1$ строка, задающие родителя для каждого элемента древа, кроме родоначальника. Каждая строка имеет вид `имя_потомка имя_родителя`.

Программа должна вывести список всех элементов древа в лексикографическом порядке. После вывода имени каждого элемента необходимо вывести его высоту.

Эта задача имеет решение сложности $O(n)$, но вам достаточно написать решение сложности $O(n^2)$ (не считая сложности обращения к элементам словаря).

Пример ниже соответствует приведённому древу рода Романовых.

Input	Output
9 Alexei Peter_I Anna Peter_I Elizabeth Peter_I Peter_II Alexei Peter_III Anna Paul_I Peter_III Alexander_I Paul_I Nicholaus_I Paul_I	Alexander_I 4 Alexei 1 Anna 1 Elizabeth 1 Nicholaus_I 4 Paul_I 3 Peter_I 0 Peter_II 2 Peter_III 2

X. *Родословная: предки и потомки*

Даны два элемента в дереве. Определите, является ли один из них потомком другого.

Программа получает на вход описание дерева, как в задаче W. Далее до конца файла идут строки, содержащие имена двух элементов дерева. Для каждого такого запроса выведите одно из трех чисел: 1, если первый элемент является предком второго, 2, если второй является предком первого или 0, если ни один из них не является предком другого.

Input	Output
9 Alexei Peter_I Anna Peter_I Elizabeth Peter_I Peter_II Alexei Peter_III Anna Paul_I Peter_III Alexander_I Paul_I Nicholaus_I Paul_I Anna Nicholaus_I Peter_II Peter_I Alexei Paul_I	1 2 0

Y. *Родословная: LCA*

В генеалогическом древе определите для двух элементов их наименьшего общего предка. Наименьшим общим предком элементов A и B является такой элемент C, что является предком A, C является предком B, при этом глубина C является наибольшей из возможных. При этом элемент считается своим собственным предком.

Формат входных данных аналогичен предыдущей задаче. Для каждого запроса выведите наименьшего общего предка данных элементов.

По-английски такая задача называется `lowest common ancestor (LCA)`.

Input	Output
9 Alexei Peter_I Anna Peter_I Elizabeth Peter_I Peter_II Alexei Peter_III Anna Paul_I Peter_III Alexander_I Paul_I Nicholaus_I Paul_I Alexander_I Nicholaus_I Peter_II Paul_I Alexander_I Anna	Paul_I Peter_I Anna

Z. *Родословная: Число потомков*

Для каждого элемента дерева определите число всех его потомков (не считая его самого).

Формат выходных данных совпадает с задачей W. Выведите список всех элементов в лексикографическом порядке, для каждого элемента выводите количество всех его потомков.

Решение должно иметь сложность $O(N)$, не считая сложности обращения к элементам словаря и сортировки результата.

Input	Output
9 Alexei Peter_I Anna Peter_I Elizabeth Peter_I Peter_II Alexei Peter_III Anna Paul_I Peter_III Alexander_I Paul_I Nicholaus_I Paul_I	Alexander_I 0 Alexei 1 Anna 4 Elizabeth 0 Nicholaus_I 0 Paul_I 2 Peter_I 8 Peter_II 0 Peter_III 3