Структуры данных. Множества

Общее представление

Перед написанием программы важно правильно выбрать структуру данных, обеспечивающую эффективное решение задачи. Одни и те же данные можно сохранить в структурах, требующих различного объема памяти, а алгоритмы работы с каждой структурой могут иметь различную эффективность.

Множество (set)

Множество в Python – это неиндексированная совокупность уникальных элементов.

Отличительные характеристики множества в Python:

- Множество не содержит дубликаты элементов;
- Элементы множества являются неизменными (их нельзя менять), однако само по себе множество является изменяемым, и его можно менять;
- Так как элементы не индексируются, множества не поддерживают никаких операций среза и индексирования.
- Можно производить операции, аналогичные операциями с множествами в математике.

Основные причины использования структуры множество для решения задач: **быстрая проверка наличия элемента во множестве**, **хранение уникальных элементов**.

Во множествах порядок хранения элементов не определён. Элементы множества хранятся не подряд, как в списке, а в виде специальной таблицы. 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 накладывается особенностями представления множества в памяти компьютера.

Создание множества

`a = set()`	*Создание пустого множества.*
`a = {1, 2, 3, 5}`	*Создание множества, заданного элементами*

`b = [4, 7, 8, 8, 9, 4]`
`a = set(b)`
`a = {4, 7, 8, 9}`
	Создание множества на основе итерируемого объекта список. В сет каждый элемент войдет только один раз.

`string ='kaktus'`
`s = set(string)`
`s = {'k', 'a', 't', 'u', 's'}`
	Создание множества на основе строки, как итерируемого объекта

`a = set()`
`a.add(1)`
`a.add(22)`
`a.add(1)`
`a = {1, 22}`
	Поэлементное добавление элементов во множество

`a = {1, 2, 3, 5}`	*Создание множества* `c` *как объединение множеств* `a` *и* `b`

`b = {0, 3, 2, 4}`
`c = a | b`
`c = {0, 1, 2, 3, 4, 5}`	

Работа с элементами множеств

Операция	*Сложность*	*Описание*
`len(a)`	`O(1)`	*Определение количества элементов множества*
`x in a`	`O(1)`	*Проверка принадлежности элемента* `x` *множеству* `a`
`a.add(x)`	`O(1)`	*Добавление элемента x во множество a*
`x = a.pop()`	`O(1)`	*Удаление из множества a некоторого элемента и возвращение его значения в переменную x*
`a.remove(x)`	`O(1)`	*Удаление элемента x из множества. Если элемента нет, то вызывается исключение KeyError*
`a.discard(x)`	`O(1)`	*Удаление элемента x из множества. Если элемента нет, исключение не вызывается.*
`a.clear()`	`O(len(a))`	*Удаление всех элементов сета*
`for x in a:`	`O(len(a))`	*Перебор всех элементов множества. Множество объект не индексируемый, но итерируемый.*
`b = list(a)`	`O(len(b))`	*Преобразование элементов сета в список.*

Если сравнивать со списками, то многие одинаковые по смыслу операции (удаление, проверка на наличие) выполняются за более привлекательную сложность O(1) вместо O(len(a)).

Логические операции над множествами

`a & b` `a.intersection(b)`	*Пересечение множеств.* *Включает элементы, которые присутствуют в обоих множествах*
`a | b` `a.union(b)`	*Объединение множеств.* *Включает элементы, которые присутствуют хотя бы в одном из множеств*
`a - b`	*Разность множеств.* *Из множества* `a` *удаляет элементы, которые присутствуют во множестве* `b`
`a ^ b` `a.symmetric_difference(b)`	*Симметричная разность множеств.* *Включает элементы, которые есть в одном из множеств, но отсутствуют в другом.*
`a <= b`	*True, если a является подмножеством множества b*
`a == b`	*True, если множества совпадают.*

Примеры операций над множествами:

```
a = set('elephant')
b = set('telephone')
a & b = {'p', 'n', 'h', 'e', 'l', 't'}
a | b = {'p', 'n', 'e', 'h', 'l', 'a', 't', 'o'}
a ^ b = {'o', 'a'}
a – b = {'a'}
b – a = {'o'}
a > {'e', 'l', 't'} == True
```

Примеры задач

Задача №1. Напишите программу, которая удаляет из строки все повторяющиеся символы.

Входные данные	Выходные данные
abc13a1b2z3c	abc132z
QWasd123	QWasd123

Решение.
Для решения этой задачи будем использовать дополнительную структуру – сет, в котором будем хранить символы, которые уже встретились в строке. Будем перебирать каждый символ строки и если он уже есть в сете используемых символов, то не будем его выводить, если нет, то выведем его на экран и добавим в сет.

```
sl = input()
s = set()
for x in sl:
    if x not in s:
        print(x, end = '')
        s.add(x)
```

В этой задаче мы используем сет для уменьшения времени работы программы. Без использования сета для проверки, встречался ли элемент раньше, нам пришлось бы перебирать все элементы строки до этого элемента, что сведет асимптотику решения этой задачи к $O(n^2)$. В рассматриваемом решение проверка наличия элемента в сете осуществляется за O(1) и время работы сводится к линейному.
Альтернативное решение задачи без множества
Вместо сета можно использовать список типа bool на 256 элементов. Индекс будет соответствовать коду символа, значение True – элемент не встречался, False – уже был.

Frozenset в Python

Frozenset (замороженное множество) – это структура с характеристиками множества, однако, как только элементы становятся назначенными, их нельзя менять. Кортежи могут рассматриваться как неизменяемые списки, в то время как frozenset - как неизменные множества. Frozenset можно использовать как элемент сета.

Используемые ресурсы:

1. Гутман Г.Н. Языки программирования: Python 3.1, учебное пособие.- Самара, Самарский государственный технический университет, 2011
2. Кириенко Д.П., Cbook
3. Саммерфилд М. Программирование на Python 3. Подробное руководство,
4. Любанович Билл, Простой Python. Современный стиль программирования. - СПб.: Питер, 2016.
5. Лутс М. Python, 4-е издание. – Пер. с англ. – Спб.: Символ-Плюс, 2010