Множества

Date	@June 4, 2025
Select	Тренировки по алгоритмам 1

https://www.youtube.com/watch?v=PUpmV2ielHA

https://contest.yandex.ru/contest/27663/problems/

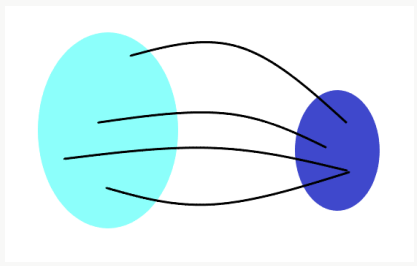
Как устроено множество

> 3 действия: добавление элемента, проверка наличия, удаление

Реализация

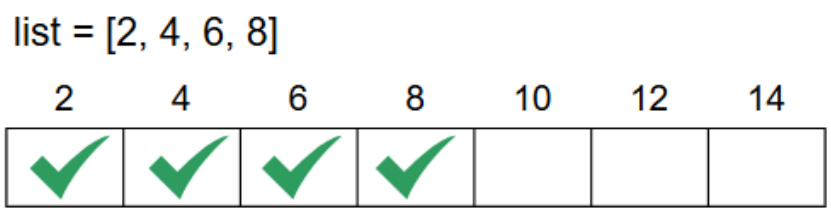
1. Придумаем функцию, которая сопоставляет каждому элементу какое-либо небольшое число
2. Вычислим функцию от элемента
3. Положим элемент в список с номером, равным значению функции

! столкнемся с тем, что элементы из меньшего множества будут описывать несколько элементов из большого (принцип Дирихле)

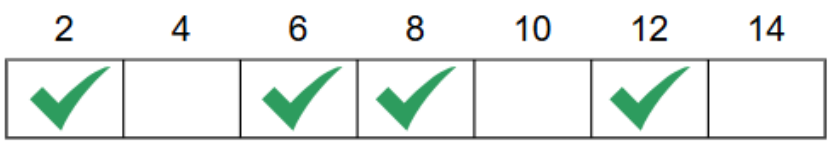


Удобная реализация хранения небольшого множества чисел, когда создается булевый массив, в котором отмечается наличие определенного элемента (удобно: добавлять, удалять, проверять наличие, неудобно: рассчитан на небольшое количество элементов)

list = [2, 4, 6, 8]



list = [2, 6, 8, 12]



Попробуем хранить большое множество таким способом

? Функция, которая переводит из большого множества в маленькое, называется **хэш-функцией**

Пример: функция - последняя цифра числа $F(x) = x\%10$

Проблема такой функции: **коллизии** - совпадение значений хэш-функции для разных параметров. В этом случае, если мы добавим число 137 и захотим проверить наличие числе 17, то из-за коллизии программа даст положительный ответ.

Одно из решений проблемы: запись в массив не пометки о том, есть ли это число, а запись самого числа (проблема: можем хранить только одно число)

Давайте для каждого возможного значения хэш-функции хранить список элементов с соответствующим хэшем (такая структура называется **хэш-таблицей**)

Обход и вывод элементов (в произвольном порядке) за $O(N + K)$, где N - количество значений хэш-функции, K - количество элементов в множестве

Удаления элемента

1. Находим элемент (в среднем за $O(K/N)$)
2. Эффективно его удаляем (удаление элемента из списка происходит за $O(\text{длины списка})$, удаление с конца списка - за $O(1)$): копируем на его место последний элемент из списка
3. Удаляем последний элемент

В чем плюсы? Мы тратим меньше операций (если бы поэлементно копировали снизу вверх элементы) и операция присваивания намного медленнее, чем чтения, т о мы сэкономили и время

Реализация (мульти)множества с хэш-функцией $F(x) = x\%10$:

```
N = 10
# Создание двумерного массива
myset = [[] for _ in range(N)]


def add(x):
    # if not find(x):
    myset[x % N].append(x)


# Линейный поиск для нахождения
def find(x):
    for el in myset[x % N]:
        if el == x:
            return True
    return False


def delete(x):
    xlist = myset[x % N]
    for i in range(len(xlist)):
        if xlist[i] == x:
            xlist[i] = xlist[-1]
            # или swap
            # xlist[i], xlist[len(xlist) - 1] = xlist[len(xlist) - 1], xlist[i]
            xlist.pop()
            return
```

Мультимножество - множество, которое может хранить несколько одинаковых элементов

В нашей реализации мы не проверяем наличие до добавления, удаление и поиск по первому найденному. При этом, если требуется удалить любое вхождение этого элемента, то придется пройтись и убедиться, что все удалено

Мультимножество → множество: проверка отсутствия на этапе добавления

Что можно хранить в множестве эффективно?

Хранить - что угодно (в компьютере все состоит из чисел)

Эффективно - только **неизменяемые** объекты (в Python почти все структуры (числа, строки, кортежи, frozenset), кроме списков, множеств, словарей и прочего)

Для неизменяемых объектов достаточно посчитать хэш один раз и им пользоваться

Хэш-функция должна давать равномерное распределение

Амортизированная сложность

Проблемы с хеш-таблицей

Перед началом работы не ясно, сколько объектов будет в множестве.

- Слишком большой размер — ест много памяти $O(N)$
- Слишком маленький размер — большой коэффициент заполнения и медленный поиск и удаление $O(K/N)$)

Хочется иметь разумный баланс, например, коэффициент заполнения не больше единицы (т.е. $\mathrm{K} <= N$). Тогда все операции в среднем будут занимать $O(1)$

Решение:

Когда таблица наполнится - увеличим ее размер вдвое и перестроим

Количество расширений: $P = \log_2 N$

Количество действий, чтобы преобразовать таблицу: $1 + 2 + 4 + 8 + \ldots + 2^P = 2^{P+1} - 1 = 2N - 1 = O(N)$

❓ **Амортизированная сложность** - среднее время выполнения операции (условно)

У нас амортизированная сложность операции $O(1)$ всего было N операций и суммарно на это ушло $O(N)$

В худшем случае отдельная операция выполняется за $O(N)$ может не подходит для систем реального времени

Задачи

Задача 1

Дана последовательность положительных чисел длиной N и число X

Нужно найти два различных числа A и B из последовательности, таких что $A + B = X$ или вернуть пару $0, 0$, если такой пары чисел нет

Решение за $O(N^2)$

Перебор числа A за $O(N)$, затем B за $O(N)$. Если их сумма равна X, то возвращаем пару.

```
# Наивная реализация (с ошибкой)

def twonumswithsumx(nums, x):
    for a in nums:
        for b in nums:
            if a + b == x:
                return (a, b)
    return (0, 0)

x = 4
nums = [2, 1, 3, 4,  0, 5]
print(twonumswithsumx(nums, x)) # (2, 2) → в условии два различных числа
```

Избавляемся от проблемы

```
def twonumswithsumx(nums, x):
    for a in nums:
        for b in nums:
            if a == b:
                continue
            if a + b == x:
                return (a, b)
    return (0, 0)

x = 4
nums = [2, 1, 3, 4,  0, 5]
print(twonumswithsumx(nums, x)) # (1, 3)
```

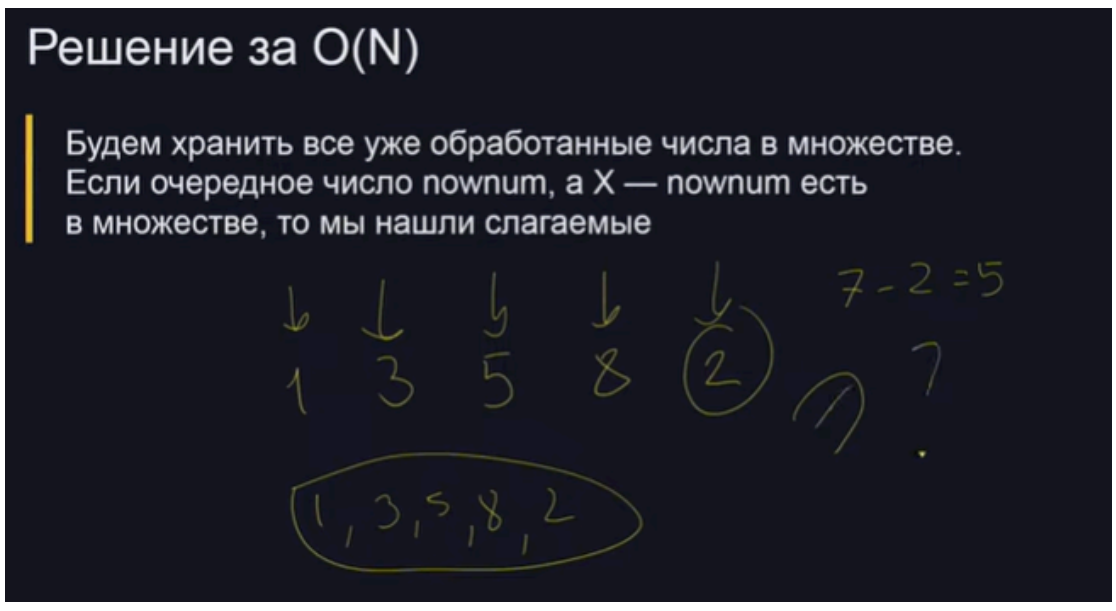
Улучшенная версия - используем индексы

```
def twonumswithsumx(nums, x):
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            if nums[i] + nums[j] == x:
                return (nums[i], nums[j])
    return (0, 0)

x = 4
nums = [2, 1, 3, 4,  0, 5]
print(twonumswithsumx(nums, x)) # (1, 3)
```

Решение за $O(N)$

При добавлении элемента el в множество (за линию) будем проверять, а не лежит ли в множестве число $X - el$




```
def twonumswithsumx(nums, x):
    prevnums = set()
```

```
    for nownum in nums:
        if x - nownum in prevnums:
            return (nownum, x - nownum)
        prevnums.add(nownum)
    return (0, 0)


x = 4
nums = [2, 1, 3, 4, 0, 5]
print(twonumswithsumx(nums, x)) # (3, 1)
```

Задача 2

Дан словарь из N слов, длина каждого не превосходит K

В записи каждого из M слов текста (каждое длиной до K) может быть пропущена одна буква. Для каждого слова сказать, входит ли оно (возможно, с одной пропущенной буквой), в словарь

Решение за $O(NK + M)$

Выбросим из каждого слова словаря по одной букве всеми возможными способами за $O(NK)$ и положим получившиеся слова в множества за $O(1)$ (тк Python посчитает хэши для каждого слова)

Для каждого слова из текста просто проверим, есть ли оно в словаре за $O(1)$

$abcd \to \{abcd, bcd, acd, abd, abc\}$

```
def wordsindict(dictionary, text):
    goodwords = set(dictionary)
    for word in dictionary:
        for delpos in range(len(word)):
            goodwords.add(word[:delpos] + word[delpos + 1:])
    ans = []
    for word in text:
        ans.append(word in goodwords)
    return ans
```