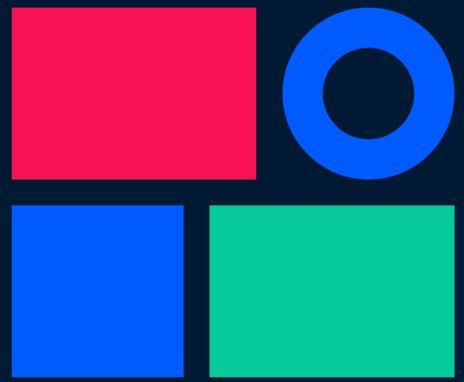


Ozon New Skills

Ozon
New Skills

Python 18+

Алгоритмы поиска и сортировки

Алгоритмы

А зачем они нужны?

Алгоритм

Конечная совокупность точно заданных правил решения задач, или же просто набор функций

Алгоритмы

Если по простому — набор готовых решений для решения задачи. Нужно быстро элемент в списке — есть готовое решение — ваша задача — только реализовать его на языке

Fizz Bizz

Первый алгоритм, который мы с вами изучим, даже не алгоритм — это скорее одна из задач на собесе. А вообще Fizz — Buzz — это просто детская считалочка, которая учит детей тренировке деления и кратности чисел

Fizz Bizz


Расскажите о себе
Fizz-Buzz могу написать
Вы Приняты!!
IT детского сада
2fan.ru
meme-arsenal.ru

Fizz Bizz

Озвучим задание: Начинающий произносит число «1», и каждый следующий игрок прибавляет к предыдущему значению единицу. Когда число делится на три оно заменяется на fizz, если число делится на пять, то произносится buzz. Числа, делящиеся на три и пять одновременно заменяются на fizz buzz. Продумайте код реализации самостоятельно

Реализация Fizz-Buzz

```
# Самое наивное решение, каждый случай -
независимая ветка
for x in range(1, 100):
    if x % 3 == 0 and x % 5 == 0:
        print("FizzBuzz", end=' ‘)
    elif x % 3 == 0:
        print("Fizz", end=' ‘)
    elif x % 5 == 0:
        print("Buzz", end=' ‘)
    else:
        print(x, end=' ')
```

Сложность алгоритмов

Или как ее определять

Оценка сложности

Сложность алгоритмов обычно оценивают по времени выполнения или по используемой памяти. В обоих случаях сложность зависит от размеров входных данных: массив из 100 элементов будет обработан быстрее, чем аналогичный из 1000.

Примеры сложности

O(n) — линейная сложность

Такой сложностью обладает, например, алгоритм поиска наибольшего элемента в не отсортированном массиве. Нам придётся пройтись по всем n элементам массива, чтобы понять, какой из них максимальный.

Примеры сложности

O(log n) — логарифмическая сложность

Простейший пример — бинарный поиск. Если массив отсортирован, мы можем проверить, есть ли в нём какое-то конкретное значение, методом деления пополам.

Примеры сложности

$O(n^2)$ — квадратичная сложность

Такую сложность имеет, например, алгоритм сортировки вставками. В канонической реализации он представляет из себя два вложенных цикла: один, чтобы проходить по всему массиву, а второй, чтобы находить место очередному элементу в уже отсортированной части.

Алгоритмы поиска

Я сейчас все тут найду

Алгоритмы поиска

Иногда вам нужно найти информацию в структурированном массиве. Для этого вам необходим какой алгоритм поиска. Мы посмотрим на один не самый эффективный алгоритм поиска — линейный, который перебирает все подряд.

Алгоритм сортировки

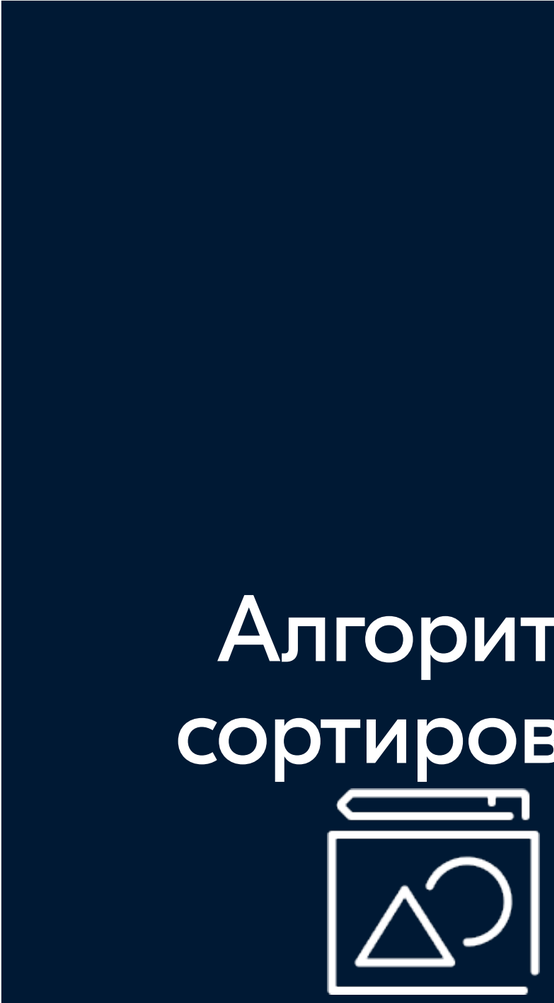
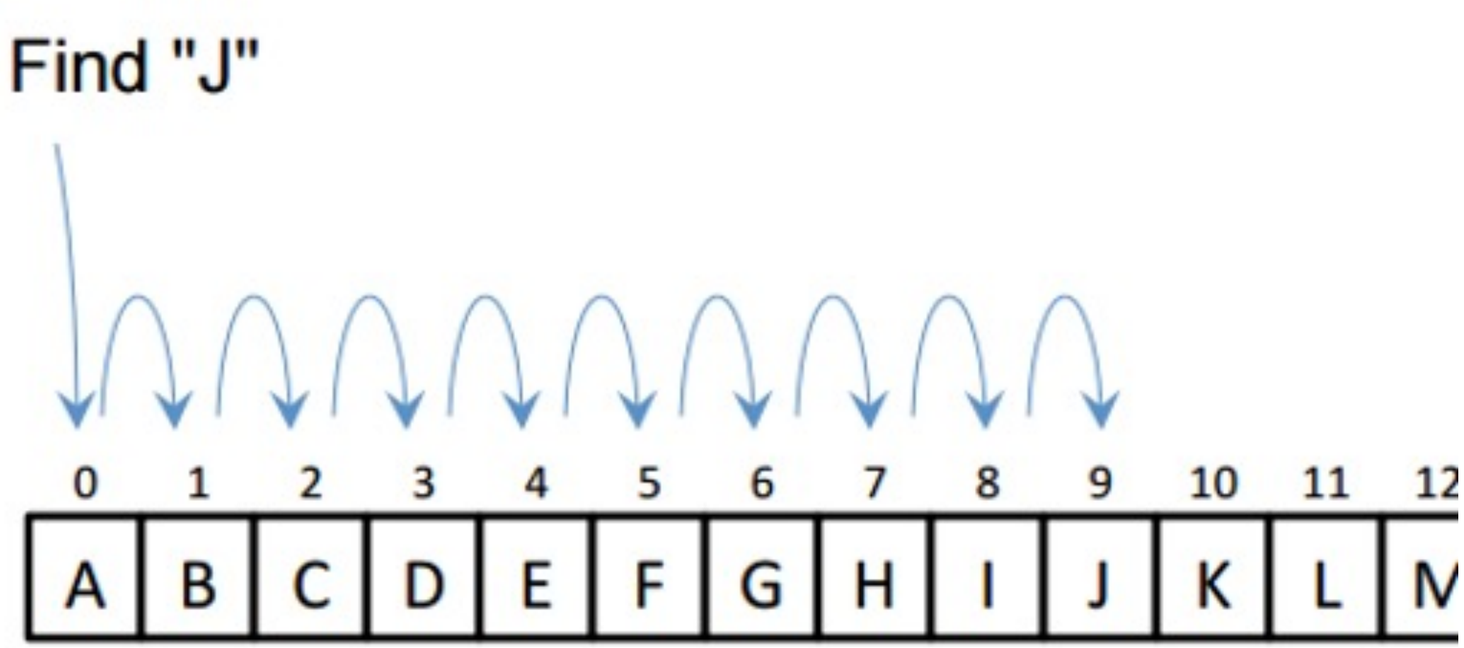


Иллюстрация работы алгоритма сортировки



Реализация линейного алгоритма

```
def search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1
```

О скорости

Однако, если мы с вами будем увеличивать размер нашего списка, скорость работы алгоритмы будет только уменьшаться. Для большей эффективности и увеличения скорости мы с вами используем другой алгоритм - бинарный

Бинарный поиск

Классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины

Бинарный поиск

У бинарного алгоритма следующий распорядок действий:

1. Определение значения элемента в середине структуры данных. Полученное значение сравнивается с ключом.
2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй.
3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.
4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

Бинарный поиск

Подумайте над его реализацией. Как лучше это осуществить? Стоит ли использовать функцию?

Бинарный поиск

```
def binary_search(arr, low, high, x):
# Проверяем базовый случай
     if high >= low:
          mid = (high + low) // 2
          if arr[mid] == x:
               return mid
          elif arr[mid] > x:
               return binary_search(arr, low, mid - 1, x)
          else:
               return binary_search(arr, mid + 1, high, x)
     else:
          return -1
```

Алгоритмы сортировки

Я сейчас все тут отсортирую

Алгоритмы

Существует много алгоритмов сортировки, но два самых известных — это сортировка пузырьком и сортировка выбором.

Сортировка пузырьком

Это учебный алгоритм, который обычно проходят одним из первых. Он эффективен только для небольших массивов.

Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован

Сортировка пузырьком

```
def bubble_sort(nums):
     while swapped:
          swapped = False
          for i in range(len(nums) - 1):
               if nums[i] > nums[i + 1]:
               # Меняем элементы
               nums[i], nums[i + 1] = nums[i + 1], nums[i]
# Устанавливаем swapped в True для следующей итерации
               swapped = True

# Проверяем, что оно работает
random_list_of_nums = [5, 2, 1, 8, 4]
bubble_sort(random_list_of_nums)
print(random_list_of_nums)
```

Сортировка вставкой

Алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

Сортировка вставкой

```
def insertion_sort(array):
    for index in range(1, len(array)):
        cV = array[index]
        cP = index
        while cP > 0 and array[cP - 1] > cV:
            array[cP] = array[cP -1]
            cP = cP - 1
    array[cP] = cV
```

Факультативная часть

Алгоритмы шифровки

Ничего вы не узнаете

Алгоритмы шифровки

Алгоритмы шифровки не очень пригодятся вам,

если вы не будете заниматься сферой кибербезопасности

Однако это нужно, чтобы вы потом не потерялись

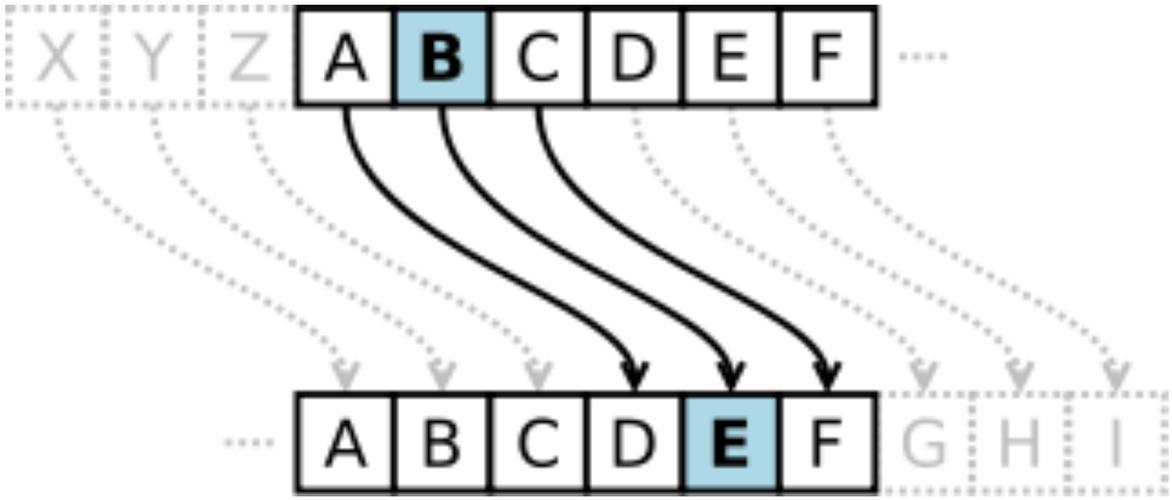
в других алгоритмах шифровки, который куда

сложнее, чем простые

Шифр Цезаря

Один из первых алгоритмов шифровки, который применялся в истории человечества — шифр Цезаря. Он представляет из себя сдвиг на определенное количество символов (которое обговорено союзными сторонами)

Шифр Цезаря



Шифр Цезаря

```
def encrypt(text, shift):
     encryption = ""
     for symbol in text:
          if symbol.isupper():
             c_unicode = ord(symbol)
             c_index = ord(symbol) - ord("A")
             new_index = (c_index + shift) % 26
             new_unicode = new_index + ord("A")
             new_character = chr(new_unicode)
             encryption = encryption + new_character
          else:
             encryption += symbol
       return encryption


print(encrypt("HELLO WORLD", 3))
```