InnoDom

"Грокаем" алгоритмы

Алгоритм

это **последовательность** шагов, выполняемых для решения определённой задачи или выполнения определённого вычисления. Он описывает, какие операции нужно выполнить, чтобы достичь желаемого результата.

Алгоритм в программировании - это последовательность шагов, необходимых для выполнения определённой задачи на компьютере. Алгоритм описывает, какие операции должны быть выполнены и в каком порядке, чтобы получить требуемый результат.
Хороший алгоритм должен быть *точным*, *эффективным*, *легко понятным* и *реализуемым* на компьютере. Он должен решать поставленную задачу корректно и за разумное время.

Виды алгоритмов

Существует множество различных типов алгоритмов, которые могут быть использованы для решения разных задач. Наиболее распространённые:

- **Жадные алгоритмы** - это алгоритмы, которые всегда выбирают локально оптимальное решение на каждом шаге. Эти алгоритмы не всегда дают глобально оптимальный результат, но могут быть эффективными в решении задач о рюкзаке или о расписании.
- **Рекурсивные алгоритмы** - алгоритмы, которые вызывают сами себя для решения более простых задач. Они широко используются для обхода деревьев, графов и других структур данных.
- **Алгоритмы поиска** - находят определённое значение в структуре данных. Некоторые из наиболее распространенных алгоритмов поиска включают в себя линейный поиск, двоичный поиск и поиск в ширину.

Виды алгоритмов

- **Алгоритмы сортировки** - упорядочивают элементы в структуре данных. Например, сортировка пузырьком, сортировка вставками, быстрая сортировка и сортировка слиянием.
- **Алгоритмы машинного обучения** - используются для обучения компьютерных систем на основе определенных данных. Они позволяют компьютеру самостоятельно находить закономерности в данных и принимать решения на основе этих закономерностей.
- **Алгоритмы шифрования** - используются для защиты информации от несанкционированного доступа. Они позволяют зашифровать данные таким образом, что только авторизованные пользователи могут получить доступ к ним.

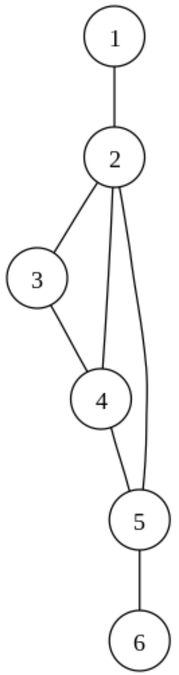
Виды алгоритмов

- **Алгоритмы компьютерного зрения** - используются для обработки и анализа изображений и видео. Они позволяют компьютеру распознавать объекты, лица, текст и другие элементы на изображении.
- **Нейронные сети** - используются для моделирования работы человеческого мозга и решения сложных задач обработки информации. Они позволяют компьютеру обучаться на основе большого количества данных и принимать решения на основе этих данных.

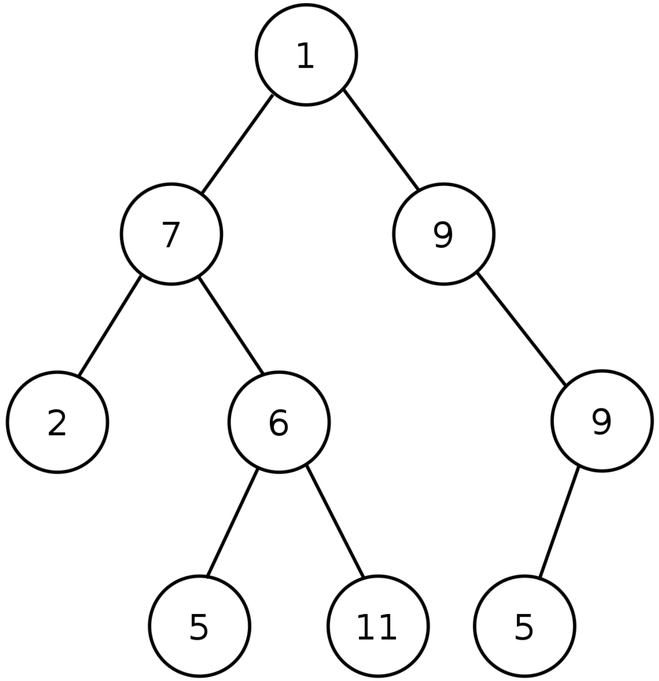
Это только небольшой набор типов алгоритмов, используемых в программировании. В зависимости от задачи, может быть использован различный тип алгоритма.

Жадные алгоритмы

это алгоритмы, которые всегда выбирают **наилучшее доступное решение** на каждом шаге, без учета будущих последствий.Обычно такие алгоритмы рассматривает раздел математического моделирования (нахождения оптимального решения той или иной задачи). Тут изучают графы и деревья.



граф



дерево

Рекурсивные алгоритмы

это алгоритмы, которые вызывают сами себя для решения подзадачи, которая является частью общей задачи. Рекурсивные алгоритмы используются для решения задач, которые могут быть разбиты на более мелкие задачи того же типа.

```
def factorial(n):

    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5))
```

Алгоритмы поиска

Алгоритмы поиска используются для нахождения элемента или набора элементов в заданной структуре данных, такой как массив или список. В Python существует несколько алгоритмов поиска, включая линейный поиск, бинарный поиск и поиск с помощью хеш-таблиц.

Алгоритмы поиска

Этот код определяет функцию `linear_search`, которая принимает на вход массив array и целевой элемент `target`. Функция перебирает каждый элемент массива по порядку, начиная с первого, и проверяет, равен ли текущий элемент целевому элементу. Если текущий элемент равен целевому элементу, то функция возвращает его индекс в массиве. Если целевой элемент не найден в массиве, функция возвращает -1.

```
def linear_search(array, target):

    for i in range(len(array)):
        if array[i] == target:
            return i
    return -1

print(linear_search([1, 3, 5, 7, 9], 5))
```

Например, вызвать функцию `linear_search([1, 3, 5, 7, 9], 5)`, то она переберет каждый элемент массива по порядку и найдет, что элемент 5 находится на индексе 2. Если мы вызовем функцию `linear_search([1, 3, 5, 7, 9], 2)`, то функция вернет -1, т.к. элемент 2 отсутствует в массиве.

Алгоритмы сортировки

Используются для упорядочивания элементов в заданной структуре данных, такой как массив или список.

В Python существует множество алгоритмов сортировки, приведём наиболее распространенные: сортировка выбором, сортировка вставками, быстрая сортировка и сортировка пузырьком.

Сортировка выбором (selection sort)

Этот код определяет функцию `selection_sort`, которая принимает на вход массив *array*. Функция перебирает каждый элемент массива по порядку, начиная с первого, и находит наименьший элемент в оставшейся части массива. Затем он меняет местами текущий элемент с найденным наименьшим элементом. Это продолжается до тех пор, пока весь массив не будет отсортирован.

```
def selection_sort(array):

    for i in range(len(array)):
        min_index = i
        for j in range(i+1, len(array)):
            if array[j] < array[min_index]:
                min_index = j
        array[i], array[min_index] = array[min_index], array[i]
    return array

print(selection_sort([1, 4, 10, -2, 10]))
```

Сортировка вставками (insertion sort)

Этот код определяет функцию `insertion_sort`, которая принимает на вход массив array. Функция перебирает каждый элемент массива, начиная со второго, и вставляет его в отсортированную часть массива, расположенную слева от текущего элемента. Это продолжается до тех пор, пока весь массив не будет отсортирован.

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


print(insertion_sort([1, 5, 32, -10, 0]))
```

Быстрая сортировка (quick sort)

Этот код определяет функцию `quick_sort`, которая принимает на вход массив array. Функция выбирает опорный элемент (*pivot*), разбивает массив на две части: одна часть содержит элементы, меньшие или равные опорному, а другая часть содержит элементы.

```
def quick_sort(array):
    if len(array) <= 1:
        return array
    else:
        pivot = array[0]
        less = [x for x in array[1:] if x <= pivot]
        greater = [x for x in array[1:] if x > pivot]
        return quick_sort(less) + [pivot] + quick_sort(greater)

print(quick_sort([1, 5, 32, -10, 0]))
```

Сортировка пузырьком (bubble sort)

Этот код определяет функцию `bubble_sort`, которая принимает на вход массив *array*. Функция перебирает каждый элемент массива, начиная с первого, и сравнивает его с каждым следующим элементом. Если следующий элемент больше текущего, они меняются местами. Это продолжается до тех пор, пока весь массив не будет отсортирован.

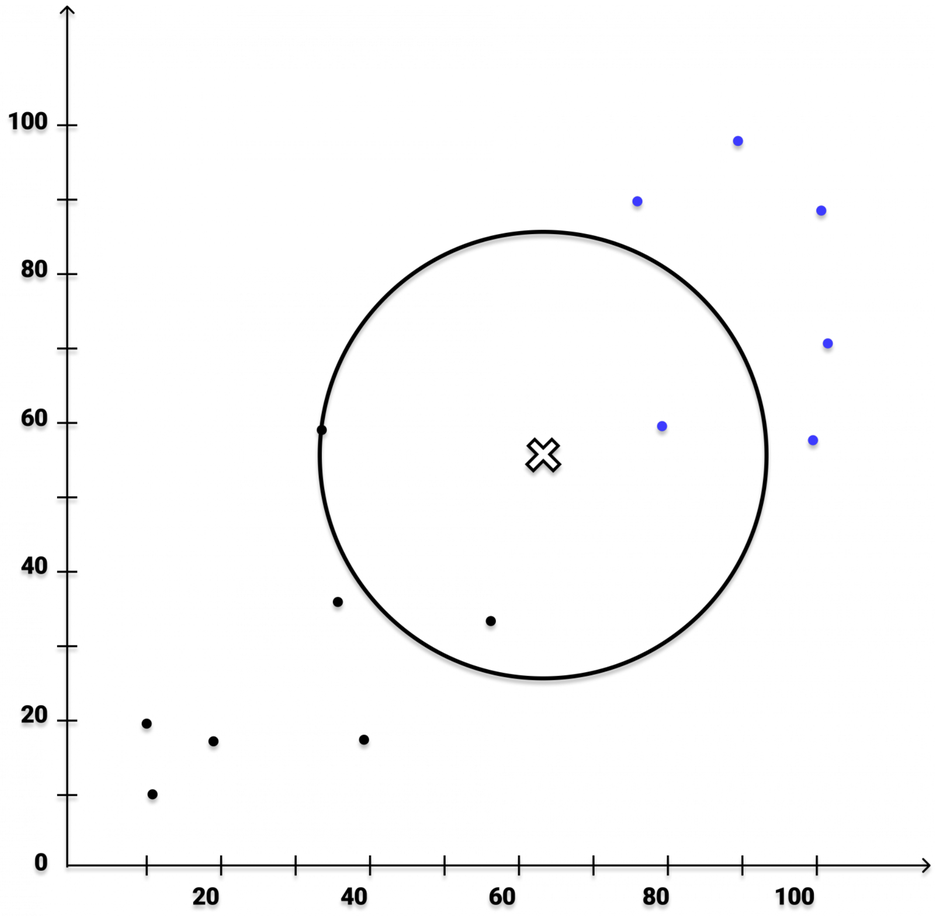
```
def bubble_sort(array):
    n = len(array)
    for i in range(n):
        for j in range(0, n-i-1):
            if array[j] > array[j+1]:
                array[j], array[j+1] = array[j+1], array[j]
    return array


print(bubble_sort([-10, 2, 30, 0]))
```

Алгоритмы машинного обучения

Нейронные сети - это алгоритмы, которые используются для моделирования работы человеческого мозга и решения сложных задач обработки информации. Они позволяют компьютеру обучаться на основе большого количества данных и принимать решения на основе этих данных.

Один из простых алгоритмов машинного обучения на Python - это **алгоритм k-ближайших соседей** (k-nearest neighbors, KNN).



Алгоритмы машинного обучения

KNN - это алгоритм классификации, который основывается на том, что объекты с похожими признаками скорее всего принадлежат к одному классу. Алгоритм классифицирует новый объект, основываясь на классах его k ближайших соседей в обучающей выборке.

Алгоритмы шифрования

это методы преобразования данных с целью их защиты от несанкционированного доступа или изменения. Они используются для шифрования информации, такой как сообщения, пароли, кредитные карты и другие конфиденциальные данные.

Один из простых примеров алгоритма шифрования на Python - это *шифр Цезаря*, простой метод шифрования, в котором каждая буква заменяется на букву, находящуюся на фиксированное число позиций в алфавите. Например, если выбрать сдвиг на 3 позиции, то буква A будет заменена на D, буква B - на E и т.д.

Алгоритмы шифрования

В этом примере функция `encrypt` принимает текст, который нужно зашифровать, и количество позиций, на которое нужно сдвинуть символы. Функция проходит по каждому символу текста и, если символ является буквой, находит его номер в алфавите, сдвигает его на заданное количество позиций и преобразует обратно в символ.Зашифрованный символ добавляется к результату. Если символ не является буквой, он просто добавляется к результату без изменений.

```
def encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            new_char = chr(ord(char.lower()) + shift)
            result += new_char.upper() if char.isupper() else new_char
        else:
            result += char
    return result

encrypted_text = encrypt("HELLO, WORLD!", 3)
print(encrypted_text)

decrypted_text = encrypt("KHOOR, ZRUOG!", -3)
print(decrypted_text)
```

Алгоритмы компьютерного зрения

это алгоритмы, которые используется для анализа и обработки изображений или видео. Эти алгоритмы используются для автоматического распознавания объектов, классификации изображений, обнаружения и распознавания лиц, сегментации изображений и других задач компьютерного зрения.

Алгоритмы компьютерного зрения могут включать в себя множество шагов, таких как предобработка изображения (например, удаление шума), выделение объектов на изображении, извлечение признаков объектов, классификация объектов и др.

Алгоритмы компьютерного зрения

Один из алгоритмов - это алгоритм распознавания лиц с использованием библиотеки **OpenCV**.
Пример кода: код использует каскадный классификатор Хаара для распознавания лиц на изображении с помощью функции *detectMultiScale()*. Затем найденные лица обводятся прямоугольниками с помощью функции *rectangle()*. Результат отображается в окне с помощью функции imshow(). Выход из программы осуществляется при нажатии клавиши «q".

```python
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

    cv2.imshow('frame', frame)

    if cv2.waitKey(1) & 6xFF == ord('q'):
        break

cap. release()
cv2.destroyAllWindows()
```

Для работы данной программы потребуется установить библиотеку opencv-python и файл xml прикрепленный к уроку.

Динамическое программирование (разделяй и властвуй)

"Разделяй и властвуй" (Divide and Conquer) - это метод разработки алгоритмов, который заключается в разбиении большой задачи на более мелкие, простые подзадачи, которые затем решаются независимо друг от друга, а затем объединяются для получения решения исходной задачи.

Классические примеры алгоритмов, использующих этот метод, включают в себя **быструю сортировку**, которая сначала разделяет массив на две части, сортирует их отдельно, а затем сливает их вместе в отсортированном порядке, и алгоритм **быстрого возведения в степень**, который рекурсивно делит задачу возведения в степень на две меньшие задачи, затем решает их и объединяет решения.

В большинстве случаев любой сложный алгоритм можно разбить на множество простых.