Алгоритмы в индустрии

10/05/2024

План лекции

- Поболтать о том, зачем вообще это все было учить
- Обсудить реальные жизненные задачи

Тут не будет правильных ответов, только личные наблюдения и мысли. Спокойно задавайте вопросы!

Проблематика

> Алгоритмы в реальной жизни программиста не нужны, и для меня от них есть только косвенная польза, так как их спрашивают на собеседованиях.

Ответим в трех частях:

- Глобальная польза от алгоритмов
- Навыки, которые алгоритмы прокачивают
- Ежедневное взаимодействие с алгоритмами

Модель простого IT-продукта

- Интерфейс для пользователя
- "Бизнес"-логика, специфична для продукта
- Логика общего вида, не привязанная к продукту

Будем обозначать за UI, Logic, Core

Пример: Subway Surfers

- UI: мобильное приложение, свайпы для смены путей.
- Logic: после второго столкновения с препятствием тебя ловит контролер.
- Core: проверка на столкновения.

Пример: Гугл-календарь

- UI: веб-сайт.
- Logic: напомнить за 15 и 5 минут до события.
- Core: база данных для сохранения событий, отправка уведомлений.

Пример: Телеграм-бот для скачивания записей лекций

- UI: Телеграм + Markdown для верстки сообщений.
- Logic: Отправка актуальной ссылки и скачанного видео в нужном разрешении на скорости x1.5.
- Core: Выгрузка видео с youtube.

Core: Алгоритмы

По определению, алгоритм это некоторая формализация процесса получения результата из аргументов (обычно через набор инструкций).

Алгоритмы не обязаны лежать в Core. Но, как правило, если в проекте используется алгоритм, он является core-модулем, потому что его вход параметризуется, и его разумно изолировать.

Скорее всего, это означает, что какой-то программист занимается конкретно этим core-модулем, а вы выступаете его пользователем.

Организация

- External service (например, REST API)
- Third-party libraries (например, open source)
- Самописные модули

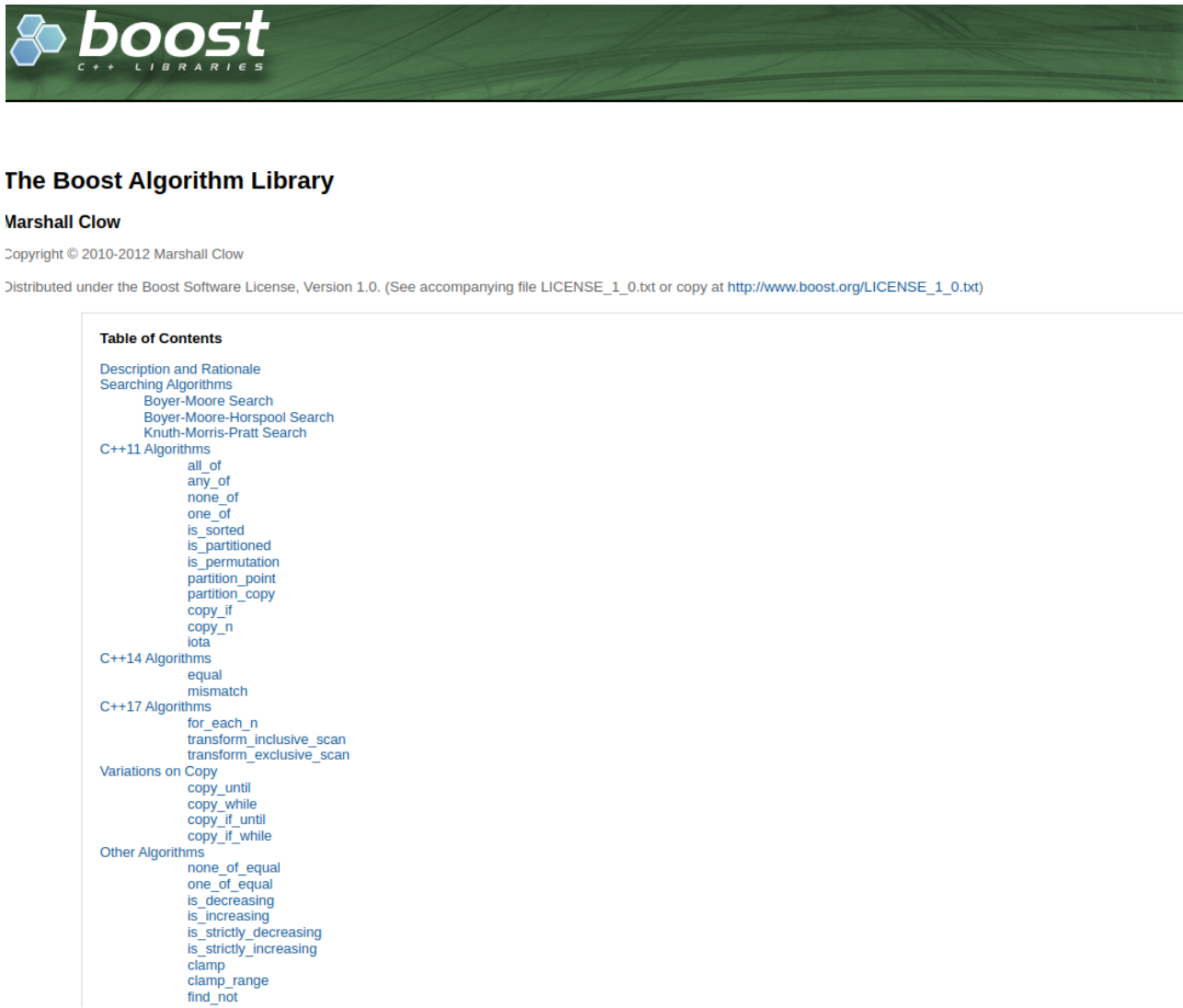
Пример: OpenAI API

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.Completion.create(
  model="text-davinci-003",
  prompt="Say this is a test",
  max_tokens=7,
  temperature=0
)
```

```
{
  // ...
      "text": "\n\nThis is indeed a test",
  // ...
}
```

Пример: Boost (на букву A)

- Accumulators
- Algorithm
- Align
- Any
- Array
- Asio
- Assert
- Assign
- Atomic



API

API является описанием интерфейса модуля. Можно считать, что модуль является черным ящиком, который соответствует API.

Предполагается, что при создании Core-компонент их API является API общего назначения и не принимает ничего, что бы относилось к Logic-компонентам.

Callback в API

При тесном взаимодействии Logic и Core можно предоставить возможность исполнения произвольного кода в виде function-as-an-argument.

```
let numbers = vec![1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let even_numbers: Vec<i32> = numbers
    .iter()
    .filter(|&n| n % 2 == 0)
    .map(|&n| n * 2)
    .collect();

println!("Doubled even numbers: {:?}", even_numbers);
```

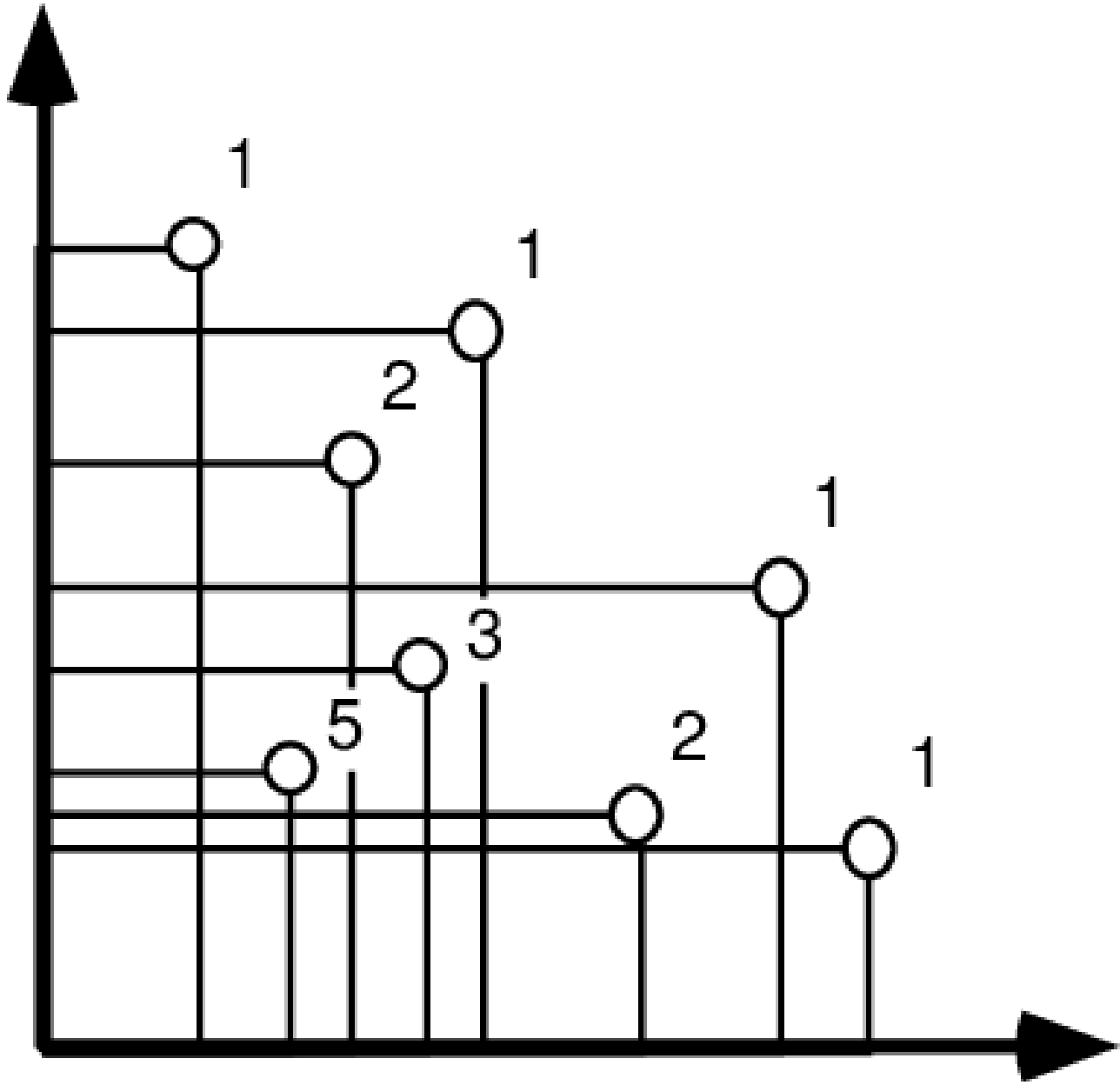
Работа с алгоритмами

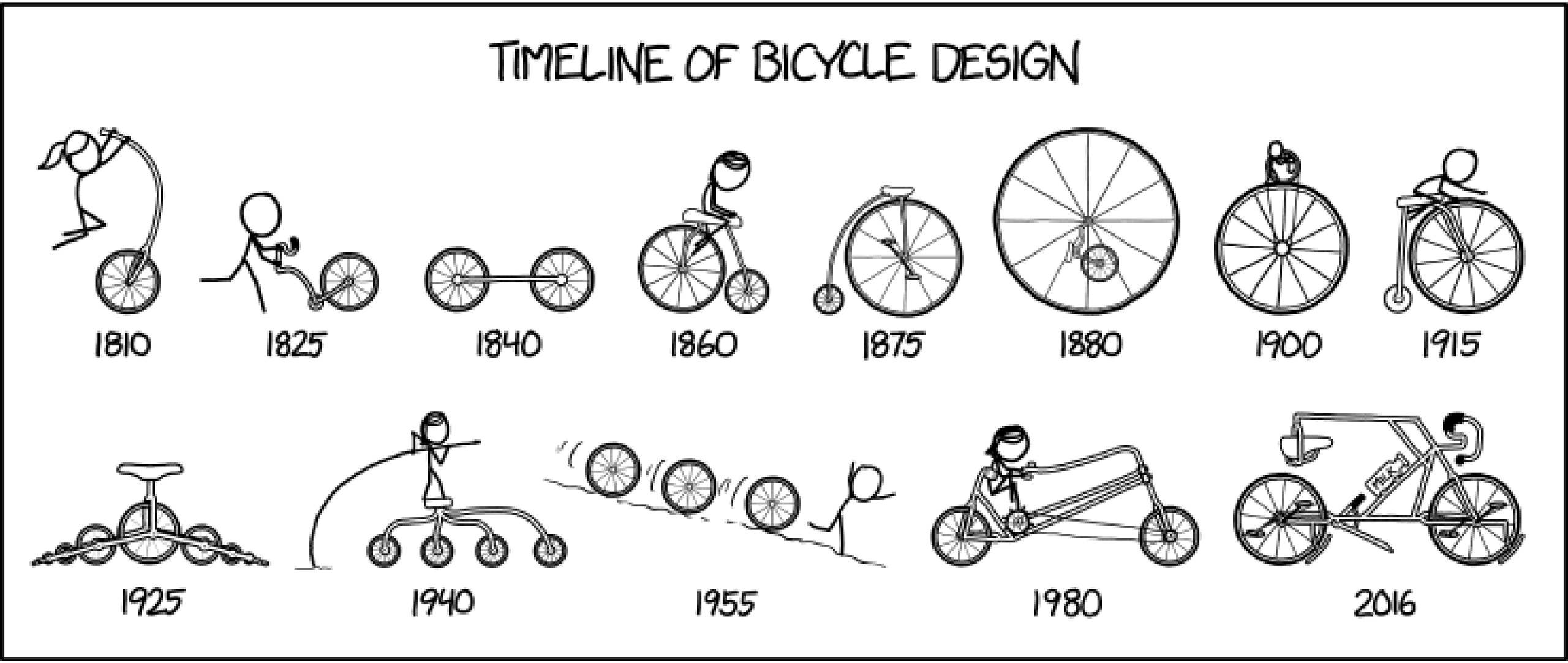
- Чтение статей
- Изучение benchmark-ов
- Чтение open-source реализаций
- Интеграция core-модулей

Полезные сайд-эффекты

- Внимание к corner cases
- Performance майндсет

Появление новых алгоритмов

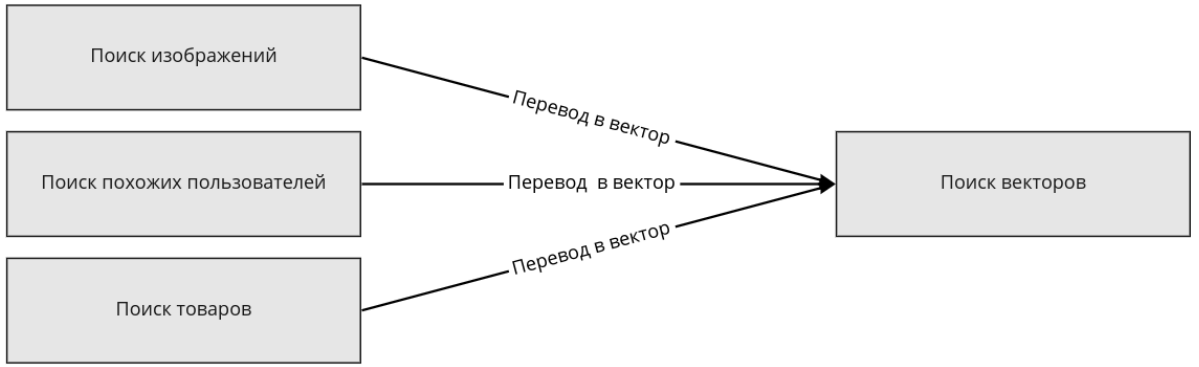



TIMELINE OF BICYCLE DESIGN
1810
1825
1840
1860
1875
1880
1900
1915
1925
1940
1955
1980
2016

Пример: поисковый сервис

Цель: сделать поисковый сервис для кастомного типа объектов

Подход: сведение задачи к поиску векторов

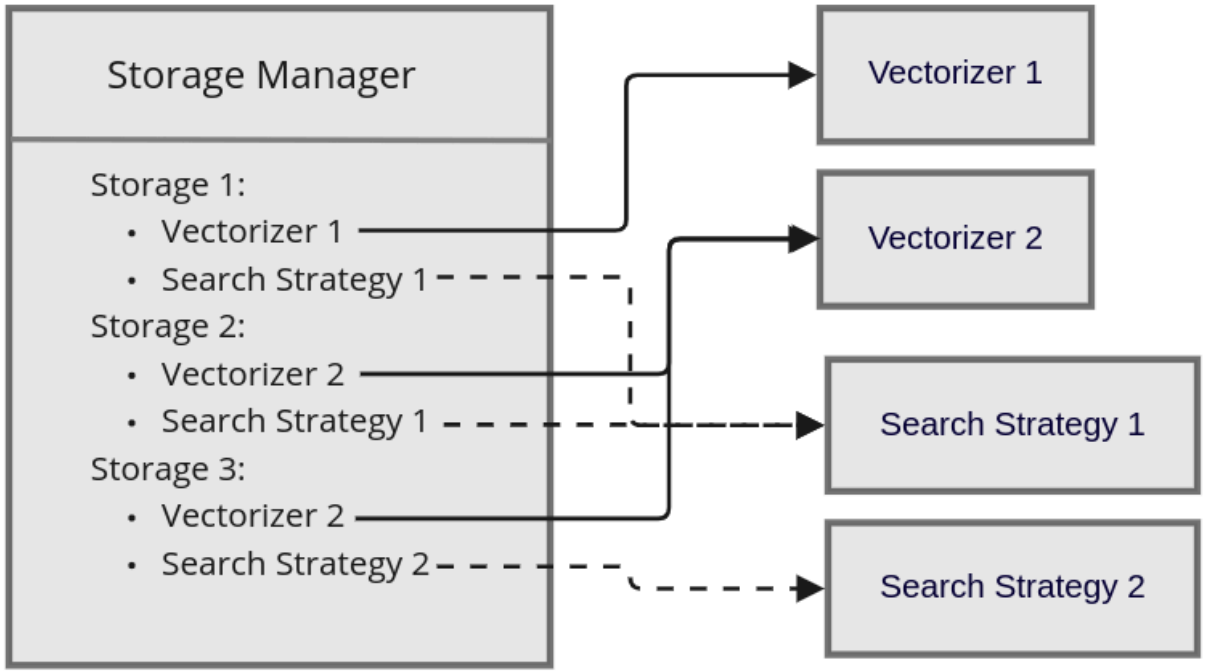


Vectorizer + Search strategy

Для того, чтобы решать поисковую задачу, можно сделать Storage с API:

- Add
- Delete
- Find top N matches

Функционал можно разбить на две Core-компоненты, которые можно комбинировать в произвольных конфигурациях.



API

```
class Vectoriser:
    def transform(object: SearchObject) -> Vector: ...

class SearchStrategy:
    def search(vector: Vector) -> List[SearchObject]: ...
    def add(vector: Vector): ...
    def remove(vector: Vector): ...
```

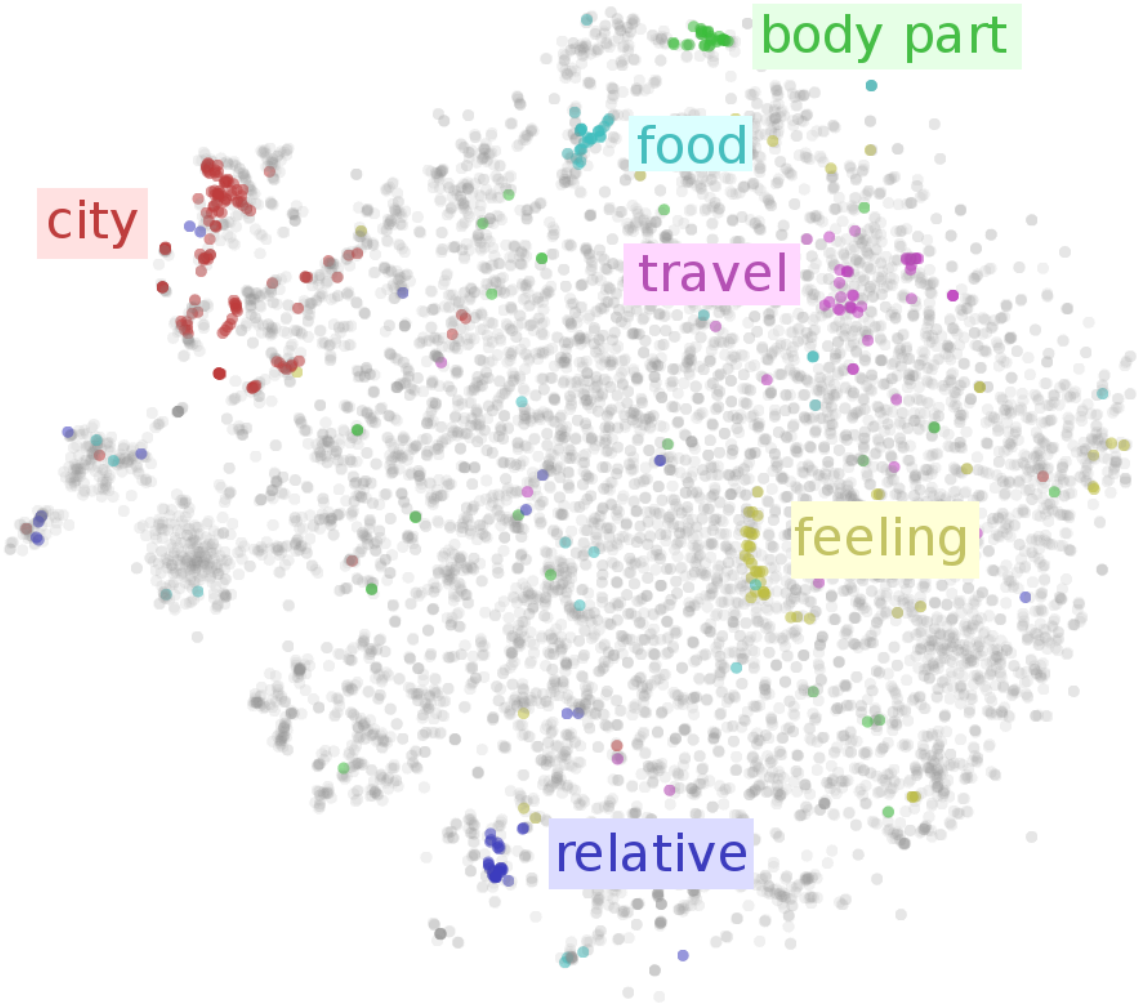
Векторизатор

Нейронные сети можно использовать для создания embedding-ов: вложений объектов в многомерное векторное пространство. При этом похожие объекты соответствуют близким векторам

Стандартная картинка: $1280 \times 720 \times 3$ чисел

Пожатый эмбеддинг: 512 чисел

Эмбеддинг для слов: Word2Vec



Как найти свой эмбеддинг?

Исследование!

Для картинок, например, нормально работает посчитать последний слой сети-классификатора, типа ResNet.

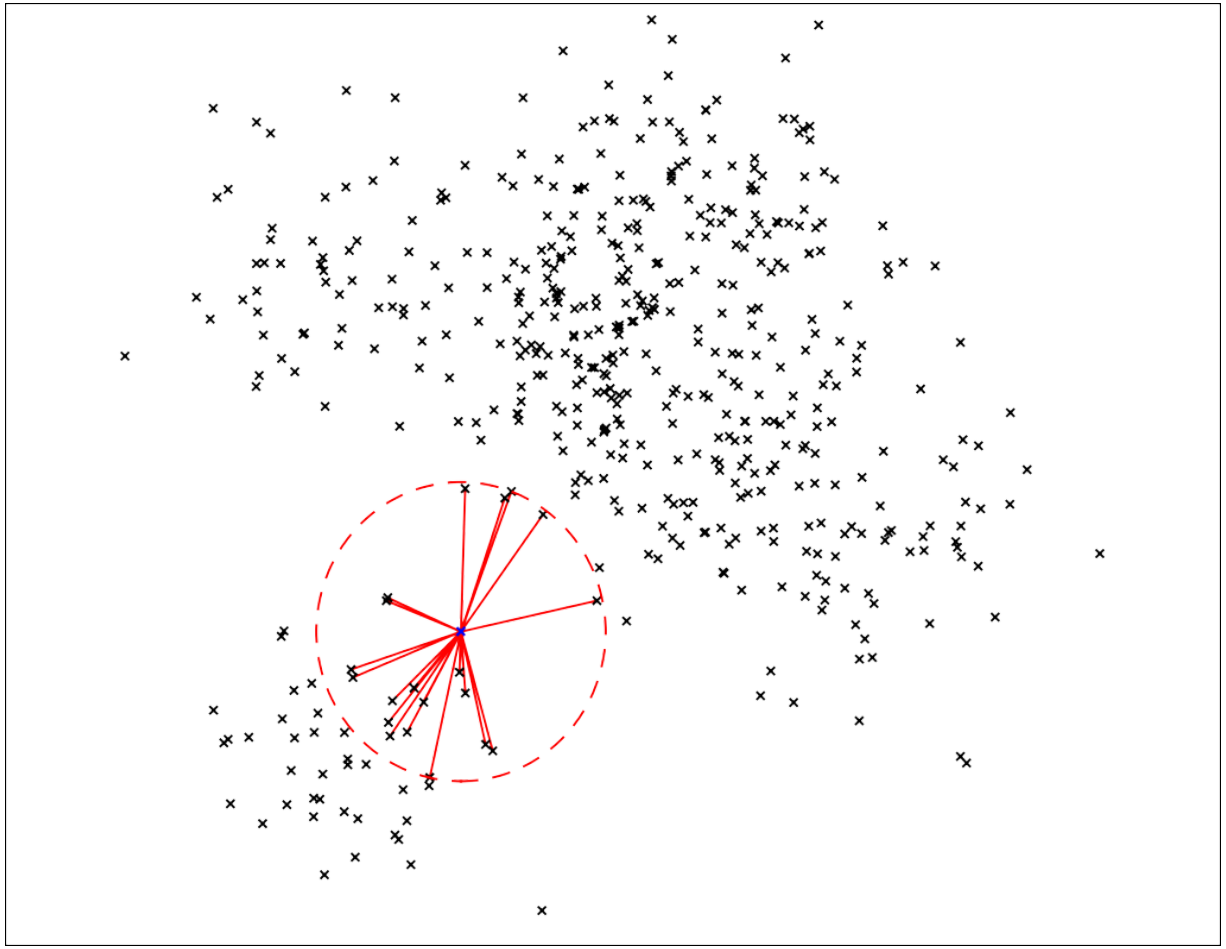
Векторизатор: реализация

Позволить добавлять кастомный python-скрипт, который работает с моделью, получает на вход объект, и выдает вектор.

Поисковая стратегия

Исследуем задачу поиска похожих векторов.

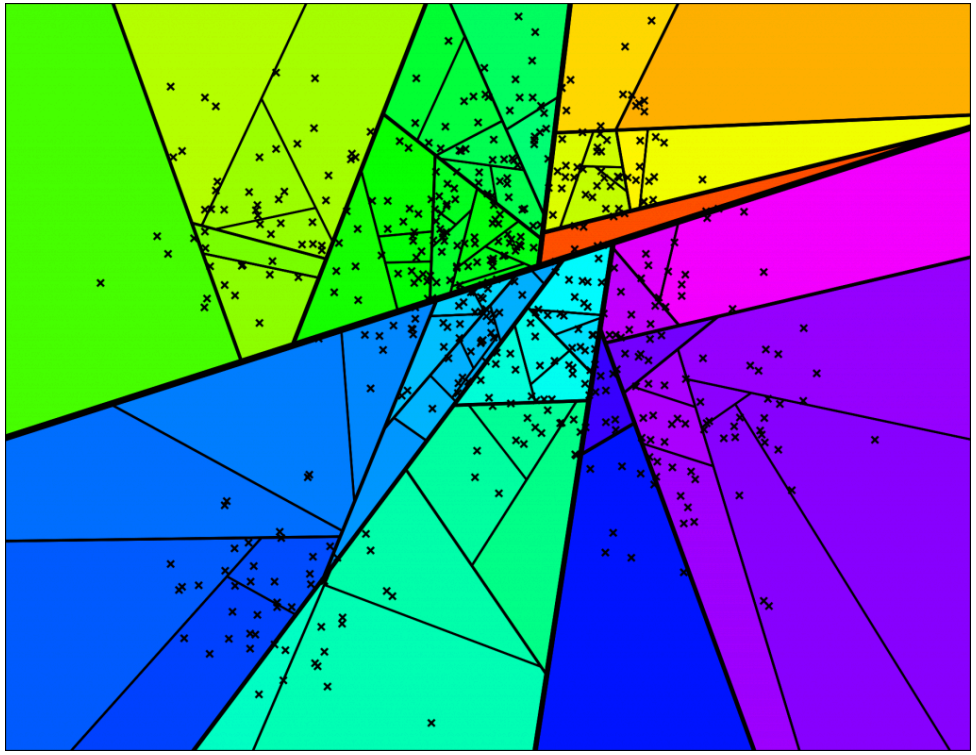
Оказывается, что это отдельный класс задач ANN (approximate nearest neighbor).

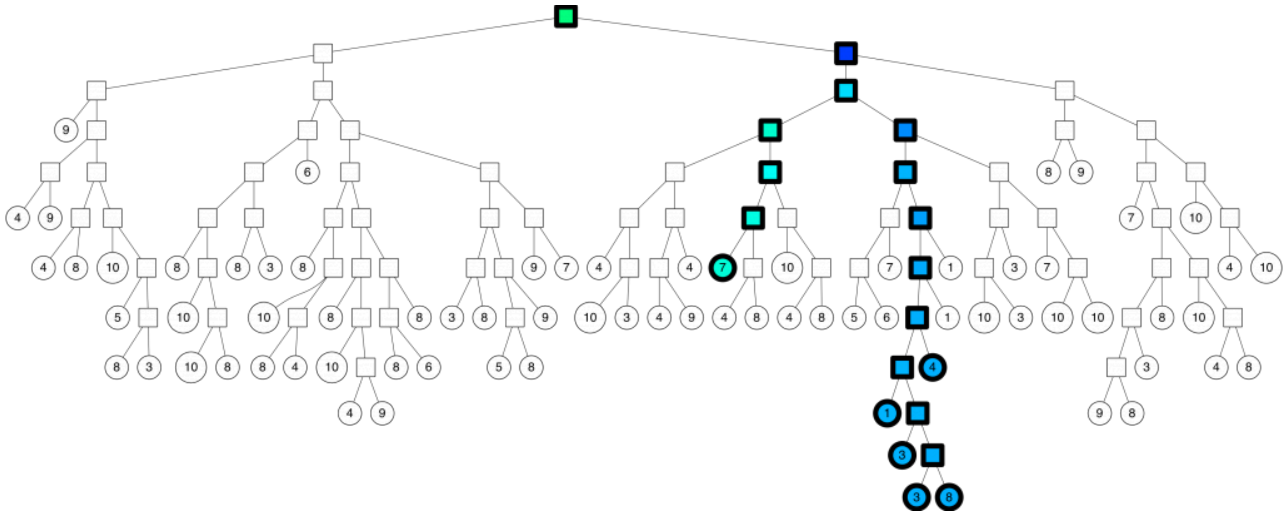


Решение задачи ANN

Как минимум 4 разных класса решений!

KD-деревья





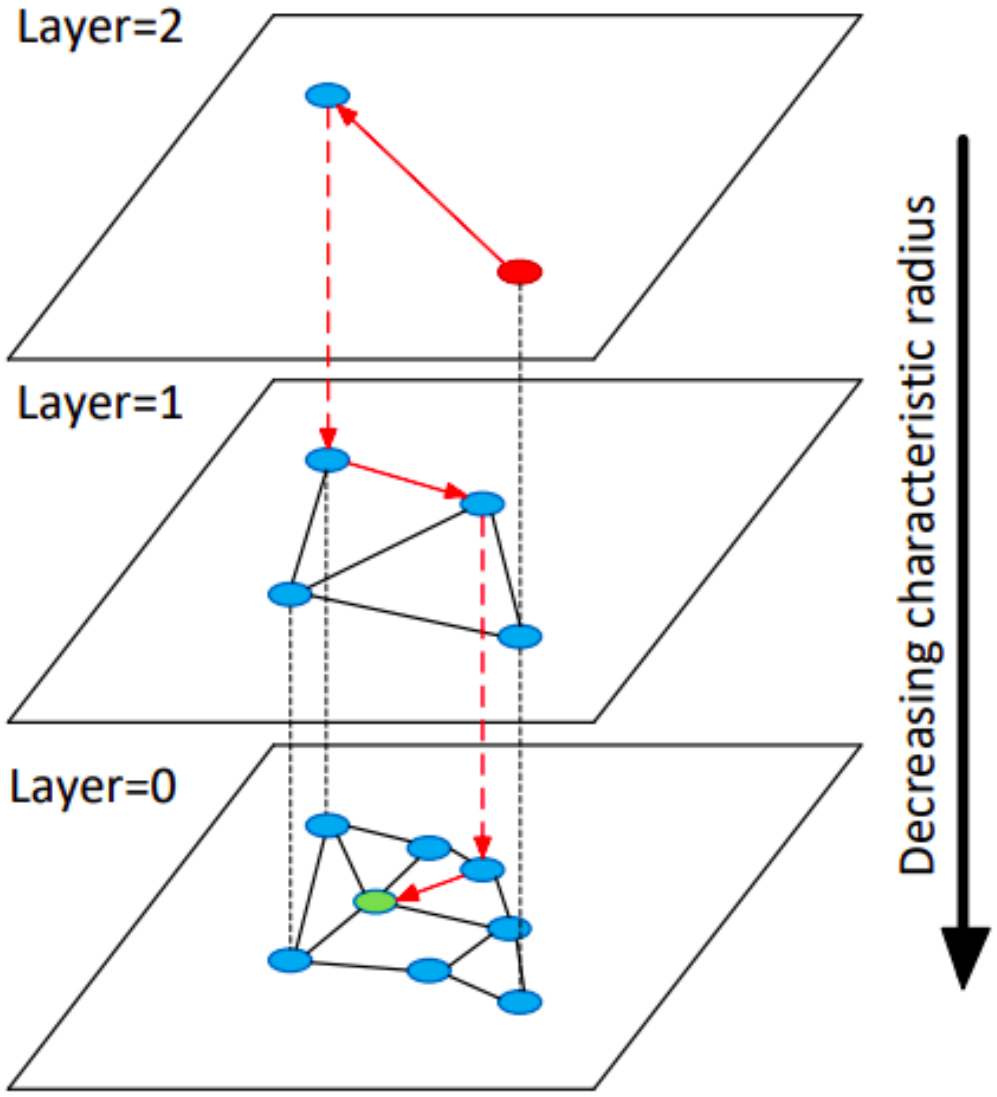
Spotify

Библиотека annoy

```
from annoy import AnnoyIndex
import random

f = 40  # Length of item vector that will be indexed

t = AnnoyIndex(f, 'angular')
for i in range(1000):
    v = [random.gauss(0, 1) for z in range(f)]
    t.add_item(i, v)

t.build(10) # 10 trees
t.save('test.ann')

# ...

u = AnnoyIndex(f, 'angular')
u.load('test.ann') # super fast, will just mmap the file
v = [random.gauss(0, 1) for z in range(f)]
print(u.get_nns_by_vector(v, 10)) # will find the 10 nearest neighbors
```

HNSW

Комбинация графов и skip-list-ов.



Faiss

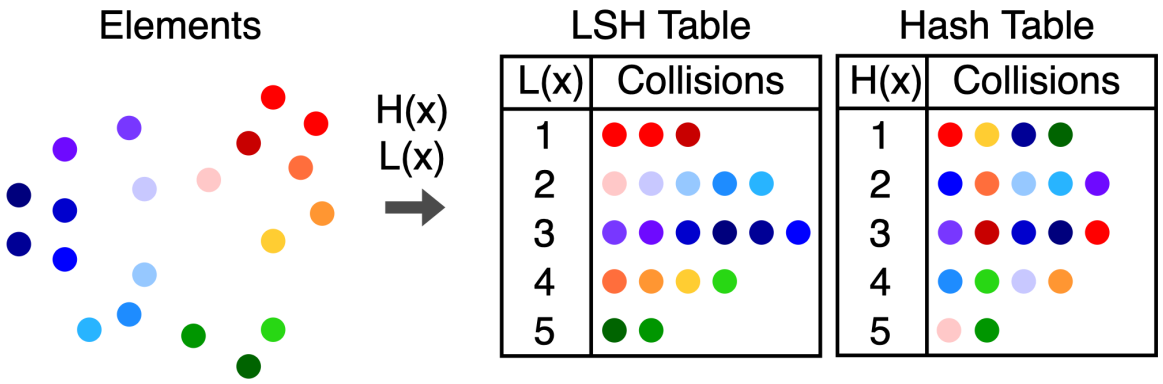
Facebook ANN Library. Есть HNSW и гораздо больше!

```
import numpy as np
import faiss
d = 64                                # dimension
nb = 100000                           # database size
index = faiss.IndexHNSWFlat(train_data)   # build the index
index.add(train)                       # add vectors to the index
print(index.ntotal)
k = 4                              # we want to see 4 nearest neighbors
D, I = index.search(test_data, k)
print(I)
print(D)
```

LSH

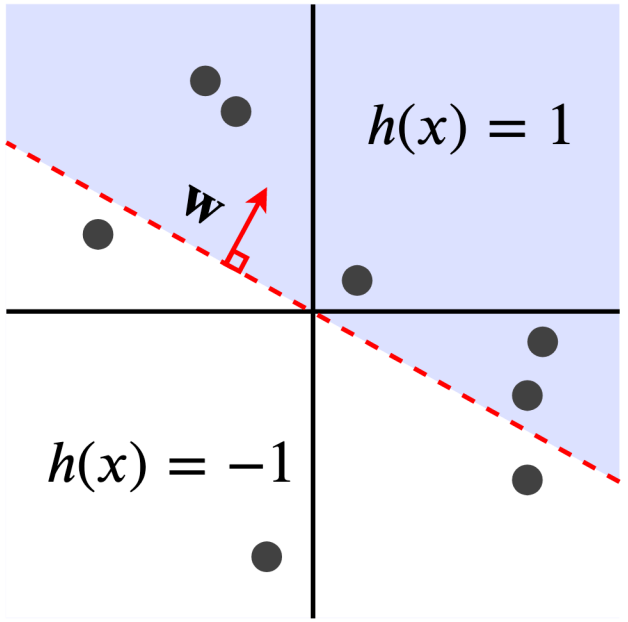
Хитрые хеш-функции, которые хешируют вектора с сохранением расстояния.

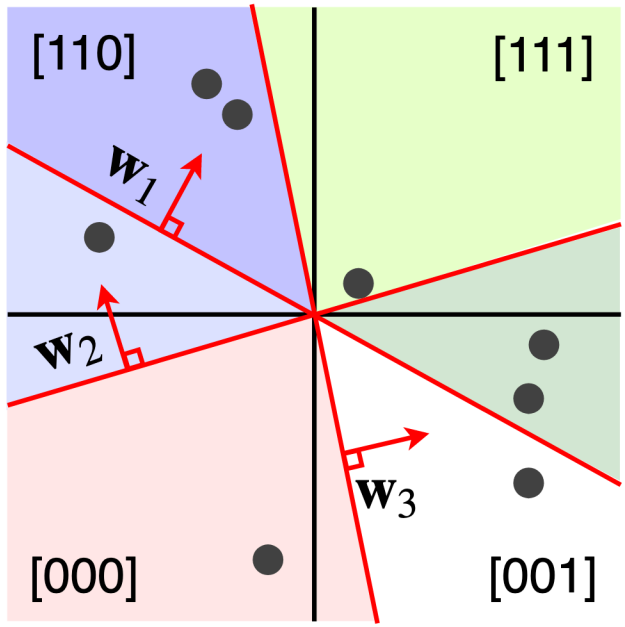
Коллизии соответствуют похожим элементам. Функция подбирается под каждое расстояние отдельно.



Косинусное расстояние

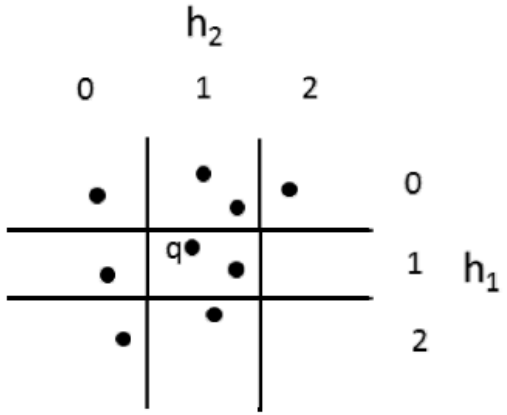
Генерация случайных прямых, которые делят пространство на "секторы". Хеш от точки соответствует сектору.



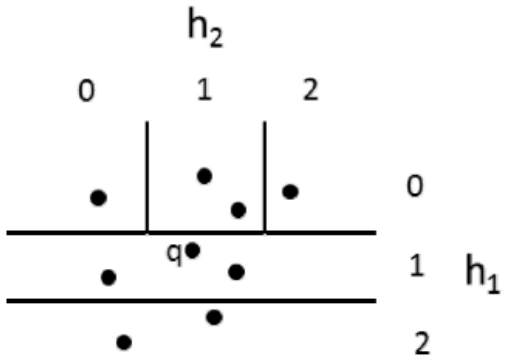


Динамические индексы

Чтобы не создавать сразу все секции хеша, мы можем увеличивать его точность в древовидной структуре



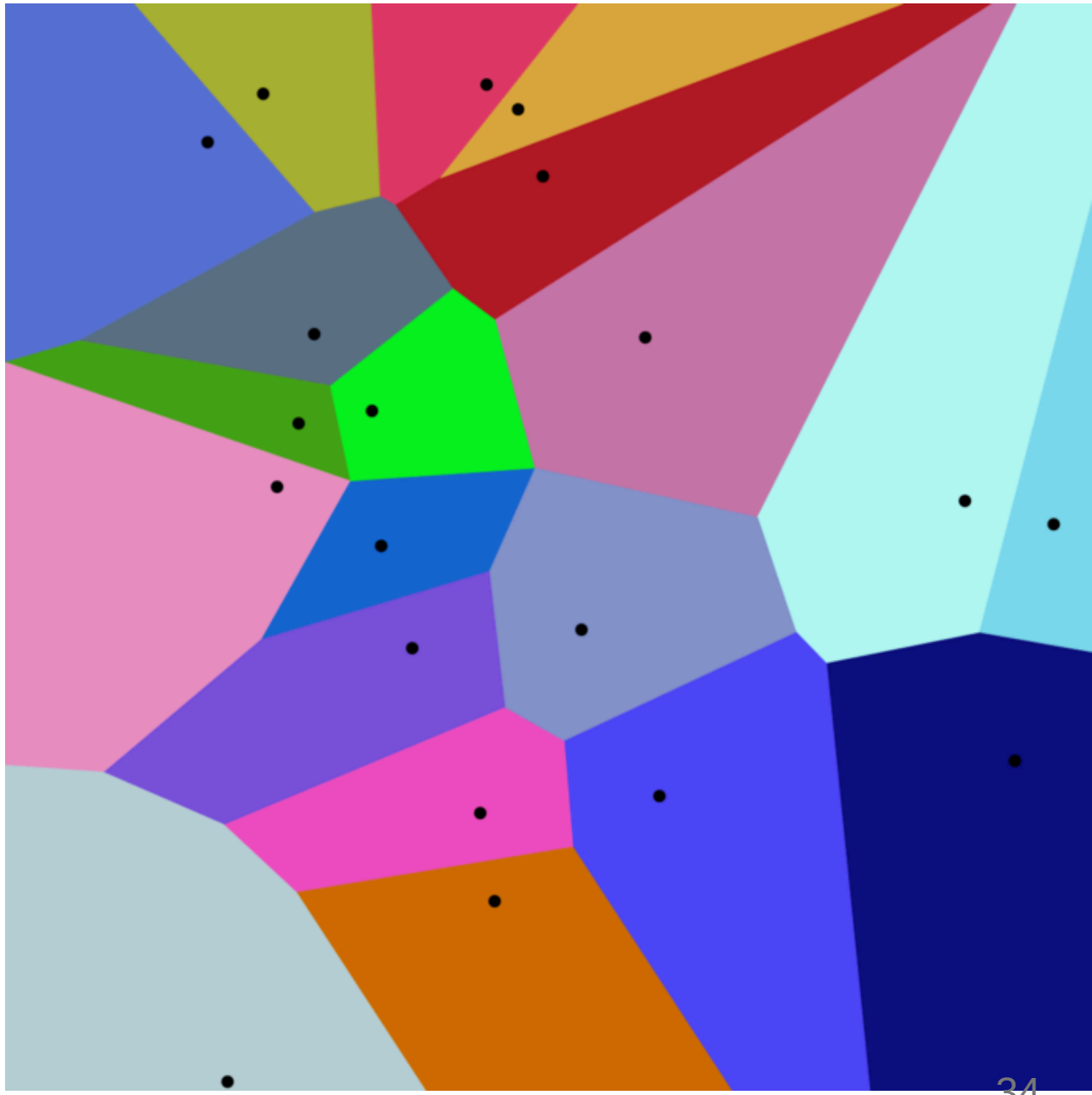
(a) Basic LSH



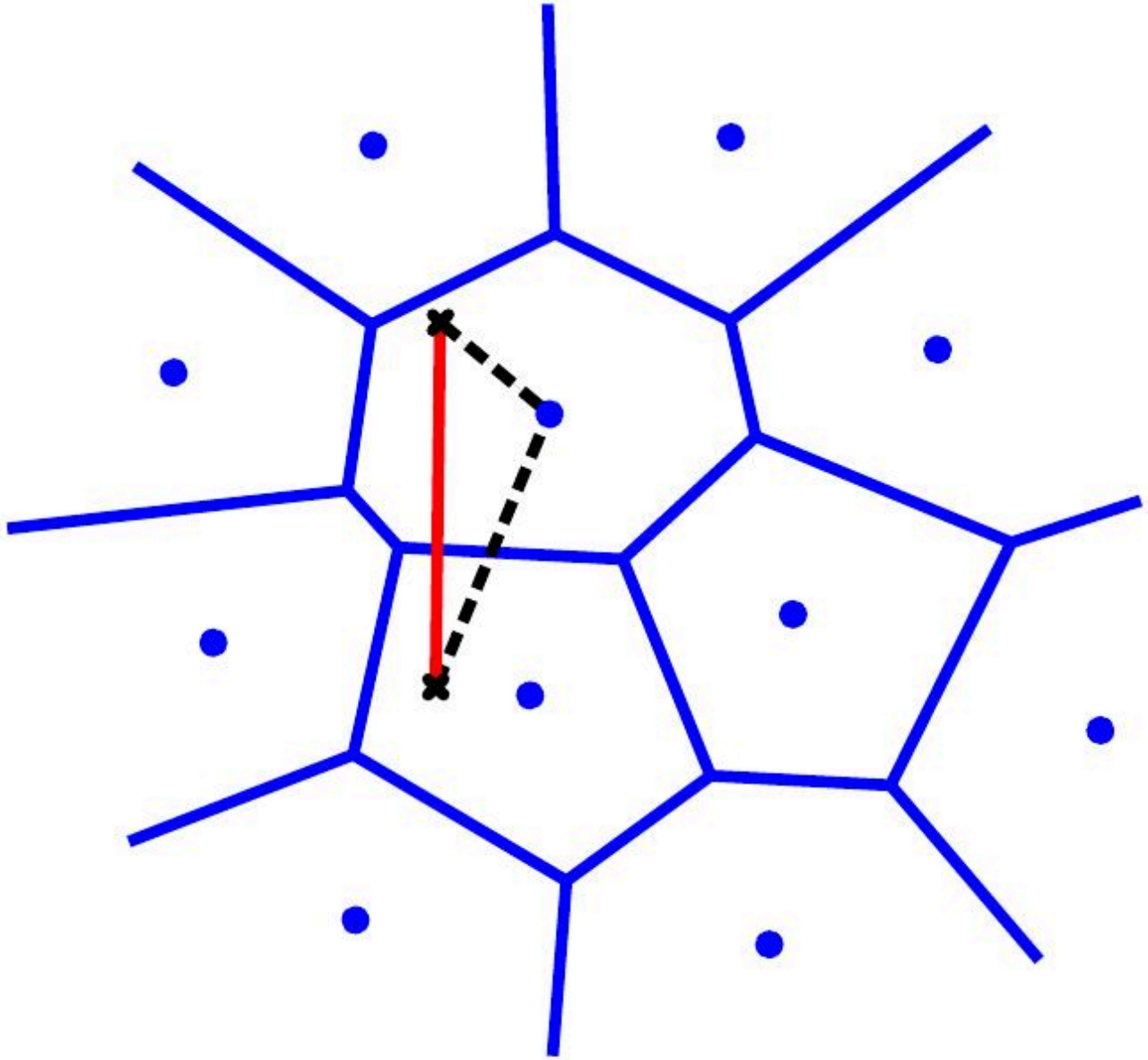
(b) DMLSH

Quantization

Делим области на кластеры, и каждую точку отождествляем с центром ее кластера.

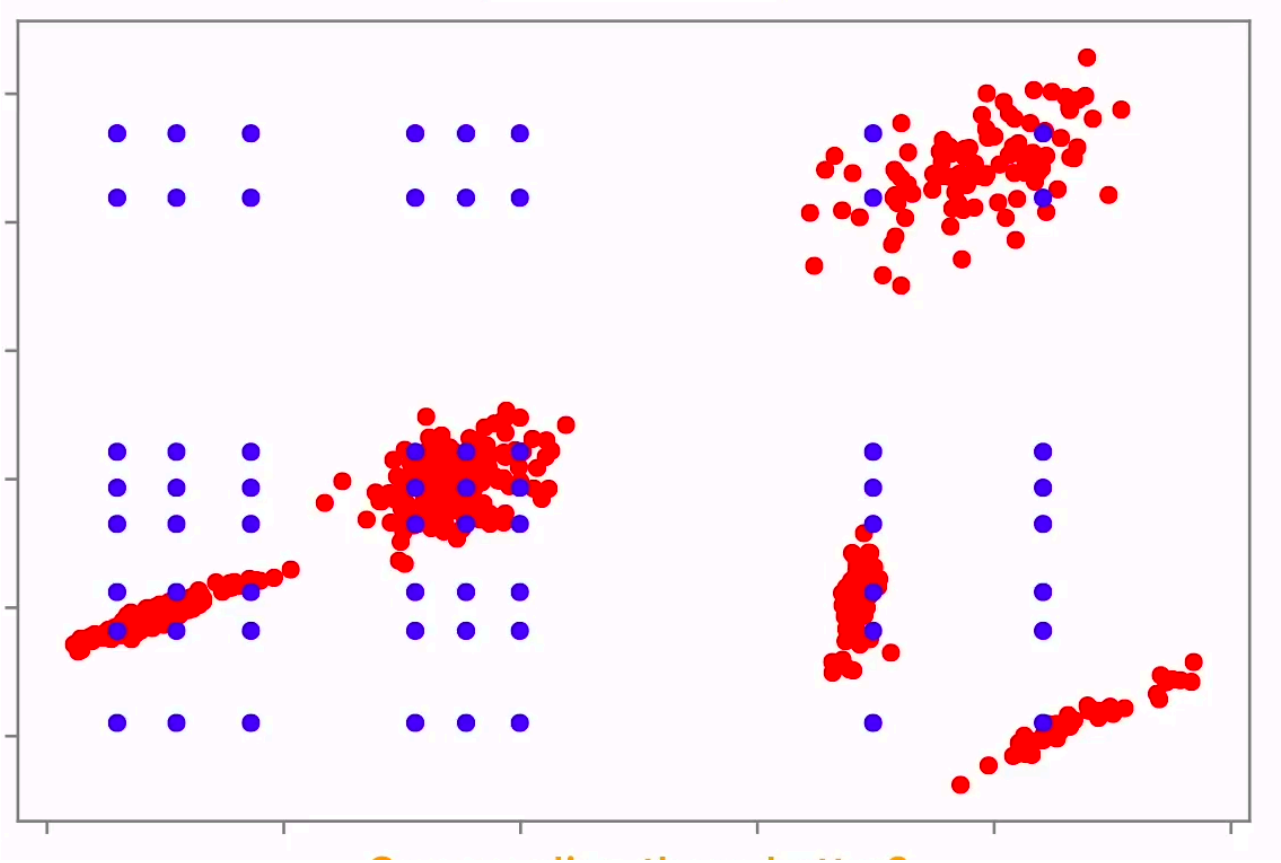


Потеря точности



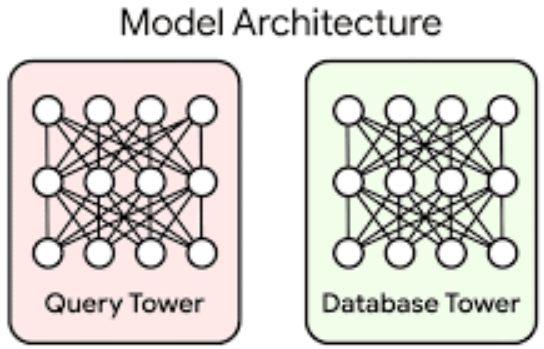
Увеличиваем количество кластеров

Кластеризуем покоординатно, за счет чего делаем большшую сетку центров размера $O(N \times M)$, храня при этом $O(N + M)$ памяти

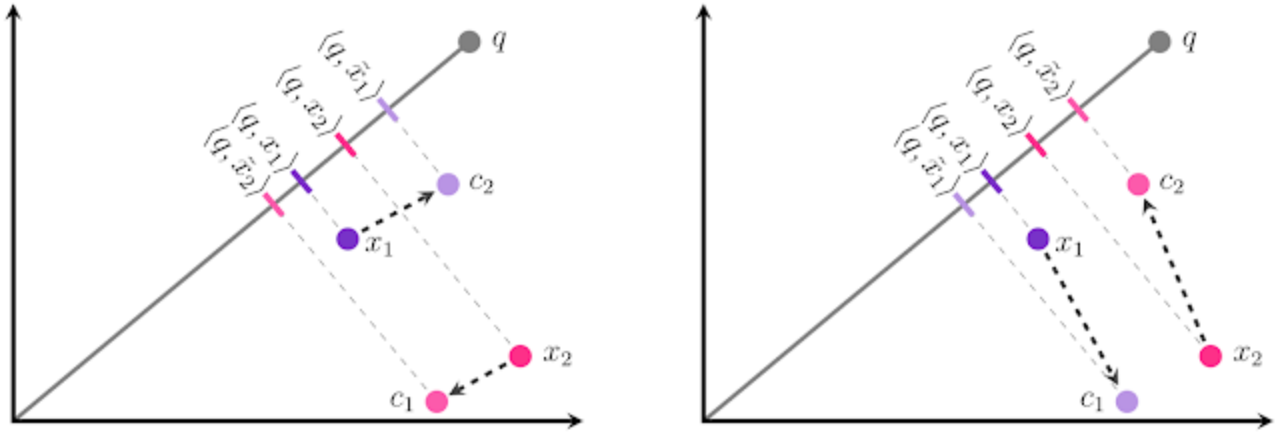


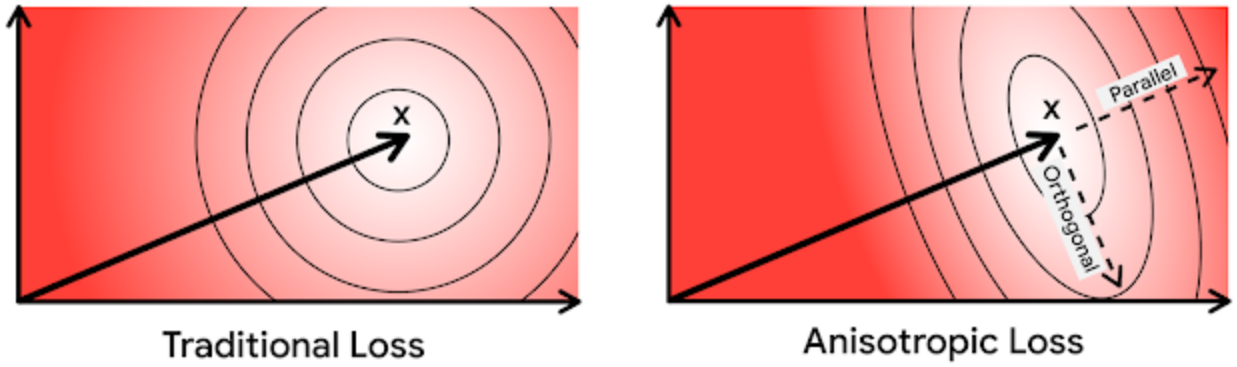
Scann

Цель такая же: найти эмбеддинг и выделить центры кластеров.



Оптимизация *похожести* скалярного произведения

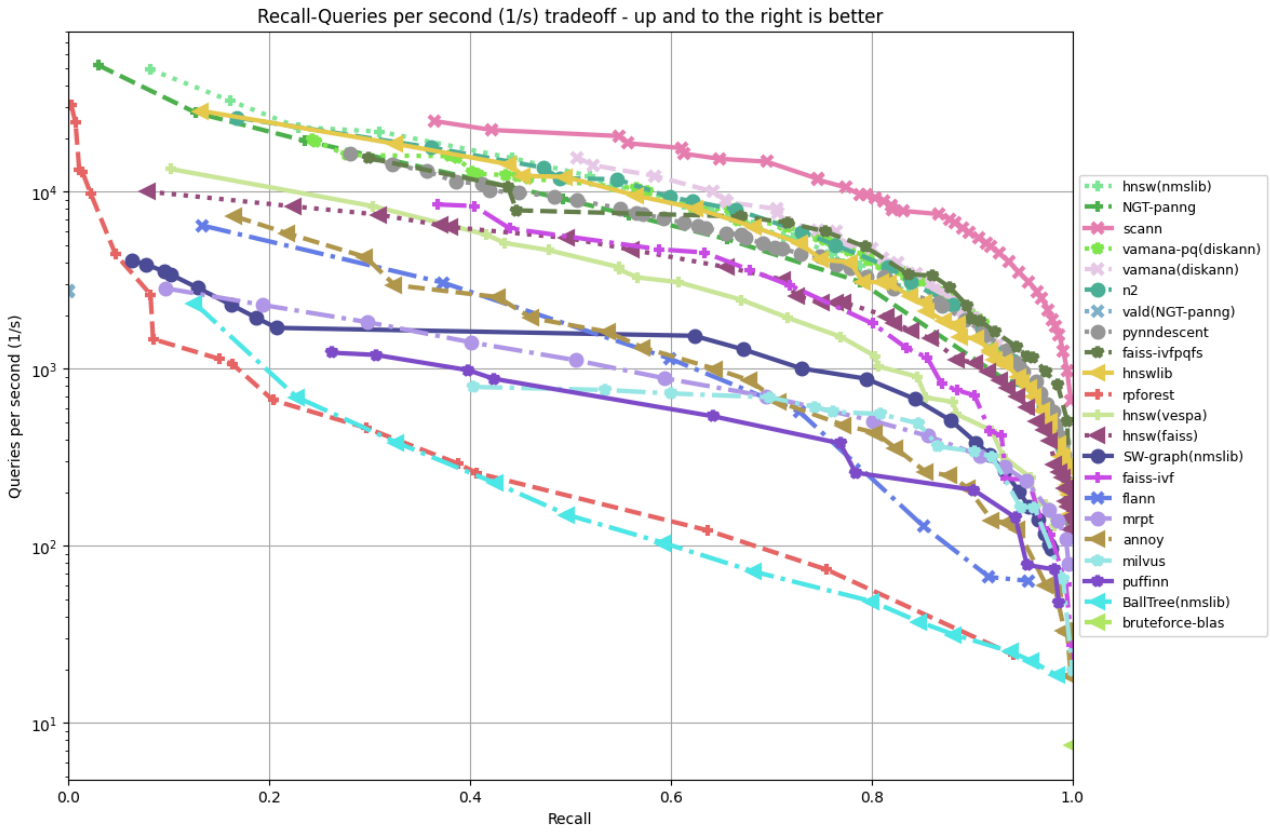




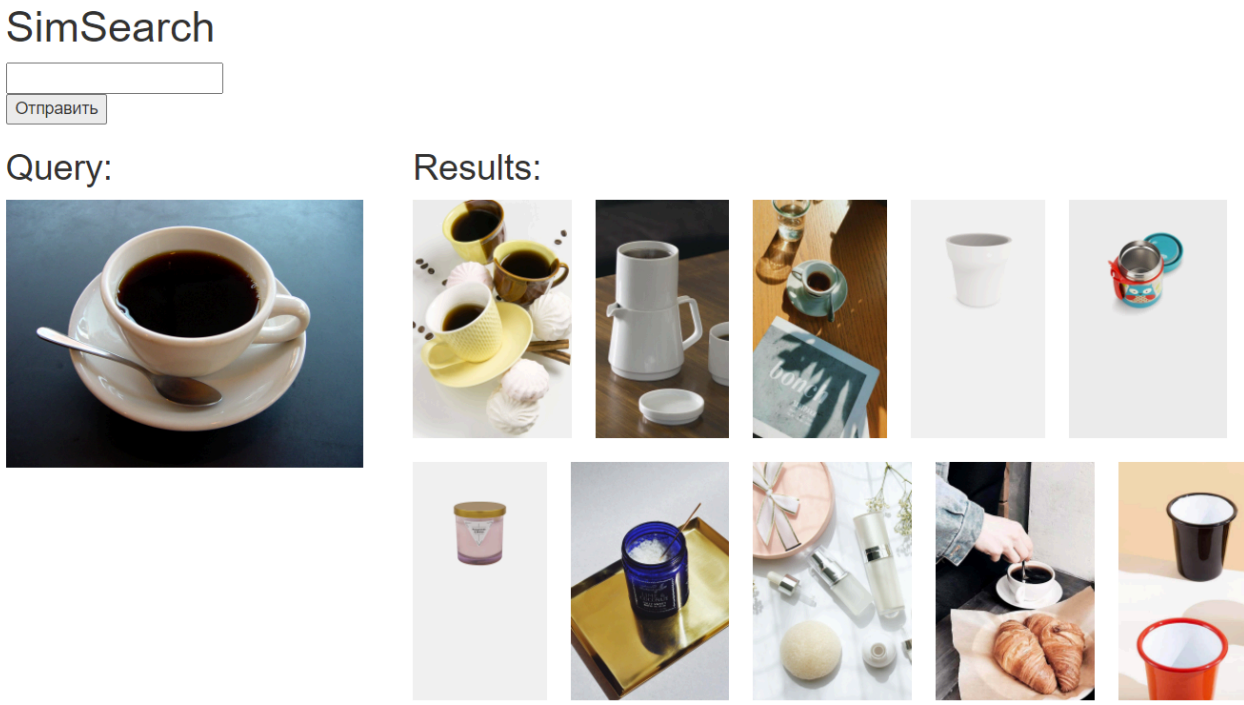
State-of-the-art 2020 года от Google

Benchmark

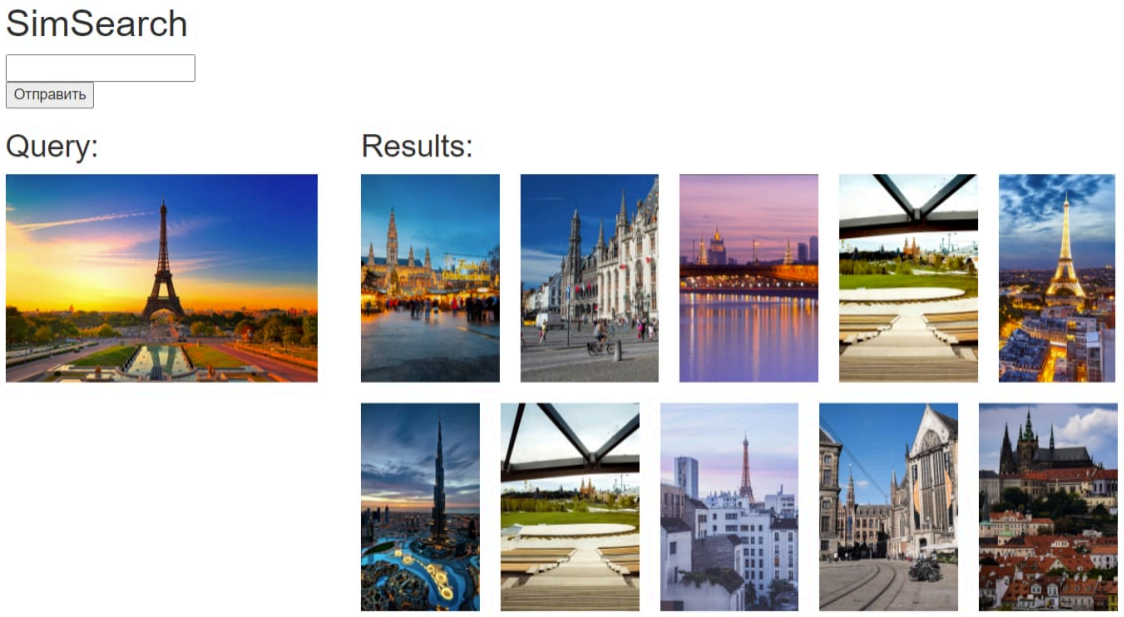
ссылка



Пример приложения



Еще примеры



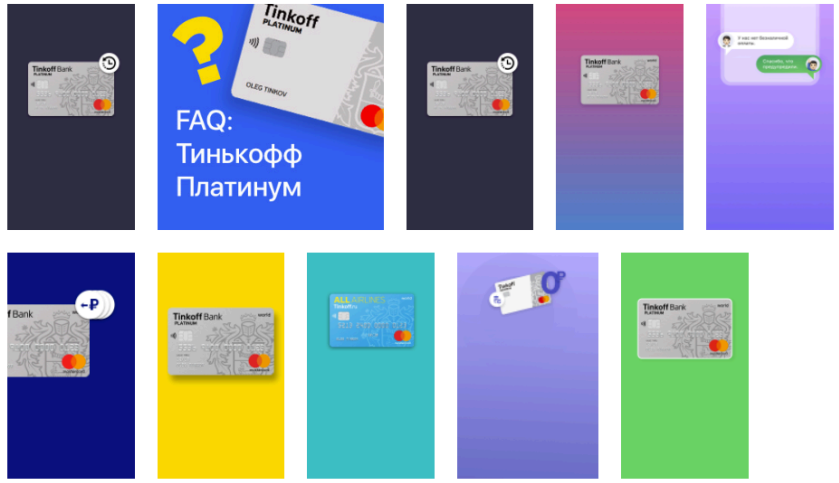
И еще

SimSearch

Отправить

Query:

Results:

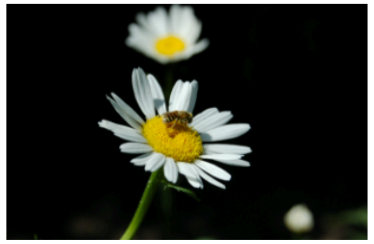


И напоследок

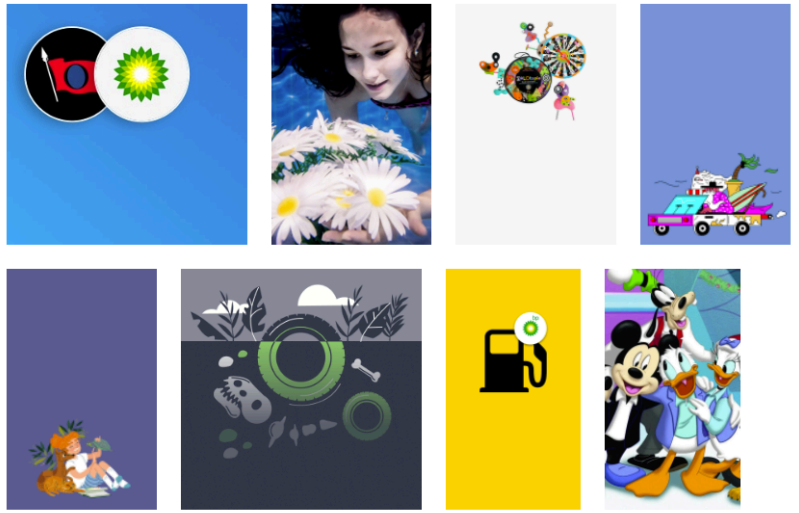
SimSearch

Отправить

Query:



Results:



Если интересно

Больше про этот сервис можно почитать в моих пдфках на гитхабе