Курс “Алгоритмы на python”

Занятие #10
Алгоритмы на строках

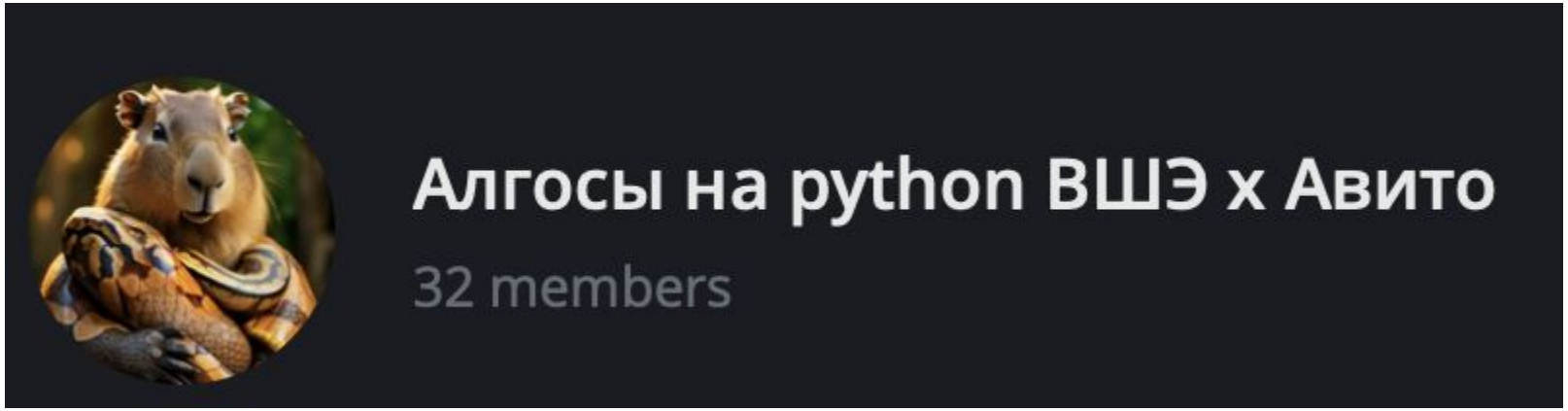
Сентябрь 2025

Avito

Единая точка входа/выхода – степик

- https://stepik.org/course/251189/

Вопросы и обсуждения – чат



Посещаемость

Орг моменты

Дедлайны

7 дз
* без штрафов 14 ноября включительно
* минус балл — 28 ноября включительно

8 дз
3 задачи — 12 баллов, за каждую по 4 балла.
Условия короткие, но, по факту, отражают все то, что мы разобрали на двух последних занятиях (кроме дейкстры).

Сроки сдачи:
до 3 декабря — без штрафа
до 17 декабря — с штрафом

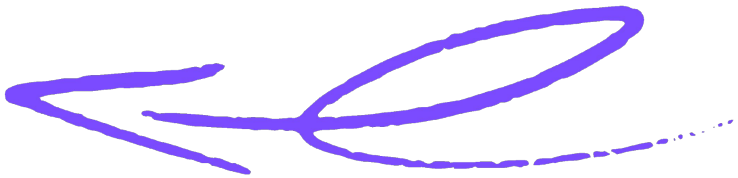
Структура курса «Алгоритмы на питоне»

1 модуль

- Введение в алгоритмы
- Базовые структуры данных
- Хеш-таблицы
- Бинарные деревья поиска
- Рекурсия
- Сортировки
- Кучи

2 модуль

- Графы
- Графы (продолжение)
- Алгоритмы на строках
- Алгоритмы в ML
- Алгоритмы в LLM



План занятия

- Часть I. Строки в python
- Часть II. Поиск подстроки в тексте
- Часть III. Метрики схожести
- Часть IV. Прямой и обратный индекс

Строки

Строки в Python

Текстовые данные в Python обрабатываются с помощью объектов str (строк)

Строки в Python

Строки — это неизменяемые последовательности символов в кодировке Unicode

Строки в Python

Строки — это неизменяемые последовательности символов в кодировке Unicode

```
>>> character = 'A'
>>> print(f"Unicode code point for '{character}': {ord(character)}")
Unicode code point for 'A': 65
>>> hex(65)
'0x41'
>>> print('\u0041')
A
>>>
```

Строки в Python

Строки записываются разными способами

- одинарные кавычки `'этот способ позволяет включать "двойные кавычки" в строку'`
- двойные кавычки `"этот способ позволяет включать 'одинарные кавычки' в строку"`
- тройные кавычки `'''тройные одинарные кавычки'''`, `"""тройные двойные кавычки"""`

Строки в Python

Строки — константные объекты

```
>>> s = 'abcd 123'
>>> s[1] = 'q'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Строки в Python

Строки поддерживают все распространенные операции с последовательностями такие, как
* сложение
* умножение
* подсчет длины последовательности
* взятие среза
* проверка наличия элементов последовательности

Строки в Python

```
>>> s = 'llm'
>>> id(s)
4300052240
>>> s = s + s
>>> s
'llmllm'
>>> id(s)
4300048448
>>> len(s)
6
>>> new_s = s * 5
>>> new_s
'llmllmllmllmllmllmllmllmllmllm'
>>> new_s[0]
'l'
>>> new_s[1:3]
'lm'
>>> new_s[:3]
'llm'
>>> 'llm' in s
True
>>>
```

Строки в Python

```
>>> s.
s.capitalize()    s.find(           s.isdecimal()     s.istitle()       s.partition(      s.rpartition(     s.swapcase()
s.casefold()      s.format(         s.isdigit()       s.isupper()       s.removeprefix(   s.rsplit(         s.title()
s.center(         s.format_map(     s.isidentifier()  s.join(           s.removesuffix(   s.rstrip(         s.translate(
s.count(          s.index(          s.islower()       s.ljust(          s.replace(        s.split(          s.upper()
s.encode(         s.isalnum()       s.isnumeric()     s.lower()         s.rfind(          s.splitlines(     s.zfill(
s.endswith(       s.isalpha()       s.isprintable()   s.lstrip(         s.rindex(         s.startswith(
s.expandtabs(     s.isascii()       s.isspace()       s.maketrans(      s.rjust(          s.strip(
```

Строки в Python

```
>>> iterable = ['hello', 'world']
>>> result = ','.join(iterable)
>>> result
'hello,world'
>>> result = ', '.join(iterable)
>>> result
'hello, world'
>>> result = 'QQ'.join(iterable)
>>> result
'helloQQworld'
```

Строки в Python

```
>>> s = 'one, two, three -- words of the string'
>>> s.split(',') # получим список слов в строке, которые разделены запятой
['one', ' two', ' three -- words of the string']
>>> s.split() # по умолчанию разделителем является пробел
['one,', 'two,', 'three', '--', 'words', 'of', 'the', 'string']
```

Строки в Python

```
>>> s = '  delete whitespaces   '
>>> s = s.strip() # удаляем символы в начале и в конце строки, по умолчанию -- пробелы
>>> s
'delete whitespaces'
>>>
>>> s = s.strip('des') # удаляем символы 'd', 'e', 's' в начале и в конце строки
>>> s
'lete whitespac'
```

Строки в Python

Ушли смотреть код токенизации в трансформерах

https://github.com/huggingface/transformers/blob/main/src/transformers/tokenization_utils.py#L982

Строки в Python

Примитивная токенизация

```
>>> text = '    Сбор и обработка данных    '
>>> res = text.strip().split()
>>> res
['Сбор', 'и', 'обработка', 'данных']
>>>
```

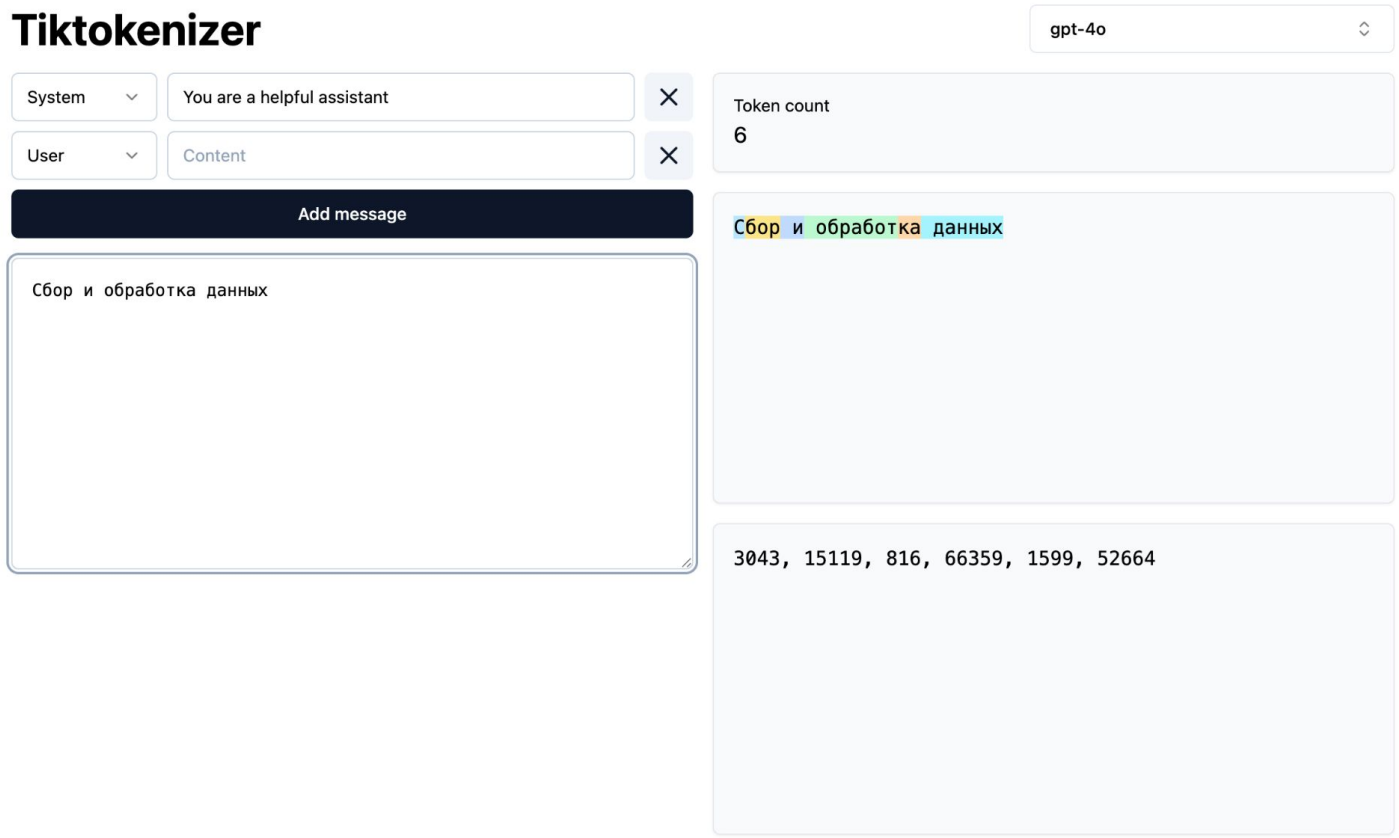
Строки в Python

Можно еще через регулярки

```
>>> text = '    Сбор и обработка данных    '
>>> words = re.findall(r'\w+', text.strip())
>>> words
['Сбор', 'и', 'обработка', 'данных']
>>>
```

Строки в Python

Как на самом деле
https://tiktokenizer.vercel.app/



Поиск подстроки

Дано:

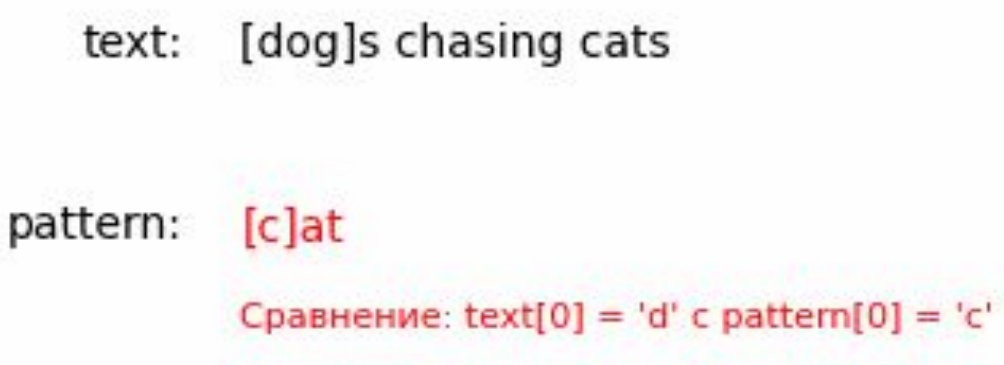
- Строка **text** длины (N).
- Подстрока **pattern** длины (M).

Хотим:

- Найти индекс первого вхождения **pattern** в **text**
- Вернуть -1, если подстрока отсутствует.

Поиск подстроки

Подход в лоб

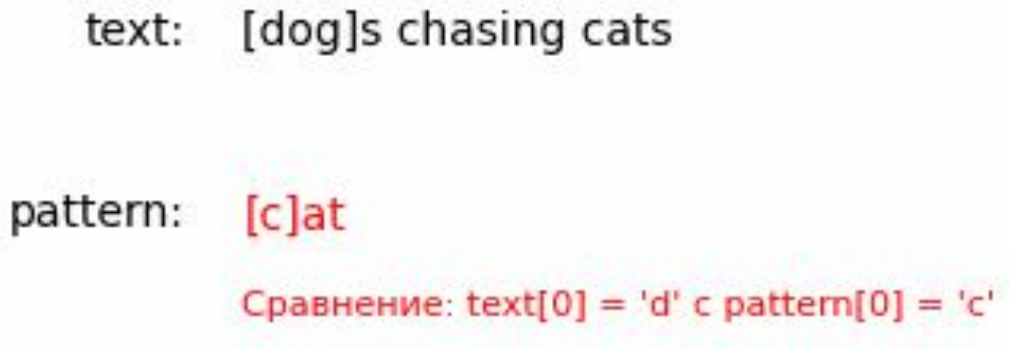


Поиск подстроки

Подход в лоб

N - M + 1
позиций

M проверок



Поиск подстроки. Алгоритм Рабина-Карпа

Хешируем подстроки для быстрого сравнения

- вычисляем хеш подстроки pattern
- для возможной позиции в строке text вычисляем хеш подстроки длиной M, начинающейся с этой позиции
- если хеши совпадают, выполняется посимвольное сравнение для подтверждения совпадения.

Поиск подстроки. Алгоритм Рабина-Карпа

Можно использовать полиномиальные хеши (будет в домашнем задании)

Поиск подстроки. Алгоритм КМР (Knuth–Morris–Pratt)

Вычисляем префикс-функцию для паттерна (структура повторяемости строки)

Поиск подстроки.
Алгоритм КМР (Knuth–Morris–Pratt)

Вычисляем префикс-функцию для паттерна (структура повторяемости строки)

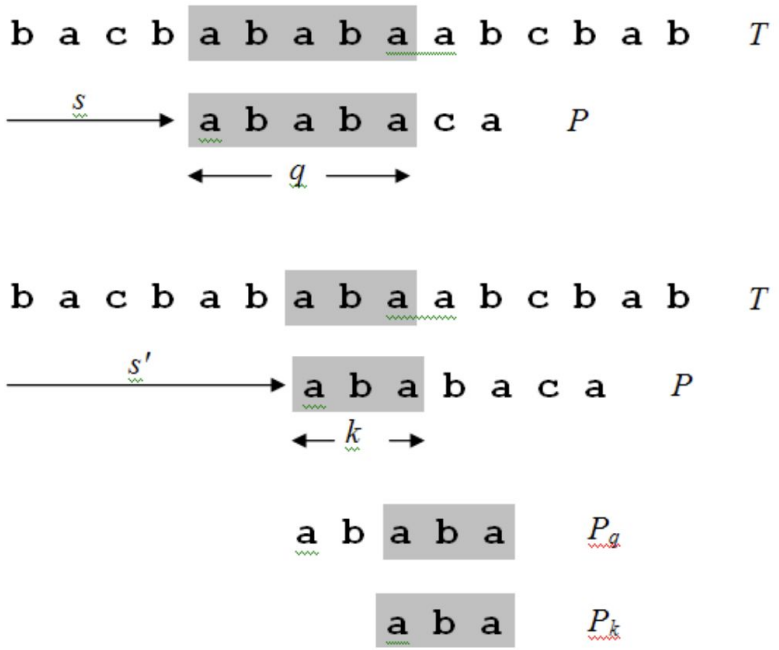
Рисуем на доске.

Поиск подстроки. Алгоритм КМР (Knuth–Morris–Pratt)

Определение. Префикс-функцией, ассоциированной со строкой $P[1..m]$, называется функция π: $\{1,2, \ldots, m\} \to \{0,1, \ldots, m-1\}$, определенная следующим образом:

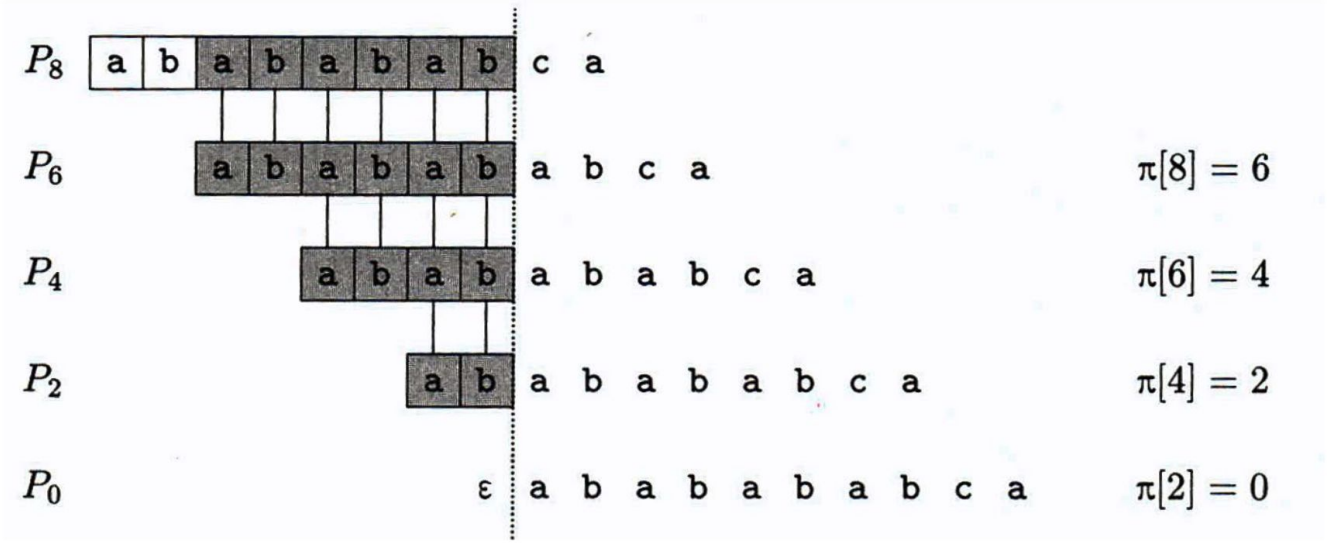
$$\pi[q] = \max\{k: k<q \ \& \ P_k \succ P_q\}$$

Иными словами, $\pi[q]$ – длина наибольшего префикса P, являющегося суффиксом P_q.



Поиск подстроки. Алгоритм KMP (Knuth–Morris–Pratt)

$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1



$$\pi^*[8] = \{8,6,4,2,0\}$$

Метрики схожести

Расстояние Левенштейна

Минимальное число односимвольных преобразований (вставка, удаление, замена), необходимых, чтобы превратить одну последовательность в другую

Метрики схожести

Расстояние Левенштейна. Алгоритм Вагнера-Фишера

Составим матрицу D и каждый элемент вычислим по следующей формуле

Метрики схожести

$$D(i,j) = \begin{cases} 0, & i=0,\ j=0 \\ i, & j=0,\ i>0 \\ j, & i=0,\ j>0 \\ \min\{ & \\ \quad D(i,j-1)+1, & \\ \quad D(i-1,j)+1, & j>0,\ i>0 \\ \quad D(i-1,j-1)+\mathrm{m}(S_1[i],S_2[j]) & \\ \} & \end{cases},$$

где $\mathrm{m}(a,b)$ равна нулю, если $a=b$ и единице в противном случае; $\min\{\,a,b,c\,\}$ возвращает наименьший из аргументов.

Метрики схожести

		Л	А	Б	Р	А	Д	О	Р
	0	1	2	3	4	5	6	7	8
Г	1								
И	2								
Б	3								
Р	4								
А	5								
Л	6								
Т	7								
А	8								
Р	9								

Метрики схожести

		Л	А	Б	Р	А	Д	О	Р
	0	1	2	3	4	5	6	7	8
Г	1	1							
И	2								
Б	3								
Р	4								
А	5								
Л	6								
Т	7								
А	8								
Р	9								

Метрики схожести

		Л	А	Б	Р	А	Д	О	Р
	0	1	2	3	4	5	6	7	8
Г	1	1	2						
И	2								
Б	3								
Р	4								
А	5								
Л	6								
Т	7								
А	8								
Р	9								

Метрики схожести

		Л	А	Б	Р	А	Д	О	Р
	0	1	2	3	4	5	6	7	8
Г	1	1	2	3	4	5	6	7	8
И	2	2	2	3	4	5	6	7	8
Б	3	3	3						
Р	4								
А	5								
Л	6								
Т	7								
А	8								
Р	9								

Метрики схожести

		Л	А	Б	Р	А	Д	О	Р
	0	1	2	3	4	5	6	7	8
Г	1	1	2	3	4	5	6	7	8
И	2	2	2	3	4	5	6	7	8
Б	3	3	3	2					
Р	4								
А	5								
Л	6								
Т	7								
А	8								
Р	9								

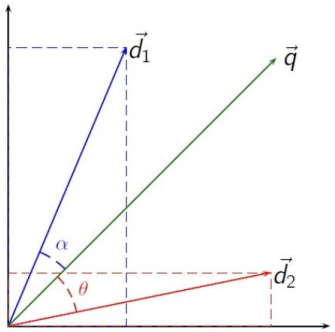
Метрики схожести

		Л	А	Б	Р	А	Д	О	Р
	0	1	2	3	4	5	6	7	8
Г	1	1	2	3	4	5	6	7	8
И	2	2	2	3	4	5	6	7	8
Б	3	3	3	2	3	4	5	6	7
Р	4	4	4	3	2	3	4	5	6
А	5	5	4	4	3	2	3	4	5
Л	6	5	5	5	4	3	3	4	5
Т	7	6	6	6	5	4	4	4	5
А	8	7	6	7	6	5	5	5	5
Р	9	8	7	7	7	6	6	6	5

Метрики схожести

Косинусное расстояние

Candidate documents from the corpus can be retrieved and ranked using a variety of methods. Relevance rankings of documents in a keyword search can be calculated, using the assumptions of document similarities theory, by comparing the deviation of angles between each document vector and the original query vector where the query is represented as a vector with same dimension as the vectors that represent the other documents.



In practice, it is easier to calculate the cosine of the angle between the vectors, instead of the angle itself:

$$\cos\theta = \frac{\mathbf{d_2} \cdot \mathbf{q}}{\|\mathbf{d_2}\| \, \|\mathbf{q}\|}$$

Where $\mathbf{d_2} \cdot \mathbf{q}$ is the intersection (i.e. the dot product) of the document (d_2 in the figure to the right) and the query (q in the figure) vectors, $\|\mathbf{d_2}\|$ is the norm of vector d_2, and $\|\mathbf{q}\|$ is the norm of vector q. The norm of a vector is calculated as such:

$$\|\mathbf{q}\| = \sqrt{\sum_{i=1}^{n} q_i^2}$$

Using the cosine the similarity between document d_j and query q can be calculated as:

$$\cos(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} d_{i,j} q_i}{\sqrt{\sum_{i=1}^{N} d_{i,j}^2} \sqrt{\sum_{i=1}^{N} q_i^2}}$$

As all vectors under consideration by this model are element-wise nonnegative, a cosine value of zero means that the query and document vector are orthogonal and have no match (i.e. the query term does not exist in the document being considered). See cosine similarity for further information.[2]

Метрики схожести

Косинусное расстояние

Примеры в коде

Наибольшая общая подпоследовательность (LCS)

Даны две строки
Найти самую длинную последовательность символов, которая встречается в заданном порядке в обеих строках. Символы не обязательно должны идти подряд

```
string_1 = "AGGTAB"
string_2 = "GXTXAYB"

LCS = "GTAB"
```

Наибольшая общая подпоследовательность (LCS)

Составим матрицу и каждый элемент вычислим по формуле

Наибольшая общая подпоследовательность (LCS)

$$LCS(i,j) = \max \begin{cases} LCS(i-1,j) \\ LCS(i,j-1) \\ LCS(i-1,j-1)+1 \quad \text{if } A[i] = B[j] \end{cases}$$

Базовый случай для этого рекуррентного соотношения — $i = 0$ или $j = 0$:

$$LCS(0,j) = LCS(i,0) = 0\,.$$

Наибольшая общая подпоследовательность (LCS)

Наибольшая общая подпоследовательность (LCS)

Поиск по документам

Структуры данных для ускорения поиска релевантных документов

Прямой индекс

Пример:

Document ID	Tokens
1	["поиск", "текст", "информация"]
2	["данные", "обработка", "анализ"]

Прямой индекс

Сложность поиска:
Для поиска всех документов, содержащих слово, необходимо:

- Перебрать все документы: $O(N)$
  , где N
  – количество документов.
- Проверить список слов в каждом документе: $O(L)$
  где L
  – среднее количество слов в документе.

Затраты по памяти:
Прямой индекс хранит все слова каждого документа, поэтому затраты на память растут линейно с количеством документов и длиной текстов.

Обратный индекс

Пример:

Token	Document IDs
"поиск"	[1, 2]
"текст"	[1]
"данные"	[2]
"анализ"	[2]

Обратный индекс

Сложность поиска:
Для поиска всех документов, содержащих слово, необходимо:

- Найти слово в индексе: $O(1)$
  (хэш-таблица).

Затраты по памяти:

- Каждое слово связано со списком документов, где оно встречается.
- Затраты на память зависят от количества уникальных слов и плотности их распределения по документам.