Рекомендательная система курсов на графах

Ахтямов Алексей (mralexeimk@yandex.ru)

October 21, 2022

Введение

В текущем документе описывается построение эффективной конфигурации базы данных PostgreSQL с таблицами курсов и тэгов, создание рекомендательной системы курсов, основанной на графах и извлечение тэгов из пользовательского ввода (в поле поиска курсов) с помощью расстояния Левенштейна.
Реализация на Python представлена тут GitHub.

Структура базы данных

База данных будет содержать 2 таблицы (**'coursers'** и **'tags'**)

Coursers:
Тип: *динамически обновляемая*
Пример создания **'coursers'** таблицы с помощью SQL запроса (нам важны только столбцы **id**, **title** (название курса) и **tags** (тэги)):

```
CREATE TABLE coursers (
    id SERIAL PRIMARY KEY,
    title VARCHAR (255) NOT NULL,
    author VARCHAR (255) NOT NULL,
    views INTEGER NOT NULL,
    likes INTEGER NOT NULL,
    tags TEXT NOT NULL)
```

Таблица будет содержать список добавленных курсов.
Столбец **tags** содержит набор строк (тэгов), разделенных специальным символом (в нашей реализации это символ **'@'**).

Tags:
Тип: *периодически обновляемая*
Заполнение происходит периодически из таблицы **'coursers'**
Создание **'tags'** таблицы с помощью SQL запроса:

```
CREATE TABLE tags (
    id SERIAL PRIMARY KEY,
    tag VARCHAR(255) UNIQUE NOT NULL,
    coursers_count INTEGER NOT NULL,
    related_tags TEXT)
```

Таблица будет хранить список всех тэгов.
Столбец **coursers_count** хранит число курсов, использующих тэг.
Столбец **related_tags** хранит индексы тэгов, связанных с текущим.
Для ускорения обновления таблицы и ограничения числа смежных вершин в графе введем ограничение на количество тэгов, связанных с текущим. Обозначим это ограничение сверху как $\mathbf{RT_{max}}$. При периодическом обновлении таблицы в столбец **related_tags** войдут индексы тех тэгов, **coursers_count** которых в таблице наибольший.

Построение графа

Визуальный пример графа с тэгами и курсами (синие - тэги, зеленые - курсы).
Обозначим число курсов как $\mathbf{C}$, а число тэгов как $\mathbf{T}$.
Граф - множество вершин, соединенных между собой каким-то образом.
Ориентированный граф - граф, с ориентированным соединением между вершинами.

Вершинами нашего графа будут все тэги и все курсы ($\mathbf{C} + \mathbf{T}$ вершин).
Каждая вершина имеет максимум $\mathbf{RT_{max}}$ выходящих из нее ребер, ведущих

к вершинам-тэгам и максимум $\mathbf{C}$ выходящих ребер, ведущих к вершинам-курсам.
В худшем случае, **когда каждый тэг есть в каждом курсе**, используемая для хранения графа память равна $O('\text{число вершин}' + '\text{ число связей}') = O((\mathbf{T} + \mathbf{C}) + (\mathbf{T} + \mathbf{C}) * (\mathbf{RT_{max}} + \mathbf{C}))$

Перебирая значения двух переменных при $\mathbf{RT_{max}} = \mathbf{5}$, получаем:

	coursersCount	tagsCount	Memory
0	1.000e+03	2.000e+04	2.113e+07
1	2.000e+03	4.000e+04	8.425e+07
2	5.000e+03	1.000e+05	5.256e+08
3	1.000e+04	2.000e+05	2.101e+09
4	5.000e+04	1.000e+06	5.251e+10
5	2.000e+05	4.000e+06	8.400e+11
6	5.000e+05	1.000e+07	5.250e+12
7	1.000e+06	2.000e+07	2.100e+13

В худшем случае, при отсутствии модерации, оптимально хранить $\sim$ 10.000 курсов и $\sim$ 200.000 тэгов с затратами меньше 1 ГБ.

Предположим, что модерация есть, тогда, пусть $\mathbf{M}$ - среднее число курсов, содержащих определенный тэг.
В среднем случае, требуемая память составляет $O('\text{число вершин}' + '\text{ число связей}') = O((\mathbf{T} + \mathbf{C}) + (\mathbf{T} + \mathbf{C}) * (\mathbf{RT_{max}} + \mathbf{M}))$
Перебирая значения трех переменных при $\mathbf{RT_{max}} = \mathbf{5}$, получаем:

	coursersCount	tagsCount	averageCoursersOnTag	Memory
0	1.000e+03	2.000e+04	20	5.460e+05
1	2.000e+03	6.000e+04	30	2.232e+06
2	5.000e+03	2.000e+05	40	9.430e+06
3	1.000e+04	5.000e+05	50	2.856e+07
4	5.000e+04	3.000e+06	60	2.013e+08
5	2.000e+05	1.400e+07	70	1.079e+09
6	5.000e+05	4.000e+07	80	3.483e+09
7	1.000e+06	9.000e+07	90	8.736e+09

Для хранения графа с миллионом курсов и 90-та миллионами тэгов требуемая память составляет $\sim 9 \cdot 10^9$ бит (4 ГБ).

Инициализация графа происходит периодически после обновления таблицы **'tags'**.

Обработка пользовательского ввода

Для поиска тэгов в пользовательской строке разобьем строку на слова (разделенные пробелом) переберем все *размещения* из количества слов по i, $\forall i \leq N$ (**N** - максимальное число слов в тэге, в нашей реализации оно равно 3).
В каждом размещении объеденим слова в строку и с помощью бинарного поиска найдем окрестность похожих на нее тэгов. Выберим из них тот, расстояние по Левенштейну до которого минимально и, если это расстояние меньше какой-то верхней границы - берем этот тэг для дальнейшей рекомендации по нему.
Расстояние Левенштейна - минимальное число операций (удаление, вставка и замена символа), необходимых для превращения одной строки в другую. Чем оно меньше - тем более похожи строки. В нашей реализации, для избежания эквивалентности коротких тэгов, операция замены стоит в 2 раза больше других операций.

После обработки мы получим список тэгов, по которым мы будем рекомендовать пользователю курсы.

Рекомендация курсов

После получения списка тэгов, добавим в этот список семантически похожие тэги.
В нашем графе, если вершина-тэг A идет в вершину-тэг B означает, что B - более высокая абстракция над A или, по-другому, A - частный случай B.
Это обеспечивается за счет ограничения $\mathbf{RT_{max}}$, которое заставляет граф строить связи с теми вершинами-тэгами, количество курсов с которыми наибольшее.
Данный факт хорошо прослеживается в данном выше визуальном примере, где такие общие тэги, как 'программирование', 'для новичков' имеют сравнительно большее число вершин, входящих в них.
Для добавления новых тэгов, переберем все смежные вершины-тэги в графе у списка тэгов и добавим в список. Проделаем эту операцию столько раз, какую глубину мы указали в конфигурации.
В итоге, мы имеем список тэгов и информацию о том, на какой глубине получен этот тэг. Переберем все эти тэги в графе и найдем все смежные вершины-курсы и добавим в нашу рекомендацию. В зависимости от глубины, ключевых слов в названии курса или рейтинга сформируем и отсортируем список рекомендуемых курсов.

Полезные ссылки

1. Python - psycopg2 библиотека (работа с базой данных)
2. Python - pyvis библиотека (работа с графами)
3. Расстояние Левенштейна