پروژه درس ساختمان داده

Search Engine

فاز اول

طراحان:

صادق جعفری

محمدرضا تاجیک

عرفان همتی

زینب باقیان

فهرست

قوانین:

- لازم است هر دو عضو گروه در در انجام پروژه مشارکت فعال داشته باشند و از کار های انجام شده توسط هم تیمی خود آگاهی کامل داشته باشند.
- در github یک repository بسازید و هر دو عضو تیم در توسعه آن کوشا و فعال باشند.
- ارزیابی شما بر اساس عملکرد برنامه شما و نیز تسلط شما بر آن انجام می شود.
- در انتخاب زبان برنامه نویسی برای انجام پروژه آزاد هستید ولی با توجه به چالش های پروژه توصیه می شود از **پایتون** استفاده کنید.
- در صورت وجود ابهام در هر بخش از پروژه تنها از تی ای های مربوطه کمک بگیرید.
- تیم تی ای مسئولیتی نسبت به بررسی کد شما و یا اصلاح آن ندارد بلکه تنها در بر طرف سازی ابهامات احتمالی صورت سوال مسئولیت خواهد داشت.

موضوع پروژه:

هدف این پروژه آشنایی شما با search engine و تجربه ی search کردن در متن است.

زمانی که شما در گوگل جست و جویی انجام میدهید؛ موتور جست و جو گوگل پس از انجام فرایندهایی به نتایجی می رسند که حاصلشان تعدادی آدرس و محتوا می باشد که برای شما نمایش داده می شود. این سایت ها و لینک ها به احتمال زیاد دربردارنده پاسخ سوالات شما هستند چرا که بیشترین ارتباط را با سوال شما و پاسخ احتمالی آن دارند. همچنین نمایش لیست سایت ها و لینک ها بر اساس معیار های مختلفی می باشد که ترتیبی خاص را رقم می زند. بر این اساس دستیابی به پاسخ موردنظرتان راحت تر و سریع تر خواهد بود.

در این پروژه بنا داریم نسخه بسیار ساده تری از عملیات های جست و جو گوگل را پیاده سازی کنیم. برای فهم بهتر موضوع این پروژه در سه فاز انجام خواهد گرفت.

ورودی و خروجی:

به عنوان ورودی به شما یک کوئری و تعدادی داکیومنت داده میشود. از شما خواسته می شود مرتبط ترین داکیومنت ها نسبت به کوئری را پیدا کنید و با ترتیب مناسب نمایش دهید. علاوه بر این برای هر داکیومنت یافت شده رفرنس دقیق آن را نیز مشخص کنید. (جمله ای که باعث انتخاب شدن داکیومنت شده را مشخص کنید)

بخش صفر:

در طول پروژه به استفاده از تابع نیاز خواهید داشت. بهتر است آن را در ابتدا پیاده سازی کنید:

- ✓ تابع Tokenizer

برای پیدا کردن کلمات در جمله ها و ذخیره سازی آن ها پس از استخراج از فایل json تابع tokenizer ای نیاز خواهید داشت تا ذخیره سازی با فرمت درست انجام شود.

```
def tokenizer(text):
    text = text.lower()
    return text.split()
```

پیاده سازی این تابع به صورت دلخواه است و این کد صرفا پیشنهادی است.

بخش اول:

در این بخش داکیومنت های گرفته شده را به ساختار داده tf-idf تبدیل کنید.

داکیومنت هایی که به شما داده می شود با فرمت json هستند و شما میتوانید این فرمت را به فرمت مورد نیاز خود تبدیل کنید.

منظور از TF-IDF یا term frequency-inverse document frequency یک آمار عددی است که برای نشان دادن اهمیت یک کلمه در یک سندی که در مجموعه ای واقع شده است استفاده می شود. TF تعداد دفعات ظاهر شدن یک اصطلاح در یک سند را اندازه گیری می کند، در حالی که IDF نادر بودن یک اصطلاح را در کل مجموعه اندازه گیری می کند. با ضرب این مقادیر در یکدیگر، TF-IDF به هر عبارت نمره ای اختصاص میدهد که نشان دهنده اهمیت آن در سند است. TF-IDF معمولاً در برنامه های کاربردی مختلف NLP، مانند موتورهای جستجو برای رتبه بندی ارتباط اسناد، طبقه بندی متن، بازیابی اطلاعات و استخراج کلمات کلیدی استفاده می شود. این به شناسایی مرتبط ترین و مهم ترین اصطلاحات در یک سند یا مجموعه کمک می کند.

بنابر این برای محاسبه TF-IDF شما باید توابعی را برای محاسبه TF و IDF داشته باشید و در نهایت با استفاده از آن ها TF-IDF را محاسبه کنید.

مقادیر TF و IDF را با مراحل زیر محاسبه کنید:

1. در ابتدا کلمات منحصر به فرد در تمام داکیومنت ها را پیدا کرده و کلمات را با ایندکس ها مپ میکنیم.
2. برای هر جمله یک وکتور به طول تعداد کلمات منحصر به فرد می سازیم و TF-IDF کلمات آن جمله را با استفاده از مپ انجام شده در این وکتور قرار میدهیم.
3. با ذخیره TF-IDF کلمات هر جمله در یک Vector میتوانیم TF-IDF یک جمله را پیدا کنیم.

پس از انجام عملیات های بالا ساخت ساختار داده TF-IDF برای جملات به پایان می رسد.

بخش دوم:

برای هر یک از داکیومنت ها یک وکتور به طول تعداد کلمات منحصر به فرد می سازیم. این وکتور باید با استفاده از جمع وکتور جملات آن داکیومنت ساخته شود.

پس از انجام عملیات های بالا ساخت ساختار داده TF-IDF داکیومنت ها به پایان می رسد.

بخش سوم:

در این بخش پس از تکمیل ساختار داده TF-IDF به دنبال پیدا کردن شباهت میان کوئری و محتوا آن داکیومنت ها خواهیم بود.

برای این بخش تابعی پیاده سازی کنید که به ازای هر جمله ورودی به دنبال پیدا کردن شباهت میان آن وکتور آن جمله(سایز وکتور به اندازه تعداد کلمات منحصر به فرد خواهد بود.) و سایر جملات داکیومنت باشد تا مشخص شود این جمله از کدام یک از جملات داکیومنت می تواند استخراج شده باشد.در ابتدا می توانید به دنبال شباهت میان وکتور کوئری و وکتور داکیومنت های مشخص شده باشید تا بتوانید آن هارا بر اساس میزان شباهت مرتب کنید و پس از آن برای پیدا کردن اینکه این شباهت ناشی از کدام جمله داکیومنت است، شباهت میان وکتور جملات و وکتور کوئری را محاسبه کنید.

مقایسه دو وکتور با استفاده از cosine similarity انجام میشود. رابطه آن مطابق زیر است:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

کلمات کوئری لزوما با کلمات ذخیره شده یکسان نیستند بلکه ممکن است تفاوت هایی داشته باشند برای حل کردن این مسئله باید با استفاده از تابع sequencematcher که در کتابخانه difflib پایتون وجود دارد مقایسه را برای کلمات مشابه نیز انجام داد.
پس از به دست آوردن جملاتی با کلمات مشابه شما باید جملاتی که بیشترین شباهت را با جمله ی مورد نظر ما دارند پیدا کرده و برگردانید.

بخش چهارم:

در این بخش پرتکرار ترین کلمات هر داکیومنت را پیدا و ذخیره کنید. این بخش و بخش بعد اساس محاسبات و عملیات هایی است که در فاز های بعد انجام خواهید داد. از جمله پیش بینی موضوع داکیومنت ها و استفاده در خوشه بندی داکیومنت ها در فاز بعدی.

تذکر: برای انجام محاسبه مربوط به این بخش حتما از ساختار TF-IDF ساخته شده برای داکیومنت ها استفاده کنید.

بخش پنجم:

در این بخش پنج کلمه مهم هر داکیومنت را پیدا کنید.

توجه کنید که کلمه مهم با کلمه پرتکرار متفاوت است. یک کلمه می تواند پرتکرار باشد ولی مهم نباشد. از طرفی ممکن است یک کلمه مهم باشد ولی پرتکرار نباشد.

معروف ترین مثال برای این بخش کلمه *the* می باشد که یکی از پرتکرارترین کلمات و همچنین از کم اهمیت ترین کلمات است.

تذکر: برای انجام محاسبه مربوط به این بخش حتما از ساختار TF-IDF ساخته شده برای داکیومنت ها استفاده کنید.