 **بازیابی پیشرفته اطلاعات**

مدرس: دکتر بیگی

شماره گروه: 5

دانشکده مهندسی کامپیوتر تهیه‌کنندگان: نیما جمالی – سپهر فعلی – سینا کاظمی

گزارش فاز اول پروژه

فهرست مطالب

[1- پیش‌پردازش اولیه 3](#_Toc56047330)

[پیش‌پردازش مستندات انگلیسی 3](#_Toc56047331)

[کلمات پرتکرار مستندات انگلیسی 3](#_Toc56047332)

[پیش‌پردازش مستندات فارسی 4](#_Toc56047333)

[کلمات پرتکرار مستندات فارسی 5](#_Toc56047334)

[2- نمایه‌سازی 6](#_Toc56047335)

[نمایه bigram 6](#_Toc56047336)

[نمایه positional 6](#_Toc56047337)

[توابع درج و حذف مستندات 6](#_Toc56047338)

[توابع ذخیره‌سازی و لود نمایه‌ها 7](#_Toc56047339)

[3- فشرده‌سازی نمایه‌ها 8](#_Toc56047340)

[فشرده‌سازی با variable byte 8](#_Toc56047341)

[فشرده‌سازی با gamma code 8](#_Toc56047342)

[ذخیره‌سازی و لود نمایه‌ها 8](#_Toc56047343)

[4- اصلاح پرسمان 8](#_Toc56047344)

[استفاده از نمایه bigram و معیار جاکارد برای یافتن کلمات مشابه 8](#_Toc56047345)

[تابع محاسبه‌ی فاصله ویرایش دو کلمه 8](#_Toc56047346)

[نمایش پرسمان اصلاح‌شده 8](#_Toc56047347)

[5- جستجو و بازیابی اسناد 9](#_Toc56047348)

[جستجوی ltc-lnc پرسمان 9](#_Toc56047349)

[جستجوی proximity با اندازه‌ی پنجره‌ی داده شده در ورودی 9](#_Toc56047350)

[6- نحوه‌ی تقسیم وظایف 9](#_Toc56047351)

[نیما جمالی 9](#_Toc56047352)

[سپهر فعلی 9](#_Toc56047353)

[سینا کاظمی 10](#_Toc56047354)

[7- مراجع 10](#_Toc56047355)

# پیش‌پردازش اولیه

در این قسمت در کلاس IRSystem تابعی به نام prepare\_text ایجاد کردیم که مستندات، زبان مد نظر و ایست‌واژه‌ها را ورودی می‌گیرد. این تابع هنگامی که کاربر در ورودی دستور prepare [lang] را فراخوانی می‌کند، صدا زده می‌شود. همچنین برای بررسی کوئری نیز از همین تابع استفاده می‌شود. در واقع اگر آرایه ایست‌واژه خالی باشد به معنای بررسی اولیه مستندات است و در صورتی که خالی نباشد، بدین معناست که یک کوئری در حال پردازش است. در هنگام ایجاد یک نمونه از کلاس، دو تابع initialize\_english و initialize\_persian صدا زده می‌شوند و متون موجود در فایل‌های csv و xml داده شده در یک مستند به فرمت زیر ذخیره می‌شوند:

document = [[title], [description]]

و برای هر زبان یک مجموعه مستند(collections) به عنوان ورودی تابع prepare\_text پاس داده می‌شود.

تابع prepare\_text بسته به زبان مورد نظر، یک سری اعمال نظیر نرمال‌سازی متنی، جداسازی، حذف حروف اضافه، یافتن ایست‌واژه‌ها و بازگرداندن کلمات به ریشه را انجام می‌دهد و در نهایت لیستی از تمامی توکن‌های باقی‌مانده، تمامی مستندات ساختار یافته، ترم‌های باقی‌مانده و ایست‌واژه‌ها را باز می‌گرداند. مستندات ساختار یافته(structured documents) آرایه‌ای از مستندات به صورت زیر است.

document = [[array of title tokens], [array of description tokens]]

در ادامه به طور مجزا به بررسی اعمال انجام گرفته برای پیش‌پردازش داده‌های هر زبان می‌پردازیم.

## پیش‌پردازش مستندات انگلیسی

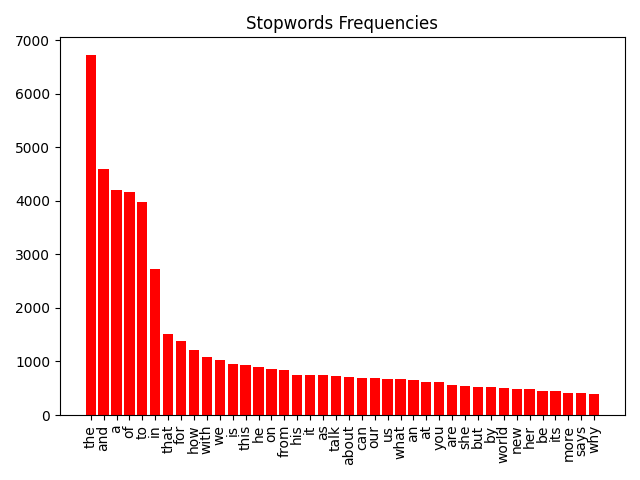
تابع prepare\_english برای پیاده‌سازی این قسمت طراحی شده‌است. ابتدا هر قسمت از مستند را به صورت جداگانه و با استفاده از تابع word\_tokenize از کتابخانه‌ی NLTK، به صورت مجموعه‌ای از توکن‌ها در می‌آوریم و تمام کلمات را نیز به حالت حروف کوچک تبدیل می‌کنیم. سپس علائمی نظیر ‘-‘، ‘,’ و ‘?’ را از کلماتی که در آنها موجودند، حذف می‌کنیم. پس از آن نیز حروف اضافه را از توکن‌ها حذف می‌کنیم. پس از یافتن ایست‌واژه‌ها ([اینجا](#_کلمات_پرتکرار_مستندات))، با استفاده از SnowballStemmer کلمات را به ریشه بر می‌گردانیم و در نهایت لیستی از تمامی توکن‌ها، مستندات ساختار یافته، تمامی‌ترم‌های باقی‌مانده و ایست‌واژه‌های زبان انگلیسی را باز می‌گردانیم.

## کلمات پرتکرار مستندات انگلیسی

در این قسمت از تابع process\_stop\_words استفاده کردیم که سایز مورد نظر برای ایست‌واژه‌ها و آرایه‌ای از تمامی توکن‌ها ورودی می‌گیرد و با استفاده از توابع کلاس Counter، توکن‌ها را بر اساس تکرار آنها و به صورت نزولی مرتب می‌کند. در نهایت هم ترم‌های مجموعه مستندات و ایست‌واژه‌ها را باز می‌گرداند. در بخش انگلیسی با امتحان کردن مقادیر مختلف به عنوان سایز مجموعه‌ی ایست‌واژه، به عدد 40 رسیدیم که تعداد توکن‌ها را از 144828 به 93694 عدد کاهش می‌داد. این آرایه به ترتیب فراوانی متشکل از کلمات زیر بود:

['the', 'and', 'a', 'of', 'to', 'in', 'that', 'for', 'how', 'with', 'we', 'is', 'this', 'he', 'on', 'from', 'his', 'it', 'as', 'talk', 'about', 'can', 'our', 'us', 'what', 'an', 'at', 'you', 'are', 'she', 'but', 'by', 'world', 'new', 'her', 'be', 'its', 'more', 'says', 'why']

نمودار فراوانی این ایست‌واژه‌ها نیز در تصویر زیر قابل مشاهده‌است.



تصویر 1: نمودار فراوانی ایست‌واژه‌های مستندات انگلیسی

## پیش‌پردازش مستندات فارسی

تابع prepare\_persian برای پیاده‌سازی این قسمت طراحی شده است، ابتدا یک لیست از حروف اضافه و علائم نگارشی و عملگرها و اعداد را انتخاب می کنیم و هر قسمت از مستند را به طور جداگانه پردازش می‌کنیم، بدین صورت که در هر بخش به‌جای اعضای لیست یک " " ( کاراکتر space ) قرار می‌دهیم. همچنین هر رشته‌ای از کلمات را که بین دو آکلاد ({ }) وجود دارد را نیز حذف می‌کنیم. سپس با استفاده از کتابخانه‌ی هضم (hazm) کلمات را در هر بخش normalize و tokenize می‌کنیم ، پس از آن در هر کلمه‌ای که نیم‌فاصله در آن وجود داشت، نیم‌فاصله (و هر کاراکتر اضافی دیگر) را حذف می‌کنیم و دو قسمت کلمه را به هم می‌چسبانیم:

می‌شود میشود

در نهایت با استفاده از stemmer که در کتابخانه هضم موجود بود کلماتی که انتهای آنها " ان - ات - ترین - تر - یی - ها - ا و … " بودند را کوتاه می‌کنیم:

کتابهایشان -> کتاب

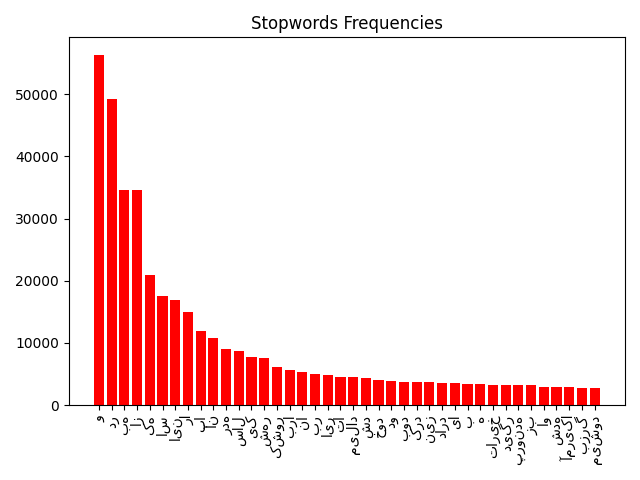
در نهایت هم پس از [پیدا کردن ایست‌واژه‌ها](#_کلمات_پرتکرار_مستندات_1)، خروجی شامل تمامی توکن‌ها، مستندات ساختار یافته، لیستی از ترم‌ها و ایست‌واژه‌ها بازگردانده می‌شود.

## کلمات پرتکرار مستندات فارسی

برای این بخش نیز از تابعی که برای پیدا کردن ایست‌واژه‌ها در متون انگلیسی استفاده شد، بهره گرفتیم و با امتحان کردن مقادیر مختلف، به عدد 40 برای اندازه‌ی ایست‌واژه‌ها رسیدیم. این عمل باعث شد تا اندازه‌ی مجموعه توکن‌ها از 1333760 به 933141 کاهش پیدا کند. این آرایه به ترتیب فراوانی از کلمات زیر تشکیل شده‌بود:

['و', 'در', 'به', 'از', 'که', 'اس', 'این', 'را', 'با', 'آن', 'رده', 'سال', 'یک', 'شهر', 'کشور', 'برا', 'نا', 'بر', 'ایر', 'تا', 'میلاد', 'شد', 'خود', 'دو', 'بود', 'کرد', 'نیز', 'دارد', 'یا', 'ب', 'ه', 'تاریخ', 'دیگر', 'پرونده', 'زب', 'او', 'شده', 'آمریکا', 'بزرگ', 'میشود']

و نمودار فراوانی این واژه‌ها نیز در تصویر زیر قابل رویت است.



تصویر 2: نمودار فراوانی ایست‌واژه‌های مستندات انگلیسی

# نمایه‌سازی

توضیحات

## نمایه bigram

متن

## نمایه positional

متن

## توابع درج و حذف مستندات

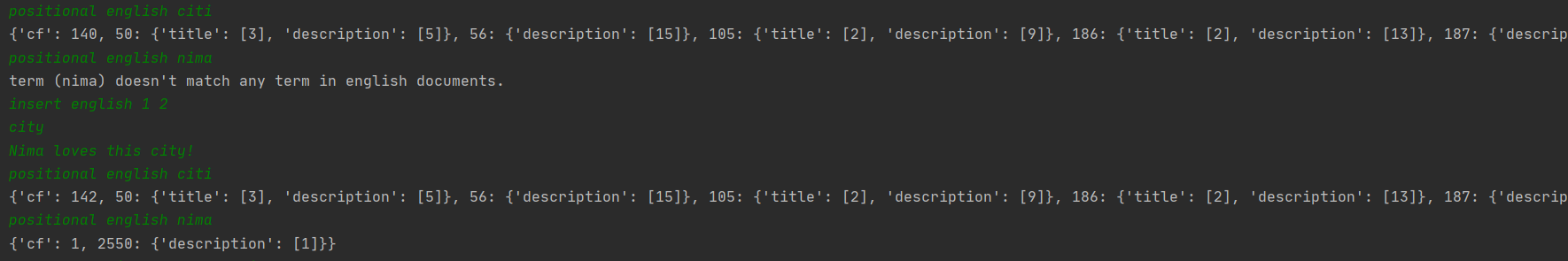
توابع این قسمت شامل توابع insert، csv\_insert، xml\_insert و delete هستند. در مجموعه مستندات فرض کردیم شناسه‌ی هر مستند برابر با شماره‌ی آن در مجموعه مستندات است و هر شماره‌ی دیگری که درج می‌شود به این اعداد اضافه می‌شود. به این منظور برای هر زبان عددی به نام docs\_size در نظر گرفته شده‌است. تابع insert مجموعه مستنداتی را ورودی می‌گیرد و با توجه به زبان ورودی، نمایه‌های bigram و positional مربوطه را نیز ورودی می‌گیرد و پس از اعمال پیش‌پردازش، تمامی توکن‌های این مستندات را به توکن‌های کلی اضافه می‌کند، مستندات را به مجموعه‌ی قبلی اضافه می‌کند و ترم‌های جدید را به دیکشنری اضافه می‌کند و سپس نمایه‌های positional و bigram را با فراخوانی توابع مربوطه به‌روز رسانی می‌کند. کاربر می‌تواند با دستور insert [lang] [docs\_number] [parts] به تعداد docs\_number داکیومنت را به صورت دستی در مجموعه مستندات درج کند.

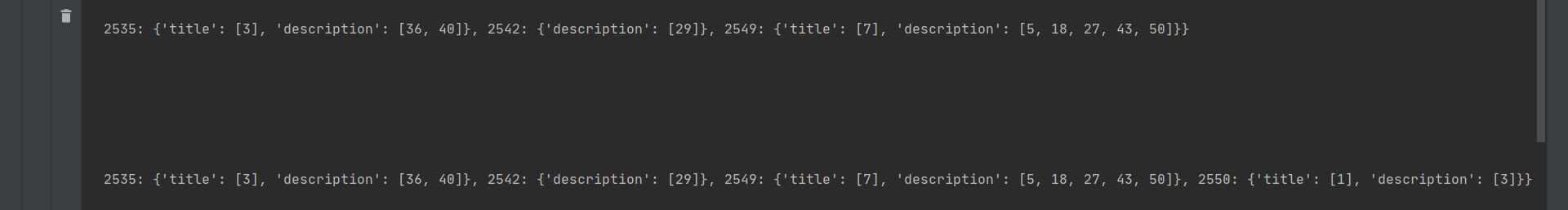
توابع csv\_insert و xml\_insert نیز زبان ورودی و path مورد نظر را دریافت می‌کنند و در صورت صحت path، پردازش‌های مربوط به csv یا xml را انجام می‌دهند. در نهایت هم تابع insert بر روی مستندات فایل ورودی اعمال می‌شود.

تابع delete اما یک شماره‌ی مستند را ورودی می‌گیرد و با توجه به زبان ورودی، اگر مستند مورد نظر در مجموعه موجود باشد، نمایه‌های bigram و positional را نیز به‌روز رسانی می‌کند. در واقع برای مجموعه مستندات بردار deleted\_documents از True یا False داریم که هنگام پاک شدن یک مستند، اندیس مربوط به آن در این بردار True می‌شود. به‌روز رسانی نمایه‌های مذکور نیز به این صورت است که به ازای هر توکن در مستند پاک شده، در نمایه positional مربوط به این ترم دیکشنری، cf یک واحد کم می‌شود و شناسه‌ی مستند هم از posting list آن کلمه پاک می‌شود. اگر cf برابر صفر شود، این ترم از نمایه positional حذف می‌گردد و در نمایه‌ی bigram نیز این ترم از posting list همه‌ی دوحرفی‌هایی که این ترم در آنها موجود بوده، پاک می‌شود. دستوری که کاربر در کنسول وارد می‌کند به فرمت زیر است:

delete [lang] [doc\_id]

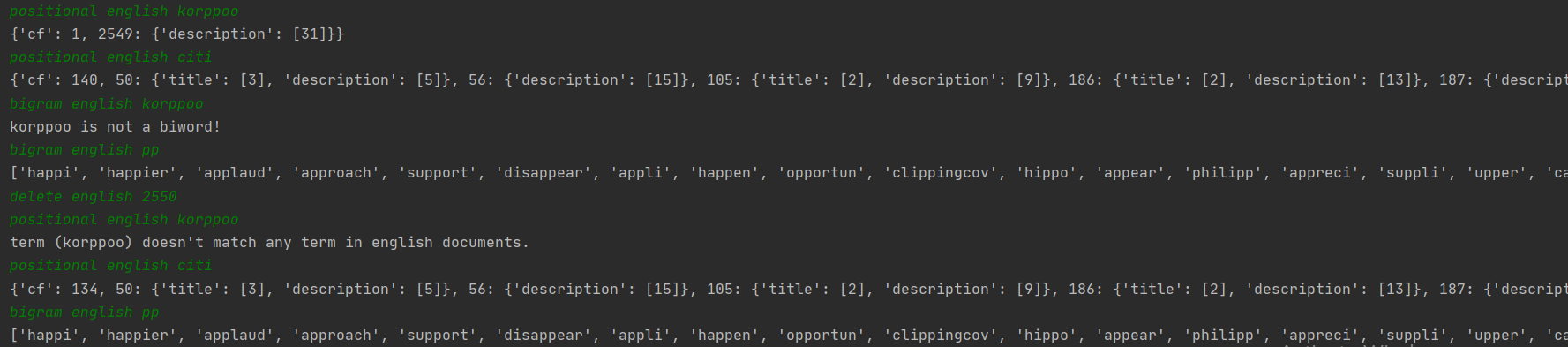
در ادامه مثال‌هایی از عملکرد توابع insert و delete مشاهده می‌کنیم. این مثال‌ها مربوط به دو اجرای مجزا هستند.





تصویر 3: مثالی از عملکرد تابع insert

در مثال بالا یک مستند با موضوع city و بدنه‌ی Nima loves this city! درج شده‌است. قبل از اعمال این درج ترم citi که stem شده‌ی کلمه‌ی city است، 140 بار در مستندات استفاده شده و ترمی به نام nima نیز وجود نداشته‌است. اما پس از درج این مستند city 142 بار در مجموعه مستندات حضور دارد و ترم nima نیز به نمایه‌ها اضافه شده‌است.



تصویر 4: مثالی از عملکرد تابع delete

در مثال بالا نیز مستند آخر یعنی 2550 پاک‌ شده‌است. ترم korppoo تنها در این مستند موجود بوده که بعد از پاک شدن چون دیگر در مستندی وجود ندارد، از نمایه‌های bigram و positional پاک شده‌است. همچنین ترم citi 6 بار در این مستند به کار رفته که با حذف آن، cf این ترم 6 واحد کاهش پیدا کرده‌است.

## توابع ذخیره‌سازی و لود نمایه‌ها

متن

# فشرده‌سازی نمایه‌ها

توضیحات

## فشرده‌سازی با variable byte

متن

## فشرده‌سازی با gamma code

متن

## ذخیره‌سازی و لود نمایه‌ها

متن

# اصلاح پرسمان

در این بخش به اصلاح پرسمان ورودی کاربر می‌پردازیم. پس از پیش‌پردازش پرسمان، با استفاده از نمایه bigram و معیار جاکارد، لیستی از کلمات پیشنهادی را برای هر کلمه‌ی پرسمان ورودی پیدا می‌کنیم و سپس فاصله‌ی ویرایش کلمه‌ی پرسمان با هر یک از این کلمات را باز می‌گردانیم و در نهایت کلمه‌ای که این فاصله ویرایش را کمینه می‌کند به عنوان کلمه‌ی پیشنهادی استفاده می‌کنیم. در نهایت نیز پرسمان اصلاح شده را به کاربر نشان می‌دهیم. در ادامه به شرح بیشتر توابع این بخش می‌پردازیم.

## استفاده از نمایه bigram و معیار جاکارد برای یافتن کلمات مشابه

توابع این قسمت شامل jaccard\_similarity و correction\_list هستند.

تابع jaccard\_similarity یک کلمه‌ی پرسمان و یک ترم دیکشنری دریافت می‌کند و مجموعه‌ی دوحرفی‌های پرسمان را محاسبه می‌کند (از $ صرفنظر می‌کنیم). در واقع معیار جاکارد به صورت حاصل تقسیم اندازه‌ی اشتراک مجموعه‌ی دوحرفی‌های پرسمان و ترم، بر اندازه‌ی اجتماع آنها محاسبه می‌شود. از این رو در posting list هر کدام از دوحرفی‌های پرسمان در نمایه‌ی bigram، اگر ترم ورودی را پیدا کند، یک واحد به اشتراک دو مجموعه اضافه می‌شود و در نهایت با استفاده از فرمول زیر معیار جاکارد بازگردانده می‌شود:

سپس نوبت به بازگرداندن لیست کلمات پیشنهادی می‌رسد. تابع correction\_list یک کلمه و یک threshold را علاوه بر زبان ورودی می‌گیرد. سپس آن کلمه‌ی پرسمان را به تعدادی دوحرفی می‌شکند و به ازای هر ترم در posting list، شباهت جاکارد بین کلمه‌ی پرسمان و ترم انتخاب شده را محاسبه می‌کند و اگر میزان این شباهت از threshold ورودی بیشتر بود، این ترم را به مجموعه کلمات پیشنهادی اضافه می‌کند و در نهایت آرایه‌ای از این کلمات باز می‌گرداند.

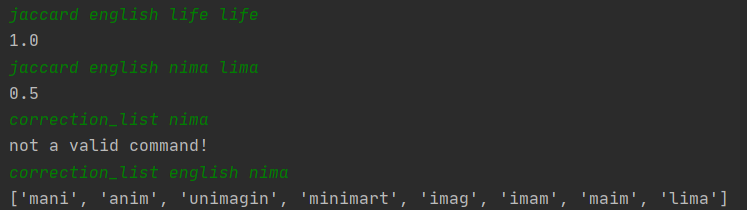
کاربر با استفاده از دو دستور زیر می‌تواند دو تابع معرفی شده در این بخش را بررسی کند:

jaccard [lang] [query] [dictionary term]

correction\_list [lang] [query]

در فراخوانی دستور دوم، ابتدا عدد 0.4 به عنوان threshold داده می‌شود اما اگر آرایه‌ی خروجی تابع تهی باشد، 0.1 از آن کم می‌شود و این عمل تا زمانی که لیست پیشنهادی تهی نباشد، ادامه پیدا می‌کند.

در ادامه مثالی از کار با دستورات jaccard و correction\_list مشاهده می‌شود.



تصویر 5: مثالی از فراخوان دستورات jaccard و correction\_list

## تابع محاسبه‌ی فاصله ویرایش دو کلمه

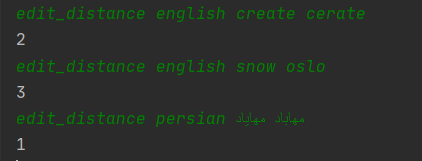
تابع edit\_distance برای این قسمت طراحی شده‌است. این تابع با استفاده از برنامه‌نویسی پویا طراحی شده است و در هر مرحله dp[i][j] به صورت زیر محاسبه می‌شود:

و در نهایت dp[len(query)][len(term)] به عنوان خروجی بازگردانده می‌شود.

در ادامه مثال‌هایی از عملکرد تابع edit\_distance دیده می‌شود.

## نمایش پرسمان اصلاح‌شده

در این قسمت از توابع دو قسمت قبلی استفاده می‌کنیم. در صورتی که پس از پیش‌پردازش پرسمان، کلمه‌ای موجود باشد که در دیکشنری موجود نباشد، با استفاده از قسمت اول لیستی از واژه‌های پیشنهادی را باز می‌گردانیم، دقیقا مانند عملیاتی که هنگام اجرای دستور correction\_list انجام می‌شود. سپس از تابع edit\_distance استفاده می‌کنیم و کلمه‌ای را که کمترین فاصله‌ی ویرایش را با کلمه‌ی پرسمان دارد، جایگزین می‌کنیم. مثالی از عملکرد این قسمت در بخش [جستجوی ltc-lnc پرسمان](#_جستجوی_ltc-lnc_پرسمان) قابل مشاهده است.



تصویر 6: مثالی از محاسبه‌ی فاصله‌ی ویرایش

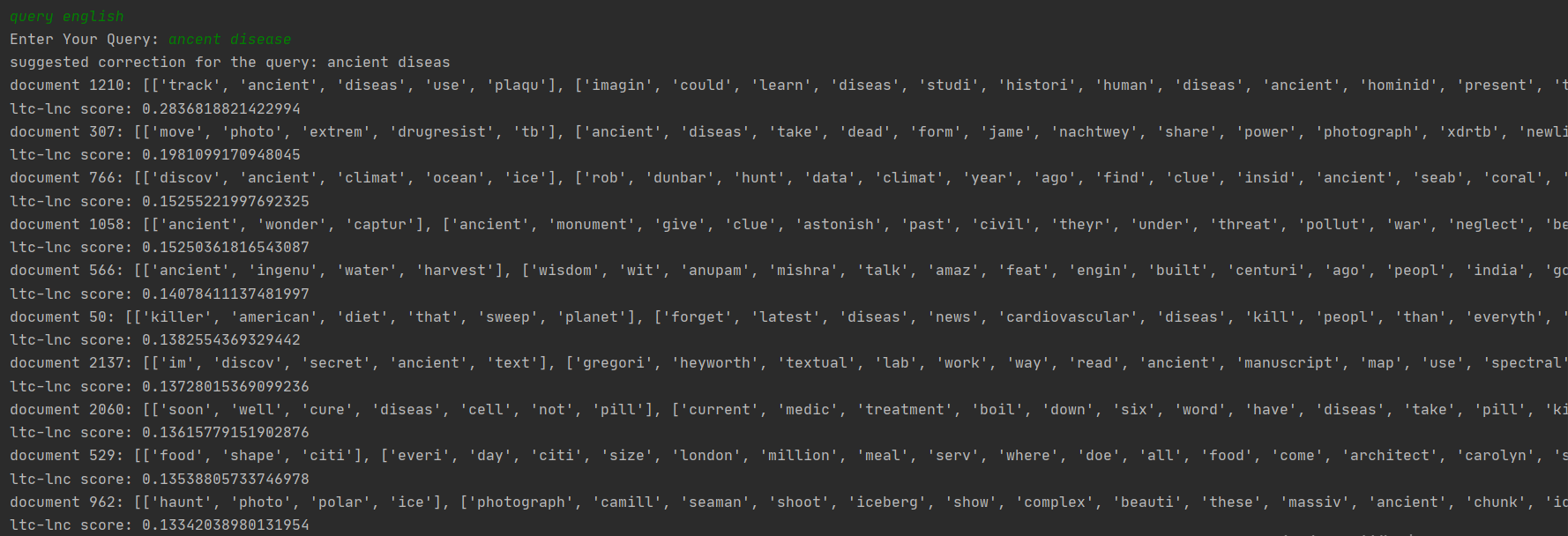
# جستجو و بازیابی اسناد

در این بخش کاربر می‌تواند پرسمان خود را وارد کند و پس از اصلاح پرسمان (در صورت نیاز)، در بخش اول می‌تواند مستندات مرتبط را بر اساس امتیاز ltc-lnc مشاهده کند و در بخش دوم نیز بعد از اصلاح پرسمان، بررسی می‌کند که در کدام مستندات تمامی این کلمات در پنجره‌ی داده شده وجود دارند و سپس آنها را به ترتیب امتیاز به کاربر نشان می‌دهد. در ادامه به شرح توابع این بخش می‌پردازیم.

## جستجوی ltc-lnc پرسمان

این بخش با فراخوانی دستور query [lang] فعال می‌شود. پس از ورود کوئری توسط کاربر، تابع اصلاح کوئری صدا زده می‌شود تا در صورت لزوم اصلاح شود. سپس تابع process\_usual\_query فراخوانی می‌شود. این تابع کوئری را به روش lnc بررسی می‌کند و طول پرسمان و تکرار هر ترم در پرسمان را به دست می‌آورد و سپس به ازای هر شناسه‌ی مستند، تابع tf\_idf صدا زده می‌شود. این تابع ابتدا مقدار عبارت (1 + log tf) × idf را برای هر ترم مستند محاسبه می‌کند و سپس با فراخوانی تابع doc\_length، طول بردار مستند را محاسبه می‌کند و در نهایت ضرب داخلی بردار مستند و بردار پرسمان را خروجی می‌دهد. سپس این امتیازات مرتب می‌شوند و در نهایت حداکثر 10 مستندی که امتیاز ltc-lnc مخالف صفر دارند، خروجی داده می‌شود.

در ادامه نمونه‌ای از فراخوانی دستور مرتبط با این بخش دیده می‌شود. شایان ذکر است که در صورتی که خود tf به‌جای 1 + log tf در نظر گرفته می‌شد، نتایج بهتری به دست می‌آمد.



تصویر 7: نمونه‌ای از پرسمان انگلیسی، همان‌طور که دیده می‌شود، ابتدا پرسمان اصلاح شده و سپس مستندات مرتبط و شواهد آن چاپ شده‌اند.



تصویر 8: نمونه‌ای از پرسمان فارسی

## جستجوی proximity با اندازه‌ی پنجره‌ی داده شده در ورودی

متن

# نحوه‌ی تقسیم وظایف

توضیحات

## نیما جمالی

1-

2-

3-

## سپهر فعلی

1-

2-

3-

## سینا کاظمی

1-

2-

3-

# مراجع