گزارش فاز **دوم** پروژه ی درس ذخیره و بازیابی اطلاعات - پویا پارسا (۹۲۳۱۰۰۵)

نسخه ای از پروژه که در هنگام نگارش این پروژه استفاده شده بود ضمیمه شده.

همچنین آخرین ویرایش این پروژه از طریق مخزن github پروژه قابل دریافت هست :

https://github.com/pi0/IR-Project

مقدمه

در این فازپروژه وارد بخش جدی تری شد و نیاز بود الگوریتم ها و پیاده سازی تا حد بسیار بیشتری بهینه شود.

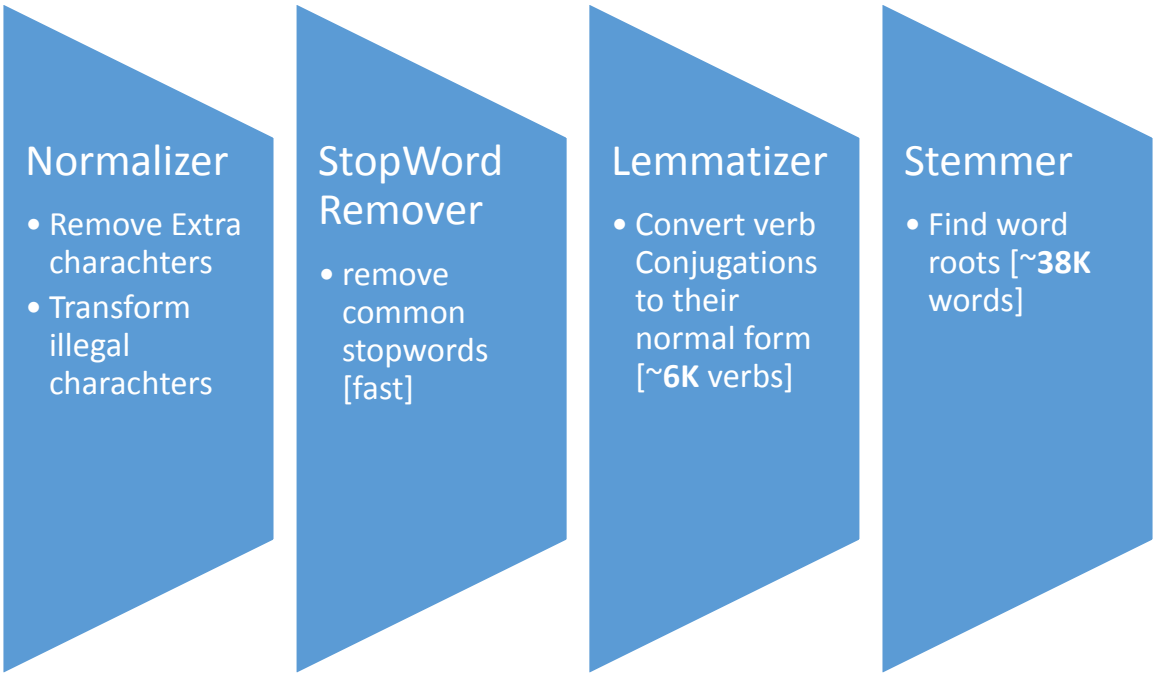
به همین منظور در اولین قدم مشکل پردازش مقالات به صورت جامعی برطرف شد و کلیه عملیات ها توسط کلاس DB Processor انجام شده و این کار به طور همزمان بین چندین Thread توسط کلاس DBProcessorJob پخش می شود.

همچنین عملیات نوشتن نتیجه در یک Thread مجزا که خودش رو با تمامی بخش های پیشین هماهنگ کرده و ترتیب مقالات را نیز حفظ می کند صورت می پذیرد.

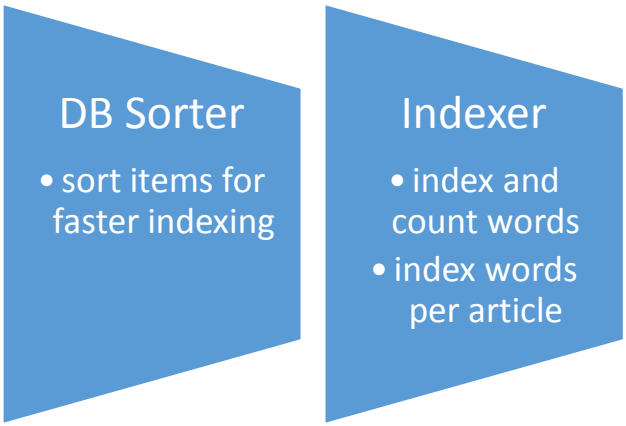
برای افزایش انعطاف پذیری و برنامه نویسی ساختار یافته تر کلیه عملیات هایی که در ادامه بر روی مقالات انجام می پذیرند تحت Processor هایی که متدهای IProcessor را پیاده سازی کرده اند به صورت موازی بر روی چند مقاله انجام می گیرد.

مراحل پردازشی

علاوه بر مراحل فاز قبل شامل Tokenizer و ... پردازش مقالات طبق روند زیر صورت می گیرد :



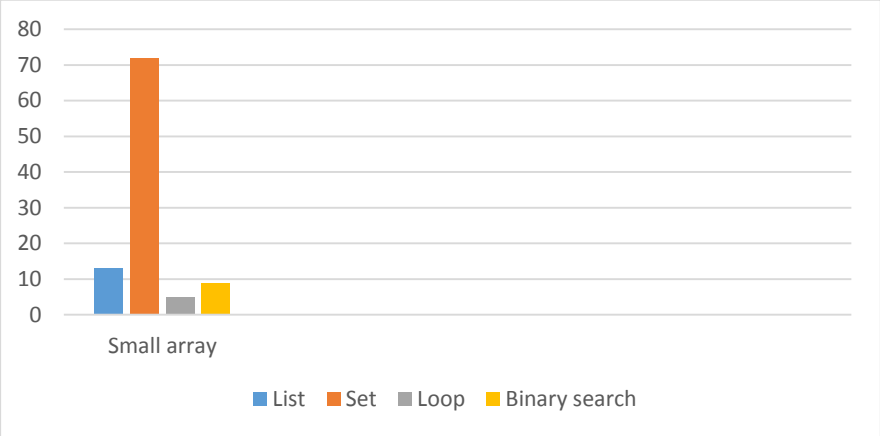
مرحله ۲:

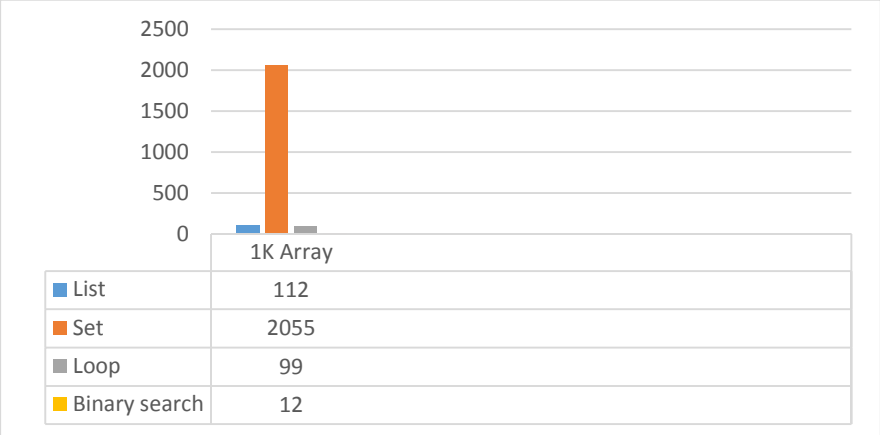


حذف کلمات متداول

برای حذف سریع کلمات متداول باید راهی برای جست و جوی سریع آنها وجود داشته باشد.

ساختارهای List و Set – روش loop و روش Binary Search بررسی شده و نتایج زیر به دست آمد :





به وضوح روش Binary Search برای جست و جو و عملیات Contains مناسب ترین است.

این روش در ساختار داده ای به نام **FastDict** در پروژه پیاده سازی شده.

زمان اندازه گیری شده برای صرفا این فیلتر حدود **۳۰** ثانیه اضافی بود و در نتیجه آن حدود **۲** دقیقه از زمان پردازشی کاسته شد.

همچنین **۲۵** کلمه متداول از نتیجه ی نهایی این گزارش افزوده شد. (فایل data/persian_stopwords.txt)

بررسی کلمات متداول

بعد از اتمام مراحل فهرست کردن کلمات ، لیست به ترتیب تعداد تکرار در هر مقاله مرتب سازی شد.

سپس ۲۱۰۰ کلمه ی پرتکرار که در بیش از ۳۶۰۰ مقاله تکرار شده بودند استخراج شد.

سپس این لیست بر اساس معیار میانگین تکرار در مقالات مرتب سازی شد و لیست زیر از آن استخراج شد. (بعضی واژه ها واقعا اضافی نبودند و به صورت دستی حذف شدند)

نسخه ی کامل این بررسی از مسیر
docs/common_words.ods
قابل دسترسی است.

word	repeats	articles count	average repeat
کرد	**1560959**	**4808**	**324**.6586938
کشور	946813	4808	196.9245008
بود	895761	4808	186.3063644
گفت	886099	4808	184.296797
سال	845340	4808	175.8194676
ایران	809782	4808	168.4238769
داشت	752705	4808	156.5526206
داد	554547	4808	115.3383943
گرفت	540766	4808	112.4721298
شد	529439	4808	110.1162646
توانست	476494	4808	99.10440932
کار	469170	4808	97.58111481
تهران	446570	4807	92.89993759
شهر	433585	4808	90.17990849
قرار	431276	4808	89.69966722
دو	424704	4808	88.3327787
روز	412747	4808	85.84588186
دولت	400503	4808	83.29929285
اسلام	377217	4808	78.45611481
شرکت	367257	4807	76.40045767
سازمان	362626	4808	75.42138103
نظر	344083	4808	71.56468386
مردم	334751	4808	69.62375208
فرهنگ	330861	4807	68.82899938
خواست	326572	4808	67.92262895
بخش	268589	4807	55.87455794
آمریکا	266832	4808	55.49750416
درصد	262910	4806	54.704536
نیست	259241	4808	53.9186772
رسید	258378	4808	53.73918469
راه	257117	4808	53.47691348
انجام	256401	4808	53.32799501
اعلام	255636	4808	53.16888519

ریشه یابی کلمات

شاید چالش برانگیز ترین و جالب ترین قسمت این پروژه به این بخش مربوط بود ☺

- طبق نموداری که در قبل آمد، ابتدا تمامی مقالات به حالت نرمال شده تبدیل میشوند.
- کلمات ابتدا بررسی می شوند که در فهرست کلمات فارسی (۳۸ هزار کلمه) قرار داند یا خیر
- سپس کلمه با حدود **۶۰۰۰** فعل صرف شده فارسی بررسی میشود که آیا فعل هست یا نه (افعال هنگام اجرا صرف و تولید می شوند)
- سپس کلمه وارد ماژول Stemmer شده و **سعی** میشود ریشه ی آن استخراج شود ، اگر ریشه ی استخراج شده در فهرست کلمات فارسی موجود باشد جایگزین خواهد شد.

لیست فعل ها و کلمات از پروژه ی فارسی هضم استخراج شده

ایجاد فرهنگ لغات و شاخص ها

بعد از تهیه ی یک نسخه ی بهینه از بانک اطلاعات مقالات نوبت به شمارش لغات آن ها می رسد.

گرچه ساختار داده های متعددی بررسی شدند و نسخه های بسیار سریعی مثل TreeSet یافت شد اما تمامی عملیات ها به صورت Multithread انجام می شوند و هیچ کدام از آنها این موضوع را تضمین نمی کردند.

ساختار داده ی سریع ConcurrentHashMap برای این عملیات انتخاب شد که عملیات درج و جست و جو در آن با O(n) انجام شده و در برابر عملیات های Multithread ایمن هست.

تکنیکی که برای index کردن استفاده شد و عملیات را تا حدود ۱۰ برابر سریع تر کرد ، افزودن فیلتر Sort قبل از شروع پردازش هر مقاله بود ، به این ترتیب به دلیل اطمینان از مرتب بودن کلمات این امکان به وجود آمد تا lookup برای هر کلمه تنها یک بار در مقاله صورت پذیرد.

تلاش دوم برای افزایش سرعت !

گرچه مرتب سازی کمک فراوانی به افزایش سرعت کرد اما باز هم نیاز به یک همزمانی دسترسی بین thread ها احساس می شد به همین دلیل از یک ساختار LRUCache برای سریع کردن در جست و جوی کلمات استفاده شد.

تلاش برای بهینه سازی حافظه !

با توجه به ماهیت عملیات و نیاز به به روز رسانی تعداد بسیار بالای کلمات و ثبت تکرار آن ها در هر مقاله ، حافظه یکی از گلوگاه های اصلی بود. به همین دلیل تمامی قسمت ها حافظه بلافاصله آزاد شده و حتی هنگام پردازش در صورتی که حافظه از حدی کمتر شود عملیات متوقف و خواندن فایل تا زمان اتمام کار بقیه ی پردازش ها و آزاد شدن حافظه متوقف می شود.

از طرفی ساختمان داده ای که برای ذخیره سازی فهرست کلمات و تکرار آن ها استفاده می شد بهینه نبود. به همین دلیل کلیه ی ساختار های داده با ساختارهای داده پیاده سازی شده کتابخانه GNU/Trove جایگزین شدند. ویژگی این کتابخانه استفاده از داده های primitive به جای نگه داری object های سنگین هست.

نتایج نهایی

بعد از انجام بهبود های این فاز سرعت بسیار خوبی برای محاسبات به دست آمد. و نسبت به حالت اولیه (۸ دقیقه) حدود ۴۰۰% زمان پردازش ها بهینه شد. و این زمان به کمتر از **۲** دقیقه رسید !

```
./scripts/normalize.sh

Normalizing database

Read 829 Items from
data/persian_stopwords.txt

Read 37522 Items from data/persian_words.txt

Loaded 6253 verbs

Starting 8 Parallel workers

Took: 00:01:15:831
```

```
./scripts/index.sh

Indexing database

Starting 8 Parallel workers

Took: 00:00:28:314
```