RI Challenge

Relatório técnico

Versão 1.0

Josué Santos Silva
Pontifícia Universidade Católica de Minas Gerais - Ciência da Computação
2016

Resumo

Relatório técnico sobre o trabalho da disciplina de Recuperação da Informação que teve com objetivo de implementar e avaliar algoritmos capazes de interpretar e enriquecer consultas de usuários, provendo resultados mais relevantes que modelos de recuperação de informação (BM-25) e de pseudo relevance feedback (Rocchio) de referência.

O relatório apresenta informações sobre a implementação algoritmos básicos para a execução do trabalho assim como tecnologias e ferramentas utilizadas no processo. Na sequência é apresentado a proposta do módulo Expansor de Consultas para a máquina de busca. Na conclusão serão apresentados os resultados comparativos de acordo com as métricas estabelecidas entre os algoritmos básicos e o módulo proposto. Palavras-chaves: recuperação da informação. relevance feedback. rocchio. bm25.

Palavras-chaves: recuperação da informação, relevance feedback, rocchio, bm25

Introdução

Tendo como objetivo do trabalho a implementação de uma máquina de busca porém com um foco no desenvolvimento de um módulo expansor de consulta, o projeto é baseado na máquina de busca de código aberto Lucene, pelo fato de a mesma já possuir uma base sólida (LUCENE, 2010; GOSPODNETIC; HATCHER, 2005; STORE) para implantação do módulo propostos. Todo o código fonte do projeto segue em licensa GPV v3 e está hospedado no site do github em https://github.com/josuesasilva/jsearch. Neste repositório estão contidos informações para a execução do projeto assim como instruções para compilação e demais comandos. Também serão mostrados a implementação de ranking utilizando modelo BM25 e modelo Rocchio de pseudo-relevance feedback, que serviram de base para a comparação de resultados obtidos no módulo proposto.

O projeto está separado logicamente em componentes Indexador, Componente Ranqueador, Componente Expansor de Consultas, Componente Gerador de Log. Ao final aspectos de efetividade (qualidade do ranking gerado em MAP, P@5 e nDCG (ROBERTSON; HULL, 2000; VOORHEES; HARMAN, 2001) e eficiência (tempo de resposta) serão reportados em relação a cada componente.

Desenvolvimento

1. Ferramentas

1.1 Apache Lucene

Criado por Doug Cutting em 2000, o Lucene é uma das mais famosas e mais usadas bibliotecas para indexação e consulta de textos, disponível em código aberto. Sob o domínio da Apache Foundation, a biblioteca, escrita em java, pode ser utilizada em qualquer aplicativo J2SE e J2EE, de código aberto ou não. Outras linguagens como Delphi, Perl, CSharp, C++, Python, Ruby e PHP devem usar os ports do Lucene para as referidas linguagens. A biblioteca é composta por duas etapas principais: indexação e pesquisa.

A indexação processa os dados originais gerando uma estrutura de dados inter-relacionada eficiente para a pesquisa baseada em palavras-chave.

A pesquisa, por sua vez, consulte o índice pelas palavras digitadas em uma consulta e organiza os resultados pela similaridade do texto com a consulta. Os índices podem ser criados em ambientes distribuídos, aumentando a performance e a escalabilidade da ferramenta. Em particular no Lucene 4.1, o codec mudou para comprimir automaticamente o armazenamento de documentos. Ele funciona através do agrupamento de documentos em blocos de 16KB e depois comprimi-los em conjunto, utilizando LZ4, um algoritmo de compressão leve. A vantagem dessa abordagem é que ela também ajuda a comprimir documentos curtos uma vez que vários documentos seriam comprimidas em um único bloco. No entanto, a fim de ler um documento único, que você precisa para descomprimir todo o bloco. De modo geral, não importa como descompressão de 16KB para 100 documentos com LZ4 é ainda mais rápido do que executar uma consulta não-trivial ou mesmo apenas buscando em um disco giratório para esses 100 documentos.

Neste projeto é foi utilizada a última versão estável do Lucene, 6.2.1. Embora já disponível compressão de documentos por DEFLATE ainda segue utilizando LZ4 que é o método padrão desde o Lucene 4.

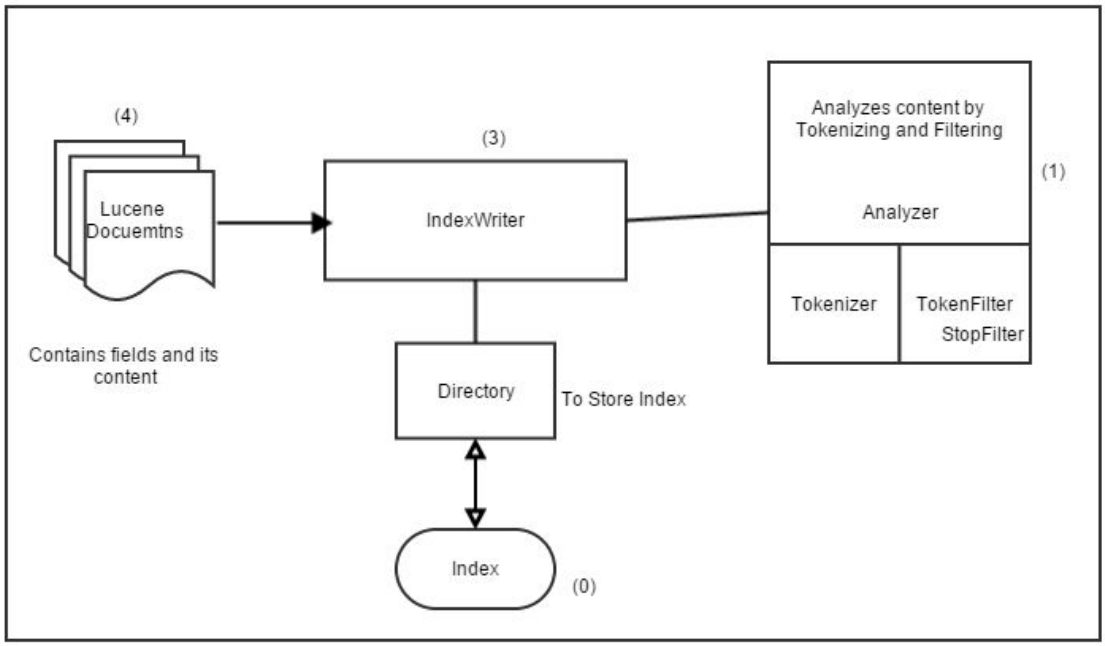


Figura 1: Componentes do Apache Lucene

2. Design da máquina de busca

2.1 Componente Indexador

Para este componente foram utilizados os recursos padrão do Apache Lucene 6, nenhum parâmetro interno como por exemplo compactação de documentos, que utiliza LZ4 desde a versão 4 do Lucene e parâmetros do BM25 não foram modificados.

Foi realizada uma modificação no *Analyzer* do Lucene, para que tanto a query inserida pelo usuário, quanto os algoritmos de ranking considerem a remoção de stopwords(este projeto considera apenas idioma inglês), e tokenização.

2.2 Componente Ranqueador

O ranking da máquina de busca proposta utiliza a função de ranking BM25(PÉREZIGLESIAS et al., 2009) na qual o Lucene possui uma implementação em sua biblioteca. Todos os parâmetros da função foram deixadas no padrão estabelecido pela função(k1 = 1.2 e b = 0.75) que é bem indicado no caso de coleção heterogênea. O componente de ranking também possui um mecanismo de expansão de de consulta, no qual foi implementado o algoritmo Rocchio(ROCCHIO, 1971; JOACHIMS, 1996), utilizando como parâmetros "alpha = 1.0"e "beta = 0.8". A primeira consulta foi realizado utilizando o ranking BM25, foram selecionados os 10 melhores resultados como os resultados relevantes para a execução do restante do algoritmo, além de que a quantidade de termos em que a consultada foi expandida foi considerado um valor de no máximo 10 termos. Na seção de resultado segue dados a respeito do desempenho do componente.

2.3 Componente Expansor de Consultas

O componente expansor foi implementado baseado no Log de consultas e em uma varredura prévia pelos documentos indexados, em busca de termos importantes.

Na primeira parte são verificados no log consultas, consultas semelhantes previamente efetuadas e seus respectivos documentos relevantes para os usuários.

Na segunda parte é efetuada uma busca com o BM25, e são retornados os 10 melhores resultados. A partir deles são retirados termos, que possuem um peso. Por fim e gerada uma lista de termos ordenada por tf*idf, além de serem atribuídos

pesos em um fator de 1.2 para termos que ocorrem no título dos documentos. O mesmo fator de 1.2 também será aplicado a termos que ocorrem em documentos relevantes obtidos no log de consultas.

Por fim os 10 melhores termos são concatenados a consulta original.

2.4 Componente Gerador de Log

O componente gerador de Log utiliza o banco de dados ObjectDB (OBJECTDB, 2016), um sistema de banco de dados relacional para armazenamento de objetos, para armazenar os dados das consultas efetuadas pelo usuário. Foi escolhido pela facilidade de integração com a aplicação Java existente, e não possuir a necessidade de executar um processo ou binário externo que possa inclusive depender do sistema operacional, estando assim contido na mesma aplicação. Além de ser mais intuitivo e possuir menor overhead em relação ao BerkleyDB, concorrente semelhante.

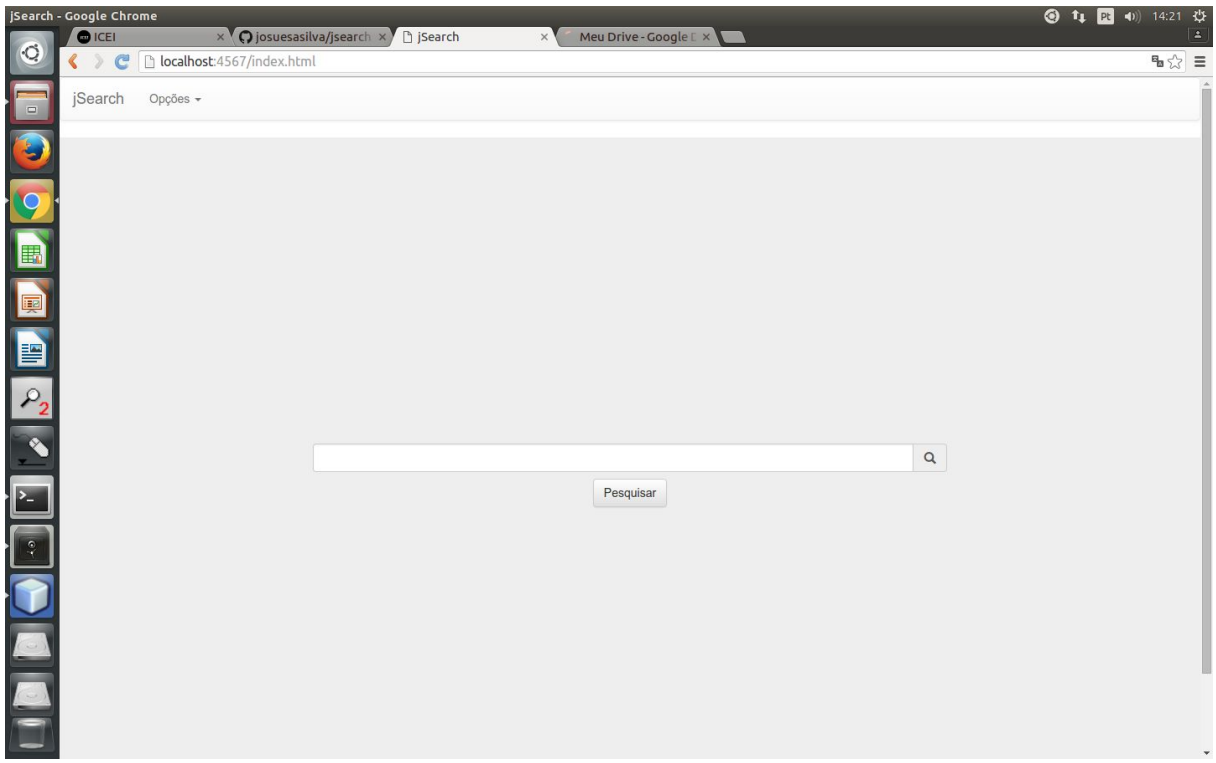
Para cada consulta são armazenados IP de origem do usuário que efetua a consulta, a consulta, o id do documento no qual o usuário interagiu e o timestamp em que a consulta foi realizada.

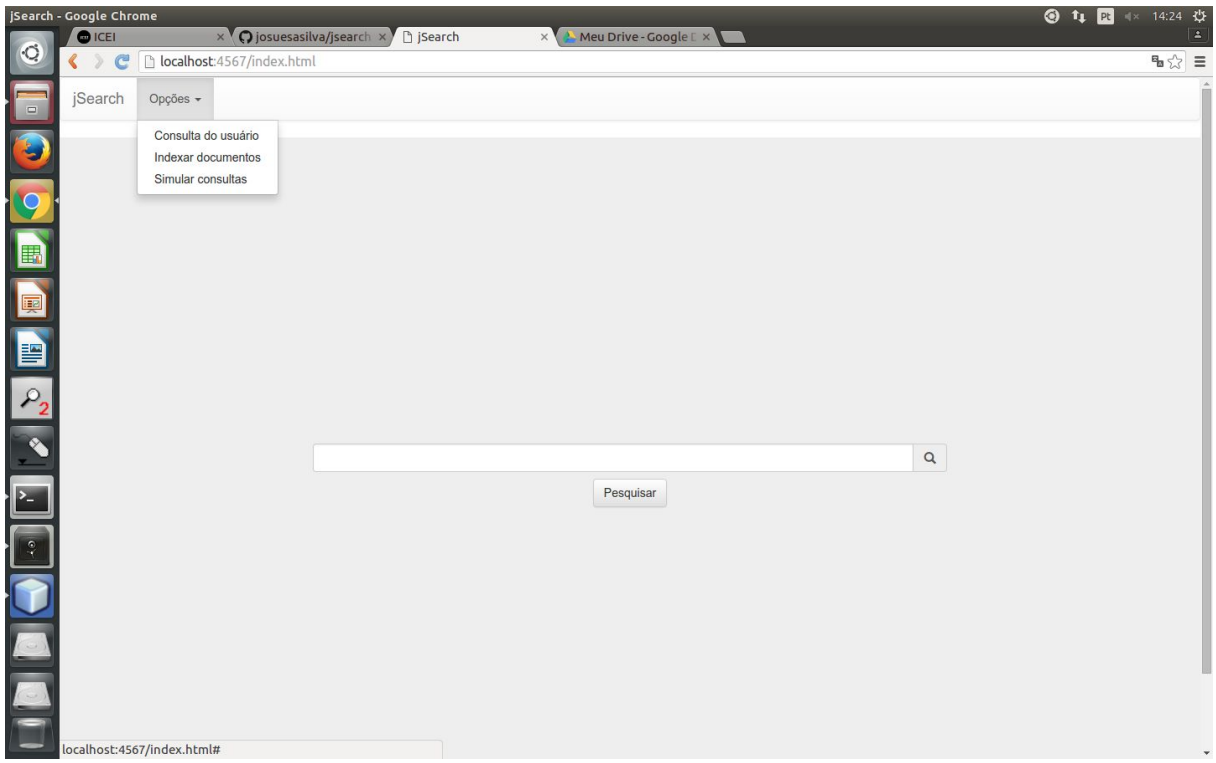
O recurso de autocompletar recupera as consultas realizadas pelo respectivo usuário considerando a princípio as consultas em um intervalo de 24 horas. São retornados no máximo 5 resultados. No final, é retornado como sugestão, uma consulta expandida utilizando o componente expansor de consulta, que gera uma nova consulta baseada em termos sinônimos. Para autocompletar, o componente verifica a similaridade entre a consulta original do usuário e as demais consultas realizadas pelo usuário contidas na base de dados e retorna as cinco melhores. O critério de similaridade utilizado é baseado na Distância de Jaro-Winkler(WIKIPEDIA, 2016). O algoritmo retorna um fator, que quanto maior mais semelhante são os strings comparados. Foi utilizado como fator de corte o valor de 0.8, ou seja, 80% de semelhança entre as consultas comparadas.

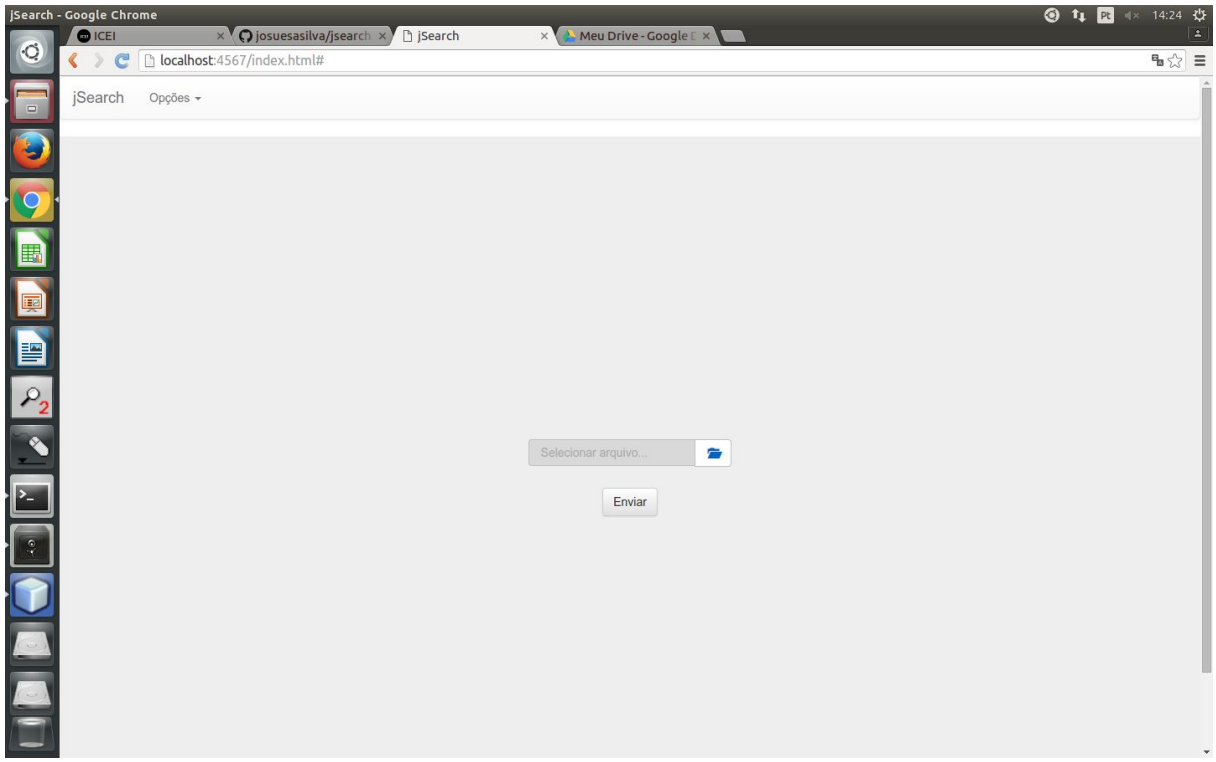
O componente expansor de consultas também deve acessar a base de logs. No caso para cada consulta enviada, será efetuada uma pesquisa na base de logs por uma query semelhante, utilizando o mesmo algoritmo do recursos de autocompletar. Como cada registro possui seu respectivo documento no qual cada usuário interagiu, serão recuperados 10 documentos. A partir destes documentos selecionados serão extraídos 5 termos que devem ser inseridos na consulta original do usuário de forma implícita.

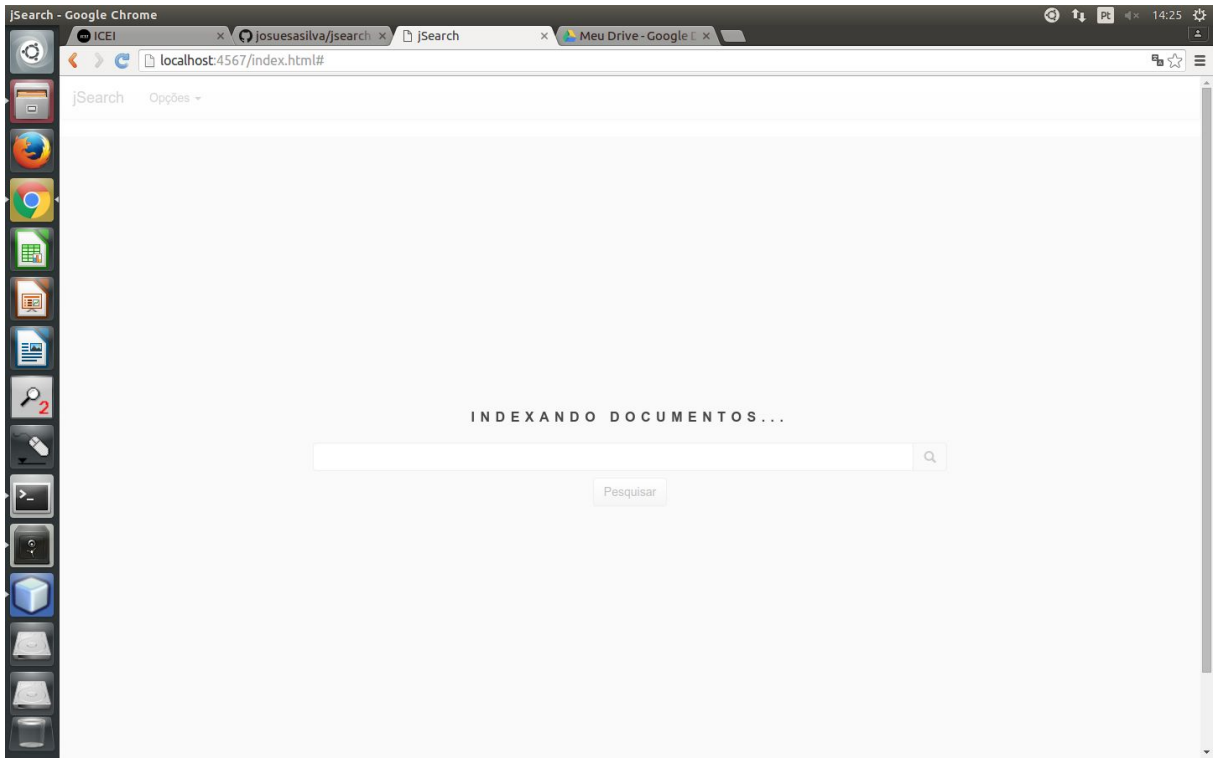
Interface

Através da interface gráfica é possível realizar consultas avulsas e enviar arquivo com consultas para avaliação de resultados dos componentes. Baseado na coleção fornecida do TREC.




jSearch - Google Chrome
ICEI
josuesasilva/jsearch
jSearch
Meu Drive - Google
localhost:4567/index.html
jSearch
Opções
Consulta do usuário
Indexar documentos
Simular consultas
Pesquisar
localhost:4567/index.html#


jSearch - Google Chrome
ICEI
josuesasilva/jsearch
jSearch
Meu Drive - Google
localhost:4567/index.html#
jSearch
Opções
Selecionar arquivo...
Enviar



Resultados

A seguir são apresentados os resultados obtidos pelas implementações do BM25, Rocchio e o expansor de consultas proposto.

Em termos gerais, partindo do oráculo da coleção do TREC e as respectivas queries, não foram obtidas melhoras em relação ao BM25. Isso ocorre pelo fato de que o expansor de consultas proposto levar em consideração interações prévias dos usuários, ou seja, a partir do log de consultas são avaliados documentos relevantes e deles são extraídos termos mais relevantes para o usuário, ou partindo de algo que foi colocado no log para viabilizar consultas mais eficientes.

Em termos de eficiência o expansor de consultas proposto obviamente possui desempenho em tempo de resposta inferior ao Rocchio e ao BM25, pelo fato de fazer uma varredura prévia no banco de dados de Log e de documentos.

Ainda de acordo com os testes efetuados partido do oráculo e das queries do TREC, avaliando cada consulta separadamente, foi notado que algumas poucas consultas obtiveram melhores resultados com o expansor de consultas. Consultas muito pequenas, em alguns momentos, foram beneficiadas, já em outros momentos nem tanto, pelo fato de se tornar ambígua. Consultas maiores (mais que 5 tokens) foram mais beneficiadas que consultas pequenas.

3.1 Ambiente de testes

Os testes foram realizados em um notebook Dell Vostro 5470, com processador i5(dois núcleos e 4 threads) quarta geração, 4Gb de memória ram DDR3, armazenamento SSD. Até o momento, nos testes não foi considerada a coleção de documentos completa.

3.2 Componente Indexador

Segue abaixo dados a respeito do desempenho do indexador:

Documentos indexados	17123	30879	1697253
Tempo de indexação	20,31 seg.	65,411 seg.	2340 seg.
Documentos indexados por segundo	843	472	725
Tempo de resposta a consultas ao índice	0,07 seg.	0,08 seg.	0.052
Taxa de compressão	25%	25%	25%

Tabela 1: Dados de desempenho

O Desempenho do algoritmo BM25, considerando um ranking entre os 10 melhores resultados. Desempenho do BM25 executando a query expandida pelo Rocchio, considerando um ranking entre os 10 melhores resultados.

3.4 Componente Expansor de Consultas

Depois de expandida a consulta inicial o ranking é gerado utilizando o BM25. Segue abaixo dados a respeito do tempo médio de resposta de consultas no BM25, Rocchio e o expansor de consultas proposto.

Algoritmo	Tempo de resposta
BM25	10 ms
Rocchio	47 ms
Expansor de consultas	11 ms

Tabela 2: Dados de desempenho

3.5 Componente Gerador de Log

Operação	Tempo de resposta
Armazenar consulta	344 ms
Recuperar log	78 ms

Tabela 3: Dados de desempenho

Consta no repositório no Github a planilha com os resultados das demais métricas requisitadas..

Referências

GOSPODNETIC, O.; HATCHER, E. Lucene. [S.l.]: Manning, 2005. Citado na página 5.

JOACHIMS, T. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. [S.l.], 1996. Citado na página 11.

LUCENE, A. Apache Lucene-Overview. 2010. Citado na página 5.

OBJECTDB. FAQ. 2016. [Online; accessed 31-August-2016]. Disponível em: . Citado na página 12.

PÉREZ-IGLESIAS, J. et al. Integrating the probabilistic models bm25/bm25f into lucene. arXiv preprint arXiv:0911.5046, 2009. Citado na página 11.

PRINCETON, U. About WordNet. 2010. [Online; accessed 31-August-2016]. Disponível em: . Citado na página 11.

ROBERTSON, S. E.; HULL, D. A. The trec-9 filtering track final report. In: TREC. [S.l.: s.n.], 2000. p. 25–40. Citado na página 5.

ROCCHIO, J. J. Relevance feedback in information retrieval. Prentice-Hall, Englewood Cliffs NJ, 1971. Citado na página 11.

STORE compression in Lucene and Elasticsearch. https://www.elastic.co/blog/store-compression-in-lucene-and-elasticsearch. Accessed: 2016-09-26. Citado na página 5. VOORHEES, E. M.; HARMAN, D. Overview of trec 2001. In:

TREC. [S.l.: s.n.], 2001. Citado na página 5. WIKIPEDIA. Jaro–Winkler distance — Wikipedia, The Free Encyclopedia. 2016. [Online; accessed 13-October-2016]. Disponível em: . Citado na página 12.