Recuperação de Informação - PA1: Web Crawler

Lorenzo Carneiro Magalhães
Universidade federal de Minas Gerais
Belo Horizonte, Brasil

1 Introduction

Nas aulas de Recuperação de Informação, estudamos técnicas para coletar, tratar e indexar eficientemente os dados advindos da web para a construção de ferramentas de busca. O primeiro passo para a implementação desse mecanismo é a implementação de um **Web Crawler**, já que sem os dados não é possível tratar e indexar nada. Assim, o trabalho visou exercitar os conceitos aprendidos dessa primeira etapa. Conforme descrito no enunciado, a implementação seguiu as políticas de coleta especificadas. Logo, o coletor respeita a autoridade do arquivo *robots.txt*, coleta apenas arquivos no formato HTML, não revisita páginas e atua em paralelo.

2 Arquitetura da solução

2.1 Armazenamento

Para o armazenamento das páginas HTML utilizei o Warcio como solicitado no enunciado. No entanto, não conhecia a ferramente e tive um pouco de dificuldade na implementação, mas observando os tutoriais no próprio repositório do github fui capaz de salvar os arquivos e lê-los depois.

O funcionamento do armazenamento não trata a concorrência de threads, pois assumi que isso não era responsabilidade do armazenamento em si e sim do coletor. Devido a esse comportamento, o código é simples. Para o usuário ou outra classe que utiliza a classe de armazenamento, há apenas uma função de adicionar mais um documento, no entanto internamente ele verifica se o buffer interno chegou em 1000 documentos. Caso afirmativo, ele salva o arquivo em um warc compactado.

Caso o número de documentos não chegue a 1000 e não há mais documentos a serem processados, o usuário pode utilizar uma função de salvamento manual. Isso só é necessário para quantidade de documentos não múltiplas de 1000 ou quando o coletor não achou novos documentos.

2.2 Politeness

A política de coleta deve respeitar as normas de cada site, portanto um módulo foi criado com esse propósito. Foram implementadas duas funções para isso, uma para checar se o domínio pode ser coletado e qual o delay para realizar uma nova requisição ao site e outra para esperar o tempo necessário.

O método de esperar é simples e apenas espera o tempo requisitado. Já o método de checagem consta com um hashmap que mapeia os dominios para seus respectivos robots.txt. Isso é útil no caso de múltiplas visitas a um mesmo domínio, de maneira que não sejam necessárias múltiplas requisições. Apesar dessa vantagem, há um custo adicional de extrair o domínio com subdomínio e salvá-lo no hashmap.

2.3 Coletor

Essa é a parte do código principal e mais complexa. O código do coletor está dividido em funções de coleta e no orquestrador e todo o processo. As funções de coleta visitam links, filtram o conteúdo quando não é HTML, extraem as informações das páginas, realizam os logs e classificam os links contidos na página como inlinks e outlinks. É importante ressaltar que considerei outlinks como quaisquer URLs com domínios ou subdomínios diferentes. Caso contrário, a função classifica como inlink. Na minha implementação, os outlinks sempre são inseridos antes dos inlinks, de maneira a incentivar a exploração.

Essa função foi bastante desafiadora de ser implementada devido a grande diversidade de links não padronizados pela web. Para solucionar esse problema, montei um pipeline para formatação da url no padrão mais comum. Empreguei condicionais para o tratamento de links locais, globais, mal-formatados e muitas vezes incorretos. Apesar dos meus esforços, percebi que não seria possível abranger todos os links.

Adicionalmente, defini um timeout de 10 segundos para páginas lentas, assim, consigo dar a possibilidade da página responder ao mesmo tempo que não espero demais. Apesar de ser relativamente bastante tempo, muitas páginas não responderam a tempo e o tempo esperando foi compensado com o número de threads.

No orquestrador, montei o fluxo de toda a execução paralelizada, preocupando com a concorrência das threads. Defini duas fronteiras para uma exploração mais controlada. A primeira fronteira contém os links de domínios já visitados, já os outra contém links de domínios não visitados. Com 70% de chance, a fronteira de domínios não visitados é escolhida para ser explorada caso não esteja vazia, permitindo com que o coletor navegue melhor pelas páginas da web.

Pensando em um contexto de ferramenta de busca, isso é essencial, já que é interessante cobrir uma variedade maior de sites a fim de atender aos variados interesses dos usuários.

3 Resultados

Em suma, com o método de duas fronteiras, foi possível aumentar consideravelmente o número de domínios visitados. A complexidade do método é baixa, mas devido ao número de operações e ao tempo de espera, o processo pode demorar. Pensando nesse tempo de espera, cheguei, através de experimentações manuais, a um valor de threads igual a 48. A variação de performance de 32 threads para cima não é tão relevante, mas abaixo desse valor o timeout de 10 segundos começa a pesar. Esse valor alto de threads definitivamente aumenta a concorrência, mas provocou ainda assim uma execução mais veloz.

Considerando a ampla utilização de estruturas como filas, conjuntos e hashmaps, considerarei o caso médio da manipulação delas como $O(1)$. Dado l links, o custo é $O(l)$. Entretanto, na exploração de um link, tenho que processar o conteúdo da página e seus links

de saída, o que é $O(p+s)$, adotando p como um custo linear advindo desse processamento e s o número de links de saída, tem-se $O(l*(p+s))$. Devido a essa eficiência, o código rodou relativamente rápido, demorando cerca de 4 horas.

Passando para uma análise dos dados coletados, que consistem em 100300 documentos HTML, foram feitos 3 gráficos a fim de realizar uma caracterização dos dados e do processo de coleta. O gráfico 1 apresenta o resutaldo desejado com a implementação da política de duas fronteiras: o número de URLs coletadas não é exponencialmente maior do que o número de domínios, fato que ocorre caso não haja nenhum tratamento.

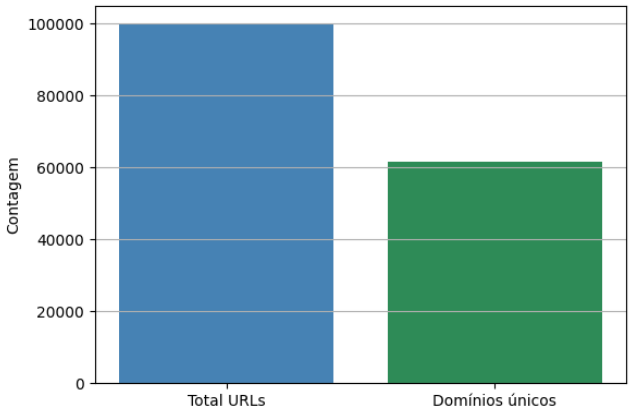

Figure 1: Comparativo entre a quantidade total de URLs coletadas e quantidade de domínios únicos

Apesar da implementação das duas fronteiras, ainda possibilitei a coleta de muitas páginas por domínio, o que, para um trabalho acadêmico, pode não ser muito bom já que não gera tanta variadede de domínio e páginas dentro de domínios. Esse fenômeno ocorre devido à presença de uma enorme quantidade de *inlinks* em sites grandes, de maneira que a fronteira de domínios já visitados fique superpopulada por URLs do mesmo domínio. Um outro ponto é que esse problema ocorre somente no início, visto que o coletor ainda não conseguiu coletar muitas URLs de domínios diferentes. Lembrando que a fronteira de domínios diferentes tem prioridade sobre a outra, espera-se que ao longo da exploração o coletor encontre muitos domínios diferentes e passe a não visitar tão frequentemente a fronteira de domínios não visitados, o que faz com que apenas poucas páginas de domínios explorados após o início sejam exploradas. Esse comportamento pode ser também um problema em coletores reais para ferramentas de busca.

Uma solução para isso no contexto acadêmico seria a limitação de páginas por domínios, o que é uma solução bem mais simples. Esse problema comentado pode ser visualmente observado em 2.

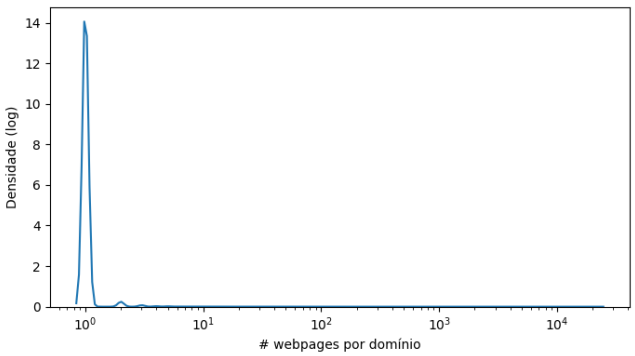

Figure 2: Histograma de páginas por domínio

Por fim, foi encontrado uma tendência de páginas HTML maiores serem menos frequentes, o que é bem natural de ser imaginado. Nessa análise, os tokens são relativos a cada caracter codificado em *utf8*.

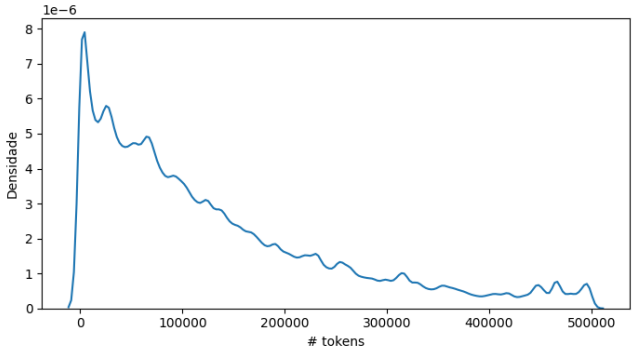

Figure 3: Histograma de tokens por página

Ainda que há melhorias a serem feitas no coletor, principalmente no tocante a um coletor robusto para a construção de uma ferramenta de busca, o resultado foi satisfatório, pois consegui entender os principais desafios enfrentados pelos pesquisadores nessa área.