Programming Assignment #1
Web Crawler

Victor Hugo Silva Moura
victorhugomoura@dcc.ufmg.br
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, BR

ABSTRACT

O objetivo do documento a seguir é detalhar o processo de implementação do Web Crawler para a disciplina de Information Retrieval, bem como destacar os desafios enfrentados durante esse processo. Além disso, ao final do documento, serão fornecidas algumas informações e estatísticas relacionadas ao crawling realizado para 100.000 páginas.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; *Document representation*; **Web crawling**.

KEYWORDS

information retrieval, web crawler, document representation

ACM Reference Format:
Victor Hugo Silva Moura. 2022. Programming Assignment #1 Web Crawler. In *Proceedings of (Information Retrieval (UFMG))*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUÇÃO

Um web crawler, ou rastreador web (em português), é um programa de computador que navega pela rede mundial de uma forma metódica e automatizada. Ele é responsável por baixar e indexar conteúdo de várias partes da internet, um processo chamado de rastreamento da rede ou indexação.

O objetivo principal de um web crawler é detectar do que se tratam (quase) todas as páginas da internet para que as informações possam ser recuperadas quando necessário. Esses rastreadores geralmente começam sua varredura pela web com uma lista de páginas pré-definidas, chamadas de seeds, que podem variar.

Levando isso em consideração, no contexto da disciplina de Information Retrieval da UFMG, foi desenvolvido um web crawler simplificado com o objetivo de tornar mais prático o aprendizado dessa importante ferramenta no processo de indexação das páginas web. Esse crawler é considerado simplificado por não apresentar características mais complexas desse tipo de software, como o parser das páginas para identificação dos tokens, criação de um índice invertido e revisitação de páginas, por exemplo.

Information Retrieval (UFMG), 12 de Maio de 2022, Belo Horizonte, MG

ACM ISBN 978-X-XXXX-XXXX-X/22/05…$0.00
https://doi.org/XXXXXXX.XXXXXXX

2 IMPLEMENTAÇÃO

A implementação do Web Crawler seguiu como padrão o modelo geral de um web crawler, que foi apresentado em classe, durante as aulas sobre esse assunto. Esse modelo pode ser visto na figura abaixo.

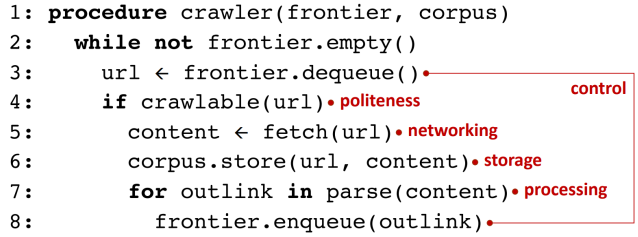

Figure 1: Modelo Geral de Execução de um Web Crawler, visto em aula

Algumas modificações foram feitas, no entanto, a fim de cumprir alguns pontos principais do trabalho, como a execução em paralelo, políticas de revisitação e a seleção de páginas HTML apenas, por exemplo. Essas modificações e a estrutura final do crawler serão apresentados nas subseções a seguir.

2.1 Política de Revisitação

A fim de garantir que as páginas que já foram visitadas anteriormente não sejam revisitadas, é necessário alguma forma de guardar a informação de quais URLs já foram consultadas. Para isso, foi feita a implementação de um set de páginas visitadas. Dessa forma, é possível guardar quais URLs o crawler já descobriu e evitar a revisitação dessas URLs.

De forma a tornar o processo um pouco mais efetivo, e evitar guardar páginas desnecessárias na fronteira de páginas a serem visitadas, as páginas já são marcadas como "visitadas" no momento em que são inseridas na fronteira, evitando assim a inserção de URLs duplicadas nessa fila. Além disso, as URLs inseridas são normalizadas para evitar que pequenas alterações na URL façam uma mesma página ser inserida múltiplas vezes.

2.2 Seleção de Páginas HTML

Em paralelo a inserção de links na fronteira e no set de páginas visitadas, é feita uma primeira checagem dos links para verificar se a página se trata de um arquivo PDF, por exemplo. Essas páginas são caracterizadas por conterem um "`.pdf`" ao final da URL. Dessa forma, essa filtragem inicial ajuda a não inserir arquivos PDF na fronteira.

Após isso, quando a página é retirada da fronteira para ser baixada, durante o processo de download da página os cabeçalhos da requisição são analisados. Caso o **content-type** da página seja "`text/html`", o que representa uma página HTML, então o processo com ela continua. Caso contrário ela é descartada.

2.3 Temporização e Acesso

No entanto, antes de fazer uma requisição da página para o host, é necessário verificar se o crawler possui acesso à página e se existe algum limite de tempo para consultas consecutivas naquele host.
Sendo assim, antes de ser feito o download da página, é necessário verificar o arquivo `robots.txt` do host, que contém informações sobre o bloqueio e acesso de páginas para alguns crawlers, além de informar o tempo mínimo para requisições consecutivas. No caso de páginas que não possuíam essa temporização, o tempo considerado foi de 100ms. A verificação desse arquivo é feita por meio da biblioteca `reppy`, que é especializada em leitura e interpretação desse tipo de arquivo. Com isso, é necessário passar consultar a URL do robots por meio de funções dessa biblioteca e verificar a possibilidade de crawling, bem como o tempo de delay entre requisições.
Caso a consulta à página seja possível, ainda falta verificar se a última consulta feita ao host está de acordo com o tempo entre requisições. Para isso, foi criado um dicionário que mapeia um host ao timestamp da última consulta feita a ele. Sendo assim, caso o host não esteja nesse dicionário, ele é incluído juntamente com o timestamp atual. Caso contrário, o timestamp da última consulta é obtido e é feita uma verificação para garantir que o tempo de delay já passou. Se esse tempo já tiver passado, a página é consultada normalmente. No entanto, se ainda não tiver passado, ela é inserida ao final da fila para posterior consulta.
Para página que por algum motivo venham a dar algum erro durante seu download, é feito um máximo de 10 tentativas de download, seguindo os parâmetros de temporização.

2.4 Salvamento das Páginas

Para o salvamento das páginas, foi utilizada a biblioteca WARCIO, do Python. Essa biblioteca fornece uma interface simples e fácil de ser utilizada para salvamento de páginas Web.
A integração dela no código ocorre em duas partes. Inicialmente, antes do início do processo de crawling, é criado o arquivo inicial de crawl e um writer da WARCIO é inicializado. Após isso, já dentro do loop principal do crawler, a escrita é realizada após o fetch da página, assim como mostrado no modelo geral na Figura 1. Para essa escrita, inicialmente os dados obtidos na requisição são preparados de acordo com o que é solicitado pelo modo de escrita avançado da biblioteca, e após isso é feita a escrita da página. Feito esse processo, o contador de páginas salvas é incrementado de um.
No entanto, antes mesmo de salvar a página, de modo a garantir que existam 1000 páginas por arquivo salvo, é checado se o contador atual é um múltiplo de 1000. Caso seja, o arquivo anterior é fechado e um novo é aberto. Além disso, o writer é redefinido para utilizar esse novo arquivo. Após isso, o salvamento segue como descrito.

2.5 Paralelização

Com a estrutura principal do crawler pronta, o que resta é apenas paralelizar o código. Sendo assim, o loop principal foi inserido em uma classe que seria posteriormente transformada em uma thread. Além disso, as principais estruturas do código, como a fronteira de páginas, o set de páginas visitados, etc, foram transformados em variáveis globais, de forma a serem compartilhados por todas as threads.
No entanto, ao compartilhar essas variáveis o processo de consulta e modificação delas pelas threads pode gerar uma condição de corrida, que é quando processos tentam modificar um recurso simultâneamente, o que pode gerar inconsistências. Nesse caso, foi necessário utilizar um `Lock`, que é uma estrutura presente na biblioteca de Threads de Python, responsável por garantir exclusão mútua entre threads. Ou seja, na região delimitada por esse mutex, como geralmente são chamados, apenas uma thread pode entrar por vez, e somente após ela sair dessa região outra thread pode entrar.
Sendo assim, inicialmente foi utizado um Lock para bloquear o acesso de múltiplas threads em todas as regiões que utilizavam variáveis compartilhadas. Porém, após algumas execuções notou-se que era necessário fazer o mesmo para o salvamento das páginas no arquivo WARC pois, sem esse controle, o acesso simultâneo de várias threads a ele poderia corrompê-lo.

3 EXECUÇÃO

O web crawler foi executado em um sistema com as seguintes especificações:

- **CPU:** AMD Ryzen 5 3600 (3,9 GHz)
- **RAM:** 16 GB DDR4 (2.800 MHz)
- **Armazenamento:** 2TB HDD (7200 RPM) + 2TB SSD
- **Rede:** Download de 250 Mb/s

Inicialmente foram feitos diversos testes pequenos, com no máximo 10.000 páginas, para garantir a funcionalidade do crawler. Com esse número já foi possível notar alguns padrões e fatores que poderiam prejudicar a execução do crawler.

Um deles foi a presença de páginas muito similares que modificavam o subdomínio do host ao invés do restante da URL. Sendo assim, alguns desses domínios encontrados foram filtrados e deixaram de ser inseridos na fronteira.

Outro fator foi a presença de páginas que ramificavam demais para o mesmo host dela, como o site do G1, por exemplo. As páginas desse host possuem muitos links que apontam para páginas dele próprio e, com isso, a fila nessas áreas fica muito limitada à sites esse host apenas. Dessa forma, é feito um shuffle/reordenação da fila periódicamente, de forma a garantir que a sequência de páginas na fila esteja quase sempre a mais equilibrada possível entre os hosts.

Com esses dois fatores garantidos foram executados novos testes para verificar o número de threads ideal para executar o programa, garantindo que haveria melhoras em relação à execução com uma thread, porém minimizando o overhead envolvido na troca de contexto entre as threads e/ou gerado pelas zonas de exclusão mútua. Para cada número de threads, foi feito um crawl de 500 páginas durante 5 vezes para cálculo de média e desvio padrão. Os resultados desse teste podem ser vistos à seguir:

Table 1: Tempo de Execução Para 500 Páginas

Nº de Threads	Tempo de Execução (s)
1	498, 77 ± 50, 44
2	262, 72 ± 10, 87
5	153, 81 ± 2, 69
10	152, 19 ± 10, 16
12*	141, 17 ± 5, 08
25	144, 34 ± 9, 65
50	197, 84 ± 9, 38
100	244, 31 ± 5, 94

* Número de Threads do Processador

Com base nos testes acima, é possível notar que a melhor configuração se deu com 12 threads, que coincidentemente é o mesmo número de threads que o processador utilizado nos testes possui. Sendo assim, o crawler foi executado para as 100.000 páginas utilizando 12 threads.

4 RESULTADOS

Após executar o crawler e fazer o download das 100.000 páginas requisitadas para o trabalho, foi feita uma análise dos dados obtidos por meio desse crawling. Abaixo temos algumas estatísticas gerais do `corpus` obtido:

- **Número de Páginas Baixadas:** 100.000
- **Número de Hosts Únicos:** 3.000
- **Tamanho Médio Por Arquivo:** 64.407,13 KB
- **Tamanho Total:** 6,14 GB
- **Tempo Médio Por Arquivo:** 3 minutos e 31 segundos
- **Tempo Total Gasto:** 5 horas e 52 minutos

Com isso, podemos ver que no final o crawler acabou gastando menos tempo por arquivo do que o tempo visto durante os testes. De acordo com os testes, seria gasto por volta de 4 minutos e 42 segundos por arquivo de 1.000 páginas. Essa redução no tempo provavelmente foi causada pelos shuffles feitos na fila de páginas, somado aos novos hosts que vão sendo descobertos com o tempo.

Considerando aspectos mais específicos do crawling, temos as seguintes informações:

Table 2: Hosts Mais Frequentes

Host	Nº de Páginas Visitadas
https://www.ultrafarma.com.br/	24347
https://en.wikipedia.org/	10524
https://empresas.americanas.com.br/	6963
https://play.google.com/	5916
https://www.tiktok.com/	3105
https://web.archive.org/	2840
https://g1.globo.com/	2750
https://store.nba.com/	2317
https://www.legislation.gov.uk/	1720
https://clube.netshoes.com.br/	1292

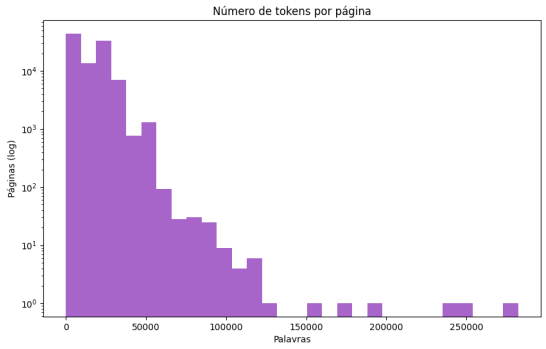

Figure 2: Histograma contendo o número de tokens por cada página visitada

Por meio da Tabela 2 podemos ver que, apesar de adotar politicas para evitar muitas páginas de um mesmo host, tivemos grande predominância de páginas da Ultra Farma, seguidas pelo Wikipedia. Mesmo assim, temos páginas interessantes que ficaram entre os hosts mais frequentes, como o Tik Tok, o Web Archive e até mesmo a loja da NBA.

Por outro lado, considerando o histograma 2 é possível ver que a grande maioria das páginas baixadas possui menos de 100.000 tokens/palavras de conteúdo. Além disso, vemos que existe uma pequena parcela que possui mais de 250.000 tokens, o que pode ser considerado um número alto de tokens para uma página web.

5 CONCLUSÃO

Por meio da execução desse trabalho, foi possível aprender um pouco mais sobre como funcionam os web crawlers e as dificuldades envolvidas durante a implementação de um software para essa finalidade.

Considero que o trabalho serviu de grande aprendizado para mim e forneceu uma forma prática de aplicar o conteúdo visto em sala de aula.

REFERENCES

[1] B Barla Cambazoglu and Ricardo Baeza-Yates. 2015. *Scalability Challenges in Web Search Engines* (1st. ed.). Morgan Claypool.
[2] Cloudfare. c2022. *O que é um web crawler? | Como funcionam os web spiders.* Retrieved 12 de Maio de 2022 from https://www.cloudflare.com/pt-br/learning/bots/what-is-a-web-crawler/
[3] W. Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. *Search Engines: Information Retrieval in Practice* (1st. ed.). Pearson.