**Recuperação de Informação**

Trabalho prático 1

Ewerton S. Santos

Departamento de Ciência da Computação

Universidade Federal de Minas Gerais

Belo Horizonte, MG - Brasil

[ewerton\_dc@hotmail.com](mailto:ewerton_dc@hotmail.com)

Matrícula: 2016058140

Belo Horizonte, MG

Dezembro, 2020

1. **Introdução**

Um motor de busca na web moderna possui três componentes principais: um coletor de páginas (***crawler***); sistema de índice; um processador de consultas. Como primeira etapa do processo de recuperação de informação, faz-se necessária a obtenção dos documentos web, por meio de um ***crawler***, no caso os arquivos HTML de cada página, para que estas sejam armazenadas, processadas e indexadas para as consultas do usuário final.

Em um primeiro momento, é necessário partir de um conjunto de sítios web relevantes, os quais vão nos levar às demais páginas do mesmo e à páginas de outros servidores de relevância. Dessa forma, em uma página obtemos novos links a serem coletadas, o que implica em um processo incremental.

1. **Desenvolvimento**

Para desenvolver o nosso coletor utilizou-se a linguagem C++, sistema operacional Ubuntu 18.04 e os componentes gratuitos da biblioteca ***Chilkat***® (possui configuração especifica para cada sistema operacional, vide reade.md), mais especificamente o módulo ***Spider***.

Para realizar o processo de coleta fora criado a função ***requestCrwaler****,* que recebe um número inteiro **n** e um string contendo a **url** do domínio a ser coletado. O método ***CkSpider.Initialize*** recebe a url base a ser raspada, abstraindo um conjunto de tarefas complexas, como por exemplo, criar uma lista de url a serem acessadas.

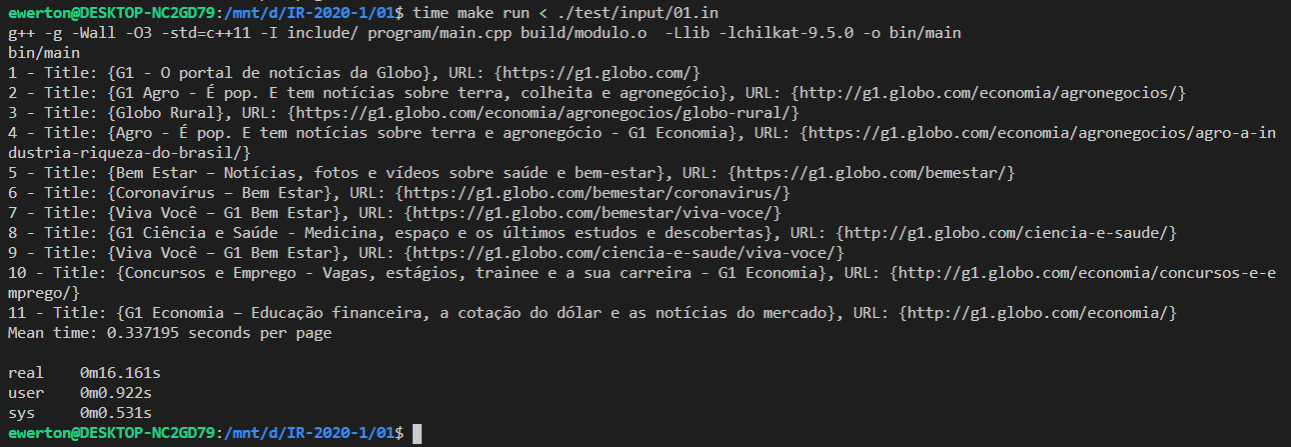
Com o método ***CkSpider.CrawNext***é possível iterar sobre a lista de urls, realizando requisição para as mesmas. O método ***CkSpider.lasHTMLtitle*** retorna o título da url requisitada e ***CkSpider.lastURL*** retorna a url para qual for realizada a requisição. Dessa forma, é possível utilizar um loop com **n** iterações para obter o título e o link das páginas continas na url fornecida à função.

No intervalor entre cada requisição, é contabilizado o tempo de cada requisição, que é utilizado para calcular o tempo médio de reposta de cada servidor.

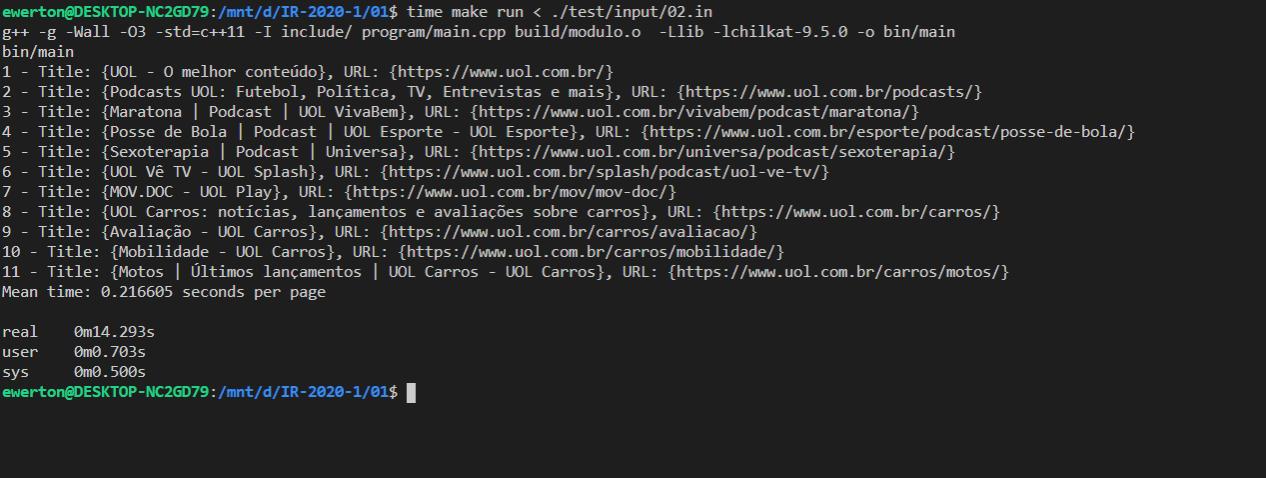
1. **Resultados**

Como teste para o funcionamento do código, foram realizados 30 testes para cada url distinta, requisitando para cada uma 10 sub links. Os outputs finais para estes testes estão na pasta ***teste/output/***.

Para o site “https://g1.globo.com/”, o tempo médio para requisição de cada página, nos 50 testes foi de 0.37822 segundos.



Para o site “https://www.uol.com.br/”, o tempo médio para requisição de cada página, nos 50 testes foi de 0.19496 segundos.



1. **Referências**

[1]. Chilkatsoft. Chilkat Reference Documentation.

[2]. Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto. 1999. Modern Information Retrieval. Boston, MA: Addison Wesley.

1. **Anexos**

[1]. main.cpp

#include <iostream>

#include "modulo.hpp"

#include <CkSpider.h>

int main(int argc, char\* argv[]){

    int n = 0;

    std::string url;

    std::string aux;

    // Get input

    std::getline(std::cin, aux);

    n = std::stoi( aux );

    std::getline(std::cin, url);

    // Call function for crawling

    requestCrwaler(n, url);

    return 0;

}

[2]. crawler.cpp

#include <iostream>

#include <iomanip>

#include <chrono>

#include <CkSpider.h>

#include "modulo.hpp"

void requestCrwaler(int n, std::string url){

    int i = 0;

    double countTime = 0.0;

    CkSpider spider;

    // Object spider for crawls pages

    spider.Initialize(url.c\_str());

    //  Begin crawling the site by calling CrawlNext repeatedly.

    for (i = 0; i < n + 1 ; i++) {

        // Start time for request URL

        auto start = std::chrono::high\_resolution\_clock::now();

        bool success;

        success = spider.CrawlNext();

        if (success == true) {

            std::cout << i+1 << " - Title: {" << spider.lastHtmlTitle()<< "}, URL: {" << spider.lastUrl()  << "}" << std::endl;

        }

        else {

            //  Did we get an error or are there no more URLs to crawl?

            if (spider.get\_NumUnspidered() == 0) {

                std::cout << "No more URLs to spider" << std::endl;

                break;

            }

            else {

                std::cout << spider.lastErrorText() <<  std::endl;

            }

        }

        // End time for request URL

        auto end = std::chrono::high\_resolution\_clock::now();

        countTime = countTime + (std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count()) \* 1e-9;

        //  Sleep 1 second before spidering the next URL.

        spider.SleepMs(1000);

    }

    // Mean time for request

    std::cout << "Mean time: " << countTime/double(n) << std::setprecision(4) <<  " seconds per page" << std::endl;

}