Assignment 2 - Information Retrieval

Luiz Philippe Pereira Amaral

[1]Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)

Abstract. *This project is a web crawler which takes as input a seed file containing a list of starting URLs and produces as output a summary of collected pages, information about total and average time and the HTML of each collected page.*
Keywords: *C++, Chilkat, web crawler*

1. Implementation

The code is written in C++ 17 and utilizes C++'s Standard Template Library (STL) and the Chilkat library. Instructions for installing Chilkat can be found at `https://www.chilkatsoft.com/downloads_CPP.asp`.

1.1. Crawler

The main entity for the project is the crawler, which is responsible for organizing multiple tasks (the short-term scheduler) and manage the long-term scheduler. Initially, all our seeds are pushed into the long-term scheduler. The crawler initializes a constant number of tasks (defined via code), each of those begin consuming from our long-term scheduler for valid URL addresses. If there are none left, the crawler awaits for the running tasks to resolve a new valid URL, then fetches this address from the long-term scheduler and starts the task. If there are no active tasks and the long-term scheduler is empty, the program will be terminated early with a ”bad seed”error message. Every collected page is summarized under the crawler's registry, and it's HTML content is stored locally. When the target number of pages is reached, the program outputs a report on the collected pages, awaits for every thread to finish and terminates the execution.

1.2. Short-term Scheduler

From the moment each task is created, it begins crawling the level 0 URL received in a new thread. After each level 1 URL visited, it pushes any outbound links found into the long-term scheduler, after visiting every level 1 URL present, the task finishes the process on that domain and fetches (from the long-term scheduler) a new valid level 0 URL to continue crawling. The task will be interrupted at any moment should the crawler reach the target number of pages to collect.

1.3. Long-term Scheduler

The long-term scheduler is simply a thread-safe queue used to organize the order in which the links are visited, thus, creating a breadth-first-like process. It uses our registry to validate and prioritize URLs from less visited domains. Every time a task from the short-term scheduler requests a new address, one URL is removed from the front of the scheduler's queue, which proceeds to check whether it is valid.

For an URL to be valid, two criteria must be matched: the URL's domain cannot be currently being visited by another task, and the URL itself cannot have been visited in the past. If an URL is evaluated and considered invalid, then it is rescheduled (sent to the end of the long-term scheduler's queue) and a new one is evaluated, otherwise, the valid URL is sent to the short-term scheduler.

1.4. Registry

The registry is a store for the crawler summary. It has one entry for each domain visited. A registry entry is a structure that contains the associated domain, the visited URLs for that domain (level 0 only), the number of pages collected for every URL under that domain, a signal that tells us whether the domain is currently being visited in some task and the total time spent visiting that domain.

1.5. Compiling and running

You can use make to build the project by simply typing ”make”on your terminal. The output is an executable file inside the build directory. You will need, however, to install the Chilkat library on your environment. Use ”make run”to execute the crawler with a predefined seed, or use ”./build/web-crawler [PATH TO SEED FILE]”to specify a custom seed.

2. Results

The chosen target number of pages was 100000 (one hundred thousand). The file used as seed for the crawler contained the following URLs:

```
ufmg.br
kurzgesagt.org
www.cam.ac.uk
www.nasa.gov
github.com
medium.com
www.cnnbrasil.com.br
disney.com.br
en.wikipedia.org
www.reddit.com
br.yahoo.com
```

The crawler reached the target number of pages in approximately 60 minutes using 50 threads, the average time per request was 1.37797 seconds. Note that this average is not equivalent to the total execution time, since only the time between the request and response is accounted, while total execution time is a measure for every code instruction. 10.968.088.732 bytes (approximately 11 gigabytes) of HTML data were downloaded, representing an average of approximately 109.68 kb per page.

```
        Crawled 100000 in 3727.89 seconds (50 simultaneous processes)
        1.37797 in average per page
```

During the execution, the crawler visited 1198 different domains. The fact that we were able to visit so many different domains is due to the way our long-term scheduler works, our breadth-first approach results in a more horizontal search, other than a vertical one, where we would collect various pages for a single domain. This result is desirable, since our collection of pages is intended to represent a general view of the web instead of a particular aspect (like news, scientific research, etc). It also improves the crawler's politeness, since we will not be overwhelming a single server with too many consecutive requests.

These results show that our approach was suitable for the given problem, considering that even with an ordinary hardware (a CPU with 4 cores and 8 threads), we have been able to iterate over a large collection of documents within a moderate period of time.