Web Crawling

Tom Taylor – 13015452

12th November 2015

1 Instructions

The crawler requires a POSIX compatible operating system (e.g. Linux, OSX, Unix) and Ruby $\geq$ 1.9. In addition, it requires the rubygem Nokogiri to be available.

The crawler itself is called **bot_tom.rb** and expects files for the three classes (Queue, Store and Page) to be in the same directory.

Execution of **bot_tom.rb** should produce the files crwal.txt and results.txt in the current working directory.

2 Program Architecture

i Components

- **bot_tom.rb** – The main script
- **Store** – A class to store the links and produce the required output
- **Queue** – A class to store and evaluate the request queue
- **Page** – A class to fetch a page and parse its links

ii Overview

1. Queue creates a Page object with a URL (initially the seed URL)
2. Page requests content from WWW and parses HTML to extract links which are returned to the Queue object.
3. Queue object parses links from Page adding any that are not yet stored, not blocked by robots.txt and within the URL restriction to itself.
4. Queue sends these to Store and then shifts the next URL from itself then goes back to step 1 until there are no more URLs.

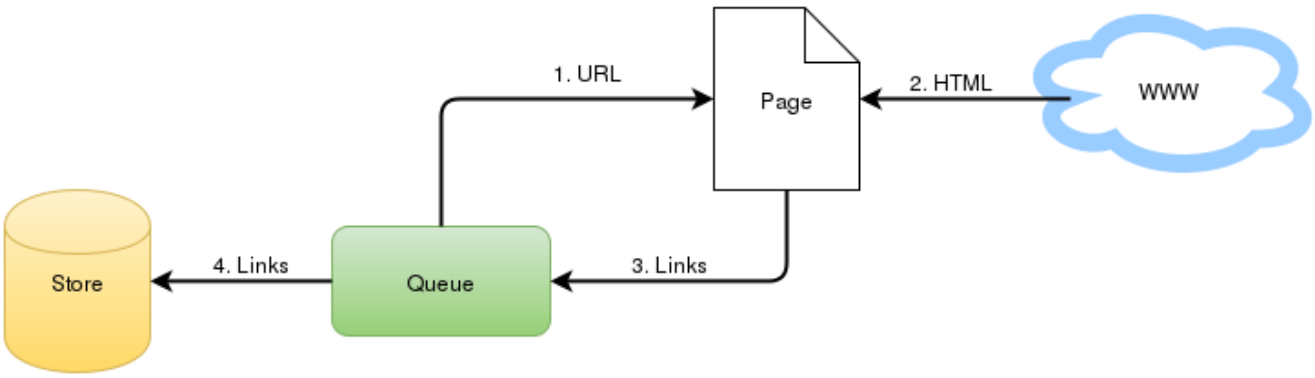

Figure 1: Summary of crawling process

iii bot_tom.rb

It contains hardcoded seed and robot URLs. It parses the robots.txt and produces an array of blocked URLs. It also initialises the Store and Queue classes before passing the seed to the first URL. On completion of the queue (i.e. when the crawling is complete) it creates the two output files and uses the dump and count methods of Store to populate them. It also has a TTL (time to live) which sets a maximum depth of links to follow. This TTL value is also manipulated to prevent the following of non HTTP links (e.g. mailto).

iv Store

Although this class only stores data as a simple hash, it could easily be overloaded to use any key-value store. Its methods are **add** to add a new URL, **exists?** to test for the existence of a URL in the store, **dump** to produce the output required for crawl.txt and **count** to produce the output required for results.txt.

v Queue

This is the main execution loop of the program. It maintains a list of URLs that neex to be indexed, calls Page to index them and adds them to the Store. The queue itself is maintained as an array and URLs are 'shifted' from the start. Links retrieved from a Page object are added to the Store and if index-able, added to the end of the queue.

vi Page

The Page class fetches a URL and parses the HTML with the Nokogiri library. Any links that are not HTTP are set to a TTL of zero (they will not be persued) and any http(s) links have their TTL decremented by one. Links are stored as an array to be later processed by Queue.

3 External code used

Nokogiri - Ruby gem for parsing HTML
`https://github.com/sparklemotion/nokogiri`