Build Your Own Web Crawler

The goal of this project is to build a very simple web crawler which fetches URLs and outputs crawl results to some sort of log or console as the crawl proceeds.

The below, example invocations refer to our reference Python implementation, but you should feel free to use whatever language you think best exemplifies your ability to write clean, readable, well-structured code. We have also seen great implementations in Java, Javascript, Go, Rust, Ruby, Scala, and probably a few others.

Basic Crawler

You should write an application that takes as input a starting URL for your crawl. The application should then do the following:

1. Fetch the HTML document at that URL
2. Parse out URLs in that HTML document
3. Log/print the URL visited along with all the URLs on the page
4. Loop back to step 1 for each of these new URLs

Your application should fetch multiple "levels" of pages, NOT just the first page and its immediate children.

It is not required, but feel free to implement some sort of stopping condition if you like.

Depending on the page structure and whether you implement some stopping condition or not, this application may never finish crawling, which is why you should output crawl results as you go. You can either write to stdout, console, or some sort of logging mechanism. You should format the output as follows:

```
<URL of page fetched>
   <URL found on page>
   <URL found on page>
   ….
<URL of page fetched>
   <URL found on page>
   <URL found on page>
   ….
….
```

Here is an example from a crawl of the Rescale website from a while ago (your actual input may differ from this as the link structure of this site changes faster than this spec is updated.)

```
$ ./crawler.py http://www.rescale.com
http://www.rescale.com
  http://www.rescale.com/booking/
```

```
  http://blog.rescale.com/events/
  http://insidehpc.com/2017/01/video-rescale-night-showcases-hpc-cloud/
  http://www.digitaleng.news/de/rescale-hosts-first-ever-rescale-night-san-francisco/
  https://blog.rescale.com/rescale-night-recap/
  http://www.rescale.com/about/
  http://www.rescale.com/investors/
  http://blog.rescale.com
  http://www.rescale.com/jobs/
  http://www.rescale.com/legal/
http://www.rescale.com/booking/
  http://blog.rescale.com/events/
  http://insidehpc.com/2017/01/video-rescale-night-showcases-hpc-cloud/
  http://www.digitaleng.news/de/rescale-hosts-first-ever-rescale-night-san-francisco/
  https://blog.rescale.com/rescale-night-recap/
  http://www.rescale.com/about/
  http://www.rescale.com/investors/
  http://blog.rescale.com
  http://www.rescale.com/jobs/
  http://www.rescale.com/legal/
http://blog.rescale.com/events/
  http://www.rescale.com
  http://blog.rescale.com/
  https://blog.rescale.com/archives/
  http://blog.rescale.com/events/
  https://blog.rescale.com/events/category/webinars/
  https://blog.rescale.com/reup/
  http://blog.rescale.com/events/
  http://blog.rescale.com/ja/events/
...
```

So in the above example, on the http://www.rescale.com page, there are the following URLs:

```
  http://www.rescale.com/booking/
  http://blog.rescale.com/events/
  http://insidehpc.com/2017/01/video-rescale-night-showcases-hpc-cloud/
  http://www.digitaleng.news/de/rescale-hosts-first-ever-rescale-night-san-francisco/
  https://blog.rescale.com/rescale-night-recap/
  http://www.rescale.com/about/
  http://www.rescale.com/investors/
  http://blog.rescale.com
  http://www.rescale.com/jobs/
  http://www.rescale.com/legal/
```

Note you do not need to worry about nesting in your output, each URL visited is printed without indent and the URLs found on the page are printed with one indentation (spaces or tab).

Only take URLs from `<a href>` tags and do not worry about handling all the different link formats (absolute, relative, anchor, etc.). To make things easier, you can just handle absolute URLs (starting with "http" and "https") if you want and ignore other formats.

Feel free to use publicly accessible libraries to fetch and parse HTML pages (Python Requests library, for example). It just needs to be clear how those libraries are obtained and used (see Submission Test and Build below).

Parallelizing Your Application

In addition to the basic crawler requirements above, your application should fetch URLs in parallel to speed up the crawl and avoid blocking the whole process on single slow page loads. We encourage you to make use of any managed thread pool functionality provided by the language you choose to simplify your implementation.

Submission Test and Build

You should submit your solution by sharing an archive of all the source code and files necessary to build and run your application with the person who sent you this challenge (via email, dropbox, github, etc). You do not need to (should not) submit an actual application binary.

We should be able to build and run your application with 1 or 2 simple commands on some unix-like system (preferably Linux or MacOS). Please document the command(s) to build and run your application in a README as part of the submission.

You should also include at least one test that can be run against your application to show it works as expected. Please also document the command to run your test(s) against the application.

Target time to complete this is 2-4 hours. Please let us know approximately how long you worked on this in your submission.

Your implementation should prioritize clean, readable, well-structured code over cleverness or complicated performance optimizations. Good luck! Feel free to email the recruiter you are working with if you have any questions.