Assignment 3

Indexing Wikipedia

In this assignment, you will use Perl and bash scripts to build a tool that you can use to see which Wikipedia pages mention different words.

You won't be indexing all of Wikipedia; that would take too long, fetching the pages would take up too much bandwidth, and your index files would take up too much space. Instead, you'll start at a given page and do a depth-first or breadth-first traversal to a limited depth.

Task 1. Building an index

Write a Perl script called `windex-build` that should be able to be executed like this:

`windex-build` *`NAME STARTURL EXCLUDEFILE [OPTIONS...]`*

There are three *required* parameters:

- *`NAME`* is the name of the index
- *`STARTURL`* is the Wikipedia URL where you would like to begin your index. For example, you might want to start at `https://en.wikipedia.org/wiki/Perl`.
- *`EXCLUDEFILE`* is a file containing words you *don't* want to index. For example, there's not much point in indexing words like `the` or and. Words should be listed one per line.

There are also two possible *optional* parameters:

- `maxdepth=`*`DEPTH`* tells `windex-build` how deeply it should traverse. The minimum possible value for *`DEPTH`* is 0, which tells it to only index the page at *`STARTURL`* and not follow any links at all.

  If a `maxdepth` option is not provided, `windex-build` should default to a depth of 3. In order to minimize the impact of this assignment on Wikipedia's servers, the largest allowable value for *`DEPTH`* is 5.

- `dir=`*`DIRECTORY`* tells `windex-build` to store its index file in the given directory. If this argument is not supplied, it should default to storing its index file in the current directory.

The index

Your script should build a file at *`DIRECTORY/NAME`* that contains each word that it found, followed by a comma, followed by a comma-separated list of the URLs of the pages that contained it. Each word should be on its own line. So one entry might look like:

```
scalar,https://en.wikipedia.org/wiki/Perl,https://en.wikipedia.org/wiki/
Variable_(computer_science)
```

Note that the line break is only there for readability. If this were a real index file, the only `n` character would be at the end of the entry.

Visiting web pages

There are many ways to fetch a web page into a Perl script. The simplest way is probably to use the `wget` command – this isn't a Perl command but it's a standard part of any Linux distribution. You can use it like this: `my @page = `wget $url`;`

This will fetch the entire text of the page into the variable `@page` – technically, it captures `wget`'s `stdout`.

If you like, you may use Perl modules from CPAN to achieve this. Many programmers on the internet say nice things about `LWP::Simple`, for example. You might also want to look into modules that can help you parse the HTML once you've got it. Perl was the original P in the LAMP stack – the Linux, Apache, MySQL, Perl/Python open source toolchain that has underpinned much of the Internet for many years – so there are many, many resources available to you that can make your job easier.

If you find yourself needing to download a package from CPAN to get your script to work, don't forget to include an instruction to do the same in your README file to ensure that your marker will be able to get your script to run.

For each web page you visit, you need to do the following things:

- **Identify the words.** You'll need to pick out the words in the body of the article. You can ignore images, punctuation, markup such as bolding and italics, and the text in the sidebar on the left. (You are free to define what a "word" is for the purposes of this exercise.)
- **Filter the words.** Check the words that you find against your list of words that should be ignored. Any words *not* on the list should be indexed.
- **Follow links.** Find the links in the body of the article. Visit any links to Wikipedia pages that you have not yet visited – ignore pages you've already indexed and links that point outside Wikipedia.

Avoiding repetition

Retrieving information from the server is likely to take up most of the running time for your index builder, so you need to minimize the amount you have to download. One obvious way to do this is to avoid retrieving pages more than once. Your program should maintain a list of visited URLs and check it in order to prevent reindexing pages.

Normalizing the URLs

Sometimes, one web page may be accessed by more than one URL. This makes it harder to avoid revisiting previously-visited pages, and may also mean that the list of URLs that your program produces will contain multiple references to the same page. You can prevent this by *normalizing* the URLs to make it easier to determine when duplicates arise.

A useful place to start looking for information about URL normalization is Wikipedia: `https://en.wikipedia.org/wiki/URL_normalization`. Not all of these techniques will be useful to you; but you should *at least* attempt to remove fragment identifiers (`https://en.wikipedia.org/wiki/Fragment_identifier`).

You can find more information about different ways that Wikipedia pages can be accessed here: `https://en.wikipedia.org/wiki/Help:URL`. Some of these methods lead to pages that you won't want to index, such as the History or Edit pages; these links should not be followed. Others represent genuine different URLs for the same page.

Wikipedia terms of use

Wikipedia's terms of use prohibit the following:

- "Engaging in automated uses of the site that are abusive or disruptive of the services and have not been approved by the Wikimedia community;
- "Disrupting the services by placing an undue burden on a Project website or the networks or servers connected with a Project website;

- "Disrupting the services by inundating any of the Project websites with communications or other traffic that suggests no serious intent to use the Project website for its stated purpose"

This suggests that our use of Wikipedia is fine *if and only if* we take care not to produce too much traffic. We definitely don't want to get Monash in trouble with Wikipedia, so you will need to take the following precautions:

- limit the maximum search depth to 5
- count how many pages you have visited in each run and exit the program if you visit more than 1000
- sleep for at least 0.2 of a second between page visits. You can use the `usleep` function from the module `Time::HiRes` to achieve this:

  ```
  use Time::HiRes "usleep";
  usleep($delay);
  ```

  where `$delay` is the desired time delay in microseconds. (Note that this is equivalent to a Python `import` statement along the lines of `from Time::HiRes import usleep`.)

It might be a good idea to limit the number of pages you visit even more stringently until you are certain that your program logic is right, especially if you are not familiar with this kind of traversal.

Task 2. Viewing the index

You should also write a bash script called `windex` that you can use to look up words in your index file. You should be able to call it like this:

`windex` *`NAME WORD`* `[`*`DIRECTORY`*`]`

There are two required parameters. As with `windex-build`, *texttt*`NAME` refers to the name of the index file you'd like to look in. *`WORD`* is the search term you'd like to look up. The optional parameter, `textitDIRECTORY`, has the same meaning it does for `windex-build`. Again, the script should default to the current directory if one is not provided.

When run, `windex` should display the URL of each page where the search term was found, one per line. If the term is not found, you should display an appropriate error message.

You should test that the two parameters are present and valid, and that *`DIRECTORY`* is also present and valid if it is used.

Your script may assume that it is being run on Linux, and that all of the usual GNU coreutils programs are present. If you need other software to be installed, you need to say so in the README.

Submission instructions

As well as your Perl and bash scripts, you must write a README that details how your marker should install and run your scripts, and any known bugs that they have.

Your Perl script, bash script, README, and any other files that your suite needs to run should be submitted on Moodle in a single .tar.gz (or, equivalently, .tgz) file no later than **9:00am on Friday 27th May**.

Marks breakdown

There are 20 total marks available for this assignment and they will be broken down as follows:

- building the index: **5 marks**
- normalizing URLs: **5 marks**
- `windex` functionality: **5 marks**
- Perl readability, commenting, style: **3 marks**
- Following the submission instructions: **2 marks**

Note that this assignment is worth **10% of your total mark** for FIT3042.