REPORT – Ivan Konstantinov s0787128

This report will begin with a description of the assumptions made for the implementation of the crawler, and then will describe the designed algorithm and the decisions that were made regarding its implementation.
To run the crawler, import *botcrawler.py* into a python shell (*import botcrawler*) and then run *botcrawler.start()*. When the crawler finishes, it will print statistical results about the pages and URLs that were encountered and processed.

1. GENERAL ASSUMPTIONS

- If a web page does not contain a text section between a pair of <!-- CONTENT --> tags, it will be ignored without further processing.
- Any pages that were put into into the frontier queue, but were not found on the server, are not considered as processed. However, if a crawler requests a page that cannot be found on the server, it will still obey the "Request-rate" directive, as specified in the "robots.txt" and will not attempt to immediately retrieve a new page.
- Local URLs are assumed to be of the following types: without a root path ("123456.html") or with a root path which is within the Informatics Network ("http://(www.)[..].inf.ed.ac.uk/[..]"), including the IR network ("http://ir.inf.ed.ac.uk").
- External URLs are assumed to be any URL that points outside the Informatics Network, as in "http://www.google.com".
- Whitespace characters can be placed in arbitrary number and in arbitrary places within an anchor (<a>) tag, excluding within the URL string. Moreover, uppercase tags and URLs are also assumed as valid. Thus, the following examples are valid examples of an anchor tag containing a URL: "<   a href="EXAMPLE.HTML">", "< a   href   =   "example.html"   >", "<a href=example.html>", "<a href="example.html"style="some-style">.
- The "Request-rate" and "Crawl-delay" directives are assumed to be equivalent. That is, a **1/20** value of "Request-rate" directive is equivalent to a value of **20** of a "Crawl-delay" directive (access a page every 20 seconds).
- Pages are assigned priorities corresponding to their numerical name; pages with larger numerical names are given higher priority

2. IMPLEMENTATION

When the *start()* function is called, the "*robots.txt*" file is requested from the server to determine any "Request-rate" or "Crawl-delay" directives that may constrain the speed of the crawler. To achieve this, a modified *URLopener* module is used to open the *robots.txt* file (the only modification to the original *URLopener* is the version name, which is changed to "TTS", to reflect the name of the crawler). After the delay value has been determined, the crawler takes the frontier priority queue and retrieves the page with the highest priority (in this case, this is the seed URL). Each time the crawler needs to retrieve a new page from the server, it checks the elapsed time between the last page request and the current time: if it is below the delay constraint (as defined by the "Request-rate" or "Crawl-delay" directives), the crawler suspends for a specific amount of time (*delay – (current_time – last_time)*), before fetching the next page.

When the page is fetched from the server, its contents, defined between the <!-- CONTENT --> and <!-- /CONTENT --> tags, are extracted using the following regular expression: **"<!--(\s)*CONTENT(\s)*-->[\s]*.*<!--(\s)*/CONTENT(\s)*-->".**
Then, all URLs are extracted by looking at all anchor (<*a*>) tags, that contain the *href* declaration (those that do not contain it are assumed not to contain any URLs). The crawler assumes that the *href* token is followed by an "equals" (=) symbol, a single opening quote (' or "), but not necessarily, the URL string (with or without a root path), and then a single closing quote (if an opening one has been seen). Three regular expressions are used to retrieve the URL:

<\s*a[^>]*href[^>]*\.[^>]* will retrieve the whole anchor tag, starting at *<a*
href(\s)*=(\s|\'|\")*[^' '|^\'|^\"]* will retrieve the *href = […].[...]* part of the extracted anchor tag
(?<=[\'|\"]).*\.[^\"|^\']* or (?<==).*\.[^' '|^\'|^\"]* will retrieve the actual URL (here, two slightly different regular expressions are used depending on whether quotes are used or not)

After retrieving all URLs from a page, they are then checked for their root path in order to determine if they are local or external. Local URLs are those that are in the Informatics network (http://inf.ed.ac.uk), or those that do not have a root path (such as "012313.html"), while all other URLs are externals and are thus ignored. To determine if the specific URL contains a root path, it is checked using the regular expression: **(\s)*http(s)?://** . Here, the "s" in *https* is considered for generality: although the crawler may not encounter any pages that need to be retrieved using the *Secure HTTP* protocol in this case, it is good practice to allow it to handle such pages should such be created in the future. To determine if the URL is local, the following regular expression is used: **(www)?[a-z0-9]*.inf.ed.ac.uk/**
All local URLs are then checked against a list containing all already processed URLs, and those that have been seen before (already contained in the list) are discarded and not queued. Two approaches were considered for implementing the list: a simple *[]* list, or a dictionary (*{}*). The motivation for the latter was that dictionary indexing would be much faster if the URLs were used as keys: in this case, it won't be necessary to go through the whole list searching for an entry that matches the specific URLs, but instead will try to access the entry that has the URL as key and will return *false* if no such key has been found. Running the crawler with both approaches showed that using a dictionary to store and keep track of already seen URLs is more efficient, if only slightly for the given task, in terms of execution time: while the upper execution time bound using the list approach was estimated at around 839 seconds, the upper bound using a dictionary was estimated at 817 seconds.
All distinct URLs are then checked against the *robots.txt* file to determine if the crawler can access them. Those that are not "crawlable" are also discarded. Queuing is done using a priority queue, as supplied by Python's *heapq* module. However, by default a priority queue in Python will assign higher priority to smaller numbers. For example, if two pages, *01234.html* and *05678.html*, are queued with priorities 1234 and 5678, respectively, *01234.html* will be retrieved first. Therefore, to the priorities will need to be reversed in order to achieve the wanted approach (as defined in the **Assumptions** section) by converting them to negative values. Thus, *05678.html* will have priority -5678 and will be retrieved first.
The following packages were used: **robotparser**, **urrlib**,**re**, **heapq**, **time, TTSOpener** (modified *URLopener)*