Text Technologies - Practical 1 - Web Crawler - Report
Ruaridh Thomson - s0786036

Designing the crawler.
The crawler is based on the given design:

- frontier: priority queue of all pages to be crawled
- initialised with a seed site
- crawler fetches the links from the seed and uses these as further seeds

At its core, given a URL the crawler is able to fetch the contents of the URL, read the HTML source from the returned data, and parse the source for further url links. On top of this the crawler manages all returned urls and 'polices' the urls to ensure they comply with given restrictions, e.g. robots.txt. A parser was created to handle fetching and parsing URLs, while a separate crawler object managers the returned URLs and provides the correct seed for the parser.

The crawler maintains three lists: links seen by the crawler, links visited, and a queue of seed links (frontier). The frontier is managed automatically as a heap (using heapq), keeping the link with the lowest number at frontier[0]. The next seed to be crawled is the last link in the frontier list. Once a link is seen it is added to a list of seen links, if it is not already present in the list. Adding a link to the frontier queue twice is prevented by checking if the link has been seen or not; it is discarded if it has. Though not necessary for crawling, the visited list gives the links fetched and parsed by the crawler — if there are no errors both the seen and visited lists will be the same. Initially the parser returned all links found in the content of a url. However, a significant speed improvement was made by checking if a link had already been seen while parsing, saving processing links we do not want to visit more than once.

To parse HTML content, after the html source has been cropped to the <!— CONTENT —> tags, the crawler uses the BeautifulSoup parsing framework to extract all <a href...> tags — removing the need to manually extract them. Normally some urls parsed need to be escaped (e.g., replacing '&' with '&'). However, due to the known environment this step has been omitted, though it would be done while parsing the urls from page content.

The crawler is designed to crawl in one pass, therefore pages that return 404 errors are not added back into the queue and are assumed to be missing. The crawler therefore has no re-visit policy. The crawler is designed to obey robots.txt by checking all links returned by the parser using the robotparser library; if the link is forbidden by robots.txt it is not visited and is discarded. A further restriction is discarding any parsed links that fall outside of the initial seed's network location (in this case: ir.inf.ed.ac.uk), and although there were no links found to point outside of this location the step performing this check is left in to demonstrate when it is checked. Furthermore, because the crawler uses a priority queue based on page numbers, it does not traverse pages in a breadth/depth-first manner.

To avoid complexity, the crawler is run on a single thread. If the crawler were to access multiple domains, considerable time could be saved through multi-threading; though this is not the case here and multi-threading was omitted.

References:
http://docs.python.org/library/urllib2.html
http://docs.python.org/library/robotparser.html
http://docs.python.org/library/heapq.html
http://docs.python.org/library/cgi.html (For url escaping)
http://www.crummy.com/software/BeautifulSoup

Restrictions:
Pages are numbered and the priority is given to pages with a larger number.
The crawler must only crawl urls parsed from <a ...> tags contained within the <!— CONTENT —> of a page.
The crawler must remain in the University domain.

Libraries:

robotparser	-	For robots.txt access and checking
urllib2	-	For accessing URL content
urlparse	-	Managing URLs
heapq	-	For maintaining frontier priority queue
BeautifulSoup	-	HTML parsing