Basic Idea

- Start crawling from the first 10 URLs returned by a Google query.
- A priority queue is used to keep track of the new URLs found and determine which URL the crawler should crawl next.
- A limited number of hyperlinks is obtained from each page, and a priority is given to each hyperlink. The priority is a random number between 0 and N - 1, where N is the limit for the number of hyperlinks that we keep from each page. At first, each hyperlink is given a random priority from 0 to N-1, and no hyperlink from the same page is assigned the same priority. After each hyperlink is assigned a priority, hyperlinks from the same domain as its parent URL or from the same domain as another hyperlink that comes before it will be assigned a priority of N, which is the lowest priority possible. This ensures that only a few URLs from the same domain will be at the front of the priority queue.
- A page is chosen to be sampled randomly. When a page is crawled, it will be chosen as one of the samples with a probability of r. (In my runs, I used r= ½ and r = 1/3).
- A dictionary is used to keep track of URLs that have been traversed.

Limitations and problem

- The crawler can get stuck in domains where there are few outgoing links to other domains. During one of the runs, the crawler was stuck at different Wikipedia domains.
- The probability of sampling a page was set too high. In different runs, I tried to use ½ and ⅓ as the probability of sampling a page. Since many of the pages crawled were related to each other, this probability might be too high to give actual good sampling statistics.
- The crawler might only be sampling a tiny part of the web within similar/related domains since this approach is more similar to BFS. A better approach is to go deeper by doing something like a DFS and jumping to a completely different domain after some time.

Major Functions

- fetch_page(url):
  Send an HTTP get request for a URL. Check the request's content type to ensure that it's of type text/html. If it is, return the response to the get request. Otherwise, return none.

- parse_response(response):
  Parse the content of the HTTP response using the lxml Python library and return an HTML tree.

- get_attribute(tree, attribute):
  Return the intended attribute from the HTML tree.

The target attribute for this task is the language of the page. The function first tries to find the ‘lang’ tag from the HTML tree. If the ‘lang’ tag is not found, it uses the method detect from the language-detection library in Python to determine the language used in the content of the HTML.

- get_links(tree, num_links):
  Return a list of hyperlinks from the parsed HTML tree.
  num_links sets a limit to the number of hyperlinks the function will return.
  The function first gets all the hyperlinks found in the HTML tree by searching through the <a> tag. If the number of links found is more than num_links, the function shuffles the hyperlinks and saves the first num_links links. After that, it checks whether the links end with an extension in the bad list, ex. Jpeg, pdf, … The function removes all the links that end with an extension in the bad list, and returns the rest of the links saved.
  ***Limitation: not all hyperlinks can be found (ex. URLs that are embedded in the text will not be found)

- cal_link_priority(url, base_url, num_links, priority, added_domain):
  Returns a priority number for the URL
  If the URL is from the same domain as the base_url(parent’s URL) or the URL is from the same domain as a URL that has already been given a priority number, the URL will be given a priority number that is of the lowest priority, which is num_links in this case. Otherwise, the priority listed in the parameters will be returned. (This priority is a random number from 0 to num_links - 1 .)

- get_base_url(url, tree):
  Returns the base_url that can be used to get the absolute URL from a relative URL.
  This function first tries to find the <base> and <base_url> tags from the HTML tree. If a proper URL exists in one of these tags, the function will return the URL directly. Otherwise, it will get the scheme and net location of the URL given in the parameters and concatenate them to form a base URL.

- normalize_url(link, base_url):
  Returns the normalized link, which is an absolute URL with the scheme and net location of the URL all in lowercase.
  The link is normalized by making the scheme and netloc of the link lowercase, and joining it will the base_url if it is a relative link.

- get_robot_text(url):
  Returns the text from the robots.txt of the given URL in a list form, where each element of the list is a line in the text.

- allowed_to_fetch(url, robot_text):
  Returns True if robot_text is none or if the crawler is allowed to access the given URL.

- shoud_sample():
  Returns True with a probability of 1 / 3 and False with a probability of 2 / 3.

- get_ggl_results(query, num_res):
  Returns the first num_res URLs from a Google query.

Popularity Estimates

- Chinese: 1.33%
- Spanish: 1.00%
- Polish: 0.78%