Homework 11: Several Hundred Degrees of Separation

36-350, Fall 2013

Due at 11:59 pm on TUESDAY, 26 November 2013

We continue building our own version of page-rank, based on actually taking a random walk across the Web. Make sure that you have read and understood the solutions to the last assignment, and that your version of R is up to date.

Previously, we saw functions which downloaded a Web page, found all the hyperlinks, extracted the links to absolute URLs, and chose a random link to follow. We also had a final over-all function, `surf.to.mk2`, which measured how many random steps it took to reach a target website. A websites page-rank is the probability of reaching it by a random walk over the web: the higher the rank, the more short paths there are to the website, through sites which are themselves highly visited by a random walk. The probability of a random walk visiting a site is the inverse of the expected number of steps it takes the walk to reach the site.

The random walker or web-surfer we built only deals with absolute URLs, beginning `http://`. Many web-pages will have lots of relative URLs, however, which point to other parts of the same website. These are very important for actually getting the over-all structure of the web graph. (If nothing else, those pages may contain many links to other websites.) So we really ought to try to handle the relative URLs.

Relative URLs are interpreted in two slightly different ways, depending on the ending of the current URL.
(i) If the current URL ends with `/`, the relative URL is just appended to the end of the current URL. Thus if we are at

`http://www.stat.cmu.edu/`

and see the URL `privacy/`, the interpretation is

`http://www.stat.cmu.edu/privacy`

(ii) If the current URL doesnt end with `/`, we remove everything after the final `/`, and then append the new relative URL[1]. If were at

[1]This ignores some subtle points about maintaining "canonical" URLs: if the relative URL begins `/`, we should really truncate that, and if it begins `../`, we should truncate one level of the directory hierarchy in the current URL. For this assignment, let the Web server handle these things.

`http://www.stat.cmu.edu/~cshalizi/statcomp/13/index.html`

and we see `lectures/20/lecture-20.pdf`, we go to

`http://www.stat.cmu.edu/~cshalizi/statcomp/13/lectures/20/lecture-20.pdf`

If were at

`http://www.youtube.com/watch?feature=player_embedded&v=zoRfgeiRD_c`

and we see the URL

`/watch?v=tz1R7nCdCbM&amp;feature=fvwrel`

then we go to

`http://www.youtube.com//watch?v=tz1R7nCdCbM&amp;feature=fvwrel`

1. (10) Write a function which takes in a vector of link anchors (such as that returned by the `raise.anchors function` from lab), and returns all the ones which do *not* contain the string `"http://`. These are presumably relative URLs. Check that it works by comparing its output on `raise.anchors(statshome)` with what you expect to see from manually inspecting the latter.

2. (a) (20) Write a function which takes the current URL (one character string) and a set of relative URLs (a vector of character strings), and returns the appropriate vector of "absolutized" URLs. Make sure it works on the examples given above. For full credit, do not use any loops. (Hint: `paste`.)

   (b) (5) What does your function give when applied to the relative URLs found in problem 1? Do these make sense?

3. Modify `surf.to.mk2` so that when it comes to a URL, it randomly chooses from among both the absolute and the relative URLs, but otherwise works as before. Call the new function `surf.to.mk3`. Check that the examples from the last assignment will still run, and that they at least sometimes follow relative URLs.

4. (10) What fraction of paths started at `http://crookedtimber.org` reach `amazon.com` within 300 steps? Reach `powells.com`? Reach `sourceforge.net`?

5. (5) What are the average path-lengths of the successful paths to those three sites? The standard deviations of the successful path-lengths?

6. (10) What are the probabilities of reaching `amazon.com`, `powells.com` and `sourceforge.net` within 300 steps, starting from `http://myblog.webbish6.com`? What are the means and standard deviations of the lengths of the successful paths?

7. (10) Write a function which takes in an origin URL, a target, a maximum number of random-walk steps and a number of random walks, and returns the fraction of random walks which succeeded, the mean length of the successful walks, and the standard deviation of successful walks. (That is, it should encapsulate what you did in problems 4–6.) Should it match your previous results if you re-run them?

IMPORTANT NOTE: If you were going to do this on any kind of scale, it would be vital to follow the Web Robots exclusion protocol (see, e.g., robotstxt.org). Failing to do so is not just extremely rude, it is also an excellent way of getting yourself in trouble with people who command serious resources online.