Programming project: Google PageRank

Victor Eijkhout

2020
Abstract

The Google search engine uses an algorithm called ‘Pagerank’ to decide on the importance of web pages, that is, how high to display them in search results. This algorithm, in its original form, can be interpreted in two different ways. Firstly, it emulates the behavior of a computer user clicking on links. Secondly, it solves a mathematical linear algebra problem.

In this project, the student first builds up a simulated internet, and models the behavior of a user randomly clicking on links. Secondly, an implementation is made that is closer to linear algebra concept. Both models are used to explore the Pagerank algorithm.

This project can be done by one or two undergraduate students, or AP high schoolers at the end of a first or semester programming course.

There is opportunity for learning some non-standard mathematics.
<table>
<thead>
<tr>
<th>Summary</th>
<th>Explore the Pagerank algorithm for ranking web search results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topics</td>
<td>Pointers, arrays, matrices, classes</td>
</tr>
<tr>
<td></td>
<td>There is a possibility of exploring sparse matrices</td>
</tr>
<tr>
<td>Audience</td>
<td>Undergraduate or AP high school</td>
</tr>
<tr>
<td>Difficulty</td>
<td>Moderate to high</td>
</tr>
<tr>
<td>Strengths</td>
<td>Explores a phenomenon that is in everyone’s experience</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>The two parts (pointer-based, matrix-based) are independent of each other.</td>
</tr>
<tr>
<td>Dependencies</td>
<td>No dependencies.</td>
</tr>
<tr>
<td>Variants</td>
<td>The matrix approach can be extended to sparse matrices.</td>
</tr>
</tbody>
</table>
Chapter 1

Google PageRank

1.1 Basic ideas

We are going to simulate the Internet. In particular, we are going to simulate the Pagerank algorithm by which Google determines the importance of web pages.

Let’s start with some basic classes:

- A Page contains some information such as its title and a global numbering in Google’s datacenter. It also contains a collection of links.
- We represent a link with a pointer to a Page. Conceivably we could have a Link class, containing further information such as probability of being clicked, or number of times clicked, but for now a pointer will do.
- Ultimately we want to have a class Web which contains a number of pages and their links. The web object will ultimately also contain information such as relative importance of the pages.

This application is a natural one for using pointers. When you click on a link on a web page you go from looking at one page in your browser to looking at another. You could implement this by having a pointer to a page, and clicking updates the value of this pointer.

**Exercise 1.1.** Make a class Page which initially just contains the name of the page. Write a method to display the page. Since we will be using pointers quite a bit, let this be the intended code for testing:

```cpp
auto homepage = make_shared<Page>("My Home Page");
cout << "Homepage has no links yet:" << endl;
cout << homepage->as_string() << endl;
```

Next, add links to the page. A link is a pointer to another page, and since there can be any number of them, you will need a vector of them. Write a method click that follows the link. Intended code:

```cpp
auto utexas = make_shared<Page>("University Home Page");
homepage->add_link(utexas);
auto searchpage = make_shared<Page>("google");
homepage->add_link(searchpage);
cout << homepage->as_string() << endl;
```
Exercise 1.2. Add some more links to your homepage. Write a method `random_click` for the `Page` class. Intended code:

```
for (int iclick=0; iclick<20; iclick++) {
    auto newpage = homepage->random_click();
    cout << "To: " << newpage->as_string() << endl;
}
```

How do you handle the case of a page without links?

### 1.2 Clicking around

Exercise 1.3. Now make a class `Web` which foremost contains a bunch (technically: a vector) of pages. Or rather: of pointers to pages. Since we don’t want to build a whole internet by hand, let’s have a method `create_random_links` which makes a random number of links to random pages. Intended code:

```
Web internet(netsize);
internet.create_random_links(avglinks);
```

Now we can start our simulation. Write a method `Web::random_walk` that takes a page, and the length of the walk, and simulates the result of randomly clicking that many times on the current page. (Current page. Not the starting page.)

Let’s start working towards PageRank. First we see if there are pages that are more popular than others. You can do that by starting a random walk once on each page. Or maybe a couple of times.

Exercise 1.4. Apart from the size of your internet, what other design parameters are there for your tests? Can you give a back-of-the-envelope estimation of their effect?

Exercise 1.5. Your first simulation is to start on each page a number of times, and counts where that lands you. Intended code:

```
vector<int> landing_counts(internet.number_of_pages(),0);
for (auto page : internet.all_pages()) {
    for (int iwalk=0; iwalk<5; iwalk++) {
        auto endpage = internet.random_walk(page,2*avglinks, tracing);
        landing_counts.at(endpage->global_ID())++;
    }
}
```

Display the results and analyze. You may find that you finish on certain pages too many times. What’s happening? Fix that.
1.3 Graph algorithms

There are many algorithms that rely on gradually traversing the web. For instance, any graph can be connected. You test that by

- Take an arbitrary vertex \( v \). Make a ‘reachable set’ \( R \leftarrow \{ v \} \).
- Now see where you can get from your reachable set:
  \[ \forall v \in V \forall w \text{ neighbour of } v : R \leftarrow R \cup \{ w \} \]
- Repeat the previous step until \( R \) does not change anymore.

After this algorithm concludes, is \( R \) equal to your set of vertices? If so, your graph is called (fully) connected. If not, your graph has multiple connected components.

**Exercise 1.6.** Code the above algorithm, keeping track of how many steps it takes to reach each vertex \( w \). This is the Single Source Shortest Path algorithm (for unweighted graphs).

The diameter is defined as the maximal shortest path. Code this.

1.4 Page ranking

The Pagerank algorithm now asks, if you keep clicking randomly, what is the distribution of how likely you are to wind up on a certain page. The way we calculate that is with a probability distribution: we assign a probability to each page so that the sum of all probabilities is one. We start with a random distribution:

**Code:**

```cpp
ProbabilityDistribution random_state(internet.number_of_pages());
random_state.set_random();
cout << "Initial distribution: " << random_state.as_string() << endl;
```

**Output**

```
[pdfsetup: Initial distribution: 0:0.00, 1:0.02, 2:0.07, 3:0.05, 4:0.06, 5:0.08, 6:0.04, 7:0.04, 8:0.04, 9:0.01, 10:0.07, 11:0.05, 12:0.01, 13:0.04, 14:0.08, 15:0.06, 16:0.10, 17:0.06, 18:0.11, 19:0.01
```

**Exercise 1.7.** Implement a class `ProbabilityDistribution`, which stores a vector of floating point numbers. Write methods for:

- accessing a specific element,
- setting the whole distribution to random, and
- normalizing so that the sum of the probabilities is 1.

- a method rendering the distribution as string could be useful too.

Next we need a method that given a probability distribution, gives you the new distribution corresponding to performing a single click.

**Exercise 1.8.** Write the method

```cpp
ProbabilityDistribution Web::globalclick ( ProbabilityDistribution currentstate );
```
Test it by

- start with a distribution that is nonzero in exactly one page;
- print the new distribution corresponding to one click;
- do this for several pages and inspect the result visually.

Then start with a random distribution and run a couple of iterations. How fast does the process converge? Compare the result to the random walk exercise above.

**Exercise 1.9.** In the random walk exercise you had to deal with the fact that some pages have no outgoing links. In that case you transitioned to a random page. That mechanism is lacking in the `globalclick` method. Figure out a way to incorporate this.

Let’s simulate some simple ‘search engine optimization’ trick.

**Exercise 1.10.** Add a page that you will artificially make look important: add a number of pages (for instance four times the average number of links) that all link to this page, but no one links to them. (Because of the random clicking they will still sometimes be reached.)

Compute the rank of the artificially hyped page. Did you manage to trick Google into ranking this page high?

### 1.5 Graphs and linear algebra

The probability distribution is essentially a vector. You can also represent the web as a matrix $W$ with $w_{ij} = 1$ if page $i$ links to page $j$. How can you interpret the `globalclick` method in these terms?

**Exercise 1.11.** Add the matrix representation of the `Web` object and reimplement the `globalclick` method. Test for correctness.

Do a timing comparison.

The iteration you did above to find a stable probability distribution corresponds to the ‘power method’ in linear algebra. Look up the Perron-Frobenius theory and see what it implies for page ranking.
Chapter 2

Style guide for project submissions

*The purpose of computing is insight, not numbers. (Richard Hamming)*

Your project writeup is at least as important as the code. Here are some common-sense guidelines for a good writeup. However, not all parts may apply to your project. Use your good judgement.

**Style** First of all, observe correct spelling and grammar. Use full sentences.

**Completeness** Your writeup needs to have the same elements as a good paper:
- Title and author, including EID.
- A one-paragraph abstract.
- A bibliography at the end.

**Introduction** The reader of your document need not be familiar with the project description, or even the problem it addresses. Indicate what the problem is, give theoretical background if appropriate, possibly sketch a historic background, and describe in global terms how you set out to solve the problem, as well as your findings.

**Code** Your report should describe in a global manner the algorithms you developed, and you should include relevant code snippets. If you want to include full listings, relegate that to an appendix: code snippets should only be used to illustrate especially salient points.

Do not use screen shots of your code: at the very least use a `verbatim` environment, but using the `listings` package (used in this book) is very much recommended.

**Results and discussion** Present tables and/or graphs when appropriate, but also include verbiage to explain what conclusions can be drawn from them.

You can also discuss possible extensions of your work to cases not covered.

**Summary** Summarize your work and findings.
2. Style guide for project submissions
Chapter 3

Bibliography