So you want to build a search engine?

Benji Altman

May 6, 2018

Contents

Abstract

There are many different types of search engines, the most complicated of which are where you do not initially have knowledge of what is being searched and that information may change. In order to tackle this we need some way of discovering possible results, matching these results to a query and ranking possible results.

1 Who this paper is for

This paper assumes the reader has no special knowledge of Mathematics, Graph theory, or the workings of the Internet. The paper does however use very common terminology from Mathematics and Graph theory without providing definitions. If a reader is unfamiliar with certain terms they are encouraged to look them up as otherwise the paper will spend to much time describing mathematical concepts that are outside the scope of this paper.

2 Introduction

So you want to build a search engine? Now there are a couple different types of search engines and not all search engines need to work the same, however there are two steps that are absolutely necessary for any searching to make sense. First you are going to need to have some set A to search through and second you will need a search function $s : Q \to \mathcal{P}(A)$[1] where Q is the set of all possible search queries. If we step away from mathematical formalities, this makes sense, as we simply are finding a way to match a search query to set of possible results.

Now what one might notice is that none of this relates to ranking the results, and this is obviously an important part of any modern search engine. The problem of ordering may, in fact, be the most interesting part of a search engine. This paper however will spend very little time talking about ordering a search result as Page Rank is already well understood and after that most of the complexity really has more to do with understanding the query which we will cover when talking about our search function.

[1] $\mathcal{P}(A)$ is the power set of A.

Now this paper will mostly focus on the discovery of set A and assumes and Internet like structure to A. That is to say that A is a large set with a directed graph structure to it. Additionally A is not static, edges in A's graph structure may change, members of A may be created or destroyed, as may the content of those members. Finally A is not initially known to us.

Now while I try to keep this paper in the abstract and I talk about A as if it's a set and does not have to be the Internet, I think it's important to talk about some of the tools that we have built to help us work with constrains the Internet put on us. This is because the Internet is so well known and what most people have experience searching. Leaving out practical information, while it may take away from the abstractness, it is the authors opinion that it is too important to leave out. In this vein we will be focusing on Google's architecture as it is so well known and information is readily available on it.[2]

3 The Internet

3.1 How it works

The Internet itself is a somewhat complex entity and a lot goes into making it work. For this we don't need to know everything but it would be good to have a rough idea of what happens when you go to a web page.

Lets say you open your web browser and go to the `www.google.com`. Now what happens is your computer sends a message out to `www.google.com` and a response is sent back to you. The web browser that you use (ie: Mozilla Firefox, Internet Explorer, Google Chrome, ...) will then display that message in a way that makes most sense.

The message that is sent to `https://www.google.com` ins't terribly complicated, however we aren't goint to deal with it's entire protocol, instead we will pretend that the message is simply the URL that you type in. Now you might see a somewhat more complicated URL then just `https://www.google.com`, maybe you'll see `https://www.google.com\mail` or even more complicated looking like

```
https://www.google.com/search?ei=k1TuWuoH0pCaBcvOuIgC\&q=hello+world\&oq=hello+
    world\&gs\_l=psy-ab.3..0j0i20i264k1l2j0l5j0i10k1j
    0.2636.3822.0.3932.11.8.0.0.0.0.252.676.2-3.3.0....0...1c.1.64.psy-ab
    ..8.3.676...46j0i67k1j0i46k1j0i131k1.0.2UY03D\_JQgE
```

we can think of this as simply sending that message to `www.google.com`, and them sending back a reply. Now you can see the actual text of the response that is sent back to you, however what is important to note is that this is simply a response. This means that `https://www.google.com` can send me any response it likes and it doesn't need to look like a web page and it does not need to be consistent. This means that there really ins't information on the Interent, just URLs and servers that give responses.

When we make a Internet search engine we are making some implicit assumptions about how websites like `https://www.google.com` are going to act. First we assume that most urls will respond with relitivily consistant responses. What is meant by this is that if I have a URL x that produces response r_0, the first time I made a request for it, then when I send another request to x and get response r_1, then r_1 should be similar in content to r_0.

Now in practice, as of writing this paper, most responses (at least for important web pages) tend to be fairly static, often being exactly the same for a length of time and not changing by a large amount when they do change. For example think of the website `https://www.wikipedia.com` that acts as a communally sourced encyclopedia. Now as anyone may make a change to any particular page on this site you might think it to be rather volitile, however these changes tend to be very minimal and wont completley change the content of the page. When we talk about crawling we will useing and working within this constraint that pages change, but only gradually.

For this section we have thought of web pages simply a responses to a URL. Now that we have justified that these responses are mostly consistent, we will think of them rather as pages.

[2]This is not fully accurate, there is some very good information on Google's inner workings that comes from before 2000, however much of of modern Google's inner workings are kept as trade secrets or simply not released.

3.2 Some terminology

In order for us to understand how a search engine like Google operates, there is some terminology pertaining the Internet that we must understand. Admittedly the definitions here are not rigorous, however the quick overview of some ideas is all that is needed for our discussion.

- **Website** - A website might be best though of as a server. Each website has a single computer[3] that deals with incoming requests and makes responses. A website is denoted by `www.google.com`, anything that is appended after that is still on the same website, that is that `www.google.com` is still sending the responses.
- **Web page** - A web page is a single page or URL on a web site. More precisely a web page is the response given to you by a website given a specific URL. In this we will think of the URL as being the name of the web page while the page itself being subject to change. That is to say if a page was to be moved from URL a to URL b we would simply see it as a's web page changing to not exist and b's web page changing to whatever a used to be.
- **URL** - The URL is the 'name' of a webpage and tells us how to get there. We have already seen some example URLs like `https://www.google.com`. We see URLs come in a a few flavors, we will break them into two catagories.
  - **absolute URL** - This a URL that has the whole name in it like the ones we have seen. This contains all the information we need to get to the specific page. These often look something like `www.domainName.topLevelDomain/optionalExtraStuff`.
  - **relative URL** - This a URL that works within a website. If I'm at `www.example.com/index` and I go to the URL `/home` there is an agreement that this link really refers to `www.example.com/home`.

  If which type of URL is not specified it should be assumed that we are referring to an absolute URL.
- **404 Error** - Sometimes web pages are removed. When we try to go to a URL that is not set to respond with anything, the website will respond with a 404 error. This becomes important as we may find that some pages that used to exist no longer do, and these must be handled in a meaningful way.

4 Discovery

You want to create a search engine that searches through some set A, however you do not have all of A, so you need some way to discover as much of A as you can. Now for this we are going to assume you already know some $S \subset A$ and that A has a directed graph structure to it that you may follow to discover new nodes. In order to do this we will need a graph searching algorithm. There are two basic algorithms available: Depth-First search and Breadth-First search.

A depth-first search starts with a single vertex v_0, it finds all children of v_0 and then repeats the process recursively on them one at a time. Notice that if v_1 is v_0's 'first' child then we will find each v_1's children, and then repeat on v_1's first child continuing in this fashion until there are no more children and we are able to work back up to v_0's second child. This is like searching by clicking the first link you find on a page until there is no links on some page and then going back to the last page and clicking it's second link. The obvious problem with this search is that it loop infinitely and to deal with this we make the slight adjustment that we never go to the same vertex twice.

A breadth-first search starts with a seed set S and finds the neighborhood of S. In this context we define the neighborhood of a set, S, to be all verticies $v \notin S$ for which there is some vertex $s \in S$ with an edge to v. In a breadth-first search, once we find the neighborhood of S we then repeat the process on the union of S with it's neighborhood and continue recursively until the neighborhood of S is empty, at this point we have discovered as much of our graph as possible given our seed set. This is like going to a page and clicking all of it's links and opening them in a new tab, then going through each tab and continually opening links in new tabs.

[3] In practice the single computer is actually multiple computers

The choice of which search to use is very situational and has much to do with the structure of the graph you are searching. Additionally with more information about the graph you may be able to do some more complicated algorithms that are more suited to your needs, or some combination between the above algorithms.

4.1 On the Internet

There are quite a few useful results that we already know about the structure of the Internet that will help us to explore it. The problem with the Internet however is that it is so large that no one could ever hope to discover all of it, so instead we must find some way of trying to get the most valuable parts of the Internet discovered.

The process we are talking about is often referred to as web crawling and a program that does this is called a web crawler or spider. Our first idea for building one might be as follows:

1. Start with some set of URLs S as our seed.
2. Enqueue each element of S to a queue Q.
3. While Q is not empty do the following:
   (a) Dequeue a URL from Q, let us call this URL u.
   (b) Make a request to URL u.
   (c) When a response is sent back, find all URLs (either absolute or relative) that one may find in that response, convert all relative URLs to absolute URLs. We will refer to this set of discovered URLs as D.
   - On a webpage there are certain places to look, for example an anchor tag may be a link to another page.
   - One may also use a regular expression to try and find things anywhere on the page that look like URLs. Even if our system is perfect sometimes pages go away so we should expect not all URLs to be valid.

   In practice we may want to use some or all of the information from the web page. This will be important in later sections, but for now just realize that we can keep track of the responses we get and deal with them how we choose.
   (d) For each URL $d \in D$, if $d \notin S$ then add d to S and enqueue it onto Q.

Now this web crawler works perfectly well, however it's going to be incredibly slow. The first step in optimizing this is always looking at what the slowest part is. Without even testing this program there is going to be a clear winner.