The most interesting graph in town!

“Information moving through cyberspace travels in tiny packets that hopscotch around the world.

This graph shows data from a two-week stretch in April 2005, Tracking packets from hub to hub and country to country.”

http://discovermagazine.com/2006/oct/map-internet-servers
The Web as a Graph

• What is the structure of the web?
  – A directed graph \( G = (N, E) \).
  – Nodes correspond to web pages, composed of text and hypertext.
  – Edges correspond to hyperlinks.

How do search engines do it?
Internet search

- Goal of a web search engine:
  - Crawl and index the web efficiently.
  - Produce satisfying search results to human queries.

- Main components required to achieve this:
  1. Web crawling -> exploring the graph
  2. Indexing -> storing nodes
  3. Searching -> finding a target node

Exploring the graph: Web crawling

- A program to explore the web in a systematic way.

- We know 3 methods for exploring a graph:
  - Breadth-first search, Depth-first search, Best-first search

- Similar ideas apply here, but there are complicating factors!
  - Large size of the graph
  - Content of nodes change over time
  - Nodes are added/removed, so structure of the graph changes also.
Web crawling strategies

- We need a **selection strategy**, to decide order in which pages are explored.

- We need a **re-visit strategy**, to decide when to re-visit nodes.

- Let’s do these two things separately.

Node selection strategy

- There are many, many nodes. Need to prioritize them somehow.

- **Key idea**: prioritize them in terms of **importance**.

- What makes a page important?
  - Intrinsic quality ("authority")
  - Popularity (# visits)
  - Connectivity (# links to it)

- Let’s say we get a number \( \text{Score}(n) \), that tells us the importance of webpage \( n \). (We’ll get back to how we calculate this later.)
A new algorithm for exploring a graph

- Assume you have a list of candidate nodes to explore.

- **Algorithm:**
  - Get a starting node from somewhere.
  - Add its neighbours to the list of candidate nodes.
  - Pick the candidate node with highest score.
  - Add its neighbours to the list of candidate nodes.
  - Continue until no more unexplored nodes.

- This is our old friend **Best-first-search.**

Re-visit strategy

- Once in a while, we need to re-visit nodes. How should we do this?

- **Two possible strategies:**
  - Re-visit all pages with the same frequency (e.g. once a month)
    - **Uniform strategy**
  - Re-visit more often pages that change more frequently.
    - **Proportional strategy**

- Which is best?
Choosing a re-visit strategy

- Surprisingly, the uniform strategy is best!
- Why?

![Graph showing the relationship between number of pages and changes in 32-day period.](image)

What next?

- Main components required to build a search engine:
  - Web crawling -> exploring the graph
  - Indexing -> storing nodes
  - Searching -> finding a target node
Web indexing

- Indexing is a method to store information collected during crawling.
- Need a BIG array to store web pages:

  What goes in this array?

  Array:

<table>
<thead>
<tr>
<th>document ID</th>
<th>length</th>
<th>URL</th>
<th>webpage (compressed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  ⇒ The Repository

Indexing all explored webpages

- Documents in the array are sorted by document ID.
  - Problem 1: Given a URL, how do we retrieve the right document?
  - Problem 2: Given a new document (with its URL), how do we pick its document ID?

- Could just use the order in which it was explored.
  - Easy to generate this document ID (just keep a counting variable).
  - Hard to retrieve later (need to search the whole array!)

- Instead: Use a special program that converts URL to document ID.
  - Easy to generate this document ID.
  - Same program is applied to address to retrieve document given its URL.

- Apply binary search to look through the array and find the correct row.
Indexing the words

- **Problem 3**: Given a set of keywords, how do we retrieve webpages with those words?
- Need another big array to store words: ⇒ The Lexicon

Array:

<table>
<thead>
<tr>
<th>word</th>
<th>documents that contain that word (by document ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>00110100 10100011 11100111  ...</td>
</tr>
</tbody>
</table>

What next?

- Main components required to build a search engine:
  - Web crawling -> exploring the graph
  - Indexing -> storing nodes
  - Searching -> finding a target node
- Goal: Return the webpages which contain the given query words.
Idea #1: Starting with something simple

- Web pages returned should contain the query words!

- Easy! Use the lexicon to find all the pages that contain the query words.

- Problem: Too many pages returned.

Idea #2: Need to prioritize!

- Prioritize pages in terms of importance (as we did for web crawling).

- How do we assess importance?
  - Webpage authors tend link to good pages.
  - Good sites (i.e. “authorities”) are cited by many other sites.
    - These should get high priority.
    - Prefer webpages with many “in” edges.
  - Good sites are cited by authorities.
    - These should also get high priority.
    - Prefer webpages with “in” links from authorities.
  - Some websites link to many others
    - These are generally less useful, and so should get lower priority.
    - Exclude webpages with many “out” edges.

- Need to incorporate all these ideas!
Calculating the “score” of a webpage

- Assume each website gets an importance score ⇒ its PageRank
- The PageRank is based on the actual graph structure, not on an actual query (this will come later).
  - Let $PR(n)$ be the PageRank of node $n$.
  - Let $C(n)$ be the number of links “out” of node $n$.
  - Let $w_1, w_2, ..., w_k$ be the pages that have “in” links to $n$.

$$PR(n) = \frac{PR(w_1)}{C(w_1)} + \frac{PR(w_2)}{C(w_2)} + ... + \frac{PR(w_k)}{C(w_k)}$$

- We have a system of $N$ linear equations with $N$ unknowns ($N = \#webpages$).
  - Use linear algebra (numerical approximation, not Gaussian elimination) to solve this system of equations.

Example of PageRank Calculation

http://en.wikipedia.org/wiki/PageRank
Using PageRank for web crawling

- Recall the Best-first-search algorithm:
  - Get a starting node from somewhere.
  - Add it's neighbours to the list of candidate nodes.
  - Pick the candidate node with highest score.
  - Add it's neighbours to the list of candidate nodes.
  - Continue until no more unexplored nodes.

- Use PageRank to order the list of candidate nodes.
  - Keep the list in sorted order at all times.
  - Every time you need to add a node, search the list of candidate nodes (e.g. using binary search) to find the spot in the list where to insert the new candidate webpage.

Using PageRank to answer queries

- Search the Lexicon for all webpages containing the query words.

- Using the Document ID, extract the content of these webpages from the Repository.

- Sort these pages in decreasing order of their PageRank.
Manipulating the PageRank

• Popular activity! Why?
  – Higher PageRank = more traffic!

• How?
  – Bribery
  – Collusion
  – Many different ways!

Manipulating PageRank by Bribing an Authority

• How do we do this?
  – Place a link to your webpage on an “authority” page.
  – However if the “authority” page has too many “out” links, it’s own PageRank is reduced, and it becomes less useful.
  – Pay an appropriate amount (based on the authority’s PageRank) to have a link from the authority to your webpage.
  – There’s a whole economy for buying and selling links.
Manipulating PageRank by Collusion

• What is collusion?
  
  A secret agreement between two or more parties for a fraudulent, illegal, or deceitful purpose.

• How does this concern PageRank?
  
  – A web designer creates many new webpages.
  
  – The designer then creates links between these webpages, so each has many “in” links.

A few more general comments

• Most ranking systems are vulnerable to manipulation.

• Users often have incentives to cheat.
  
  E.g. Study of eBay has shown that buyers are willing to pay an extra 8% to buy from a seller with a high reputation.

• Many search engine companies spend substantial resources trying to outsmart such people.
Take-home message

- Concepts to understand:
  - Web crawling and the best-first search algorithm
  - PageRank (the main ideas that come into the equation, not necessarily how to solve the system of equations.)
  - The Repository and the Lexicon: their role and their content

- Understand the role of basic searching and sorting algorithms in the web.

- **Midterm**: Tuesday Oct.18, in class.
  
  Closed book. No calculators/dictionaries. All material up to lecture 13.