SEARCHING

Unit 2

Introduction

- Let's say that you have a list of documents and you're interested in reading about those that are related to the phrase "World heritage sites".
- A bruit force and naïve solution would be to read each document and keep only those in which you can find the term "World heritage sites."
- You could even count how many times you found each of the words in your search term within each of the documents and sort them according to that count in descending order.
- **Information retrieval** (IR) or simply **searching** is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Introduction

- The naïve IR solution is full of problems. For example, as soon as you increase the number of documents, or their size, its performance will become unacceptable.
- There are sophisticated and robust libraries available that offer scalability and high performance. The most successful IR library in the Java programming language is **Lucene**. Lucene can help you solve the IR problem by indexing all your documents and letting you search through them at lightning speeds.
- State-of-the-art searching even goes well beyond indexing such as link analysis, user click analysis, and natural-language processing. These techniques strengthen the searching functionality.

Link Analysis

- The **PageRank algorithm** is a link analysis algorithm that makes google special. This algorithm was introduced in 1998, at the seventh international World Wide Web conference (WWW98), by Sergey Brin and Larry Page in a paper titled “The anatomy of a large-scale hypertextual Web search engine.”
- Around the same time, Jon Kleinberg at IBM had discovered another link analysis algorithm, the **Hypertext Induced Topic Search (HITS) algorithm**. HITS didn’t have the degree of commercial success that PageRank did. **HITS** is now part of the **Ask** search engine (www.Ask.com).

PageRank Algorithm

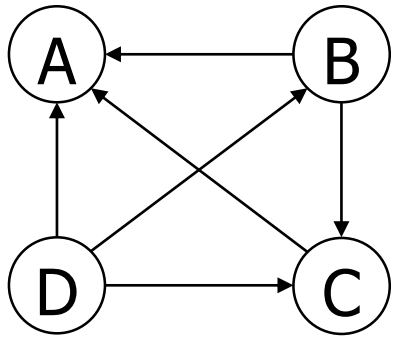
- The key idea of PageRank algorithm is to consider hyperlinks from one page to another as recommendations or endorsements.
- So, the more endorsement a page has the higher its importance should be. In other words, if a web page is pointed to by other, important pages, then it's also an important page.
- **Simplified Algorithm:**
  - The page rank value of any page *u* can be expressed as:

    $$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

    i.e., the PageRank value for a page *u* is dependent on the PageRank values for each page *v* contained in the set B_u (the set containing all pages linking to page *u*) divided by the number *L(v)* of links from page *v*.

PageRank Algorithm

- Links from a page to itself are ignored. Multiple outbound links from one page to another page are treated as a single link. PageRank is initialized to the same value for all pages. Hence, the initial value of page rank for each page is 1/n where n is the number of pages.
- The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.
- **Example:**
  - Assume a small universe of four web pages: **A**, **B**, **C**, and **D**. Suppose that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages.



PageRank Algorithm

- The initial page rank value for each page, PR(A) = PR(B) = PR(C) = PR(D) = 1/4 = 0.25.
- Thus, in the first iteration, since page B has two outbound links, it would transfer half of its existing value (0.125) to page A, Page C would transfer all of its existing value (0.25) to page A and since, page D has three outbound links, it would transfer one third of its existing value (approximately 0.083) to page A. Hence, the completion of this iteration, page A will have a PageRank of approximately 0.458. That is,

  PR(A) = PR(B)/L(B) + PR(C)/L(C) + PR(D)/L(D)
  = 0.25/2 + 0.25/1 + 0.25/3
  = 0.125 + 0.25 + 0.083 = 0.458
- Similarly, we can calculate page rank values of other remaining pages in the first iteration. Hence, the page rank value of page B will be 0.083, the page rank value of page C will be 0.208, and the page rank value of page D will be 0.
- We repeat this process until the difference between two successive page rank values is small enough. This smallness is also known as the **convergence criterion** or **threshold**.
- The difference between page rank values in two successive iterations is calculated by adding the absolute value of the difference between the new and the old PageRank values.

PageRank Algorithm

- **Damping Factor:**
  - Sometimes our surfer may get bored, or interrupted, and may jump to another page without following the linked structure of the web pages. To account for these arbitrary jumps, we use damping factor.
  - This parameter determines the amount of time that our surfer will surf by following the links versus jumping arbitrarily from one page to another page. The value of damping factor will be generally set around 0.85.
  - As the value of damping factor approaches to zero, it is assumed that our surfer is choosing his next destination at random, not on the basis of links. As the value of damping factor approaches to one, a surfer closely follows the links. Now, the page rank value of any page u can be expressed as:

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

where N is the number of pages and d is the damping factor.

PageRank Algorithm

- Alternatively, we can also use matrices to find page rank of pages with initial page rank vector, $R_0 = \begin{bmatrix} 1/N \\ 1/N \\ \vdots \\ 1/N \end{bmatrix}$ and page rank R_i is calculated as

$$R_i = \begin{bmatrix} (1-d)/N \\ (1-d)/N \\ \vdots \\ (1-d)/N \end{bmatrix} + d \begin{bmatrix} l(p_1,p_1) & l(p_1,p_2) & \dots & l(p_1,p_N) \\ l(p_2,p_1) & l(p_2,p_2) & \dots & l(p_2,p_N) \\ & & \vdots & \\ l(p_N,p_1) & l(p_N,p_2) & \dots & l(p_N,p_N) \end{bmatrix} R_{i-1}$$

where, the adjacency function $l(p_i, p_j)$ is the ratio between the number of links outbound from page p_j to p_i to the total number of outbound links of page p_j. The adjacency function is 0 if page p_j does not link to page p_j , and normalized such that, for each *j*

$$\sum_{i=1}^{N} l(p_i, p_j) = 1$$

HITS Algorithm

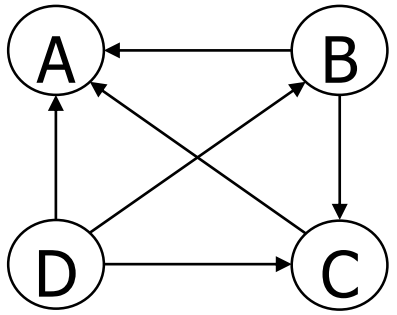
- In HITS algorithm, the first step is to retrieve the most relevant pages to the search query. This set is called the ***root set*** and can be obtained by taking the top pages returned by a text-based search algorithm. These are *potential authorities*.
- Find all pages that link to a page in the root set. These are *potential hubs*.
- A ***base set*** is generated by augmenting the root set with all the web pages that link to a page in the root set.
- Consider all edges (hyperlinks) connecting nodes in the base set. This graph is now used to find the important web pages.
- The difference between HITS and PageRank is that rather than taking the full network as in PageRak, we start with the subset of the network.

HITS Algorithm

- **Algorithm:** This algorithm performs series of iterations and in each iteration, it keeps track of ***authority score*** and ***hub score*** of every node. The initial score of authority and hub for each page is 1.
  1. Apply the **Authority Update Rule**: each node's *authority score* is the sum of *hub scores* of each node that points to it.
  2. Apply the **Hub Update Rule**: each node's *hub score* is the sum of *authority scores* of each node that it points to.
  3. Normalize the scores by dividing each *hub score* by the sum of all hub scores, and dividing each *authority score* by the sum of all authority scores. If we do not normalize, these scores will grow after every iteration.
  4. Repeat the process k times.

HITS Algorithm

- **Example:** Consider the graph with base set as given below.



Initially:

auth(A) = auth(B) = auth(C) = auth(D) = 1

hub(A) = hub(B) = hub(C) = hub(D) = 1

After first iteration:

After applying update rules for authority and hub:

auth(A) = 3, auth(B) = 1, auth(C) = 2, auth(D) = 0

hub(A) = 0, hub(B) = 2, hub(C) = 1, hub(D) = 3

After applying normalization:

auth(A) = 3/6, auth(B) = 1/6, auth(C) = 2/6, auth(D) = 0

hub(A) = 0, hub(B) = 2/6, hub(C) = 1/6, hub(D) = 3/6

HITS Algorithm

After second iteration:

After applying update rules for authority and hub:

auth(A) = 1, auth(B) = 3/6, auth(C) = 5/6, auth(D) = 0

hub(A) = 0, hub(B) = 5/6, hub(C) = 3/6, hub(D) = 1

After applying normalization:

auth(A) = 6/14, auth(B) = 3/14, auth(C) = 5/14, auth(D) = 0

hub(A) = 0, hub(B) = 2/14, hub(C) = 1/14, hub(D) = 3/14

After certain iterations, authority and hub scores for each node will converge to a unique value. And the node with highest authority will have highest priority.

HITS Algorithm

- We can also use matrices to find authority and hub scores. Let M be the adjacency matrix of the graph. Let h_i be vector of hub scores after i iterations and a_i be vector of authority score after i iterations. Then, we can calculate authority and hub score after iteration i using

$$a_i = M^T h_{i-1}$$

$$h_i = M a_{i-1}$$

As the values may increase beyond bounds, normalization is done such that maximum value is 1.

Exercise: Use both PageRank and HITS algorithm in the graph below up to iteration 2.