Eric Roberts
CS 106B

Handout #53
March 9, 2015

More Algorithms

More Algorithms for Trees and Graphs

Eric Roberts
CS 106B
March 9, 2015

Outline for Today

- The plan for today is to walk through some of my favorite tree and graph algorithms, partly to demystify the many real-world applications that make use of those algorithms and partly to emphasize the elegance and power of algorithmic thinking.
- These algorithms include:
  - Google's Page Rank algorithm for searching the web
  - The Directed Acyclic Word Graph format
  - The union-find algorithm for efficient spanning-tree calculation

Page Rank

The heart of the Google search engine is the page rank algorithm, which was described in a 1999 by Larry Page, Sergey Brin, Rajeev Motwani, and Terry Winograd.

The PageRank Citation Ranking:
Bringing Order to the Web

January 29, 1998

Abstract

The importance of a Webpage is an inherently subjective matter, which depends on the reader's interests, knowledge and attitudes. But there is still much that can be said objectively about the relative importance of Web pages. This paper describes PageRank, a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them.

We compare PageRank to an idealized random Websurfer. We show how to efficiently compute PageRank for large numbers of pages. And we show how to apply PageRank to search and to user navigation.

Page Rank Algorithm

The page rank algorithm gives each page a rating of its *importance*, which is a recursively defined measure whereby a page becomes important if other important pages link to it.

One way to think about page rank is to imagine a random surfer on the web, following links from page to page. The page rank of any page is roughly the probability that the random surfer will land on a particular page. Since more links go to the important pages, the surfer is more likely to end up there.

Markov Processes

A simple example of a Markov process is illustrated by this table, which shows the likelihood of a particular weather pattern for tomorrow given the weather for today.

If today is \ *Tomorrow will be*			
	0.85	0.10	0.05
	0.60	0.25	0.15
	0.40	0.40	0.20

Markov Processes

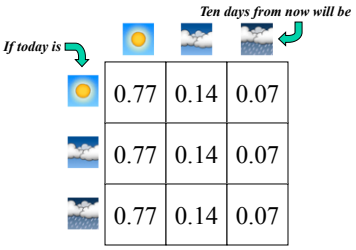
What, then, is the likely weather two days from now, given that you know what the weather looks like today?

If today is \ *The day after tomorrow will be*			
	0.81	0.13	0.06
	0.72	0.18	0.10
	0.66	0.22	0.12

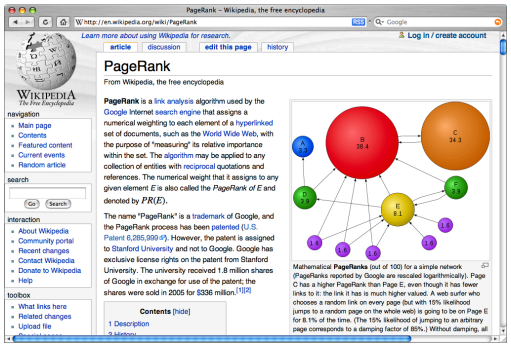
Markov Processes

What if you then repeat the process for ten days?

That far out, it doesn't matter what today's weather is.

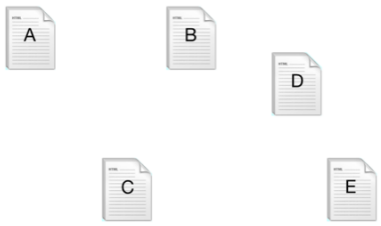


Google's Page Rank Algorithm



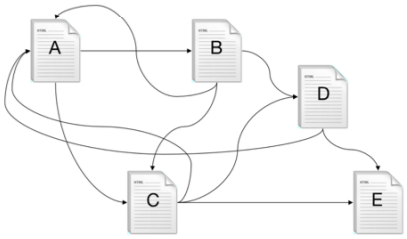
The Page Rank Algorithm

1. Start with a set of pages.



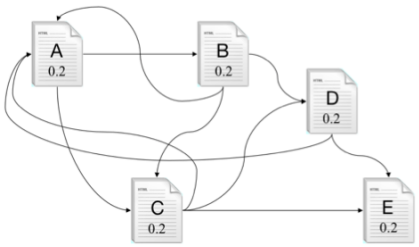
The Page Rank Algorithm

2. Crawl the web to determine the link structure.



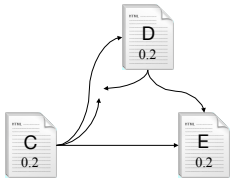
The Page Rank Algorithm

3. Assign each page an initial rank of 1 / *N*.



The Page Rank Algorithm

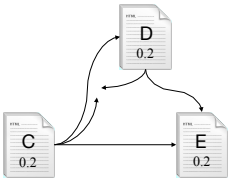
4. Successively update the rank of each page by adding up the weight of every page that links to it divided by the number of links emanating from the referring page.
   - In the current example, page E has two incoming links, one from page C and one from page D.
   - Page C contributes 1/3 of its current page rank to page E because E is one of three links from page C. Similarly, page D offers 1/2 of its rank to E.
   - The new page rank for E is



$$PR(E) = \frac{PR(C)}{3} + \frac{PR(D)}{2} = \frac{0.2}{3} + \frac{0.2}{2} \approx 0.17$$

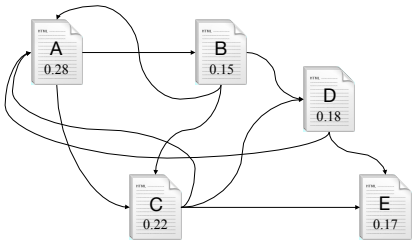
The Page Rank Algorithm

5. If a page (such as E in the current example) has no outward links, redistribute its rank equally among the other pages in the graph.
   - In this graph, 1/4 of E's page rank is distributed to pages A, B, C, and D.
   - The idea behind this model is that users will keep searching if they reach a dead end.



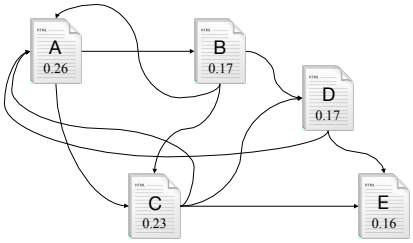
The Page Rank Algorithm

7. Apply this redistribution to every page in the graph.



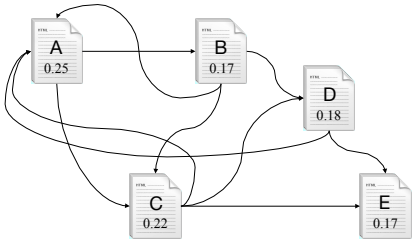
The Page Rank Algorithm

8. Repeat this process until the page ranks stabilize.

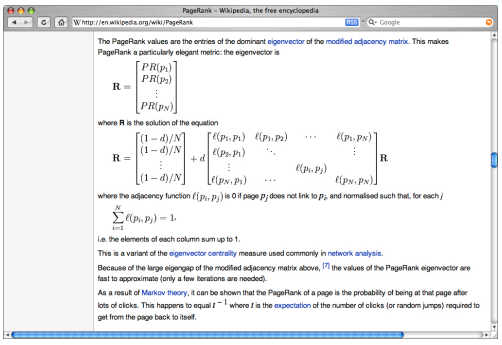


The Page Rank Algorithm

9. In practice, the Page Rank algorithm adds a ***damping factor*** at each stage to model the fact that users stop searching.

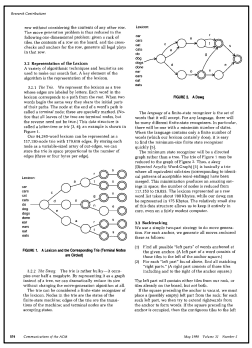


Page Rank Relies on Mathematics



Directed Acyclic Word Graphs

- Although the current version of the `Lexicon` class uses a trie as its underlying representation, it is possible to implement a more space-efficient version using a ***Directed Acyclic Word Graph*** or DAWG, which was first described in a 1988 paper by Appel and Jacobson.
- The DAWG structure is based on a much older data structure called a ***trie***, developed by Ed Fredkin in 1960. (The trie data structure is described in the text on page 490.)



The Trie Data Structure

- The trie representation of a word list uses a tree in which each arc is labeled with a letter of the alphabet. In a trie, the words themselves are represented implicitly as paths to a node.

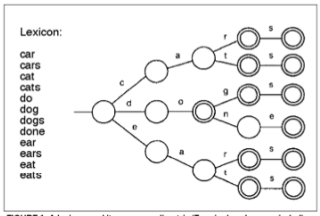


FIGURE 1. A lexicon and its correponding trie (Terminal nodes are circled)

Going to the DAWGs

- The new insight in the DAWG is that you can combine nodes that represent common endings.

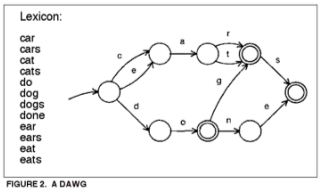


FIGURE 2. A DAWG

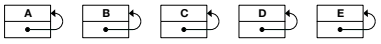
- As the authors note, "this minimization produces an amazing savings in space; the number of nodes is reduced from 117,150 to 19,853."

Fast Set Operations

- If you implement Kruskal's algorithm for finding a minimum spanning tree, one of the important operations consists of merging sets of connected nodes.
- Making this algorithm efficient requires fast implementations of the following operations:
  - Given a node, you need to be able to determine which set contains that node. This operation is called ***find***.
  - Given two nodes that are part of different sets, reconfigure the data structure so that those two sets are merged into one. This operation is traditionally called ***union***.
- If an algorithm requires only the *union* and *find* operations (as opposed to all the other operations available in the complete set class), it is possible to run that algorithm so that it runs essentially in constant time.

The Union-Find Algorithm

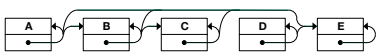
- The best-known algorithm for implementing these restricted set operations quickly is called the ***union-find*** algorithm.
- In the union-find algorithm, sets are represented as a forest of trees. Thus, if the elements of a collection are represented by the letters **A**, **B**, **C**, **D**, and **E**, the entries initially form a forest of five disjoint nodes, as follows:



- Each node in the tree contains a parent pointer, which initially points to the node itself.
- Each set in this disjoint forest is identified by the address of one of its elements, which is called a ***representative***. In the union-find algorithm, this representative always appears at the end of the parent-pointer chain.

The Union-Find Algorithm

- In the most primitive form of the algorithm, the `find` and `union` operations have the following implementations:
  - `find` follows the parent pointers until a node points to itself.
  - `union` calls `find` to identify the representatives and makes one point to the other.
- The following diagrams illustrates the results of calling the `union` operation on the pairs **A**–**B**, **D**–**E**, **A**–**C**, and **A**–**D**:



- It is easy to improve the performance of this algorithm by changing the parent links to point directly to the representative as the recursion unwinds.

Ackermann's Function

- In 1975, Robert Tarjan proved that the amortized cost of the optimized union-find algorithm is $O(\alpha(N))$, where $\alpha(N)$ is the inverse of the ***Ackermann function*** $f(N) = \mathrm{A}(N, N)$, which is named after its inventor, Wilhelm Ackermann.
- The function $\mathrm{A}(m, n)$ is defined recursively as follows:

$$\mathrm{A}(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ \mathrm{A}(m - 1, 1) & \text{if } m \neq 0 \text{ and } n = 0 \\ \mathrm{A}(m - 1, \mathrm{A}(m, n - 1)) & \text{if } m \neq 0 \text{ and } n \neq 0 \end{cases}$$

- The Ackermann function grows extremely quickly, so that the number of digits in $\mathrm{A}(5, 5)$ exceeds the estimated number of atoms in the universe.
- Because the Ackermann function grows quickly, its inverse grows slowly, which means that $\alpha(N)$ is less than 5 for any conceivably practical value of N.