PageRank

Author: Logan Roche

Table of Contents

Introduction

The PageRank algorithm, developed by Larry Page and Sergey Brin in the late 1990s at Stanford University, is one of the foundational technologies that powered the early success of the Google search engine. At its core, PageRank is a method for ranking web pages based on their importance within a network of hyperlinks. The algorithm was inspired by the academic citation system, where the value of a research paper is often inferred by the number of citations it receives, especially from influential sources.

The purpose of PageRank is to assign a numerical score to each node (or web page) in a graph, representing the probability of a "random surfer" landing on that page. This score is determined by the structure of incoming links, with the intuition that links from highly ranked pages carry more weight than those from less significant pages. By iteratively calculating these scores, PageRank identifies the most relevant and authoritative nodes in the graph.

PageRank has widespread applications beyond web search. It has been used in analyzing social networks, ranking academic papers in citation networks, identifying influential players in sports, and even in biological sciences to prioritize important genes or proteins in a network. Its ability to reveal hierarchical structures in large datasets makes it a powerful tool in various domains.

The importance of PageRank lies not only in its historical significance but also in its enduring relevance. Despite the evolution of search algorithms, the principles underlying PageRank remain a cornerstone of modern information retrieval and network analysis, influencing the way we understand and interact with connected data in the digital age.

Intuition

At a high level, the PageRank algorithm works by simulating the behavior of a "random surfer" navigating through a network of web pages. Imagine a person who starts at a random web page and either clicks on a link to move to another page or randomly jumps to a completely different page. The goal of PageRank is to determine the likelihood of the surfer being on each page at any given time, with these probabilities representing the importance of the pages.

The algorithm operates on the following key principles:

1. Link Importance:
    - A page becomes more important if it is linked to by other important pages. The idea is recursive: a page's rank depends on the ranks of the pages linking to it.
2. Even Distribution:
    - Initially, all pages are assigned equal rank (e.g., 1/N, where N is the total number of pages). Over time, these ranks are updated based on the incoming links.
3. Rank Contribution:
    - When a page has multiple outgoing links, its importance is evenly distributed among the pages it links to. For instance, if a page with a high rank links to five other pages, each linked page gets an equal share of its rank.
4. Damping Factor:
    - To model the possibility of a random jump to any page, the algorithm introduces a "damping factor," typically set to 0.85. This means that 85% of the time, the surfer follows a link on the page, and 15% of the time, they jump to a random

page. This factor ensures that every page has a non-zero rank, even if it has no incoming links.

5. Convergence:
    - The algorithm iteratively updates the ranks of all pages based on the contributions from their incoming links. This process is repeated until the ranks stabilize, meaning the difference between updates becomes negligible.

To summarize, PageRank intuitively captures the idea that important pages are those linked to by other important pages. By modeling a realistic surfing behavior, it assigns ranks that reflect the relative importance of nodes in a network.

Run-Time Analysis

Step-by-Step Analysis

1. **Initialization**:
    - Initializing the PageRank scores to 1/N for all N nodes takes O(N).
2. **Main Loop**:
    - For each iteration:
        - For each node v, compute its new PageRank based on contributions from its incoming neighbors.
        - Across all nodes, this is proportional to the total number of edges E (since the sum of in-degrees equals the total number of edges in the graph).
        - Updating and normalizing PageRank values across N nodes adds an O(N) overhead per iteration.
    - Total cost per iteration is therefore O(E+N)
3. **Convergence**:
    - The algorithm runs for I iterations until convergence. I depends on the graph's structure and the convergence threshold ϵ, but it is generally considered logarithmic relative to the graph size for most practical cases.
4. **Overall Runtime**:
    - $O(I \cdot (E + N))$
    - I: Number of iterations to convergence.
    - E: Total number of edges in the graph.
    - N: Total number of nodes in the graph.

Pseudocode

Main Program

1. **Build Graph:**
    - Prompt the user to enter the number of nodes.
    - Prompt the user to enter the number of edges.
    - Create the specified number of nodes.
    - For each edge:
        - Input the start and end node indices.
        - Establish the relationship (start → end).
2. **Compute PageRank:**
    - Set the damping factor to 0.85.
    - Initialize all PageRank values to 1.0.
    - Iterate until convergence:
        - Update PageRank for each node.
        - Normalize the PageRank values.
3. **Print Results:**
    - Display the PageRank for each node in a formatted table.

Classes

Class Node

- **Attributes:**
  - name: Node identifier.
  - children: Outgoing links (list of nodes).
  - parents: Incoming links (list of nodes).
  - pagerank: Current PageRank value.

Class Graph

- **Attributes:**
  - nodes: List of all nodes in the graph.

Functions

Function: Build_Graph

1. Input:
   - Number of nodes.
   - Number of edges (connections between nodes).
2. Create:
   - Nodes and store them in the graph.
3. Establish edges:

    - For each edge, link the start node to the end node.
4. Output: Graph with nodes and connections.

Function: Update_PageRank(page, damping_factor, n)

Input:

- page: A reference to the current node for which we are recalculating PageRank.
- damping_factor: The damping factor used in the PageRank formula.
- n: The total number of nodes in the graph.

Process:

1. Initialize pagerank_sum to 0.0.
2. Retrieve the list of parent nodes from page->parents.
3. For each parent in page->parents:
    - Determine the contribution from this parent to the current node's PageRank:
        - If parent->children is not empty, the parent's contribution is (parent->pagerank / number_of_children_of_parent).
        - Accumulate this value into pagerank_sum.
4. (This effectively sums up the fraction of each parent's PageRank that is passed along the link to page.)
5. Compute the random walk (teleportation) component:
    - random_walk = (1 - damping_factor) / n
6. This represents the probability of jumping to any node at random.

7. Combine the random walk and the damped contribution sum to obtain the new PageRank for page:
    - page->pagerank = random_walk + (damping_factor * pagerank_sum)

Output:

- The page->pagerank value is updated in place.

Function: PageRank_One_Iteration(graph, damping_factor)

Input:

- graph: The graph containing all nodes.
- damping_factor: The damping factor for the PageRank formula.

Process:

1. Let node_list = graph->nodes, an array or vector of all nodes in the graph.
2. Retrieve the total number of nodes n = node_list.size().
3. For each node i in node_list:
    - Call Update_PageRank(node_list[i], damping_factor, n) to recalculate that node's PageRank based on its parents' current values.
4. After updating the PageRank for each node, you must ensure that all PageRank values sum to 1.0.
    - Initialize total_pagerank = 0.0.
    - For each node in node_list:

        - Accumulate node->pagerank into total_pagerank.
5. Normalize:
    - For each node in node_list:
        - node->pagerank = node->pagerank / total_pagerank
6. This scaling ensures that the total PageRank distribution is a proper probability distribution summing to 1.

Output:

- All nodes in graph->nodes now hold updated and normalized PageRank values after one full iteration.

Function: Compute_PageRank(graph, damping_factor)

Input:

- graph: The graph containing all nodes, each with an initial PageRank (often initialized to 1/n).
- damping_factor: The damping factor for the PageRank formula.

Process:

1. Set epsilon to a small number, e.g. 1e-6, which determines convergence sensitivity.
2. Initialize has_converged to false.
3. While has_converged is false:
    1. Set has_converged = true (assume convergence until a difference is detected).

2. Create an array old_pageranks to store the current PageRank values before this iteration:
    - For each node i in graph->nodes:
        - Append graph->nodes[i]->pagerank to old_pageranks.
3. Call PageRank_One_Iteration(graph, damping_factor) to perform a single iteration:
    - This updates and normalizes the PageRank values of every node.
4. Compare the updated PageRank values with the old values to check if convergence criteria are met:
    - For each node i:
        - Compute difference = |old_pageranks[i] - graph->nodes[i]->pagerank|
        - If difference > epsilon:
            - Set has_converged = false
            - Break out of the loop since we found a node that changed significantly.
4. (If after checking all nodes, has_converged remains true, it means all changes were below the threshold epsilon, so the algorithm terminates.)
5. When the loop ends, the PageRank values in graph->nodes have converged. The final stable distribution of PageRank scores is now available in each node's pagerank attribute.

Output:

- The graph->nodes array contains stable, converged PageRank values for each node.

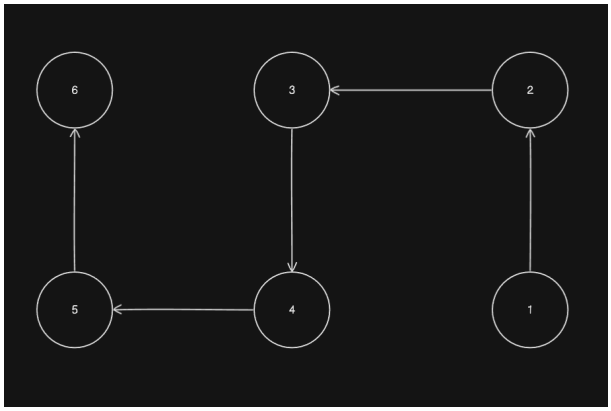
  These values represent the steady-state probability distribution of a random surfer visiting each node.

Function: Print_Graph

1. Input: Graph.
2. Output:
    - For each node, display its name and PageRank value.

Example Runs

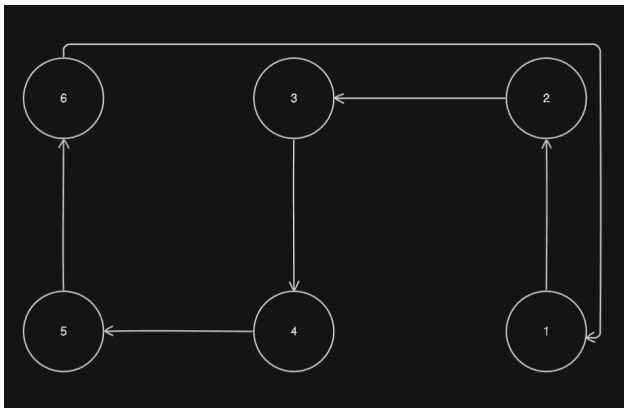
1.




```
Page Rank:
[0.061, 0.112, 0.156, 0.193, 0.225, 0.252]
```

- The Nodes have an increasing Rank due summing the value of all previous links. Thus the first node has no links pointing at it resulting in the lowest rank while the last node has the highest rank due to all nodes having a link, direct or indirect, to the node.

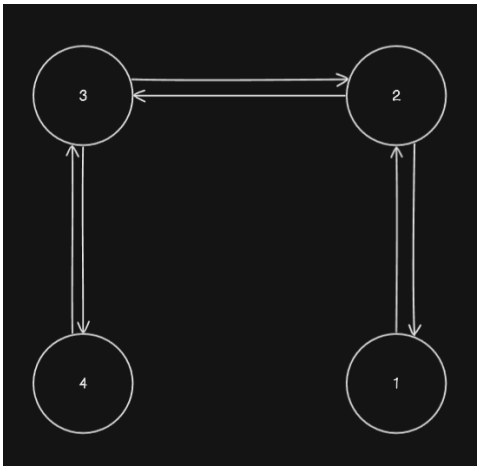
2.




```
Page Rank:
[0.167, 0.167, 0.167, 0.167, 0.167, 0.167]
```

- The Nodes have equal rank due to there being a circular sequence of links

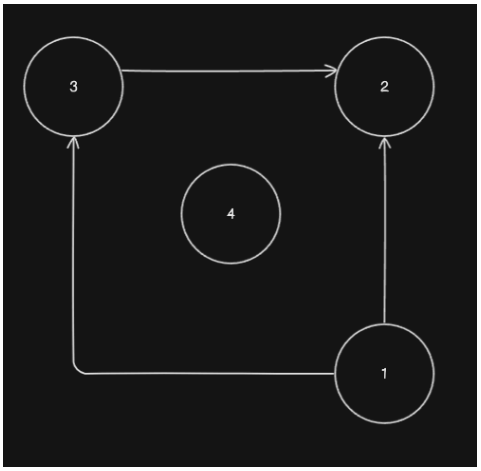
3.




```
Page Rank:
[0.175, 0.325, 0.325, 0.175]
```

- The central nodes, 2 and 3, have a higher rank due more links being directed at them. Nodes 1 and 4 share the same rank due to having each link pointing at them carry equal amounts of weight despite being disconnected.

4.




```
Page Rank:
[0.116, 0.602, 0.166, 0.116]
```

- Nodes 1, 2, and 3 are all as expected. Node 4 still has a page rank despite not being pointed at by a link due to the randomness of a user not following a link. This is represented through the damping factor where the user has a 15% chance to not follow a link and jump to a random website

Citations

- "PageRank." *Wikipedia*, https://en.wikipedia.org/wiki/PageRank. Accessed 5 Dec. 2024.
- Gillis, Andrew. "PageRank: Understanding Google's Algorithm." *Towards Data Science*, 17 June 2020, https://towardsdatascience.com/pagerank-3c568a7d2332. Accessed 5 Dec. 2024.
- *Eraser.io*. https://www.eraser.io/. Accessed 5 Dec. 2024