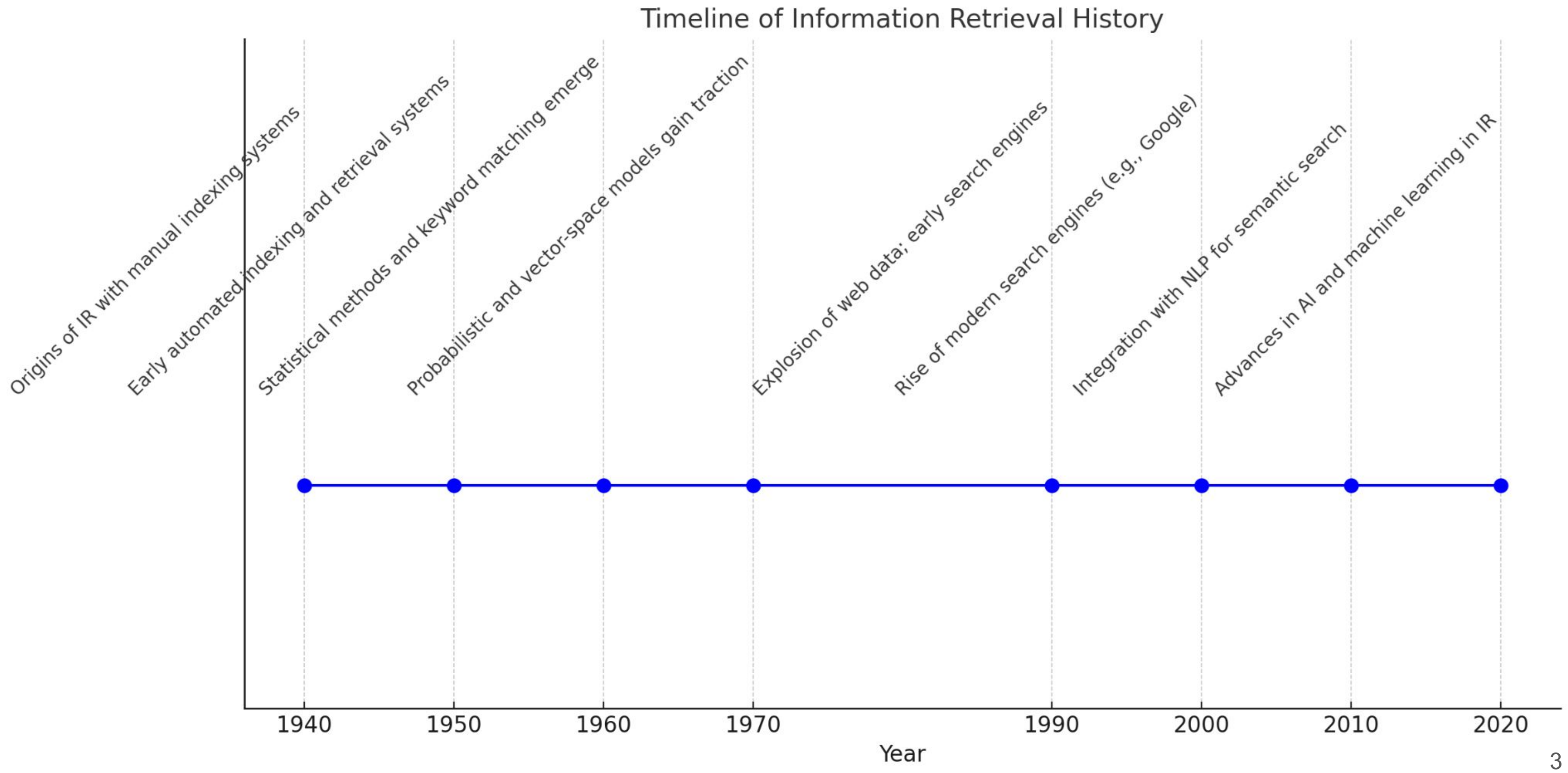
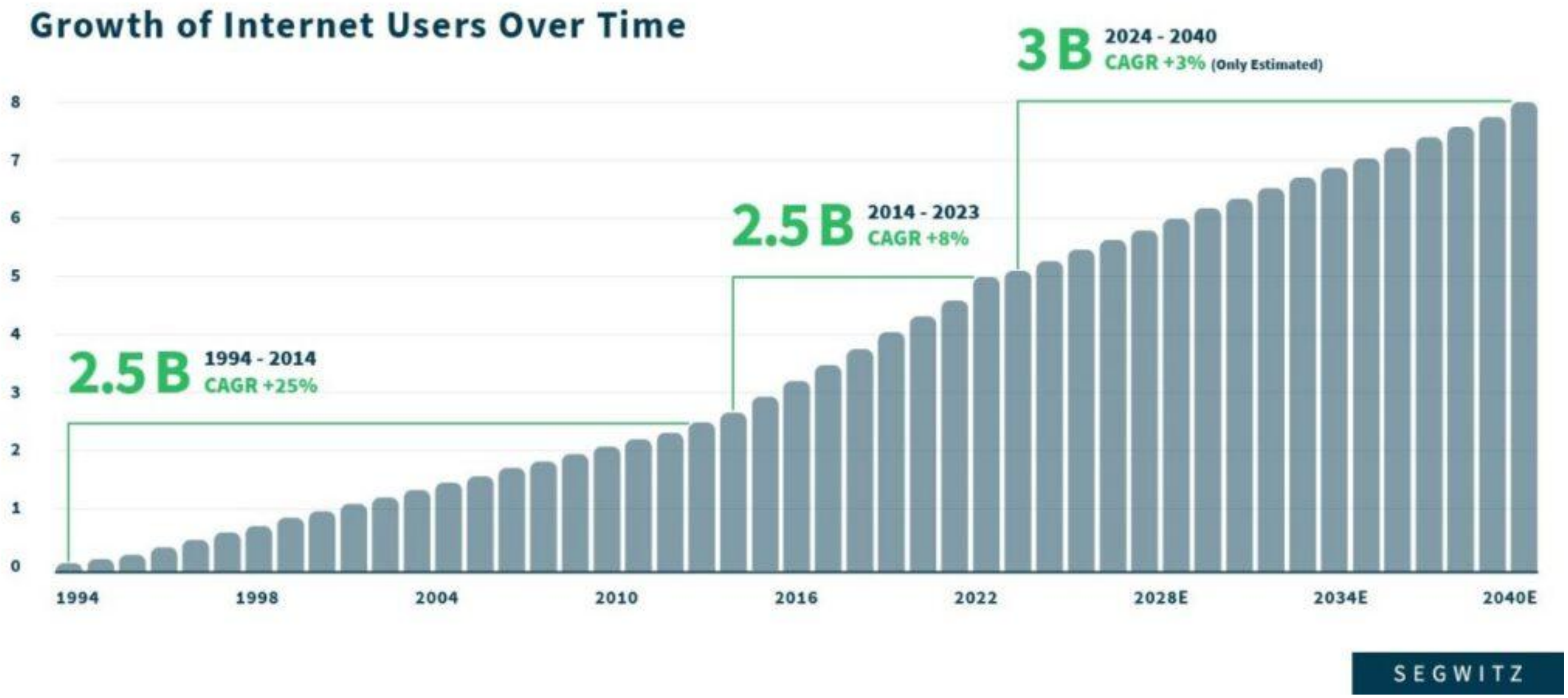
Web Search and Crawling

Jan, 2025

Outline

Background and history





Source: https://segwitz.com/the-age-of-next-internet-users-what-should-you-know/

Characteristics of the Web:

Enormous Scale:

- The web contains billions of documents and is constantly growing.
- This vastness makes it challenging to index and retrieve information effectively.

Dynamic Nature:

- Web content changes frequently, with pages being added, updated, or deleted.
- Maintaining an up-to-date index is a complex task for search engines.

Heterogeneity of Content:

- The web hosts a wide range of content types, including text, images, videos, and interactive elements.
- These diverse formats require specialized retrieval techniques.

Link Structure:

- The web's hyperlink structure provides valuable information about the relationships between pages.
- Algorithms like PageRank use these links to assess the importance of a page.

Varied Quality:

- Web content ranges from high-quality, authoritative sources to spam and misinformation.
- Evaluating the credibility and relevance of content is a critical challenge.

User-Generated Content:

- The web is increasingly driven by user contributions, such as blogs, social media, and forums.
- This content is often informal and lacks traditional editorial oversight.

Multilingual and Multicultural:

- The web includes content in many languages and reflects diverse cultural perspectives.
- Effective IR systems must handle multilingual search and context understanding.

Search Engine Dependency:

- Most users rely on search engines to navigate the web.
- This dependency places a high demand on search engines to deliver accurate and relevant results.

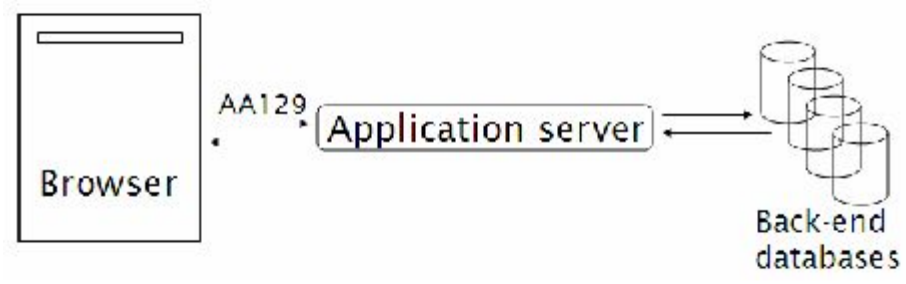
…

Economic Influence:

- Web content is heavily influenced by economic factors, including advertising and SEO (Search Engine Optimization).
- This introduces biases in how information is presented and retrieved.

Long Tail of Queries:

- A significant portion of web searches consists of rare or unique queries, often referred to as the "long tail."
- IR systems must handle both popular and niche search terms effectively.



Web graph

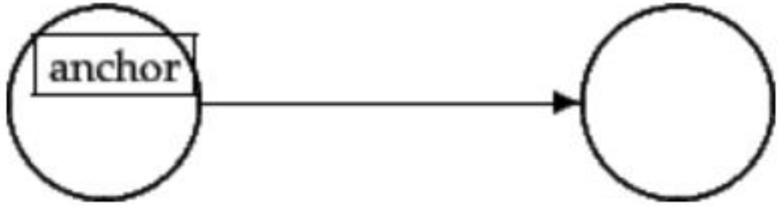


Figure 19.2: Two nodes of the web graph joined by a link.

Key Concepts:

1. **Nodes and Edges**:
   - **Nodes**: Represent individual web pages.
   - **Edges**: Directed hyperlinks from one page to another.
2. **In-links and Out-links**:
   - **In-links**: Hyperlinks directed toward a page.
   - **Out-links**: Hyperlinks originating from a page.
   - **In-degree**: Number of in-links to a page.
   - **Out-degree**: Number of out-links from a page.

. . .

3. Anchor Text:

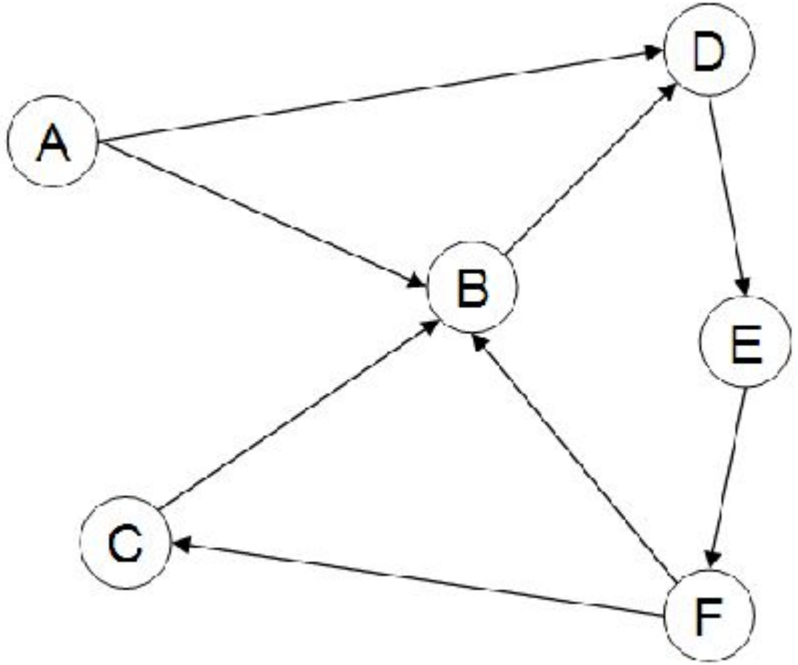
- The clickable text in a hyperlink, providing context about the linked page.

4. Connectivity:

- The web graph is not strongly connected; some pages cannot be reached from others by following hyperlinks.

5. Degree Distribution:

- The distribution of in-links follows a power law, indicating that a few pages have a high number of in-links, while most have few.



A sample small web graph.In this example we have six pages labeled A-F. Page B has in-degree 3 and out-degree 1.

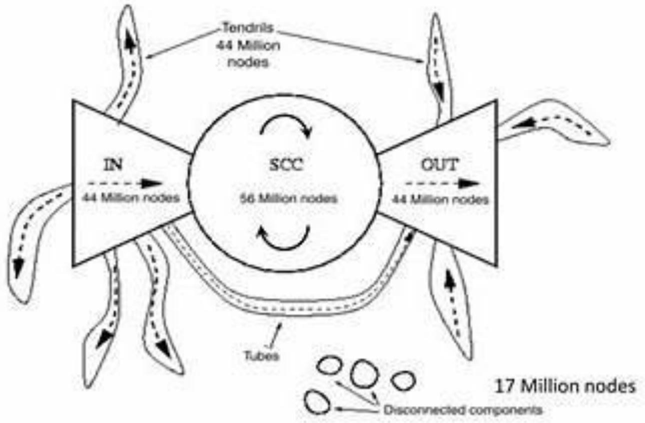
This example graph is not strongly connected: there is no path from any of pages B-F to page A.

…

The **bowtie structure** is a conceptual model that describes the organization of the World Wide Web as a directed graph, where web pages are nodes and hyperlinks are directed edges.

This model was introduced by Broder et al. in their 2000 paper, "Graph Structure in the Web."

Bow-Tie Structure of the Web



Broder et. al (Graph Structure of the Web, 2000)
Examined a large web graph (200M pages, 1.5B links)

Bowtie Structure

The web graph exhibits a "bowtie" structure with three main components:

- **IN**: Pages that can reach the central core (SCC) but cannot be reached from it.
- **SCC (Strongly Connected Component)**: A set of pages where each page can be reached from any other within the set.
- **OUT**: Pages that can be reached from the SCC but do not link back to it.

Additional components include "tendrils" and "tubes" connecting these regions.

. . .

Web Crawling and Indexing:

- Crawlers aim to traverse the bowtie structure, but disconnected components and tendrils can make discovery incomplete.
- Efficient crawling strategies prioritize SCC and frequently updated OUT components.

Link Analysis Algorithms:

- **PageRank**: Assigns higher importance to pages in the SCC due to dense link interconnections.
- **HITS**: Identifies hubs and authorities, often focusing on pages in the IN and OUT components.

What is Search Engine?

- A software program that helps people find the information they are looking for online using keywords or phrases.

- A search engine is software that helps users find relevant information from the vast library of data available on the World Wide Web.

- It allows users to search for various types of content, including queries, documents, images, videos, webpages, and other resources.

- Search engines work by crawling the web and searching through large databases available on the internet to generate the required information efficiently.

History

- Archie was the first well developed search engine in the year 1990.
    - It used to search files by matching their names and indexing on FTP server.
- In 1992 Veronica search engine was developed for Gopher based websites.
- Later in 1993 W3Catalog and Aliweb were formed which were web search engines. WebCrawler search engine was the first to allow users search keywords.
- Finally in 1994, the search engine Yahoo! was developed which gained immense popularity.
    - Earlier it was just a directory but in 1995 search feature was also added.
- In 1998 when Google was founded as till now is the most used and preferred browser all across the globe.
    - However, there have been other frequently used search engines formed after google like, Baidu, Bing, Yandex, etc.

Search Engine Architecture (1995)

(handful of people)

data acquisition

data analysis

index building

query processing

Search Engine Architecture (2000)

(dozens of people)

data acquisition

data analysis

index building

query processing

Search Engine Architecture (2010+)

(hundreds of people)

data acquisition

data analysis

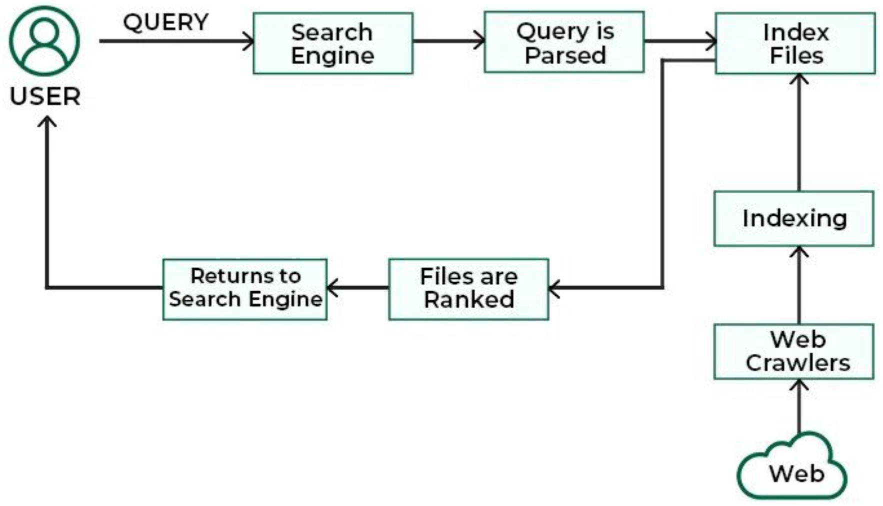
index building

query processing

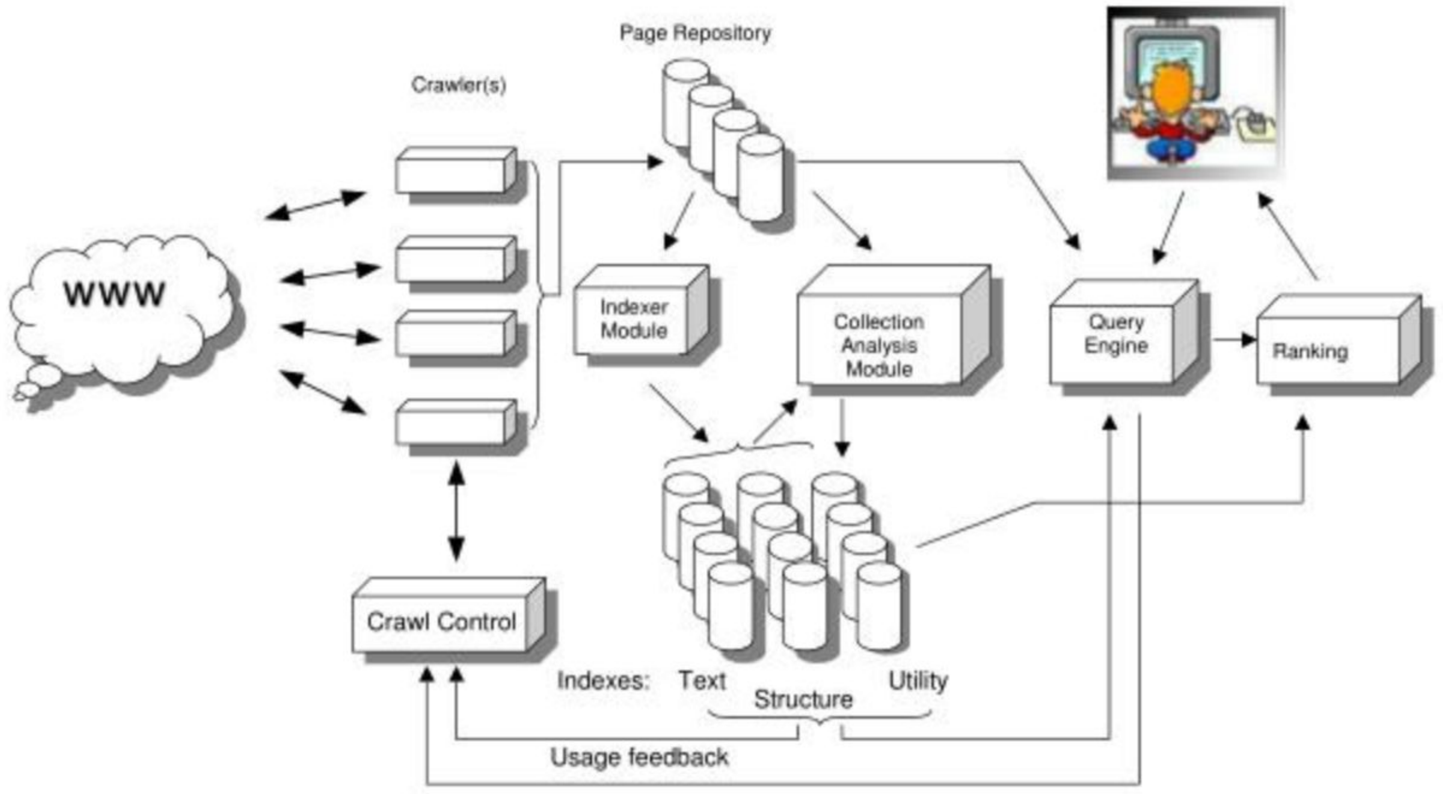
Architecture of web search engines

Components of a Search Engine:

- **Crawlers**: Continuously traverse the web, fetching pages for indexing.
- **Indexer**: Organizes and processes fetched content to enable fast retrieval.
- **Query Processor**: Analyzes user queries to retrieve relevant documents.
- **Ranker**: Assigns scores to documents based on relevance and quality.



General Search Engine Architecture[1]



…

Crawling: Web crawlers (also called spiders or bots) traverse the internet systematically.

- They follow links from one webpage to another, collecting data about the content and structure of the web pages they visit.

Indexing: The data collected during crawling is stored and organized in a structured format called an **index.**

- This index acts as a vast library where information is categorized and tagged to make it easily retrievable.

…

Ranking: When a user searches for a query, the search engine's algorithm analyzes the indexed data and determines the relevance of web pages based on various factors.

- These may include keyword usage, site quality, user experience, backlinks, and freshness of content.
- The ranking algorithm ensures that the most relevant and authoritative pages appear higher in the search results.

Retrieval and Display: Once the ranking is complete, the search engine retrieves the most relevant results and displays them on the Search Engine Results Page (SERP).

- This entire process, from crawling to displaying results, happens almost instantaneously.

How Queries are Processed in Search Engines

- Queries in search engines are processed within seconds, but the backend involves complex operations.
- **Indexing** and **Querying** are the two main building blocks of search engine processing.

Indexing

- **Web Crawling**:
  - Spiders (or bots) crawl across the World Wide Web and collect data.
- **Data Storage**:
  - Collected data is stored in a database, known as text acquisition.

...

- **Tokenization**:
  - Data is broken into tokens or keywords.
  - These tokens are associated with specific documents.
- **Index Creation**:
  - Tokens are used to create an organized index, enabling quick retrieval of information by the search engine.

Querying

- **Query Input**:
  - When a user submits a search, a query is generated.
- **Parsing**:
  - The search engine parses the query and looks for matching documents in the index.
- **Ranking**:
  - A ranking algorithm evaluates the relevance of documents.
  - Results are ranked, with the most relevant ones presented at the top.

Web Crawling:

- Web crawling involves systematically gathering web pages to build an index that supports search engines.
- The goal is to gather as many relevant and interconnected web pages as efficiently as possible.

Challenges of Crawling:

- The decentralized and uncoordinated nature of the Web creates challenges, such as dealing with dynamic content, duplicate pages, and differing access restrictions (e.g., robots.txt files).
- Scalability is a major concern, particularly for large-scale search engines.

Web Crawler (Spider):

- A web crawler or spider is the program or component that performs crawling. It follows hyperlinks on web pages to discover new content and builds an index of pages.

Features a crawler must provide

Robustness:

The Web contains servers that create spider traps, which are generators of web pages that mislead crawlers into getting stuck fetching an infinite number of pages in a particular domain.

- Crawlers must be designed to be resilient to such traps.
- Not all such traps are malicious; some are the inadvertent side-effect of faulty website development.

Politeness:

Web servers have both implicit and explicit policies regulating the rate at which a crawler can visit them.

- These politeness policies must be respected.

Features a crawler should provide

Distributed:
The crawler should have the ability to execute in a distributed fashion across multiple machines.

Scalable:

The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth.

Performance and efficiency:

The crawl system should make efficient use of various system resources including processor, storage and network bandwidth.

Quality:

Given that a significant fraction of all web pages are of poor utility for serving user query needs, the crawler should be biased towards fetching ``useful'' pages first.

…

Freshness:

In many applications, the crawler should operate in continuous mode: it should obtain fresh copies of previously fetched pages.

- A search engine crawler, for instance, can thus ensure that the search engine's index contains a fairly current representation of each indexed web page.
- For such continuous crawling, a crawler should be able to crawl a page with a frequency that approximates the rate of change of that page.

Extensible:

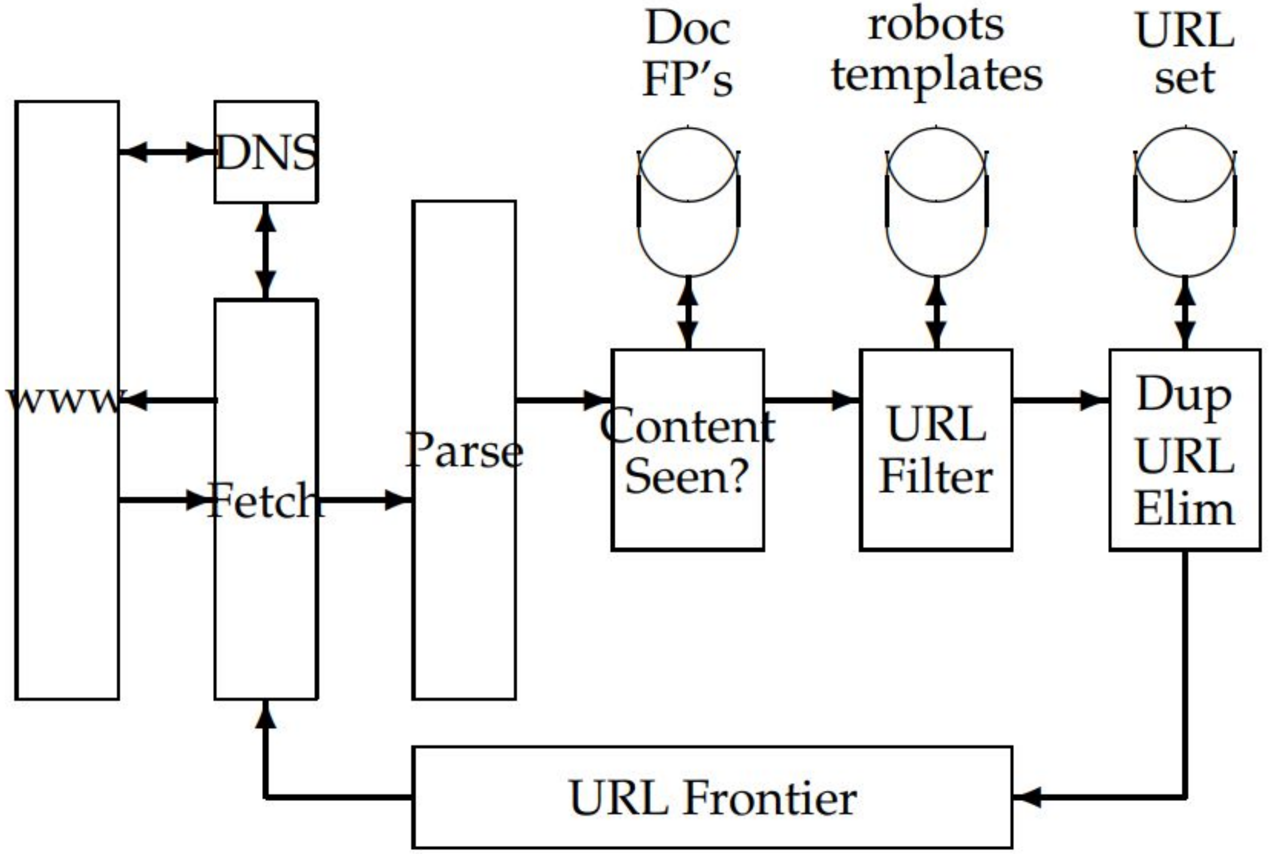
Crawlers should be designed to be extensible in many ways - to cope with new data formats, new fetch protocols, and so on.

- This demands that the crawler architecture be modular.

Basic Operation of a Crawler

1. **Seed Set**:
   - The crawler starts with a predefined set of URLs, known as the **seed set**.
   - These are the starting points for the crawler's traversal of the web.
2. **Fetching Web Pages**:
   - The crawler picks a URL from the seed set or the **URL frontier**.
   - It fetches the web page at this URL using HTTP or HTTPS requests.
3. **Parsing**:
   - The fetched page is parsed to:
     - **Extract text**: The content of the page is sent to a text indexer for indexing.
     - **Extract links**: The hyperlinks (URLs) embedded in the page are identified.
4. **URL Frontier**:
   - The extracted links are added to the **URL frontier**, which is a queue of URLs waiting to be crawled.
   - URLs in the frontier are processed iteratively, and crawled URLs are removed.
5. **Continuous Crawling**:
   - For pages that are frequently updated, the crawler re-adds the URLs of fetched pages to the frontier for periodic revisits, ensuring content freshness.

Basic Crawler Architecture



…

The process involves several key components and steps:

1. **URL Frontier**: This is a queue that holds URLs yet to be fetched.
    - In continuous crawling, URLs may re-enter the frontier for periodic re-fetching to ensure content freshness.
2. **DNS Resolution Module**: Before fetching a page, the crawler resolves the domain name of the URL to an IP address to locate the appropriate web server.
3. **Fetch Module**: Utilizing protocols like HTTP, this module retrieves the web page corresponding to the URL.
4. **Parsing Module**: Once fetched, the page is parsed to extract textual content and hyperlinks. The text is forwarded to the indexer, while the links are processed for potential inclusion in the URL frontier.
5. **Duplicate Elimination Module**: This component checks if a fetched page's content has been previously encountered to avoid redundant processing.
    - Techniques like checksums or more advanced methods such as shingles are employed for this purpose.

…

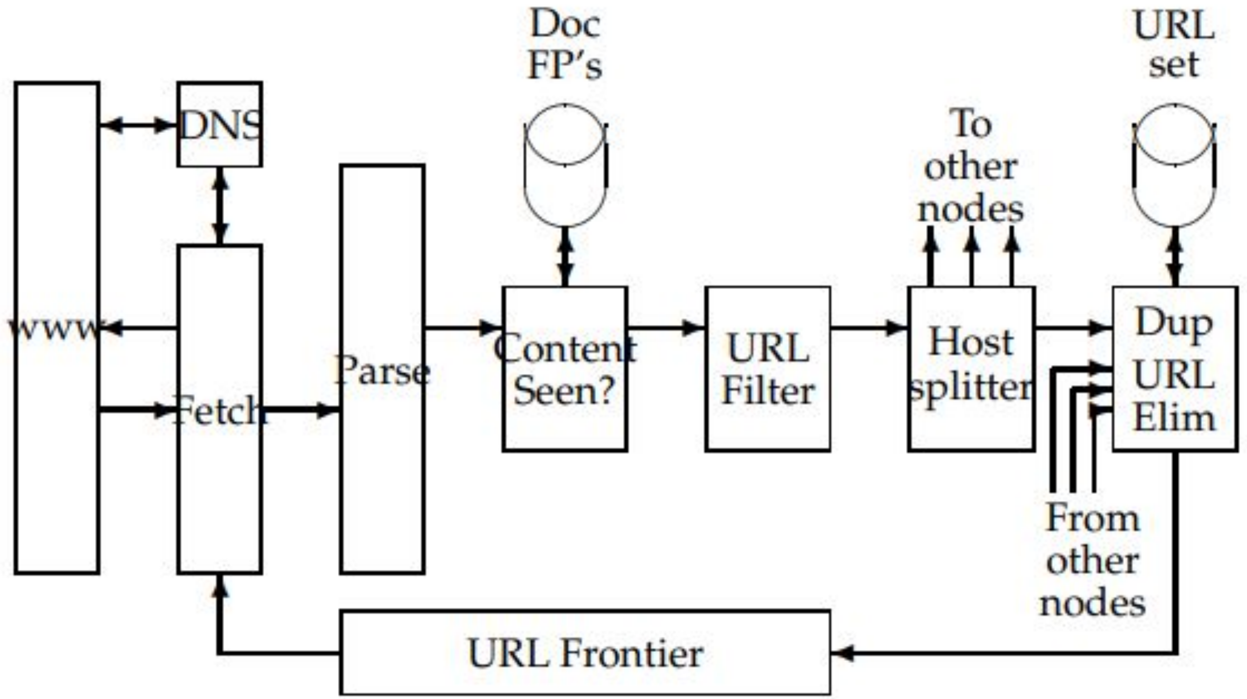
The crawler operates through multiple threads, each performing the following cycle:

- **URL Selection**: A thread selects a URL from the URL frontier.
- **Fetching**: The fetch module retrieves the web page.
- **Temporary Storage**: The fetched page is stored temporarily for processing.
- **Parsing**: The parsing module extracts text and links.
- **Content Indexing**: Extracted text, along with formatting information (e.g., bold terms), is sent to the indexer.

...

- **Link Processing**: Extracted links undergo several checks:
  - **Duplicate Detection**: The system verifies if the content at the URL has been previously encountered using methods like fingerprints or shingles.
  - **URL Filtering**: Links are assessed against predefined criteria to determine their eligibility for crawling.
    - For example, certain domains might be excluded or included based on the crawler's objectives.
  - **Robots Exclusion Protocol Compliance**: Before adding a URL to the frontier, the crawler checks the site's `robots.txt` file to respect the site's crawling policies.
    - To optimize this process, a cache of recently fetched `robots.txt` files is maintained, considering that many links often belong to the same host.
    - However, since URLs might remain in the frontier for extended periods, it's crucial to perform this check immediately before fetching to account for any changes in the `robots.txt` file.

Distributing the crawler



…

Distributing a web crawler across multiple nodes is essential for scaling and efficiency.

- Handle the vast expanse of the web more effectively.

- **Distributed Threads**: Crawler threads can operate on different processes across various nodes in a distributed system.
  - This distribution is crucial for scaling and can be beneficial in geographically dispersed systems where each node crawls hosts "near" it.
  - However, geographic proximity doesn't always align with network routes or the physical location of servers.
- **Partitioning Hosts**: Hosts can be assigned to crawler nodes using a hash function or a tailored policy.
  - For example, a node in Europe might focus on European domains, though this isn't always reliable due to the non-geographic nature of internet routing and domain naming.

Communication Between Nodes: Each node replicates the standard crawling process with a key difference: after the URL filtering stage, a **host splitter** directs each URL to the appropriate crawler node responsible for that URL's host.

- This ensures an even distribution of the crawling workload.

Content Seen Module: Determining if content has been previously encountered is complex in a distributed system:

- **Partitioning Fingerprints/Shingles**: Since identical or similar content can appear on different servers, fingerprints or shingles of documents must be partitioned across nodes based on properties like the fingerprint value modulo the number of nodes.
- **Remote Procedure Calls (RPCs)**: Most "Content Seen?" checks may require RPCs due to the distribution of fingerprints, though batching requests can improve efficiency.
- **Low Locality**: There's minimal repetition in document fingerprints, making caching ineffective.
- **Content Updates**: For continuous crawling, outdated fingerprints must be removed when documents change. This necessitates storing the fingerprint alongside the URL in the URL frontier.

DNS resolution

In web crawling, **DNS resolution** is the process of converting a hostname (e.g., `www.example.com`) into its corresponding IP address (e.g., `207.142.131.248`).

- This translation is essential for the crawler to locate and retrieve web pages.

Challenges in DNS Resolution for Web Crawlers:

- **Performance Bottleneck**: DNS resolution can be time-consuming due to its distributed nature, sometimes taking several seconds or more.
    - This latency can hinder the crawler's goal of fetching numerous documents per second.
- **Synchronous Operations**: Standard DNS lookup implementations are often synchronous, meaning a thread waits for a response before proceeding.
    - This waiting can block other operations, reducing the crawler's efficiency.

Strategies to Mitigate These Challenges

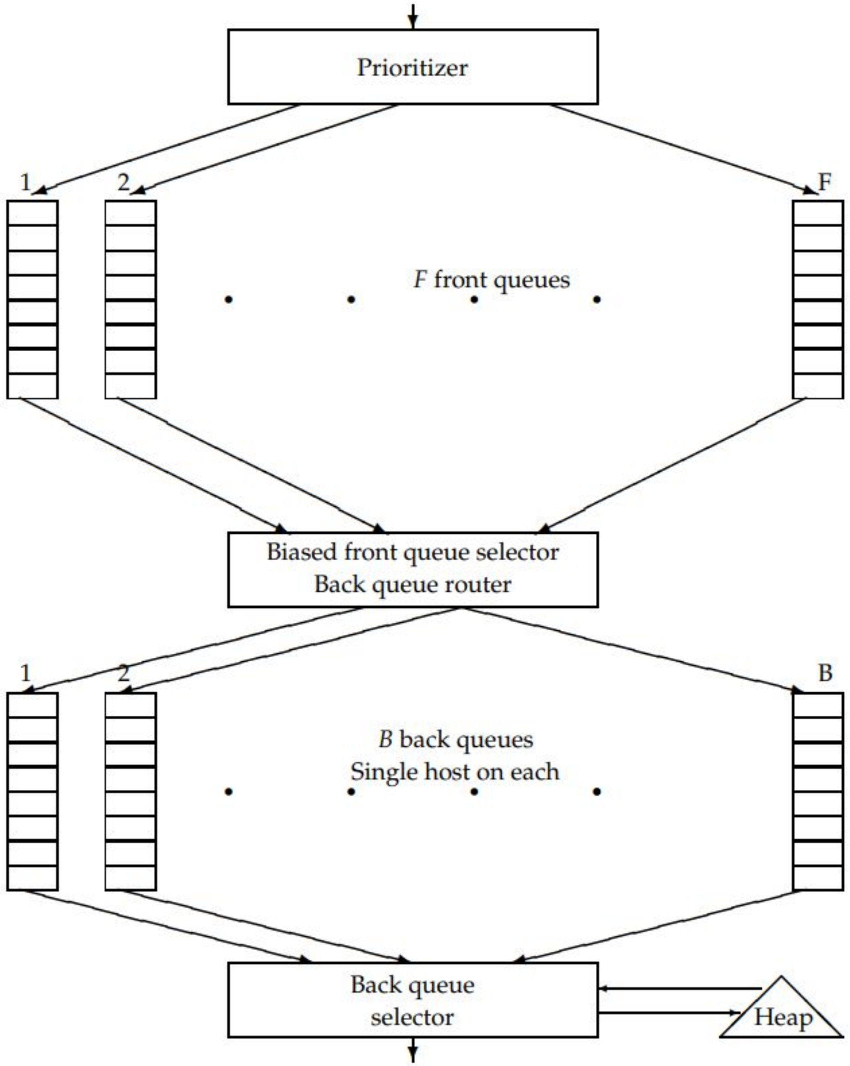
1. **Caching**: Storing results of recent DNS lookups locally allows the crawler to reuse them, reducing the need for repeated resolutions.
    - However, adhering to politeness policies (like waiting between requests to the same host) can limit the effectiveness of caching.

2. **Asynchronous Resolution**: Implementing a custom DNS resolver enables the crawler to perform non-blocking DNS queries. In this setup:
    - A crawler thread sends a DNS query and enters a timed wait, allowing it to handle other tasks or resume after a set period.
    - A dedicated DNS thread listens for responses and notifies the appropriate crawler thread upon receiving the necessary information.
    - If no response is received within the time limit, the crawler thread retries the query, with increasing wait times between attempts (e.g., starting at one second and increasing up to 90 seconds).

URL Frontier

The **URL frontier** is a critical component in a web crawler, managing URLs that are queued for fetching. It ensures efficient and polite crawling by considering two main factors:

1. **Prioritization**: Pages that are of high quality and frequently updated are assigned higher priority, ensuring they are crawled more often.
   - This prioritization is determined by evaluating both the change rate and the quality of the page.
2. **Politeness**: To prevent overwhelming any single host with rapid successive requests, the crawler enforces delays between accesses to the same host.
   - This is managed by organizing URLs into **front queues** (based on priority) and **back queues** (based on host).
   - A scheduling mechanism ensures that only one connection is open to a host at a time, with appropriate waiting periods between requests.



1. Prioritizer

- The **prioritizer** assigns importance to URLs based on factors like page quality, update frequency, or other relevance metrics.
- URLs are distributed into multiple **front queues (F)**, with each queue corresponding to a different priority level.

Front Queues (F)

- URLs in the **front queues** are grouped based on their priority.
- Higher-priority URLs are processed earlier, ensuring that critical or frequently updated pages are fetched sooner.
- A **biased front queue selector** picks a URL from one of these queues based on their priority weights.

Back Queue Router

- After selecting a URL from the front queue, the **back queue router** directs the URL to the appropriate **back queue (B)** based on its host.
- This routing ensures that URLs belonging to the same host are grouped together.

Back Queues (B)

- Each **back queue** corresponds to a specific host, and URLs in these queues are fetched sequentially.
- This design prevents overloading a single host by enforcing a delay between successive requests to the same host, maintaining crawler politeness.

Back Queue Selector

- The **back queue selector** picks a back queue to fetch from, ensuring that the delay for the host has elapsed before making a new request.
- The selector often uses a **heap structure** to efficiently manage delays and determine which queue is ready for fetching.

Overall Workflow

1. URLs enter the system and are assigned to front queues based on their priority.
2. The biased front queue selector picks a high-priority URL for processing.
3. The back queue router assigns the URL to a back queue corresponding to its host.
4. The back queue selector ensures politeness and fetches URLs in the appropriate order.

Distribution Indexes

In large-scale information retrieval systems, **distributing indexes** across multiple machines is essential for efficient query processing and scalability. There are two primary strategies for distributing indexes:

1. **Term-Partitioned Index (Global Index Organization)**:
    - The dictionary of index terms is divided into subsets, with each subset and its corresponding postings stored on a separate node.
    - **Advantages**:
        - Potential for high concurrency, as different queries may be processed by different nodes simultaneously.
    - **Challenges**:
        - Multi-word queries require merging postings from multiple nodes, leading to significant inter-node communication overhead.
        - Load balancing is complex due to the dynamic nature of query term distributions and their co-occurrences.
        - Implementing dynamic indexing (updating the index with new documents) is more difficult.

…

Document-Partitioned Index (Local Index Organization):

- The document collection is divided, and each node maintains an index for a specific subset of documents.
- **Advantages**:
  - Reduces inter-node communication, as each node can independently process its subset of documents.
- **Challenges**:
  - Global statistics, such as inverse document frequency (idf), must be computed across the entire collection, requiring coordination among nodes.
  - Determining an optimal partitioning strategy is non-trivial. Assigning all pages from a single host to one node can lead to uneven load distribution.
  - Hashing URLs to distribute documents more uniformly helps balance the load but necessitates broadcasting queries to all nodes and merging results.

…

A common heuristic is to partition the document collection into indexes of documents that are more likely to score highly on most queries and low-scoring indexes with the remaining documents.

- The system searches the low-scoring indexes only when there are too few matches in the high-scoring indexes.

Connectivity Servers

A **connectivity server** is a specialized component in web search engines designed to efficiently handle connectivity queries on the web graph.

- These queries typically involve determining which URLs link to a specific URL (in-links) and which URLs a specific URL links to (out-links).
- Crucial for applications like crawl control, web graph analysis, crawl optimization, and link analysis.

Challenges in Building a Connectivity Server:

- **Data Volume**: With billions of web pages, each containing multiple links, the sheer volume of link data is immense.
  - For instance, representing four billion pages with an average of ten links each would naively require approximately 320 gigabytes of memory (4 billion pages × 10 links/page × 8 bytes per link).
- **Efficient Query Support**: Beyond compressing the web graph to fit into memory, the system must support rapid responses to connectivity queries, necessitating efficient data structures and storage methods.

…

Strategies for Efficient Connectivity Servers:

1. **Integer Representation**: Assign a unique integer identifier to each URL, allowing for more compact storage and efficient processing.

2. **Adjacency Tables**: Construct tables where each row corresponds to a web page (identified by its unique integer) and contains a sorted list of integers representing pages it links to (out-links) or pages linking to it (in-links). This structure is analogous to an inverted index and reduces storage requirements by approximately 50%.

..

3. Exploiting Similarity and Locality:

- **Similarity**: Many web pages, especially within the same site, share common link structures due to consistent templates.
  - By identifying prototype rows for these similar pages, storage can be optimized by referencing these prototypes rather than storing duplicate information.
- **Locality**: Pages often link to other pages within the same domain or host.
  - By ordering URLs lexicographically and assigning integers accordingly, many links can be represented as small integers, further reducing storage needs.

4. Gap Encoding: Instead of storing absolute identifiers for each link, store the difference (gap) between consecutive entries in the sorted list.

- This method leverages the typically small differences between consecutive integers in a sorted list, allowing for more efficient compression.

Link analysis

- examines the structure of hyperlinks between web pages to assess their importance and relevance.
- Pivotal in enhancing web search results by leveraging the interconnected nature of the web.

Key Concepts:

1. **Web as a Graph**:
   - The web can be visualized as a directed graph where pages are nodes, and hyperlinks are edges connecting these nodes.
2. **Hyperlink Endorsement**:
   - A hyperlink from page A to page B is often considered an endorsement, suggesting that page B holds valuable information.

Web as a graph

The concept of the **web as a graph** models the World Wide Web as a directed graph, where:

- **Nodes** represent individual web pages.
- **Directed edges** (links) denote hyperlinks from one page to another.

Key Intuitions:

1. **Anchor Text as Descriptive Labels**: The text within a hyperlink (anchor text) pointing to page B often serves as a concise description of page B's content.
2. **Hyperlinks as Endorsements**: A hyperlink from page A to page B can be viewed as an endorsement or a signal of trust, indicating that the creator of page A finds page B valuable or relevant.

Anchor text

- refers to the clickable text within a hyperlink that points to another web page.
- For example, in the HTML snippet `<a href="http://www.acm.org/jacm/">Journal of the ACM</a>`,

  "Journal of the ACM" is the anchor text leading to `http://www.acm.org/jacm/`.

Descriptive Role of Anchor Text:

- In some cases, the target page provides the same description as the anchor text, but often, it does not accurately describe itself.
- For example:
  - IBM’s homepage (`http://www.ibm.com`) might not contain the term "computer" even though IBM is recognized as a leading computer company.
  - Similarly, Yahoo’s homepage (`http://www.yahoo.com`) may not contain the word "portal," though it is widely viewed as one.

...

Bridging the Gap:

- There is often a disconnect between the words on a webpage and how users describe it. Anchor text can bridge this gap:
  - It reflects how others perceive or describe a webpage.
  - It can help search engines index pages for terms that aren't explicitly present on the page itself.

Anchor Text in Search Engine Indexing:

- Search engines use anchor text as indexing terms for the target page:
  - For instance, if many hyperlinks to `http://www.ibm.com` include the word "computer," the search engine will index IBM's homepage under "computer."
- Anchor text terms are weighted based on frequency and are penalized for overuse of generic terms like "Click" or "here" (similar to TF-IDF scoring).

Machine-Learned Scoring:

- The relevance and weight of anchor text are determined through machine learning algorithms to optimize search rankings.

Impact of Anchor Text:

- **Positive Effects**:
  - Users searching for "big blue" can find IBM’s homepage because "big blue" is a widely recognized nickname for IBM.
- **Negative Effects**:
  - Misleading or derogatory anchor text, such as "evil empire," can influence search results. This has been exploited in coordinated campaigns (e.g., Google bombing).
  - Search engines actively combat such misuse to prevent anchor text spam.

Extended Anchor Text:

- The surrounding text of an anchor (referred to as extended anchor text) can also be used for indexing. For example:
  - "There is good discussion of Vedic scripture <a>here</a>."
- Research has explored the optimal width of the surrounding text to use effectively.

Given a query,

- a web search engine computes a composite score for each web page that combines hundreds of features such as cosine similarity and term proximity together with the PageRank score.
- This composite score, is used to provide a ranked list of results for the query.

Two popular algorithms to rank web pages by popularity

1. HITS - Hypertext Induced Topic Search
2. PageRank algorithm

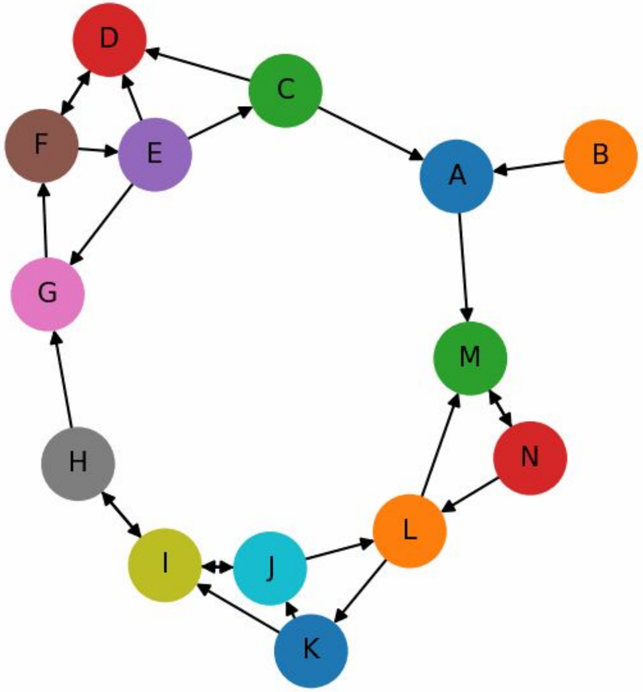
PageRank

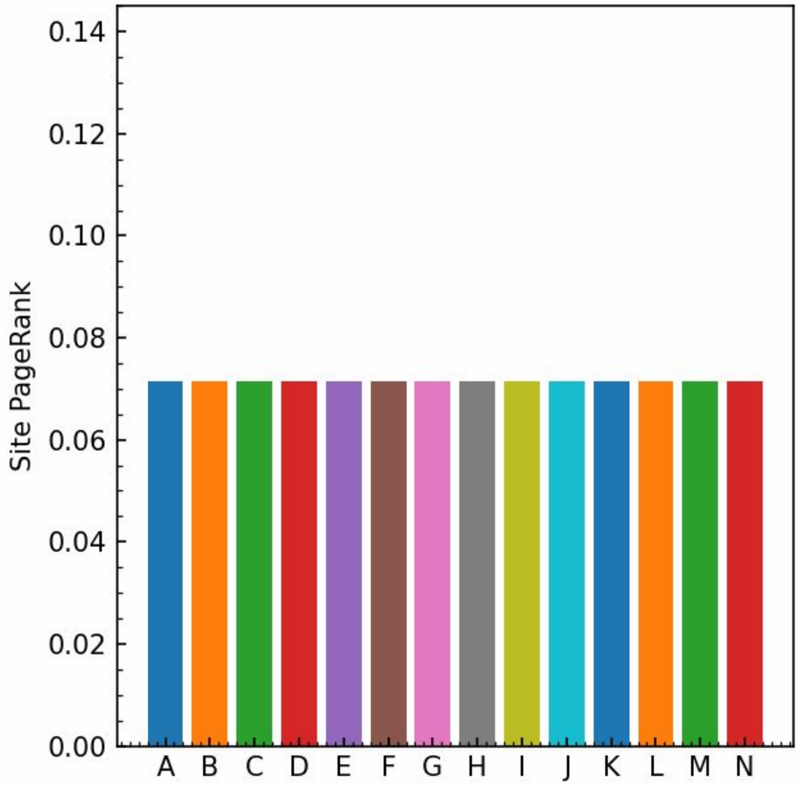
PageRank is a numerical score (between 0 and 1) assigned to every node (web page) in the web graph, derived solely from the link structure.

- It represents the importance of a web page based on how often it is visited during a "random walk" of the web graph.

According to Google:

> PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is.
>
> - The underlying assumption is that more important websites are likely to receive more links from other websites.

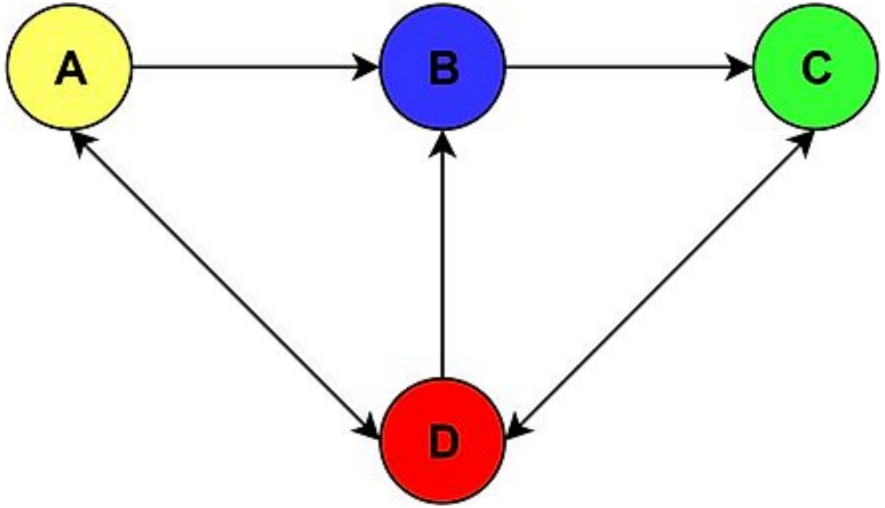

D
C
F
E
A
B
G
M
H
N
L
I
J
K


Site PageRank
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
A
B
C
D
E
F
G
H
I
J
K
L
M
N

PageRank Algorithm

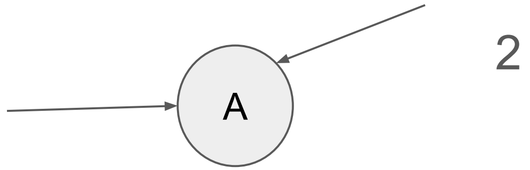
The WWW hyperlink structure forms a huge directed graph where the nodes represent web pages.

- Directed edges are the hyperlinks
- Generally graph traversal methods (most cases BFS) are used to Crawl the web.

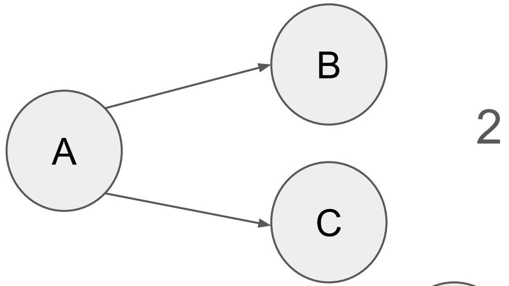


Link Types

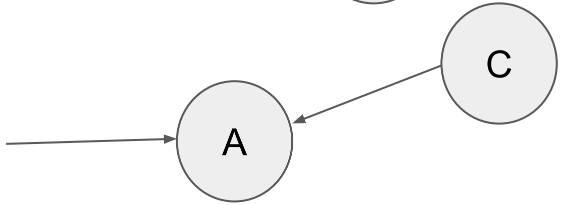
Inbound Links



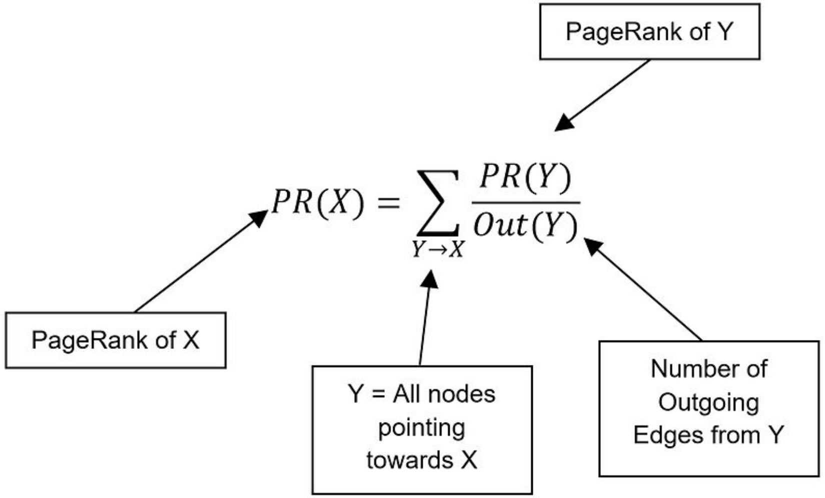
Outbound Links



Dangling Links



PageRank Algorithm: Iterative approach



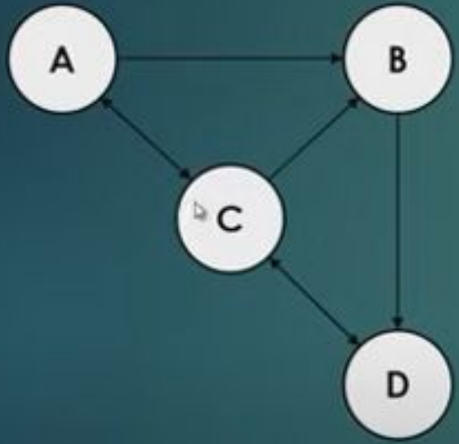
$$PR(X) = \sum_{Y \to X} \frac{PR(Y)}{Out(Y)}$$

PageRank of Y
PageRank of X
Y = All nodes pointing towards X
Number of Outgoing Edges from Y

We have to initialize page ranks at the beginning: all the pages are given equal page rank 1/n -where n is the number of pages.

So, we make several Iterations until convergence.

Example

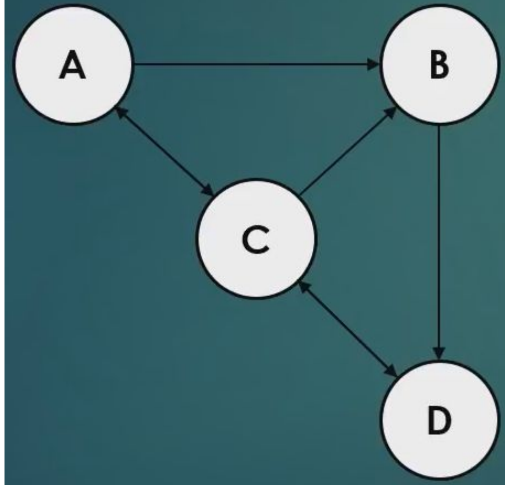


	Iteration 0	Iteration 1	Iteration 2	PageRank
A	1/4	1/12	1.5/12	1
B	1/4	2.5/12	2/12	2
C	1/4	4.5/12	4.5/12	4
D	1/4	4/12	4/12	3

Matrix Representation

We can use matrix operation instead of the iterative approach.

- Transition matrix

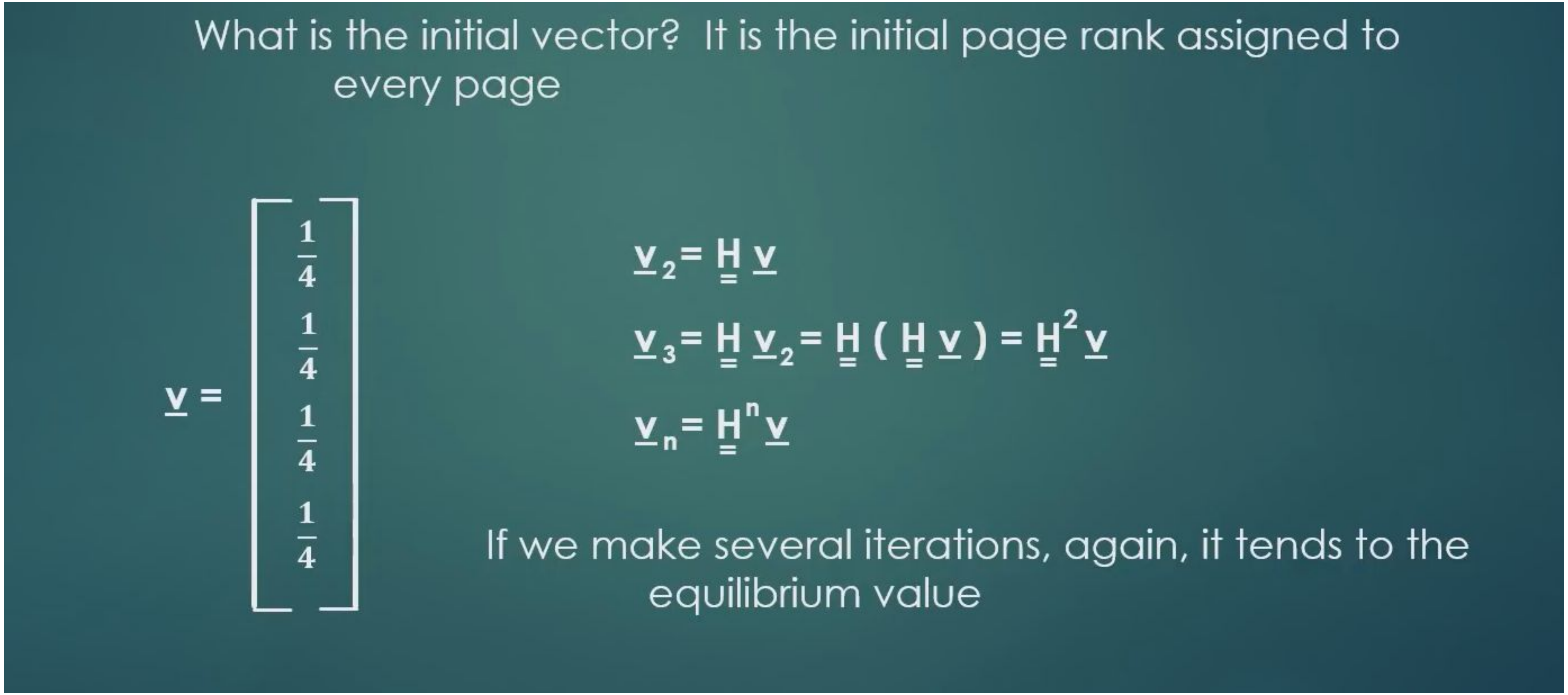


$$\begin{bmatrix} 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & 0 & 1 \\ 0 & 1 & \frac{1}{3} & 0 \end{bmatrix}$$

$$PR_{t+1} = \underline{\underline{H}}\ PR_t$$

„power method”

...



What is the initial vector? It is the initial page rank assigned to every page

$$\underline{v} = \begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}$$

$$\underline{v}_2 = \underline{\underline{H}}\,\underline{v}$$

$$\underline{v}_3 = \underline{\underline{H}}\,\underline{v}_2 = \underline{\underline{H}}\,(\,\underline{\underline{H}}\,\underline{v}\,) = \underline{\underline{H}}^2\,\underline{v}$$

$$\underline{v}_n = \underline{\underline{H}}^n\,\underline{v}$$

If we make several iterations, again, it tends to the equilibrium value

Random Surfer Model

The model assumes that a user:

1. Starts on a random webpage.
2. Follows a hyperlink from the current page to another page with equal probability (if there are multiple links).
3. Occasionally "jumps" to a completely random page, regardless of links, to avoid getting stuck on pages with no outgoing links or in loops.

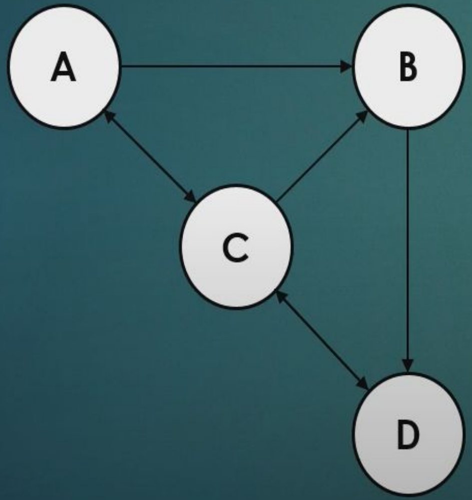
This behavior is controlled by a **damping factor (d)**, which is the probability of the user continuing to follow hyperlinks. The remaining probability, 1−d, represents the likelihood of jumping to a random page.

...

- A "random surfer" begins on a web page and performs a random walk:
  - From a current page (node **A**), the surfer randomly selects one of the outgoing links with equal probability.
  - For example, if node **A** links to nodes **B**, **C**, and **D**, the surfer proceeds to any of them with a probability of 1/3.
- Nodes visited more frequently during this process are considered more important and receive higher PageRank scores.

...

Importance of a web page is measured by its popularity
How many incoming links it has



PageRank can be defined by the probability that a random surfer on the web starts on a random page
+ follows hyperlinks AND visits the given page

→ sum of column values equals **1** because of the probabilities
→ it is like **Markov-chains**
→ the „transition matrix" defines the next steps
→ stationary distribution: is the final **PageRank** vector

$$\underline{\underline{H}}\,\underline{x} = \underline{x}$$

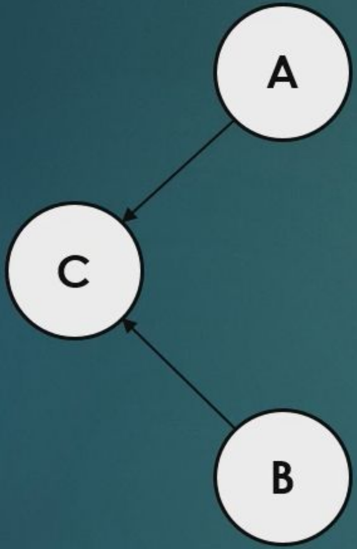
…

Problem with Pages Having No Out-Links:

- If a page (node **A**) has no outgoing links, the random walk cannot continue.
- To address this, the **teleport operation** is introduced:
    - The surfer can "teleport" to any node in the web graph, chosen uniformly at random.
    - Probability of landing on any specific page during teleportation = 1/N, where N is the total number of web pages.

Teleportation and Random Walk Combined:

- At any node:
    - With probability α= (e.g., 0.1), the surfer teleports to a random page.
    - With probability 1−α (e.g., 0.9), the surfer continues the standard random walk by following an outgoing link.



<u>Dangling nodes</u>: nodes with no outgoing edges

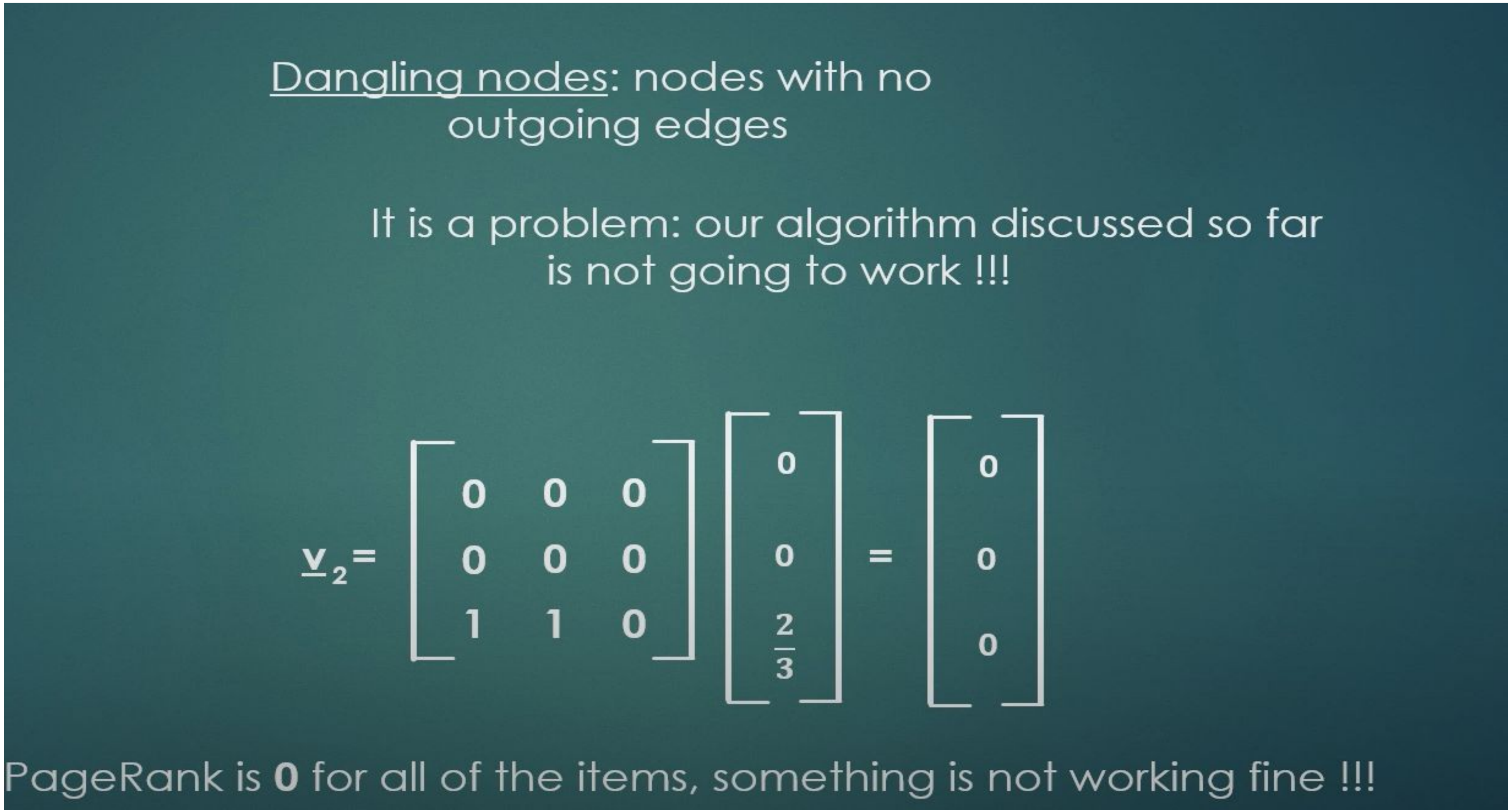
It is a problem: our algorithm discussed so far is not going to work !!!

$$\underline{v}_0 = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}$$

the inital page ranks

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

transition matrix



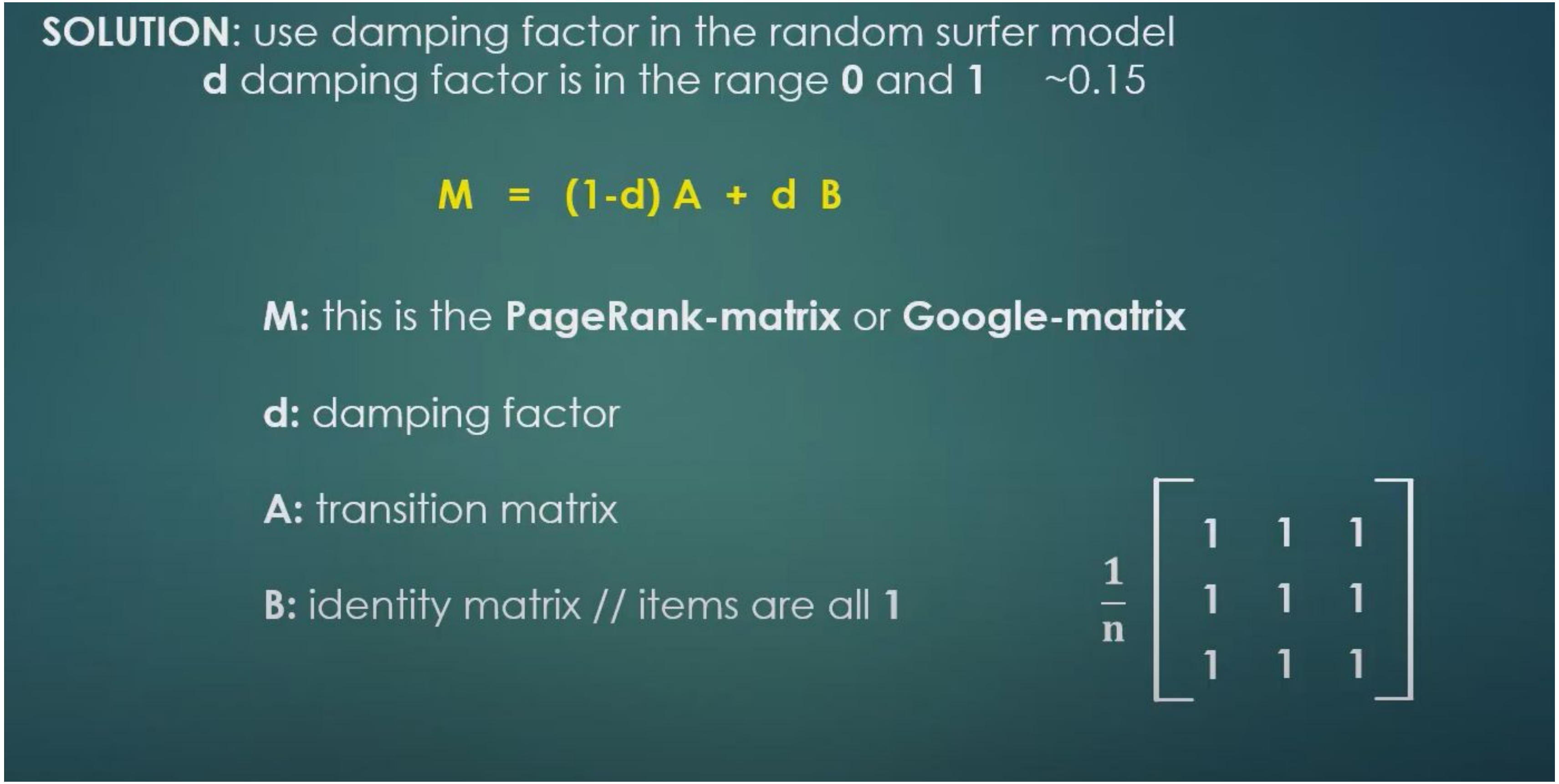
Dangling nodes: nodes with no outgoing edges

It is a problem: our algorithm discussed so far is not going to work !!!

$$\underline{v}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \frac{2}{3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

PageRank is **0** for all of the items, something is not working fine !!!

…



SOLUTION: use damping factor in the random surfer model

d damping factor is in the range **0** and **1** ~0.15

$$M = (1-d)\,A + d\,B$$

M: this is the **PageRank-matrix** or **Google-matrix**

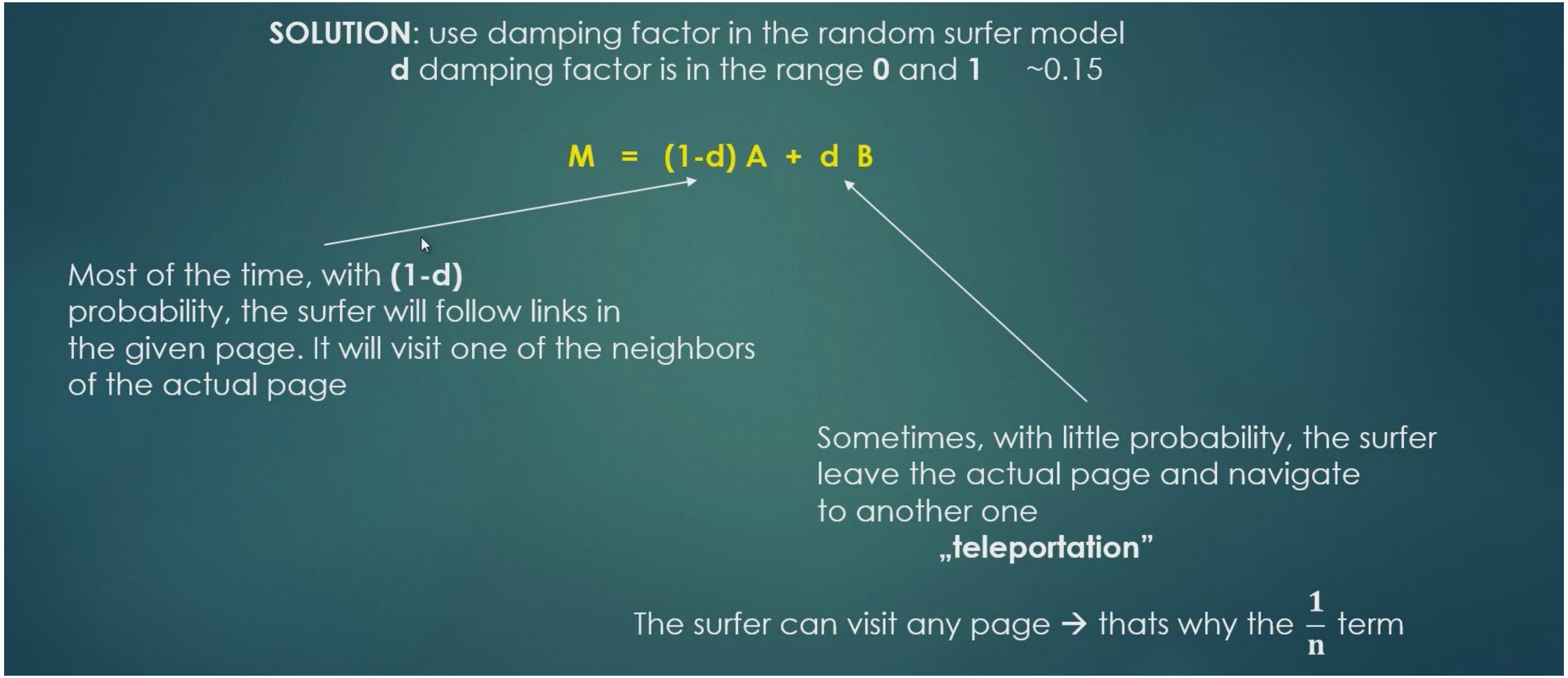
d: damping factor

A: transition matrix

B: identity matrix // items are all **1**

$$\frac{1}{n}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

...



SOLUTION: use damping factor in the random surfer model

d damping factor is in the range **0** and **1**   ~0.15

$$M = (1-d)\,A + d\,B$$

Most of the time, with **(1-d)** probability, the surfer will follow links in the given page. It will visit one of the neighbors of the actual page

Sometimes, with little probability, the surfer leave the actual page and navigate to another one

„teleportation"

The surfer can visit any page → thats why the $\frac{1}{n}$ term

HITS

The **HITS** algorithm (**Hyperlink-Induced Topic Search**) is a link analysis algorithm proposed by Jon Kleinberg to rank webpages based on their importance in relation to a query.

It distinguishes between two types of pages in a network:

- **Hubs**: Pages that link to many other relevant pages.
- **Authorities**: Pages that are linked to by many good hubs.

HITS calculates two scores for each webpage:

- **Hub Score**: Measures how well a page links to authoritative pages.
- **Authority Score**: Measures how well a page is linked to by hub pages.

…

Definitions

Authority: pages that provide an important, trustworthy information on a given topic

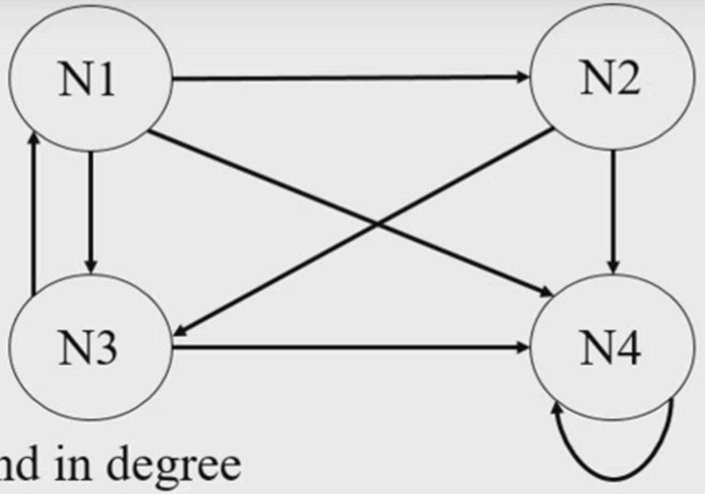
Hubs: pages that contain links to authorities

Indegree: number of incoming links to a given node, used to measure the authoritativeness

Outdegree: number of outgoing links from a given node, here it is used to measure the hubness

HITS based on IN/OUT degree method.

Graph with nodes



Ranks using out degree and in degree

Nodes	Out-degree(Hub)	In-degree(Authority)
N1	3	1
N2	2	1
N3	2	2
N4	1	4

HITS Algorithm: Assignment

Consider,
U = A*V (Hub weight Vector)

V = A^TU (Authority weight Vector)

where, A^T = Transpose of A

Calculate the hubs and authority score for k= 1 , from above graph.

For K=2,

U’ = sqrt(∑(Hub-node)^2)

V’ = sqrt(∑(Authority-node)^2)

Now use U_1 and V_1 as, and rank them again until convergence

[(previous-node U)/U’	for eg:	[8/U’
(previous-node U)/U’		7/U’
(previous-node U)/U’		6/U’
(previous-node U)/U’]		4/U’]