Modern Web Search

Architecture, Algorithms, and Analysis

Image Sources: www.chegg.com, yorko.github.io, linkurious.com, bytebytego.com, fri-datascience.github.io, analytics4all.org, medium.datadriveninvestor.com, en.wikipedia.org

Introduction Part I: Foundations

Scarcity vs. Abundance & Evaluation Metrics

The Information Retrieval Problem

Scarcity vs. Abundance

Classic IR (e.g., library systems) dealt with **scarcity**: finding the one relevant document in a pile.

Web Search deals with **abundance**: finding the *best* document among millions of relevant ones.

Goal: Satisfy user intent (Navigational, Informational, Transactional) with high precision.

🎯Precision

Fraction of retrieved docs that are relevant. Critical for web search (users rarely look past page 1).

Recall

Fraction of relevant docs that are retrieved. Less critical on the web due to redundancy.

The Paradigm Shift

Classic IR (Library Science)

Context: A lawyer searching Case Law.

Problem: Scarcity. There might be only one relevant precedent.

Goal: Find everything. Missing a document is expensive.

Metric: High **Recall**.

Web Search

Context: "Chocolate Cake Recipe".

Problem: Abundance. There are 5 million relevant recipes.

Goal: Find the *best* one immediately.

Metric: High **Precision** (specifically at the top).

Evaluation Metrics

Basic Metrics

Precision = Rel_Retrieved / Total_Retrieved

Recall = Rel_Retrieved / Total_Relevant

Recall is impossible to calculate on the web (we don't know Total_Relevant).

Rank-Aware Metrics

P@K: Precision at top K results (e.g., P@10). "Did I get good stuff on the first page?"

MAP: Mean Average Precision. Rewards placing relevant items higher.

MRR: Mean Reciprocal Rank. For factoid queries (1/Rank of first correct answer).

Partial Relevance

NDCG: Normalized Discounted Cumulative Gain.

- Handles non-binary scores (0=Spam, 5=Perfect).
- Penalizes good docs if they appear low in the list.

Exercise: Calculating Metrics

Scenario

Query: "Jaguar" (Intent: Animal)

Results:

1. Doc A (Car)
2. Doc B (Animal)
3. Doc C (Animal)
4. Doc D (OS)
5. Doc E (Animal)

Calculations

P@1: 0/1 = 0.0

P@3: 2/3 = 0.66

Average Precision (AP):

```
(P@2 + P@3 + P@5) / 3 = (0.5 + 0.66 + 0.60) / 3 = 0.58
```

Introduction Part II:
Web Search Entities

IPs, Domains & DNS

HTML, CSS & JS

The Address Book

IP Address

`142.250.185.78`

The "GPS Coordinate". Machines use this.

Pros: Direct, Fast.

Cons: Hard to remember, can change.

Domain Name

`www.google.com`

The "Business Label". Humans use this.

DNS: The phonebook that translates Domain → IP.

Virtual Hosting: One IP can host 1000 domains.

Domains & DNS

TLD

Top Level Domain. `.com`, `.org`, `.gov`. Hints at the nature of the site.

Subdomain

`blog.site.com`. Often treated as a separate "site" by search engines depending on context.

DNS

Domain Name System. The phonebook of the internet. Resolves `google.com` to IP `142.250.190.46`.

Impact on Crawling: A crawler must resolve DNS for every new host. Efficient DNS caching is critical for performance.

Anatomy of a URL

Uniform Resource Locators (URLs) are the addresses of the web. Understanding their structure is vital for crawling.

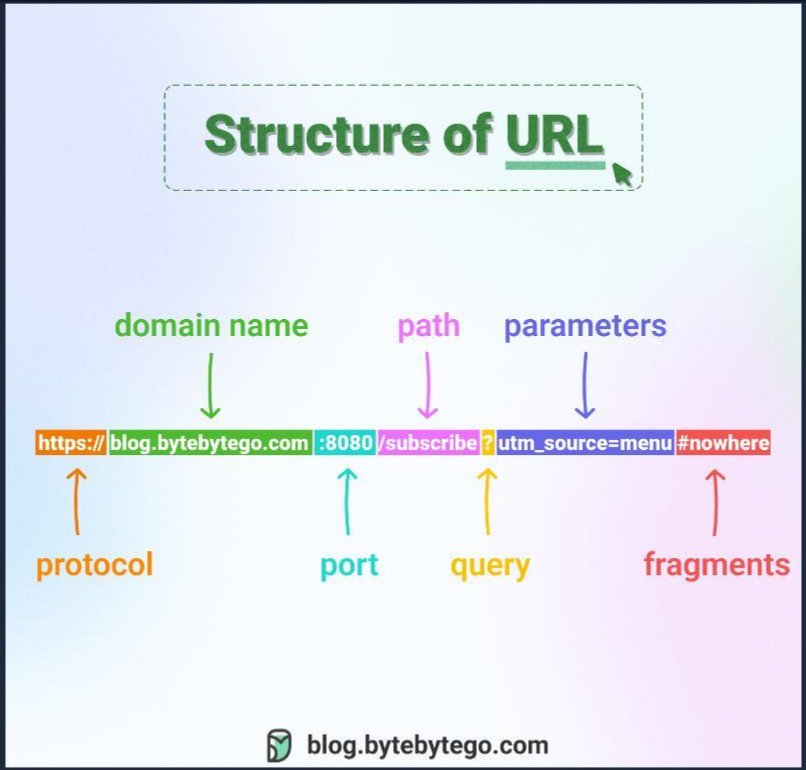
`https://` **Protocol:** How to transfer (HTTP/S).

`www.example.com` **Host/Domain:** The server address.

`/path/to/page` **Path:** Resource location on server.

`?id=123` **Query:** Parameters for dynamic pages.

`#section` **Fragment:** Anchor within the page (not sent to server).



Anatomy of a URL – example

```
https://maps.google.com:443/search?q=paris#center
```

- https:// Protocol (How to talk)
- maps Subdomain (Specific service)
- google.com Domain + TLD (Entity)
- :443 Port (Door number)
- /search Path (File location)
- ?q=paris Query (Parameters)
- #center Fragment (Browser scroll pos - NOT sent to server)

Semi-Structured Data

The web is **semi-structured**. It has tags (HTML), but the text within is free-form.

Unstructured

Plain text, images, video. Hardest to process. Requires NLP and Computer Vision.

Semi-Structured

HTML, XML, JSON. Tags provide clues, but content varies. The standard for web pages.

Structured

Databases, CSVs, Schema.org. Highly organized. Ideal for "rich snippets" (prices, ratings).

The Language of the Web

HTML (Structure)

The Nouns & Verbs.

```
Title Link
```

This is what crawlers care about most.

CSS (Presentation)

The Adjectives.

```
.price { color: green; }
```

Crawlers use this to detect hidden text (spam).

JS (Behavior)

The Actions.

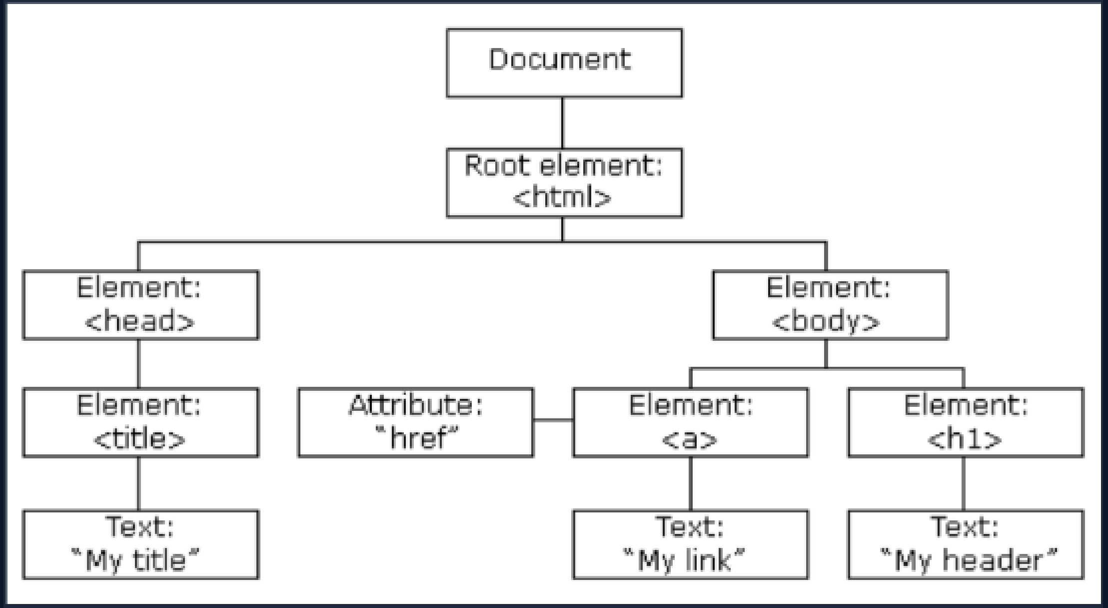
```
btn.onclick = () => {
fetch('/data') }
```

Challenge: Crawlers must execute JS to see content on modern sites.

HTML: The Language of Search

Search engines don't "see" pages like humans. They parse the **DOM** (Document Object Model).

- **Tags:** </div>, provide semantic weight.
- **Links:** are the edges of the web graph.
- **Meta:** provides snippets.



The JavaScript Challenge

Client-Side Rendering

Modern sites (React, Vue) send empty HTML shells.

Content loads via JS.

Simple Crawlers see nothing.

Modern Crawlers must be "Headless Browsers" (e.g., Puppeteer) to execute JS, render the DOM, and *then* extract links.

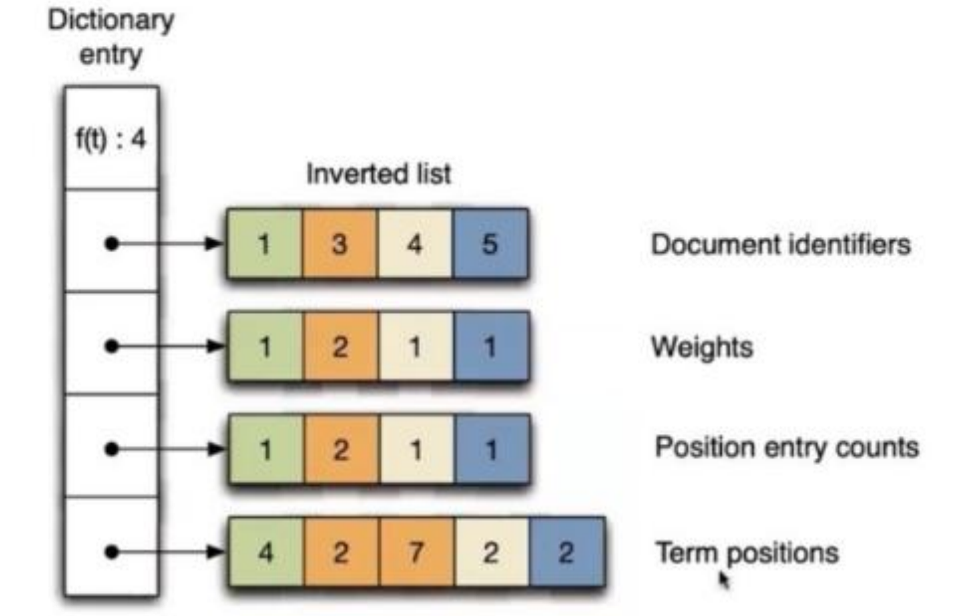
Web Search - Part I
Flow & Architecture

From Classic IR to Distributed Web Search

Classic IR Pipeline

The standard process for static document collections (e.g., legal corpuses).

1. **Tokenization:** Splitting text into words.
2. **Normalization:** Lowercasing, removing punctuation.
3. **Stopword Removal:** Removing "the", "and“ & "is".
4. **Stemming/Lemmatization:** "Running" -> "Run".
5. **Indexing:** Creating the Inverted Index.



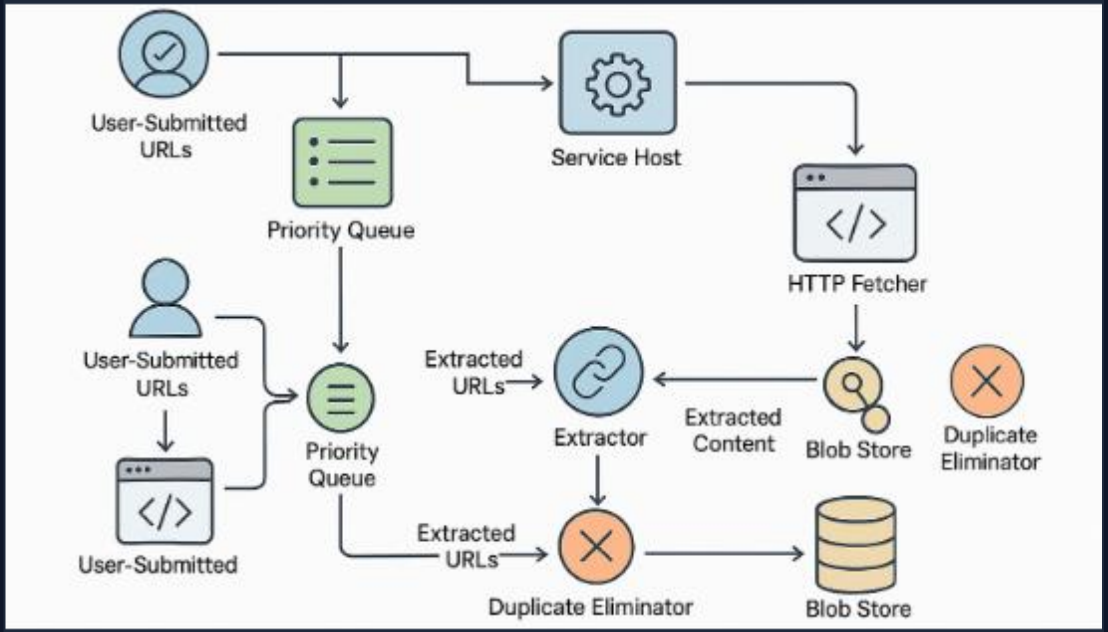
B. Cambazoglu and R. Baeza-Yates, "Scalability Challenges in Web Search Engines"

Web Search Architecture

The Big Picture

The system is cyclic. The crawler feeds the indexer, which feeds the query processor. Analytics feed back into crawling priority.

- **URL Frontier:** The "To-Do" list of links.
- **Fetcher:** Downloads pages (async/multithreaded).
- **Parser:** Extracts links and text.
- **Inverted Index:** Maps words to document IDs.



The Infinite Web

Why is it hard to measure?

- **Dynamic Content:** Calendar pages that generate infinite URLs (/day/1, /day/2...).
- **The Deep Web:** Content behind login screens or unlinked databases.
- **Soft 404s:** Pages that say "Error" but return a "200 OK" status.

~60 Trillion

Known Pages (estimated)

Crawlers must prioritize. You cannot index everything.

Why Distributed?

No single machine can hold the web. We scale horizontally.

Sharding: Splitting the index by Document ID (each machine holds a subset of docs) or by Term (each machine holds a subset of words).

MapReduce: The paradigm invented by Google to process these massive datasets (e.g., counting links, building the index).

Challenges

- Fault Tolerance (Machines die).
- Consistency (Index updates).
- Latency (Querying 1000s of machines in <200ms).

Understanding Intent

Queries are not just keywords; they are goals.

Navigational

"Facebook", "United Airlines"

User wants to go to a specific site. **High Precision** required on the #1 result.

Informational

"Who is the CEO of Apple?", "Symptoms of flu"

User wants to learn. Requires authoritative content.

Transactional

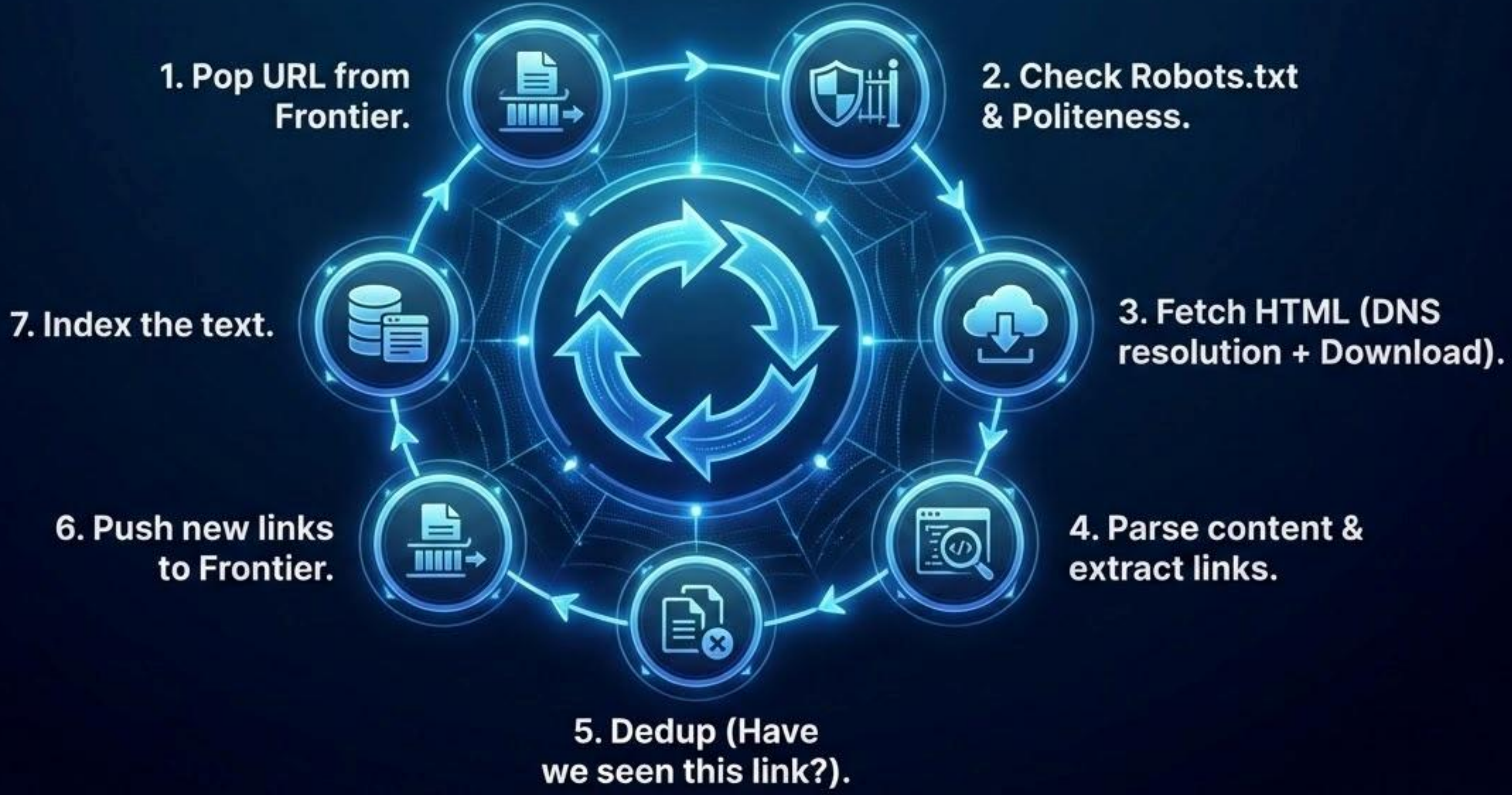
"Buy iPhone 15", "Download WinZip"

User wants to do something. Requires commerce/action links.

Web Search - Part II: Expanding Classic IR to Web Search

The flow, robots.txt & Scraping

The Spider’s Loop



Politeness & Robots.txt

DoS vs DDoS

DoS: Single crawler hammering a server. Crash caused by speed.

DDoS: Distributed crawler (1000 machines) hitting a server at once.

Solution: Crawl-delay and Robots.txt.

```
User-agent: * Disallow: /admin/ Crawl-delay: 5
User-agent: BadBot Disallow: /
```

Web Search Additions

Web search requires distinct components on top of Classic IR.

🕷️The Crawler

Documents don't just "exist". They must be actively discovered and fetched from remote servers.

Link Graph

Maintains the map of links between pages to compute authority scores (PageRank).

Deduping

Detecting mirrors, scraping, and versioning. ~30% of the web is duplicate content.

Scraping vs. Crawling

Crawling (Discovery)

Goal: Discovery & Indexing.

Scope: Broad / The whole web.

Output: A list of URLs and cached text.

Example: Googlebot, Bingbot.

Tool: requests, Scrapy.

Scraping (Extraction)

Goal: Extraction.

Scope: Narrow / Specific sites.

Output: Structured data (prices, names).

Example: Price comparison, Lead gen.

Tool: BeautifulSoup (Static), Selenium (Dynamic).

Static Scraping

The Mechanics

Pros: Very fast, low CPU usage.

Cons: Cannot see content loaded by JavaScript.

Libraries: requests, BeautifulSoup.

```
import requests from bs4
import BeautifulSoup
# 1. Fetch
raw HTML response = requests.get("https://example.com")
# 2. Parse
soup = BeautifulSoup(response.text, 'html.parser’)
# 3. Extract specific element  (e.g. $19.99  price)
price = soup.find('span', class_='price').text
print(price) # Output: $19.99
```

Dynamic Scraping

The Mechanics

Pros: WYSIWYG. Can handle infinite scroll and SPAs (React/Vue).

Cons: Slow. Launches a full browser instance.

Libraries: Selenium, Puppeteer.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

# 1. Launch Browser
driver = webdriver.Chrome()
driver.get("https://example.com/dynamic")

# 2. Wait for JS to render
time.sleep(2)

# 3. Extract content generated by JS
price = driver.find_element(By.CLASS_NAME, "price").text
print(price) driver.quit()
```

Web Search – Part III: Crawling

Crawling, Near-Duplicate Detection

The "Seed" Problem

You can't crawl the web if you don't know where to start. The web graph must be traversed from known entry points.

What makes a good seed?

- **High Out-Degree:** Links to many other pages.
- **Trustworthy:** Not spam.
- **Centrality:** Stable and long-standing.
- **Topical Coverage:** Represents the domain well.

Strategies for Finding Seeds

Directories

DMOZ (archived), BOTW, and niche directories are human-curated lists of quality sites.

Institutions

University homepages (.edu) and Government portals (.gov) are high-trust hubs.

Inverse Links

Use existing indices (like Ahrefs/Majestic) to find who links *to* known authorities.

#Social

Highly shared URLs on Twitter/Reddit often point to fresh, relevant content.

Curated Lists

"Best of" blog posts and Wikipedia "External Links" sections.

HITS

Run HITS on a small set to identify "Hubs", then use those Hubs as seeds for a larger crawl.

Crawling Strategy: BFS

Breadth-First Search

Goal: General Coverage.

Logic: Use a FIFO Queue. Crawl level *d* before level *d+1*.

Why? Prevents getting stuck in "rabbit holes" (infinite calendars or archives) on a single site.

Example

- Seed: Home (d=0)
- Q: [Sports, News] (d=1)
- Visit Sports: Adds [Article A, Article B]
- Q: [News, Article A, Article B]
- Visit News: (Crucial: We visit News *before* Article A).

Code: Simple BFS Crawler

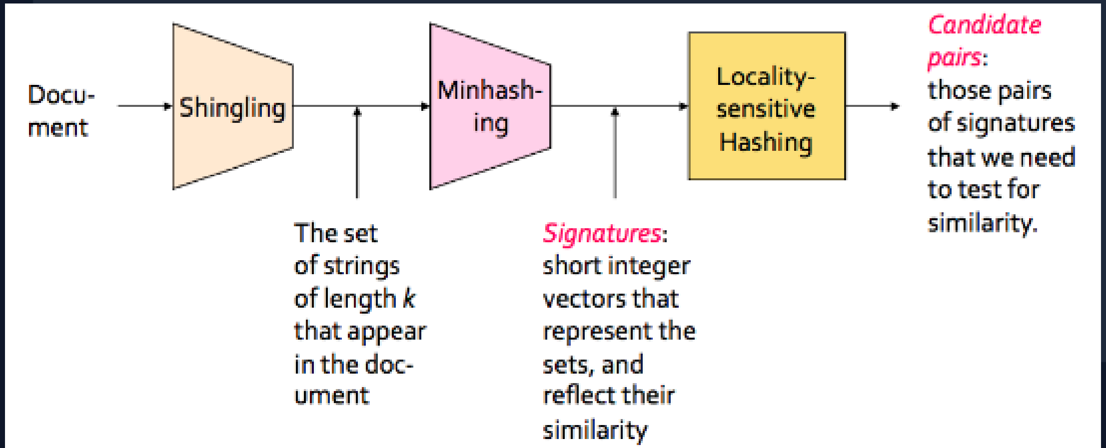
```
def simple_BFS_crawler(start_url, max_pages=10):
    visited = set()
    queue = deque([start_url])
    domain = urlparse(start_url).netloc

    while queue and len(visited) < max_pages:
        url = queue.popleft()
        if url in visited:      continue
        try:
            # 1. Fetch
            response = requests.get(url, timeout=5)

            # 2. Parse
            soup = BeautifulSoup(response.text, 'html.parser’)
            visited.add(url) print(f"Crawled: {url}")

            # 3. Extract Links
            for link in soup.find_all('a', href=True):
                abs_url = urljoin(url, link['href’])
                # Filter: Internal links only
                if urlparse(abs_url).netloc == domain:
                    queue.append(abs_url)
        except Exception as e:
                print(f"Error: {e}")
```

Near-Duplicate Detection

The Problem

Approximately 30% of the web consists of near-duplicates (mirrors, revisions, spam). Exact match detection fails on dynamic timestamps or minor edits.

Exact hash matching (MD5) fails if one-byte changes (e.g., dynamic timestamp).



The Solution: Shingling

Shingles: Convert text into sets of N-grams (e.g., "a rose is a rose").

Jaccard Coefficient: Measure similarity by dividing the intersection of shingle sets by their union.

MinHashing: A technique to approximate Jaccard similarity efficiently at scale.

Near-Duplicate Detection

Jaccard Similarity

Measures overlap between two sets.

Set A: {apple, banana}

Set B: {apple, cherry}

Intersection: {apple} (Size 1)
Union: {apple, banana, cherry} (Size 3)

Jaccard = 1 / 3 ≈ 0.33

Pros: Exact.

Cons: Storing sets is expensive.

MinHash (Approximation)

Goal: Estimate Jaccard with constant space.

1. Create random permutations of vocabulary.
2. For each doc, find the *first* word in the permutation that exists in the doc.
3. If MinHashes match, docs are likely similar.

Insight: $P(h(A) = h(B)) = \text{Jaccard}(A,B)$.

Web Search – Part IV: Link Analysis

Weighted Graphs, PageRank, HITS & Quality

Weighted Graphs in Search

Dijkstra (Shortest Path)

Use Case: Latency Minimization.

Weights: Ping time (ms).

Method: Priority Queue. Always expand the "cheapest" node (*d[v]*).

Result: Find the fastest server to crawl.

Prim (MST)

Use Case: Network Backbone.

Weights: Cost of laying cable/bandwidth.

Method: Grow a single tree. Always add the cheapest edge connecting Tree to Non-Tree.

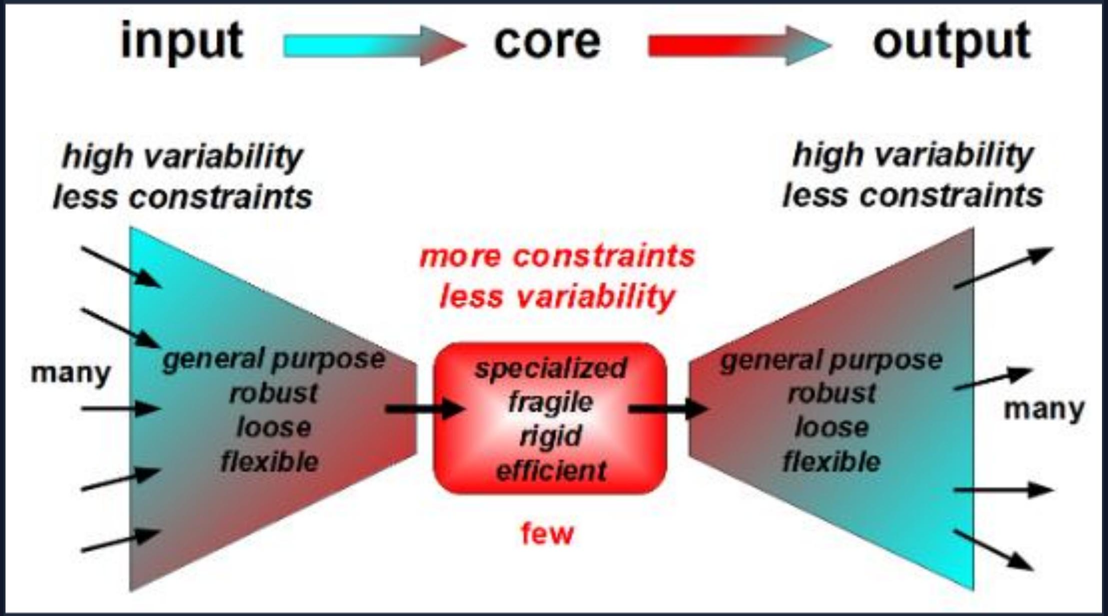
Result: Connect all data centers with min cost.

The Web as a Graph

The web is a directed graph *G=(V, E)*.

- **Vertices (*V*):** Web Pages.
- **Edges (*E*):** Hyperlinks.

Links are not just transport; they are **votes**. A link from A to B is an endorsement of B by A.



The Bow-Tie Structure

The web is not one big blob. It has distinct regions (Broder et al., 2000).

IN

New pages. Can reach SCC but cannot be reached from it.

SCC

Strongly Connected Component. The core. Everyone can reach everyone.

OUT

Corporate sites. Reached from SCC, but link nowhere.

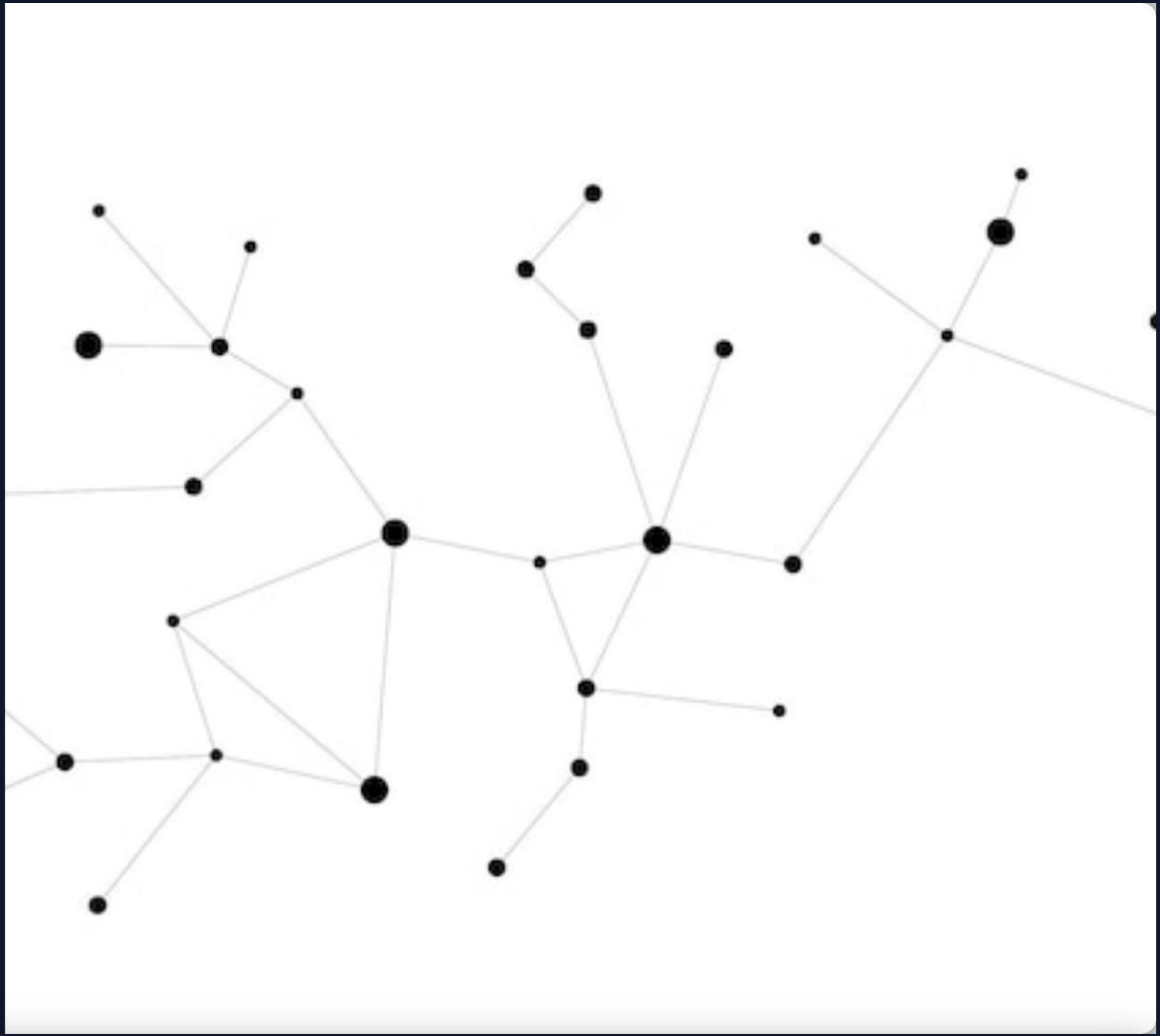
Tendrils

Isolated islands and tubes connecting IN to OUT directly.

The Bow-Tie Structure

The Web is not a purely connected graph. It forms a distinct macroscopic structure:

- **SCC (28%):** Strongly Connected Component. You can navigate from any page to any other.
- **IN (22%):** Pages that link to the SCC but cannot be reached from it (New pages).
- **OUT (22%):** Pages reached from SCC but do not link back (Corporate sites).
- **Tendrils:** Disconnected components and tubes.



Finding the SCC (Kosaraju)

The Algorithm

1. Run DFS on Graph G. Record "Finish Times".
2. Compute Transpose Graph G^T (Flip all arrows).
3. Run DFS on G^T, processing nodes in decreasing order of Finish Times.
4. Each resulting tree is an SCC.

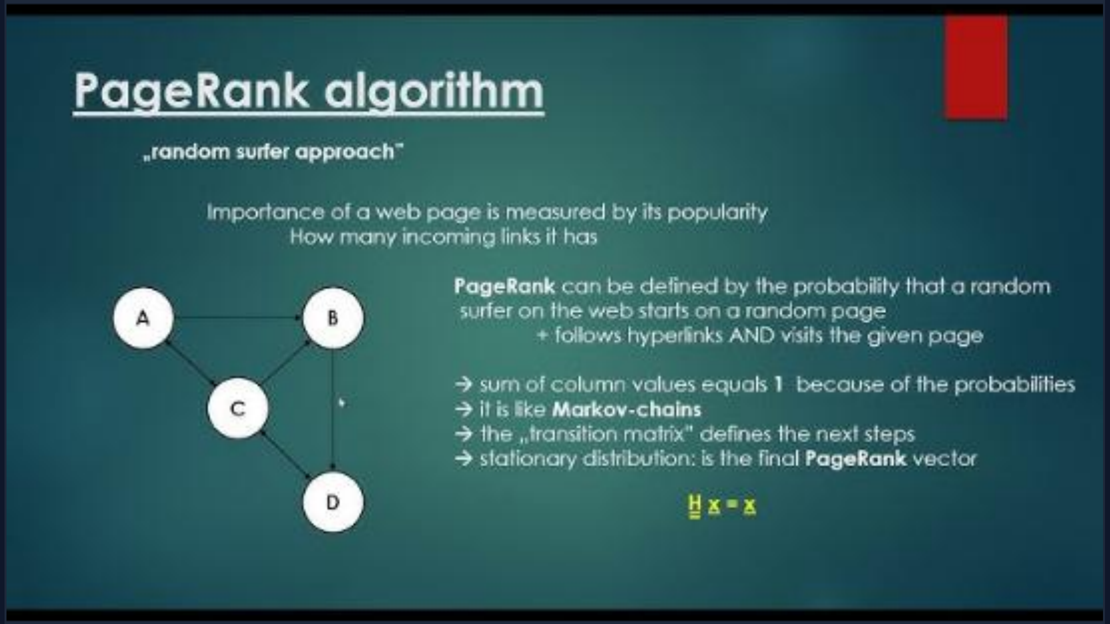
Why Transpose?

By flipping edges, we trap the DFS inside the SCC. It can't "leak out" to other components because the bridges connecting components have been reversed.

PageRank (PR): The Random Surfer

Imagine a surfer clicking random links forever.

PageRank (PR) is the probability that the surfer is on a specific page at any given time.



More incoming links = Higher probability.

Links from high-probability pages = More weight.

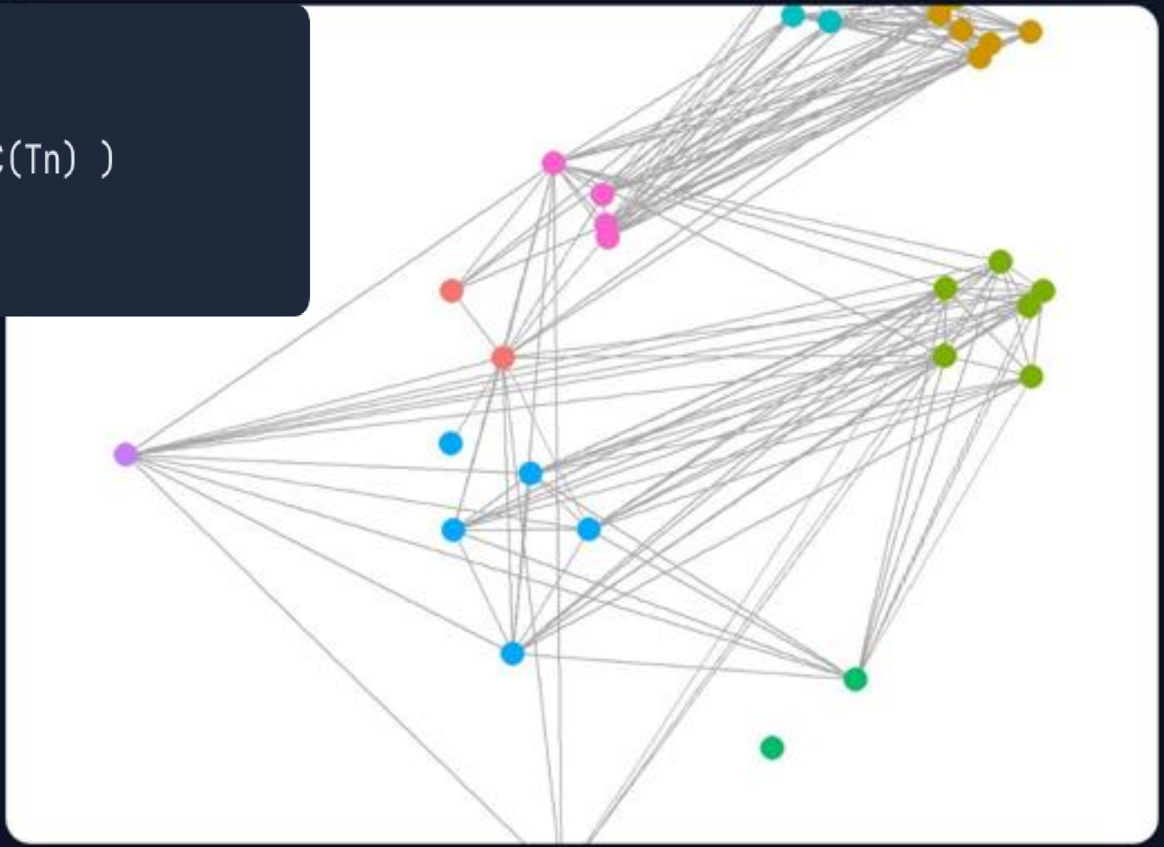
Damping factor (d): Probability of clicking (usually around 0.85).

Teleportation (1-d): Prevents getting stuck in dead ends (Sink nodes).

Page Rank - The Formula

Recursive definition with Damping Factor.

```
PR(A) = (1-d) + d * ( PR(T1)/C(T1) + ... + PR(Tn)/C(Tn) )
```



- **PR(A):** PageRank of page A.
- **d:** Damping factor (~0.85). Probability of clicking.
- **(1-d):** Teleportation. Probability of jumping to a random page (solves dead ends/sinks).
- **$C(T_n)$:** Count of outbound links from page T_n.

PageRank Calculation

Scenario

Graph: Node A links to Node B (A -> B).

Damping (d): 0.85 (Prob. of clicking).

Teleport (1-d): 0.15 (Prob. of jumping).

$$PR(A) = (1-d) + d * \Sigma\, (PR(T)/C(T))$$

Step 1: Calculate PR(A)
Incoming Links: None.

PR(A) = (1-d) = 0.15

Step 2: Calculate PR(B)
Incoming Links: A.

PR(B) = (1-d) + d * (PR(A) / OutDegree(A)) =
= 0.15 + 0.85 * (0.15 / 1) =
= 0.15 + 0.1275 =
= 0.2775

Result: B is more important than A.

PageRank

The Random Surfer

PR represents the probability of a random user landing on a page.

HITS (Alternative)

Hubs: Good lists of links.

Authorities: Good content.

Mutually reinforcing relationship (Recursive).

HITS: Hubs & Authorities

Proposed by Kleinberg. Query-dependent.

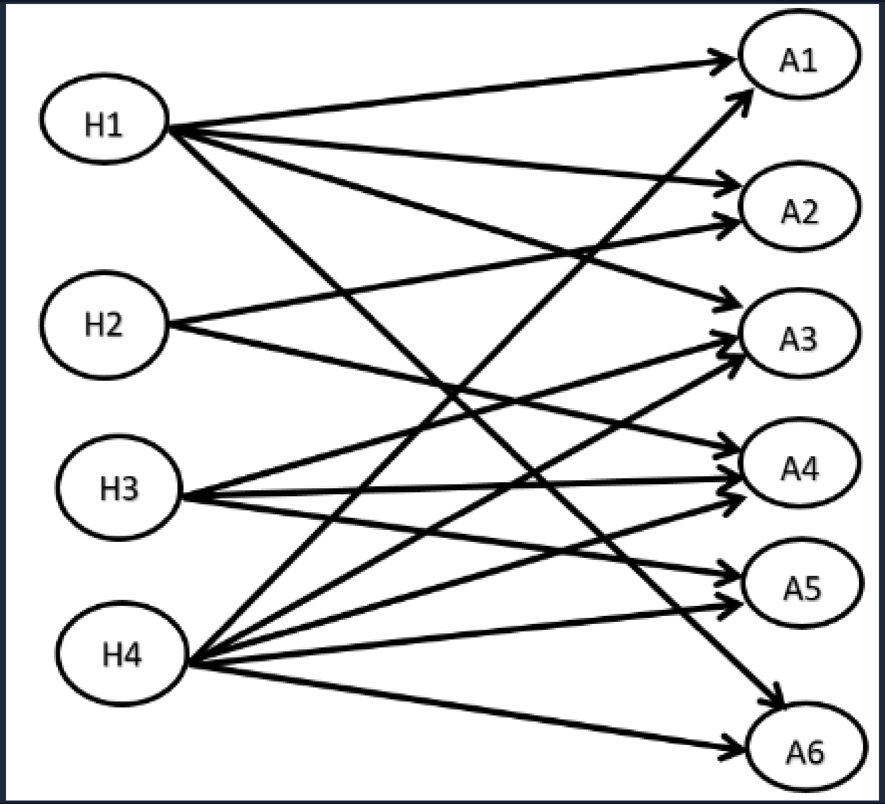
Authorities: Pages with good content (linked to by many hubs).

Hubs: Pages with good lists of links (link to many authorities).

Mutually reinforcing:

Good hubs point to good authorities.

Good authorities are pointed by good hubs.



HITS Calculation

Scenario

Graph: Hub H points to Auth A1 and Auth A2.

Goal: Find best Hubs/Auths.

Start: All scores = 1.

Iter 1: Update Authorities (Sum of Hubs)
Auth(A1) = Hub(H) = 1
Auth(A2) = Hub(H) = 1

Iter 1: Update Hubs (Sum of Auths)
Hub(H) = Auth(A1) + Auth(A2) = 1 + 1 = 2

Result: Hub H score grew because it points to valid Auths.

The Power of Anchor Text

Descriptive Power

Anchor text (the clickable text in a link) often provides a better description of the target page than the page itself.

- **Example:** Many people link to the IBM home page using the text "computer giant", even if the page itself only says "International Business Machines".
- **Google Bombs:** Historically used to manipulate rankings by mass-linking with specific phrases (e.g., "miserable failure").

Quality Signals

TrustRank

Start with a seed set of trusted pages (universities, news).

Propagate trust down the link graph.

Spam is usually far from trust.

Freshness

How often to recrawl?

News: Minutes.

Blogs: Daily.

Static: Monthly.

Use adaptive crawl schedules based on change history.

Web Search – Part V: Spam Filter

The Spam Arms Race

If ranking = money, people will cheat.

Content Spam

Keyword stuffing, hidden text, scraped content, auto-generated gibberish.

Link Spam

Link farms, paid links, comment spam, buying expired domains.

Cloaking

Showing search engines one version of a page and users another (e.g., porn/gambling).

Generative vs. Discriminative

Discriminative (Decision Boundary)

Logic: "Draw a line between Cats and Dogs."

Examples: KNN, SVM, Neural Networks.

Model: $P(Y|X)$ directly.

Generative (Probability Model)

Logic: "Learn what a Dog looks like. Learn what a Cat looks like. Compare new image to both."

Examples: Naive Bayes, LDA.

Model: $P(X|Y)$ and $P(Y)$.

Bayes' Theorem & Naive Assumption

$$P(Class|Data) = [P(Data|Class) * P(Class)] / P(Data)$$

Key Terms

- Posterior: $P(C|D)$ - What we want.
- Likelihood: $P(D|C)$ - Prob. of data given class.
- Prior: $P(C)$ - Baseline prob. of class.
- Evidence: $P(D)$ - Constant (can ignore).

The "Naive" Assumption

Features are **independent** given the class.

```
P(Words | Spam) ≈ P(Word1 | Spam) *
P(Word2 | Spam)...
```

This turns a complex joint probability into simple multiplication.

Spam Filtering: The Setup

Training Data

We analyze existing emails to calculate probabilities.

Priors:

- P(Normal page) = 0.625
- P(Spam page) = 0.375

Likelihoods P(Word|Class)

Word	Normal	Spam
Moodle	0.38	0.12
Grade	0.30	0.25
Test	0.23	0.00
Money	**0.07**	**0.63**

Classifying: "Money Grade"

Score for Normal (N)

$P(N) * P(Money|N) * P(Grade|N)$

$= 0.625 * 0.07 * 0.30$

$= 0.013$

Score for Spam (S)

$P(S) * P(Money|S) * P(Grade|S)$

$= 0.375 * 0.63 * 0.25$

$= 0.059$

Result: SPAM (0.059 > 0.013)

Practical Issues & Solutions

1. Underflow

Multiplying many small probabilities results in computer zero.

Solution: Log Probabilities

$$log(a \cdot b) = log(a) + log(b)$$

We sum logs instead of multiplying.

2. Zero Frequency

If a word (e.g., "Test") never appears in Spam, probability becomes 0, wiping out the whole score.

Solution: Smoothing (Laplace)

Add 1 to all counts so no probability is ever exactly zero.

Summary

We have traversed the stack from the physical **Infrastructure** (IPs/DNS), through the **Crawling** layer (BFS/Politeness), modeled the **Web Graph** (SCC/Bow-Tie), applied **Link Analysis** (PageRank), and finally implemented **Machine Learning** (Naive Bayes) to classify content.