Design A Web Crawler

Understand the problem

A web crawler is known as a robot or spider. It is widely used by search engines to discover new or updated content on the web. Content can be a web page, an image, a video, a PDF file, etc. A web crawler starts by collecting a few web pages and then follows links on those pages to collect new content.

Crawlers are used in many real-world applications:

Search Engine Indexing
Search engines like Google, Bing, and DuckDuckGo use crawlers to visit and index billions of pages across the internet, helping users quickly find the most relevant results.

Market & Competitor Research
Brands and analytics companies use crawlers to:

- Monitor competitor websites for new products, features, or pricing
- Collect product reviews or announcements
- Power price comparison tools (e.g., for flights or hotels)

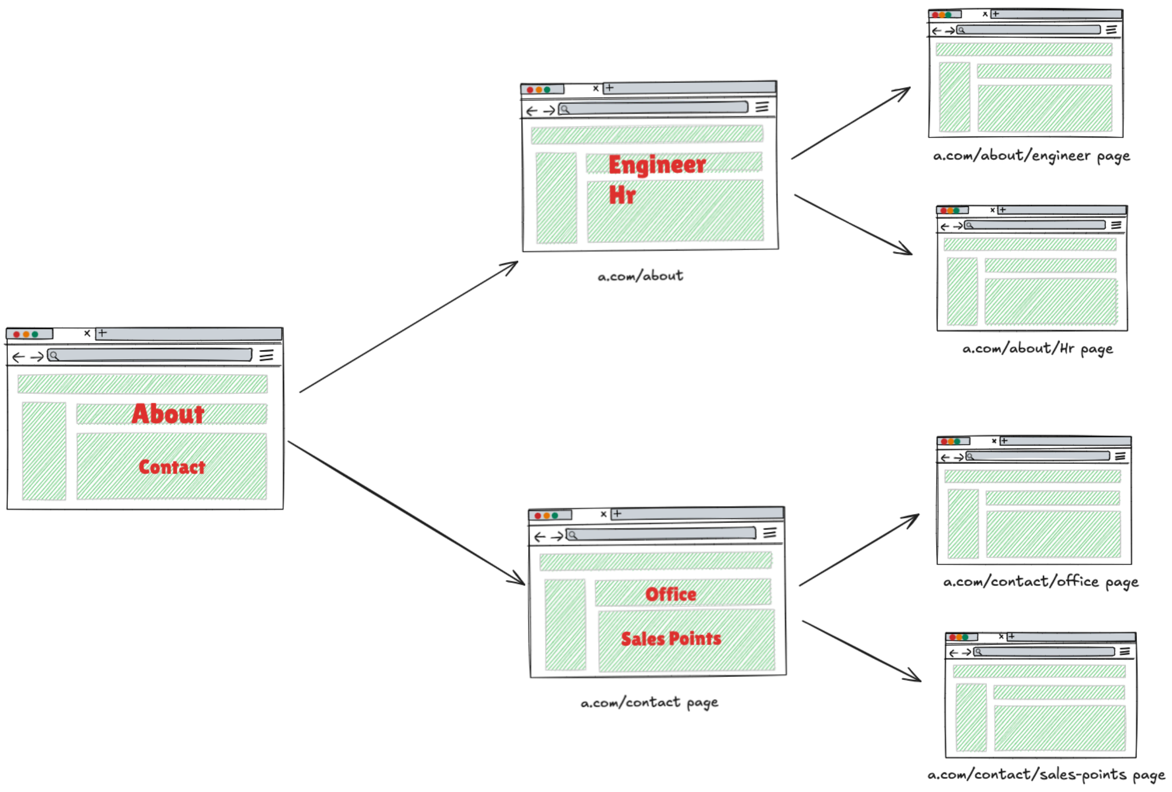
Content Archiving
Organizations like Web Archive or national libraries crawl websites to save copies for historical or legal purposes:

- Preserves content even if a website is taken down or changed
- Useful for research, digital preservation, or audits

How Crawling Works (Simplified Algorithm)

1. Start with a few seed URLs
2. Download each page
3. Extract links from the page
4. Add new links to the list of pages to visit
5. Repeat the process

This loop continues until there are no more new URLs—or until limits like crawl depth or page count are reached.



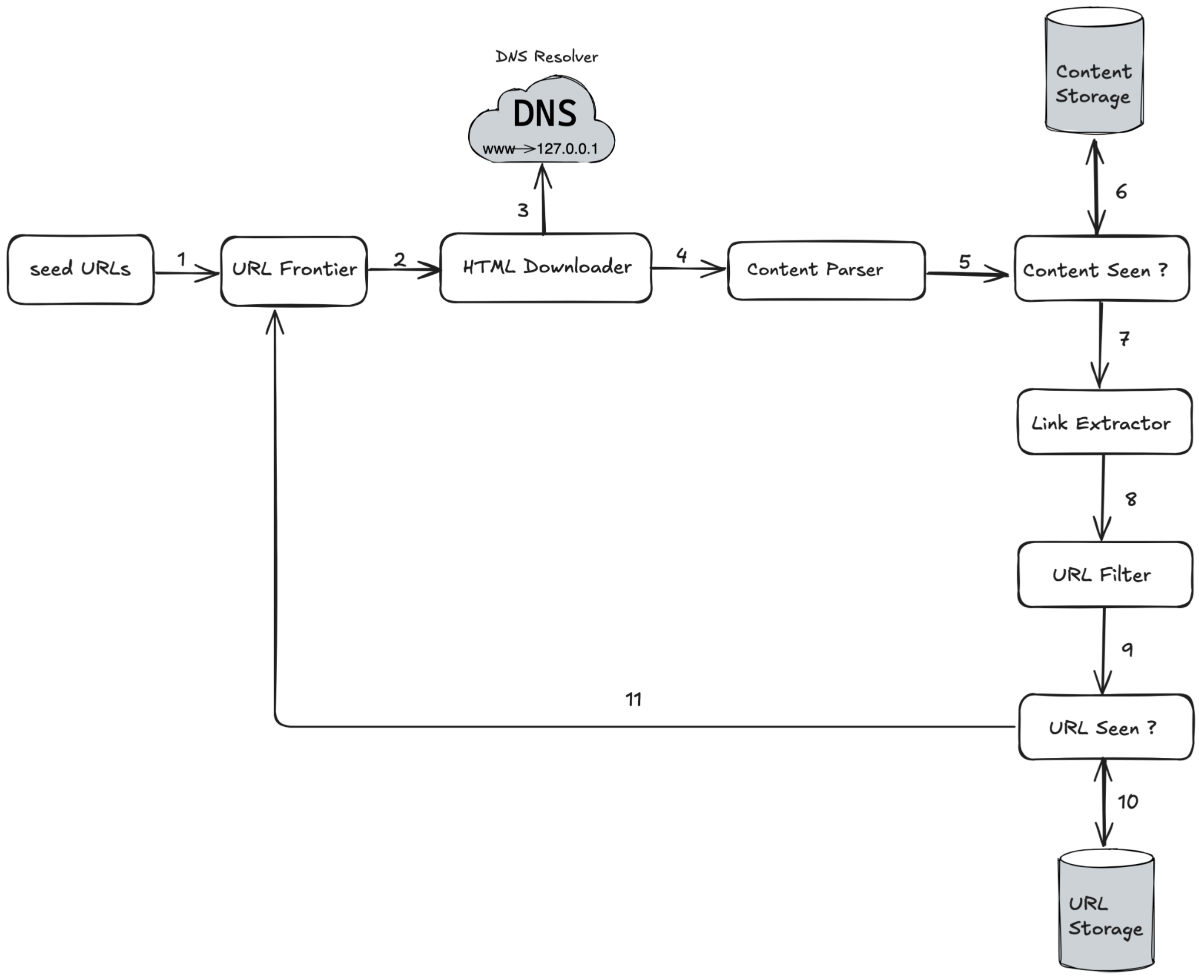
Design Scope Clarification Questions

- What is the main purpose of the crawler?
- How many web pages does the web crawler collect per month?
- What content types are included? HTML only or other content types such as PDFs and images as well?
- Shall we consider newly added or edited web pages?
- Do we need to store HTML pages crawled from the web?
- How do we handle web pages with duplicate content?

Non-Functional Requirements

• Scalability:
The internet is massive — with billions of pages. A good crawler should scale well and be able to handle large volumes efficiently. This can be done using techniques like parallel processing.
• Robustness:
A crawler must be resilient to these edge cases — able to catch errors gracefully and keep going.(Bad HTML, unresponsive servers, crashes, malicious links)
• Politeness:
The crawler should not make too many requests to a website within a short time interval. Making too many requests in a short time can overwhelm websites or get the crawler banned.
• Extensibility:
A well-designed crawler should be easy to update. For example, if support for crawling new content types like PDFs, images, or videos is needed in the future, it should only require small code changes, not a full redesign.

High Level Design



Step 1: Add seed URLs to the URL Frontier
Step 2: HTML Downloader fetches a list of URLs from URL Frontier.
Step 3: HTML Downloader gets IP addresses of URLs from DNS resolver and starts downloading.
Step 4: Content Parser parses HTML pages and checks if pages are malformed.
Step 5: After content is parsed and validated, it is passed to the “Content Seen?” component.
Step 6: “Content Seen” component checks if a HTML page is already in the storage.
• If it is in the storage, this means the same content in a different URL has already been processed. In this case, the HTML page is discarded.
• If it is not in the storage, the system has not processed the same content before. The content is passed to Link Extractor.
Step 7: Link extractor extracts links from HTML pages.
Step 8: Extracted links are passed to the URL filter.
Step 9: After links are filtered, they are passed to the “URL Seen?” component.
Step 10: “URL Seen” component checks if a URL is already in the storage, if yes, it is processed before, and nothing needs to be done.
Step 11: If a URL has not been processed before, it is added to the URL Frontier.

Seed URLs:
These are the starting points for the crawler. Seed URLs as the first pages the crawler visits. From there, it follows links to discover more pages across the web. A well-chosen seed URL leads to many other useful pages, so it's important to start with good ones.

URL Frontier:
The URL Frontier is where the crawler keeps track of pages it hasn't visited yet. It's usually implemented as a FIFO queue — the first URL added is the first one crawled. As the crawler discovers new links, they get added to this queue for future crawling.

HTML Downloader:
This component is responsible for actually fetching the web pages. It takes a URL from the frontier and makes an HTTP request to download the page's HTML content.

DNS Resolver:
To download a web page, a URL must be translated into an IP address. The HTML Downloader calls the DNS Resolver to get the corresponding IP address for the URL.

Content Parser:
After downloading a page, it needs to be parsed — turning raw HTML into a structured format the crawler can work with. Parsing also validates the content to avoid storing broken or junk pages.

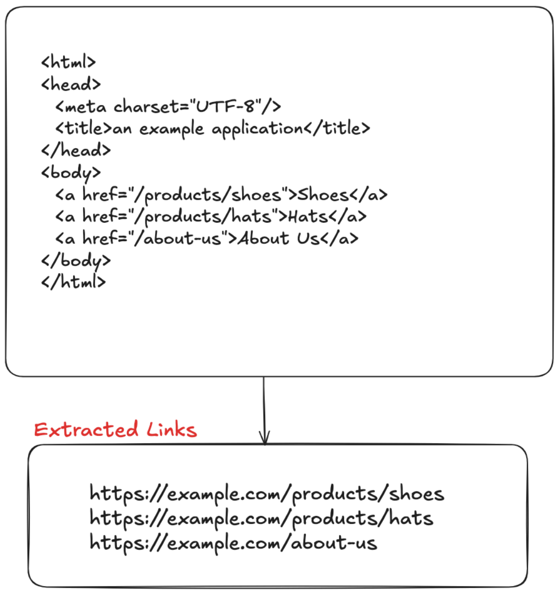
Content Seen? (Content Deduplication) :
Not all pages are unique. Some websites serve the same content under different URLs. To avoid storing duplicates, crawlers compare the hash of a page (a fingerprint of the content) instead of comparing it character by character — which would be too slow at scale.

Content Storage:
This is where the crawler saves downloaded HTML pages. Storage is typically split between:

- Disk: Used for storing most of the content, since it's cheaper and can hold large volumes.
- Memory: Used for frequently accessed content to speed things up.

The right balance between memory and disk depends on how often content is needed and how large the dataset is.

URL Extractor:
Once a page is downloaded and parsed, this component scans the HTML and pulls out all the hyperlinks. These links are then normalized and added to the URL Frontier to be crawled next — completing the cycle.


```
<html>
<head>
  <meta charset="UTF-8"/>
  <title>an example application</title>
</head>
<body>
  <a href="/products/shoes">Shoes</a>
  <a href="/products/hats">Hats</a>
  <a href="/about-us">About Us</a>
</body>
</html>
```

```
https://example.com/products/shoes
https://example.com/products/hats
https://example.com/about-us
```

URL Filter:
Before adding a URL to the crawl queue, the URL Filter checks whether it should be skipped. This includes:

- URLs with unwanted file types (e.g., .pdf, .zip, .exe)
- Links known to return errors or redirects
- URLs from blacklisted or disallowed domains

This filtering step helps reduce unnecessary requests and focuses the crawler on relevant content

URL Seen?:
To avoid crawling the same page more than once, the crawler keeps a record of already seen URLs. This check is done before adding a URL to the frontier.
It prevents:

- Redundant crawling
- Unnecessary load on websites
- Infinite loops due to circular links

URL Storage:
This is a persistent store where all visited URLs are saved. It acts as a history log and can be useful for:

- Auditing what has been crawled
- Avoiding re-crawling the same URLs in future runs
- Analyzing crawl coverage over time