PROBLEM STATEMENT

01
Simple web search engine for indexing & searching HTML pages.

02
We have used Python for development. Python is well known for its simple syntax and strong support for main operating systems.

03
Tarantula lets users search for their query from a large set of documents and the results are displayed according to their ranks. Tarantula uses PageRank Algorithm.

04
Algorithms for full-text searches are among the most important collective intelligence algorithms.

05
It is widely believed that Google's rapid rise from an academic project to the world's most popular search engine was based largely on the Page Rank algorithm.

PROPOSED SYSTEM

We propose to build a web application for a search engine. This includes PC as well as mobile devices.
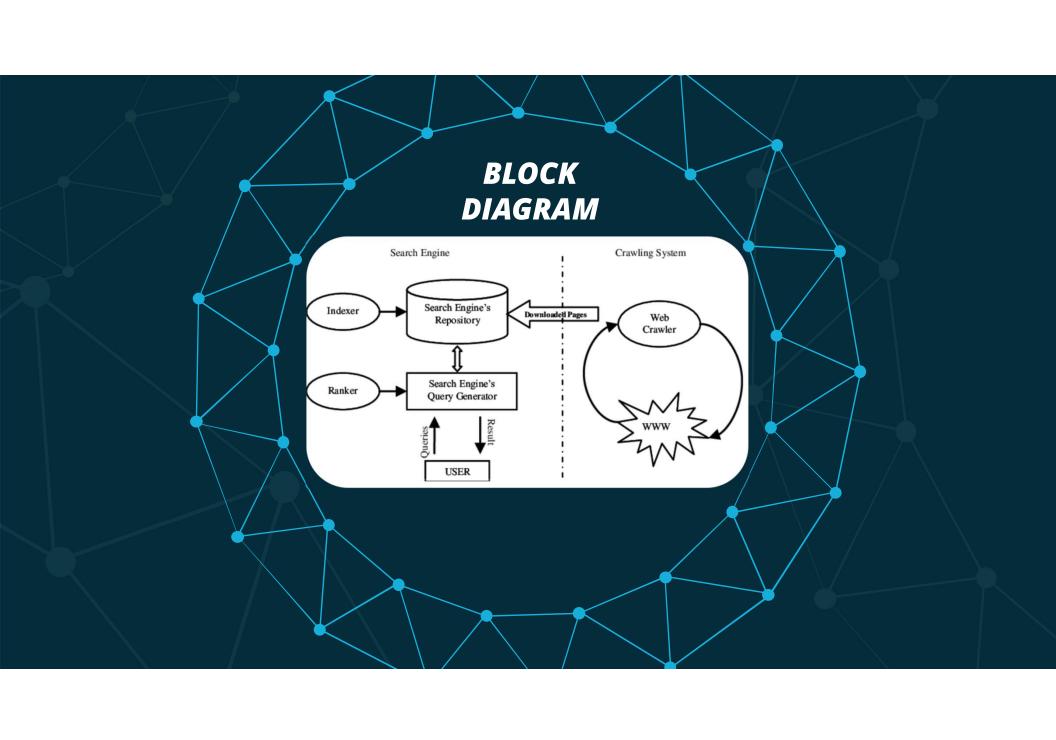
Extracted data is stored in the database and then pre-processed and indexed.

First we will build our custom Web Crawler which will crawl the WWW recursively in a DFS manner.

Pages are assigned scores using metrics like location score, frequency score, inbound link score, distance score, & PageRank score.

BLOCK DIAGRAM

PROBLEMS FACED

Crawling & Indexing

Crawling is a time consuming process & requires a lot of bandwidth

01

Storage Issue

Storing the crawled data

02

Ranking

Different algorithms for ranking and it is a time consuming process

03

Problems

SOLUTION TO PROBLEMS

Solution

01

Crawling & Indexing

Parallel Crawling and threading can be used

02

Storage Issue

Faster Disk read write disk is required to stored crawled data

03

Ranking

Ranking Can be improved by Increasing the number of iterations

IMPLEMENTATION

FRONT-END

CSS

AJAX

HTML

JavaScript

BACK-END

Python

BeautifulSoup

Flask
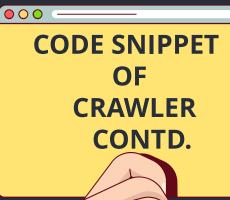
Selenium
WebDriver

SQLite

CODE SNIPPET
OF
CRAWLER

```python
def crawl(self,pages,depth=1):
        options = webdriver.ChromeOptions()
        options.add_experimental_option("prefs", prefs)
        driver = webdriver.Chrome(options=options)
        driver.set_page_load_timeout(75)
        driver.implicitly_wait(75)
        for i in range(depth):
            newpages = set()
            for page in pages:
                try:
                    driver.get(page)
                    c = driver.page_source
                except:
                    print("Couldn't open page : {}".format(page))
                    driver.close()
                    driver = webdriver.Chrome(options=options)
                    driver.set_page_load_timeout(75)
                    driver.implicitly_wait(75)
                    continue
                soup = BeautifulSoup(c,'html.parser')
                for script in soup(["script", "style"]):
                    script.decompose()
                self.addToIndex(page,soup)
                links = soup('a')
                for link in links:
                    if ('href' in dict(link.attrs)):
                        url = urljoin(page,link['href'])
```

CODE SNIPPET OF CRAWLER CONTD.

```python
        if url.find("'")!=-1 or url[0:11] == 'javascript' or url[0:3] == 'tel'
            continue

    url = url.split('#')[0]
    url = url.split('%')[0]


    if url[0:4] == 'http' and not self.isIndexed(url):
        newpages.add(url)
        pages_file = open('pages','wb')
        newpages_file = open('newpages','wb')
        pickle.dump(newpages, newpages_file)
        pickle.dump(pages, pages_file)
        newpages_file.close()
        pages_file.close()
    linkText=self.getTextOnly(link)
    self.addLinkRef(page,url,linkText)
images = soup('img')
for image in images:
    alt = ''
    src = ''
    title = ''
    if ('src' in dict(image.attrs)):
        imageUrl=urljoin(page,image['src'])
        imageUrl=imageUrl.split('?')[0]
        imageUrl=imageUrl.strip()
        if imageUrl[0:5] == 'data:':
            continue
```

CODE SNIPPET OF CRAWLER CONTD.

```python
                    if ('alt' in dict(image.attrs)):
                        alt=image['alt']
                        alt = alt.replace("'", ' ')
                        alt = alt.replace('"', " ")
                        alt = alt.strip()
                    if ('title' in dict(image.attrs)):
                        title=image['title']
                        title = title.replace("'", ' ')
                        title = title.replace('"', " ")
                        title = title.strip()
                    # print(f'Image- {imageUrl}')
                    urlid = self.getEntryId('urllist','url',page)
                    insertImageQuery = f"INSERT INTO images(siteurlid, imageurl, alt, title) VALUES ('{urlid}','{imageUrl}','{alt}','{title}')"
                    # print(insertImageQuery)
                    self.con.execute(insertImageQuery)
            self.dbcommit()
        pages=newpages
driver.close()
```

```python
def calculatePageRank(self,iterations=100):
    # clear out the current PageRank tables
    self.con.execute('DROP TABLE IF EXISTS pagerank')
    self.con.execute('CREATE TABLE pagerank(urlid PRIMARY KEY,score)')

    # initialize every url with a PageRank of 1
    self.con.execute('INSERT INTO pagerank SELECT rowid, 1.0 FROM urllist')
    self.dbcommit()

    for i in range(iterations):
        print ("Iteration %d" % (i))
        for (urlid,) in self.con.execute('SELECT rowid FROM urllist'):
            pr=0.15
            # Loop through all the pages that link to this one
            for (linker,) in self.con.execute(f'SELECT DISTINCT fromid FROM link WHERE toid={urlid}'):
                # Get the PageRank of the linker
                linkingpr = self.con.execute(f'SELECT score FROM pagerank where urlid={linker}').fetchone()[0]
                # Get the total number of links from the linker
                linkingcount = self.con.execute(f'SELECT COUNT(*) FROM link WHERE fromid = {linker}').fetchone()[0]
                pr += 0.85*(linkingpr/linkingcount)
            self.con.execute(f'UPDATE pagerank SET score = {pr} WHERE urlid = {urlid}')
        self.dbcommit()
```

APPLICATIONS / FEATURES

Web Search
This feature provides websites and URLs as search results.

Image Search
This feature populates all images based on the user query the results are displayed according to descending order of clicks

Torrent Search
This search enables the user to search all popular and trending torrents .The torrents are displayed according to seeders and leechers count.
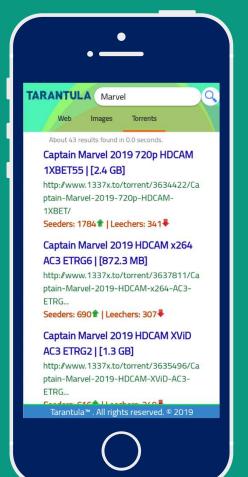
Voice Typing
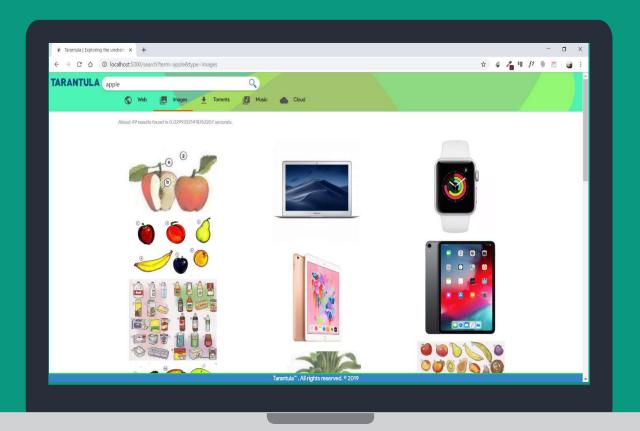This feature helps user to input search query by his speech

I'm Feeling Lucky
This feature redirects the user to website having the highest rank.

RESULTS

MOBILE VIEW

LAPTOP VIEW

Convenient

This system is easy to use and is very simple to handle.

Dynamic

As it processes data based on the crawled data and this crawled data is provided as input for ranking and accuracy can be improved by increasing the number of iterations in Page Rank Algorithm.

01   02

04   03

ADVANTAGES

Effective

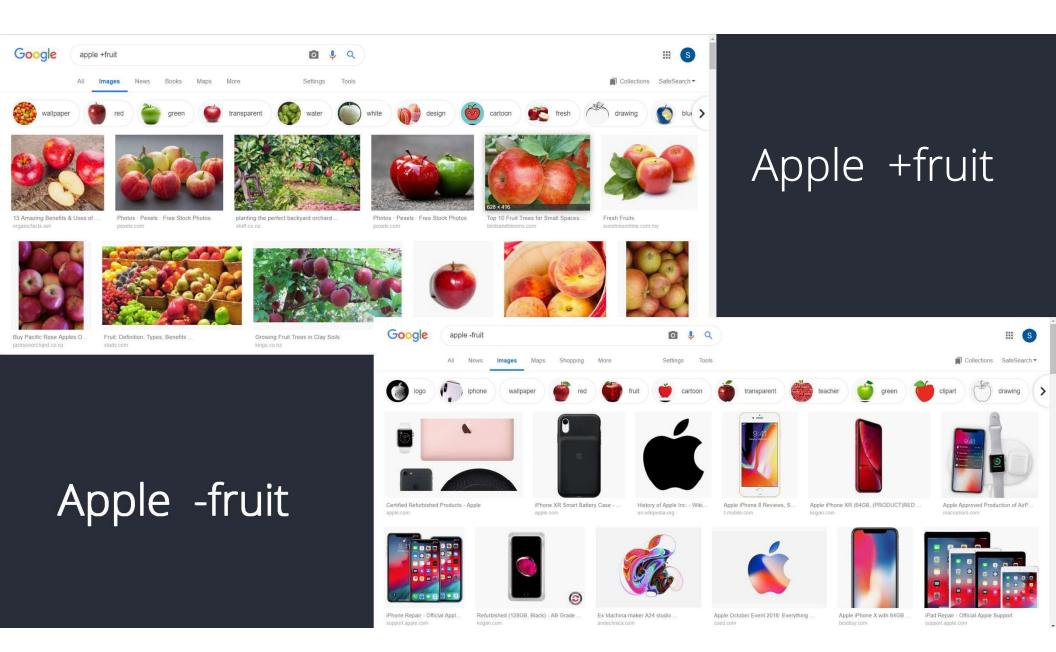Ranking and Indexing is very effective as it helps the user to search query easily.

Flexible

It is quite flexible and can be run on any system.

Apple +fruit

Apple -fruit

THANK YOU FOR
YOUR ATTENTION!

OUR
TEAM MEMBERS

Hemendra M. Naik
Web Developer

About Me:
"Well-versed in technology and writing code to create systems that are reliable and user-friendly."

Shubham Vaity
Web Developer

About Me:
"Solid background in managing interactive elements of websites to create a fulfilling user experience based upon client guidelines."