Activity based

Report on

Artificial Intelligence

Submitted to Vishwakarma University, Pune

Under the Initiative of

Artificial Intelligence Project 2

By

Student Name: Atharva Sanjay shevate

SRN No: 202201727

Roll No: 02

Div: E

Third Year
Engineering

Faculty In-charge: Prof. Richa Agnihotri

Department of Computer Engineering

Faculty of Science and Technology

Academic Year 2024-

2025

Web Crawling and Page Indexing using Breadth First Search:

Problem Statement:

Implement the web crawler using the Breadth First Search algorithm to systematically traverse and index web pages. Consider techniques to avoid redundant crawling, handle dynamic content loading, and respect the directives outlined in the robots.txt file. Create an intuitive graphical interface for the AI-powered web crawler. The GUI should allow users to input the seed URL, visualize the BFS exploration process, and display information about indexed pages. Implement features like progress tracking, a list of indexed pages, and options for users to pause or stop the crawling process. Experiment with different websites to evaluate the crawler's adaptability and assess the usability of the GUI in different scenarios.

Objectives:

- To design and implement an **AI-powered web crawler** using the **Breadth First search**
- To develop a **graphical user interface (GUI)** that allows user interaction with the crawler.
- To systematically **index web pages** starting from a seed URL while respecting robots.txt.
- To provide features such as **progress tracking**, **URL visualization**, **pause/resume/stop** crawling, and **save indexed URLs**.
- To handle **dynamic content** and prevent **redundant crawling**.
- To evaluate crawler **adaptability** across different websites and assess GUI **usability**.

Implementation Plan:

- **Design a user-friendly GUI** with controls to input seed URL, start/pause/stop
- **Initialize data structures** like a queue for BFS, a set for visited URLs, and a list for storing indexed pages.
- **Validate and normalize URLs** to ensure consistent and secure crawling (http/https only).
- **Implement robots.txt parsing** using RobotFileParser to respect site crawl rules.
- **Develop BFS crawling logic** to fetch pages level-by-level and extract all valid links.
- **Handle exceptions and timeouts** to avoid crawler crashes from broken or slow pages.

- **Update GUI dynamically** with crawled pages, logs, and progress bar in real-time using threading.
- **Add functionality to pause/resume/stop crawling** without freezing the interface.
- **Implement save feature** to export the list of indexed URLs to a local .txt file for analysis.

1] Methodology:

Identify Dataset:

- **Dynamic source**: Live web pages from the internet.
- The dataset is **collected during the crawl**, starting from a **seed URL**.

Preprocess Dataset:

- Normalize URLs.
- Filter out non-HTML resources.
- Parse and respect robots.txt.
- Avoid revisiting duplicate URLs.

Implement Algorithm:

- **BFS Traversal**: Simulated via a queue structure (though current code uses BFS with a stack, you'll convert it to a queue for actual BFS).
- Crawl and index pages in a level-order manner.
- Extract links from crawled pages using BeautifulSoup.

Implementation of GUI:

- GUI developed with tkinter and enhanced using ttkbootstrap.
- Features:
    - URL input
    - Start/Pause/Stop buttons
    - Indeterminate progress bar
    - Indexed URL display
    - Logs tab for crawling status

- Save to file functionality
- Interactive link opening
- Real-time hover preview

Verify Output with Expected Output:

- URLs should load in a browser and be valid.
- URLs from robots.txt disallowed paths should not appear.
- No duplicate entries should be present.
- GUI should reflect real-time crawling progress and activity.

Validation and Testing:

- Evaluate with diverse websites (static and dynamic).
- Monitor performance and crawling coverage.
- Manually verify robots.txt compliance.
- Test pause/resume functionality and edge case handling (timeouts, broken links).

3| Source Code:

```
import tkinter as tk
import threading
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse
from urllib.robotparser import RobotFileParser
import time
import webbrowser
import ttkbootstrap as tb
from ttkbootstrap.constants import *
from tkinter import messagebox
from collections import deque

class WebCrawlerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("🕸 BFS Web Crawler")
        self.root.geometry("1000x700")
```

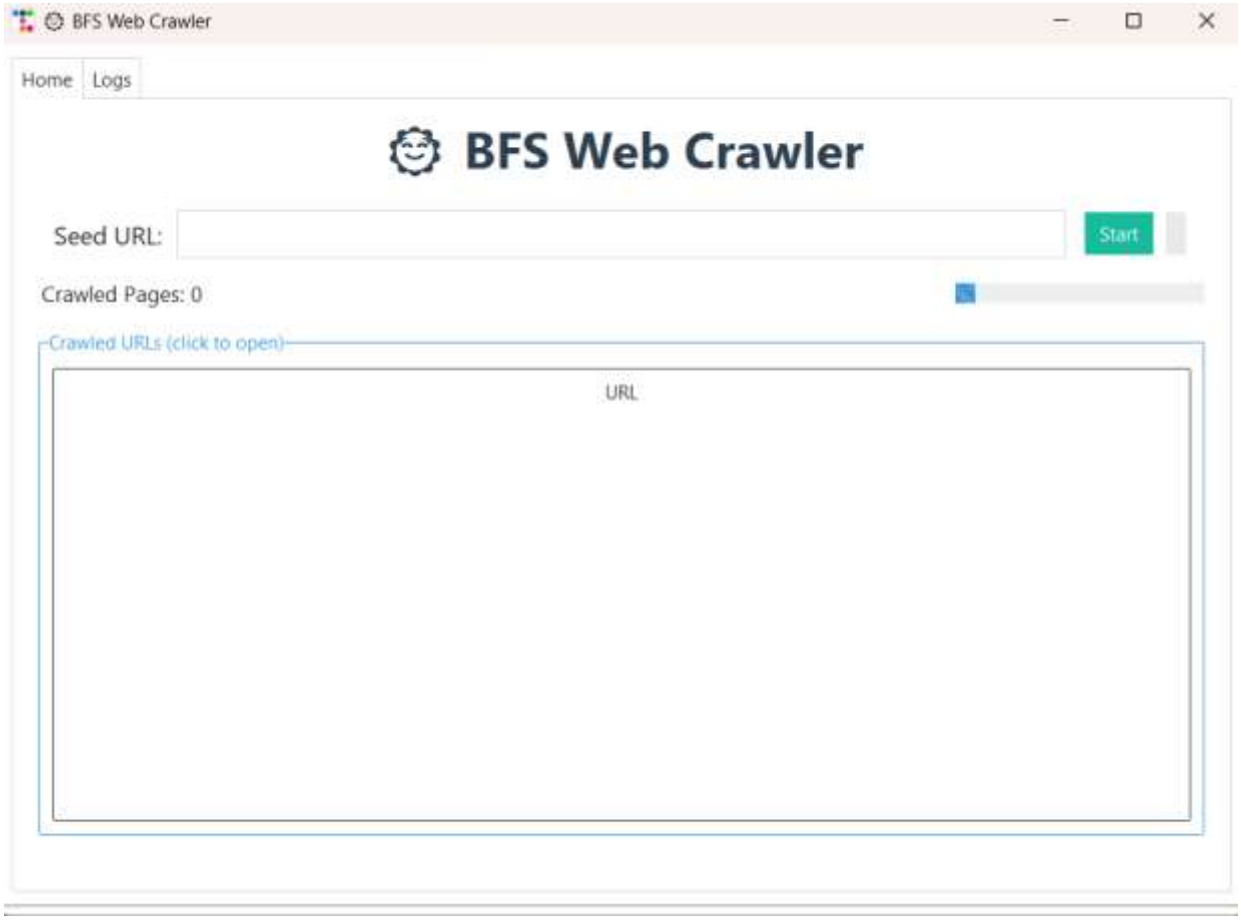
```
        self.visited = set()
        self.queue = deque()
        self.running = False
        self.paused = False
        self.hover_url = tk.StringVar()

        self.style = tb.Style("flatly")
        self.setup_gui()

    def setup_gui(self):
        self.notebook = tb.Notebook(self.root)
        self.notebook.pack(fill=BOTH, expand=True, padx=10, pady=10)

        self.tab_main = tb.Frame(self.notebook)
        self.tab_logs = tb.Frame(self.notebook)

        self.notebook.add(self.tab_main, text="Home")
        self.notebook.add(self.tab_logs, text="Logs")

        header = tb.Label(self.tab_main, text="🕸 BFS Web Crawler", font=("Segoe UI", 24,
"bold"), bootstyle="primary")
        header.pack(pady=(10, 10))

        url_frame = tb.Frame(self.tab_main, padding=10)
        url_frame.pack(fill=X, padx=20)

        tb.Label(url_frame, text="Seed URL:", font=("Segoe UI", 12)).pack(side=LEFT)
        self.url_entry = tb.Entry(url_frame, width=70, font=("Segoe UI", 11))
        self.url_entry.pack(side=LEFT, padx=10)

        self.start_button = tb.Button(url_frame, text="Start", bootstyle="success",
command=self.start_crawl)
        self.start_button.pack(side=LEFT, padx=5)

        self.pause_button = tb.Button(url_frame, text="Pause", bootstyle="warning",
command=self.toggle_pause, state=DISABLED)
        self.pause_button.pack(side=LEFT, padx=5)

        self.stop_button = tb.Button(url_frame, text="Stop", bootstyle="danger",
command=self.stop_crawl, state=DISABLED)
        self.stop_button.pack(side=LEFT, padx=5)
```

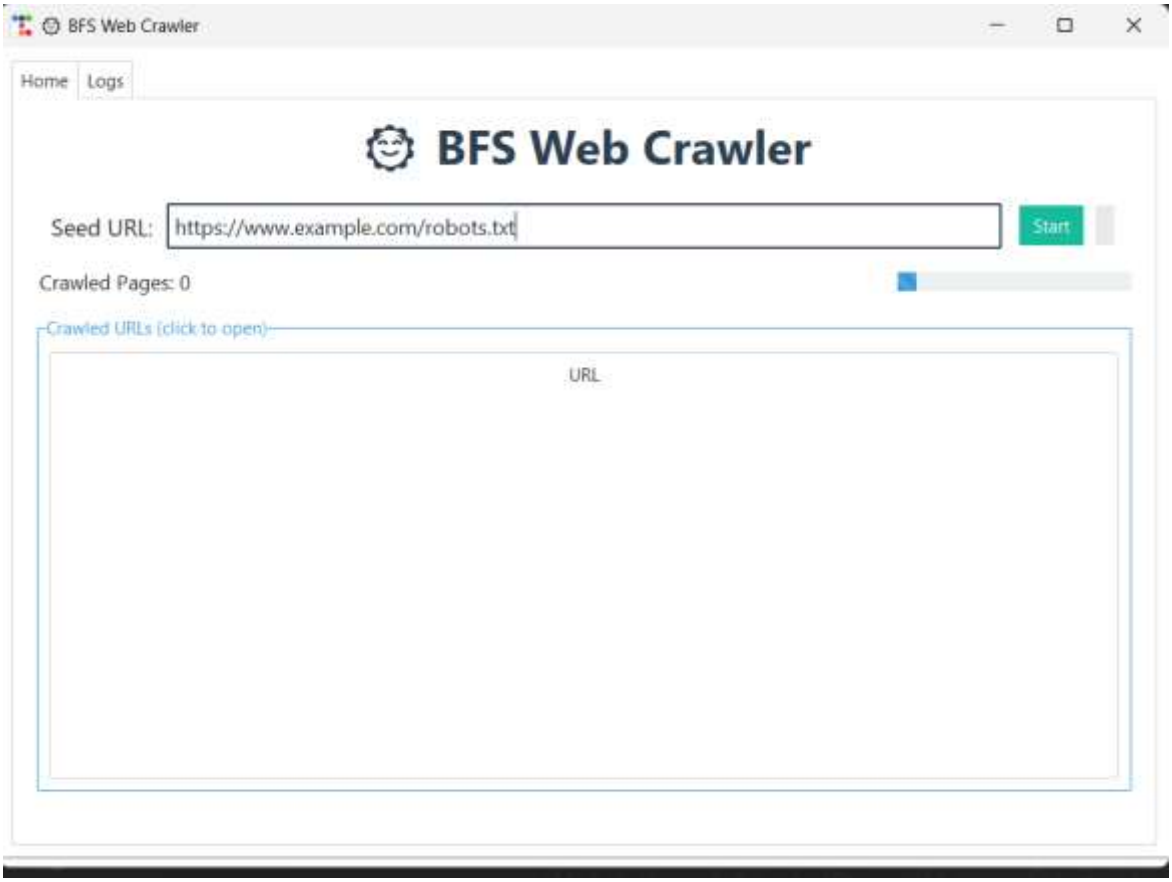
```
        self.save_button = tb.Button(url_frame, text="💾 Save URLs", bootstyle="info",
command=self.save_to_file)
        self.save_button.pack(side=LEFT, padx=5)

        progress_frame = tb.Frame(self.tab_main, padding=(20, 5))
        progress_frame.pack(fill=X)

        self.progress_label = tb.Label(progress_frame, text="Crawled Pages: 0", font=("Segoe UI",
10))
        self.progress_label.pack(side=LEFT)

        self.progress_bar = tb.Progressbar(progress_frame, mode="indeterminate", bootstyle="info-
striped", length=200)
        self.progress_bar.pack(side=RIGHT)

        list_frame = tb.Labelframe(self.tab_main, text="Crawled URLs (click to open)",
padding=10, bootstyle="info")
        list_frame.pack(fill=BOTH, expand=True, padx=20, pady=10)

        self.url_list = tb.Treeview(list_frame, columns=("url"), show='headings')
        self.url_list.heading("url", text="URL")
        self.url_list.column("url", anchor="w")
        self.url_list.pack(fill=BOTH, expand=True)

        self.url_list.bind("<Button-1>", self.handle_click)
        self.url_list.bind("<Motion>", self.update_hover)

        self.hover_label = tb.Label(self.tab_main, textvariable=self.hover_url, font=("Segoe UI",
9), bootstyle="secondary")
        self.hover_label.pack(pady=(0, 10))

        self.log_text = tb.ScrolledText(self.tab_logs, height=30)
        self.log_text.pack(fill=BOTH, expand=True, padx=10, pady=10)

        self.status = tb.Label(self.root, text="Ready", relief="sunken", anchor="w", font=("Segoe
UI", 9))
        self.status.pack(fill=X, side=BOTTOM)

    def toggle_pause(self):
        self.paused = not self.paused
        self.pause_button.config(text="Resume" if self.paused else "Pause")

    def start_crawl(self):
```

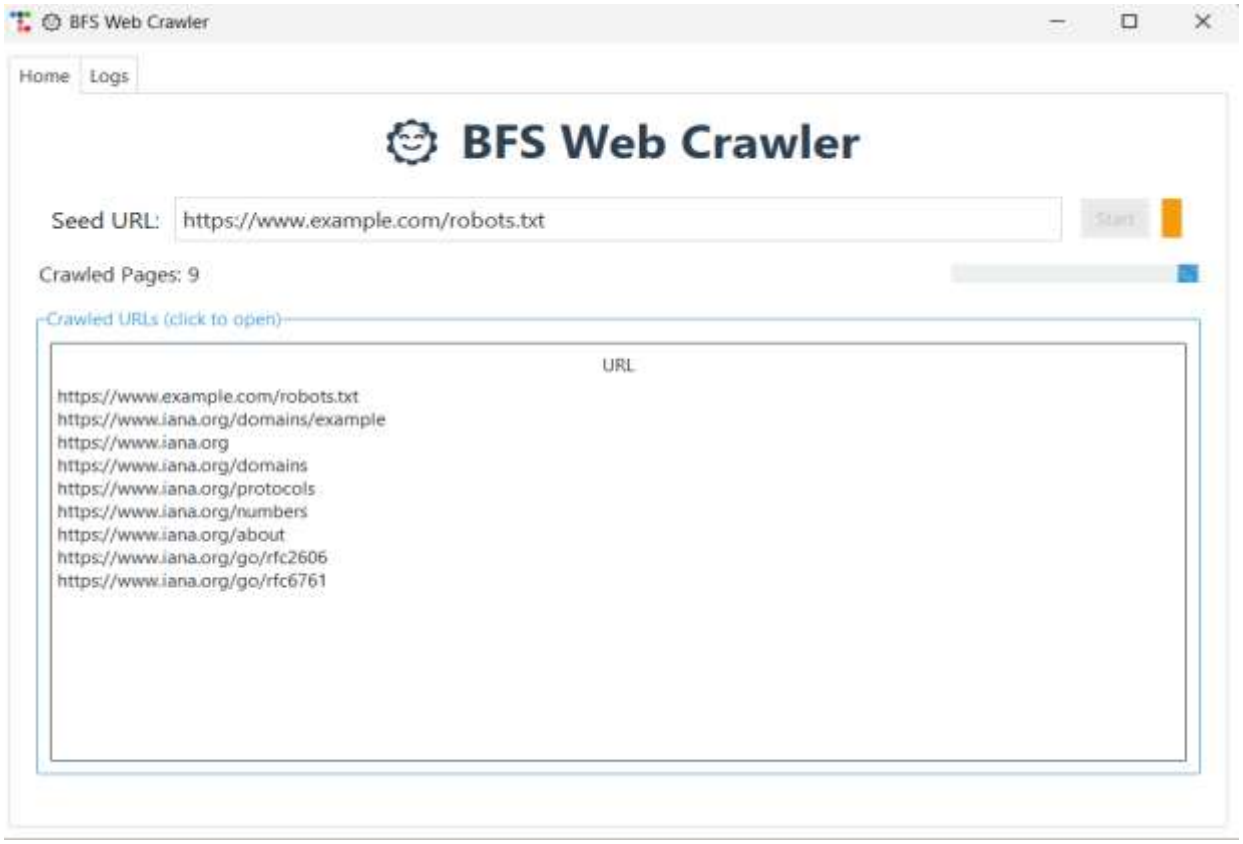
```
        seed_url = self.url_entry.get().strip()
        if not seed_url:
            messagebox.showerror("Error", "Please enter a valid URL.")
            return

        self.visited.clear()
        self.queue = deque([seed_url])
        self.running = True
        self.paused = False
        self.progress_label.config(text="Crawled Pages: 0")
        self.status.config(text="Crawling started")
        self.url_list.delete(*self.url_list.get_children())
        self.log_text.delete("1.0", "end")
        self.start_button.config(state=DISABLED)
        self.pause_button.config(state=NORMAL)
        self.stop_button.config(state=NORMAL)
        self.progress_bar.start(10)

        threading.Thread(target=self.bfs_crawl, daemon=True).start()

    def stop_crawl(self):
        self.running = False
        self.status.config(text="Crawling stopped")
        self.pause_button.config(state=DISABLED)
        self.stop_button.config(state=DISABLED)
        self.start_button.config(state=NORMAL)
        self.progress_bar.stop()

    def bfs_crawl(self):
        crawled_count = 0
        while self.queue and self.running:
            if self.paused:
                time.sleep(0.5)
                continue

            current_url = self.queue.popleft()
            if current_url in self.visited:
                continue

            self.visited.add(current_url)
            crawled_count += 1

            self.root.after(0, lambda url=current_url: self.url_list.insert("", "end", values=(url,)))
```

```
            self.root.after(0, lambda: self.progress_label.config(text=f"Crawled Pages:
{crawled_count}"))
            self.root.after(0, lambda url=current_url: self.log_text.insert("end", f"Crawled: {url}\n"))

            if not self.is_allowed_by_robots(current_url):
                continue

            try:
                response = requests.get(current_url, timeout=5, headers={'User-Agent':
'BFSWebCrawlerBot'})
                if 'text/html' not in response.headers.get('Content-Type', ''):
                    continue

                soup = BeautifulSoup(response.text, 'html.parser')
                for tag in soup.find_all('a', href=True):
                    href = tag['href']
                    full_url = urljoin(current_url, href)
                    norm_url = self.normalize_url(full_url)
                    if norm_url and norm_url not in self.visited:
                        self.queue.append(norm_url)
            except Exception as e:
                self.root.after(0, lambda: self.log_text.insert("end", f"Error crawling {current_url}:
{e}\n"))
                continue

        self.root.after(0, self.done_message)

    def normalize_url(self, url):
        parsed = urlparse(url)
        if parsed.scheme not in ['http', 'https']:
            return None
        return parsed.scheme + "://" + parsed.netloc + parsed.path.rstrip('/')

    def is_allowed_by_robots(self, url):
        parsed = urlparse(url)
        base_url = f"{parsed.scheme}://{parsed.netloc}/robots.txt"
        try:
            rp = RobotFileParser()
            rp.set_url(base_url)
            rp.read()
            return rp.can_fetch("*", url)
        except:
            return True
```

```
    def done_message(self):
        self.progress_bar.stop()
        self.status.config(text="Crawling finished.")
        messagebox.showinfo("Done", "Crawling finished or stopped.")
        self.start_button.config(state=NORMAL)
        self.pause_button.config(state=DISABLED)
        self.stop_button.config(state=DISABLED)

    def handle_click(self, event):
        item = self.url_list.identify_row(event.y)
        if item:
            url = self.url_list.item(item, "values")[0]
            webbrowser.open(url)

    def update_hover(self, event):
        item = self.url_list.identify_row(event.y)
        if item:
            url = self.url_list.item(item, "values")[0]
            self.hover_url.set(f"🔗 {url}")
        else:
            self.hover_url.set("")

    def save_to_file(self):
        urls = [self.url_list.item(i, "values")[0] for i in self.url_list.get_children()]
        if not urls:
            messagebox.showinfo("Info", "No URLs to save.")
            return
        with open("crawled_urls.txt", "w") as f:
            f.writelines(url + "\n" for url in urls)
        self.status.config(text="URLs saved to crawled_urls.txt")
        messagebox.showinfo("Saved", "URLs saved to crawled_urls.txt")

# Run
if __name__ == "__main__":
    root = tb.Window(themename="flatly")
    app = WebCrawlerGUI(root)
    root.mainloop()
```


BFS Web Crawler
Home
Logs
BFS Web Crawler
Seed URL:
Start
Crawled Pages: 0
Crawled URLs (click to open)
URL


BFS Web Crawler
Home
Logs
BFS Web Crawler
Seed URL:
https://www.example.com/robots.txt
Start
Crawled Pages: 0
Crawled URLs (click to open)
URL



Conclusion

The BFS Web Crawler is a robust and user-friendly Python application that demonstrates the practical implementation of a breadth-first search algorithm for web crawling, enhanced with a modern graphical interface using ttkbootstrap. It efficiently explores web pages from a seed URL, respects robots.txt rules, and provides real-time updates, logging, and interactive controls for managing the crawling process. With features like pause/resume, URL preview, and data export, it serves as both an educational tool and a foundation for more advanced web crawling and data mining projects.