Buscando palavras em um site

Você recebeu um código Python que implementa um crawler para buscar uma palavra específica em todas as páginas de um site. O programa navega recursivamente pelas páginas do site, começando de uma URL inicial, e verifica se a palavra está presente no conteúdo de cada página. No entanto, para sites grandes ou com muitas páginas, o processo pode ser lento, pois cada página é processada sequencialmente.

Para melhorar o desempenho do programa, você deve paralelizar a busca utilizando threads . A ideia é dividir o trabalho de processamento das páginas entre várias threads, permitindo que múltiplas páginas sejam acessadas e analisadas simultaneamente.

Objetivo

Modifique o código fornecido para implementar a busca em threads separadas. Cada thread deve ser responsável por acessar e processar uma página específica do site. Além disso, certifique-se de que as threads trabalhem de forma coordenada para evitar processamento duplicado ou loops infinitos. Por fim, os resultados das buscas sejam combinados corretamente no final.

Requisitos

1. Use a biblioteca threading do Python para criar threads.
2. Divida o trabalho de processamento das páginas entre várias threads.
3. Certifique-se de que cada thread processe apenas uma página específica.
4. Garanta que as threads não conflitem ao acessar ou modificar essas estruturas compartilhadas .
5. Mantenha a estrutura modular do código original, criando funções específicas para cada tarefa.
6. Adicione uma função principal (main) que inicie as threads e aguarde sua conclusão.
7. O programa deve exibir uma lista de URLs e indicar se a palavra foi encontrada em cada página.
8. Certifique-se de que os resultados finais estejam consistentes e completos.
9. Execute o programa com diferentes sites e palavras para verificar se a paralelização funcionou.
10. Meça o tempo de execução com e sem threads para comparar o desempenho.

Código a ser paralelizado

```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def buscar_palavra_no_site(url_inicial, palavra, profundidade_maxima=3):
    urls_visitados = set()
    resultados = {}

    def buscar_recursivo(url_atual, profundidade_atual):
        if profundidade_atual > profundidade_maxima or url_atual in urls_visitados:
            return
        urls_visitados.add(url_atual)

        try:
            print(f"Buscando em: {url_atual} (Profundidade: {profundidade_atual})")
            response = requests.get(url_atual, timeout=10)
            response.raise_for_status()  # Lança exceção para erros HTTP

            # Analisa o conteúdo HTML
```

```
            soup = BeautifulSoup(response.text, 'html.parser')

            # Verifica se a palavra está no conteúdo da página
            conteudo = soup.get_text().lower()
            palavra_encontrada = palavra.lower() in conteudo
            resultados[url_atual] = palavra_encontrada

            # Extrai todos os links da página
            links = soup.find_all('a', href=True)
            for link in links:
                url_completa = urljoin(url_inicial, link['href'])

                # Garante que só navegamos dentro do mesmo domínio
                if url_completa.startswith(url_inicial):
                    buscar_recursivo(url_completa, profundidade_atual + 1)

        except requests.exceptions.RequestException as e:
            print(f"Erro ao acessar {url_atual}: {e}")

    # Inicia a busca recursiva
    buscar_recursivo(url_inicial, profundidade_atual=1)
    return resultados

# Exemplo de uso
if __name__ == "__main__":
    url_inicial = input("Digite a URL inicial do site (ex.: https://www.exemplo.com): ")
    palavra = input("Digite a palavra a ser buscada: ")

    resultados = buscar_palavra_no_site(url_inicial, palavra)

    print("\nResultados da busca:")
    for url, encontrada in resultados.items():
        status = "Encontrada" if encontrada else "Não encontrada"
        print(f"{url}: Palavra '{palavra}' {status}")
```