25.03 Algoritmos y Estructuras de Datos

Level 5: EDAoogle

Introducción

En esta práctica vamos a implementar un motor de búsqueda de páginas web.

HTML

El lenguaje HTML es el estándar fundamental para la creación de páginas web. Veamos un ejemplo de una página web:

```
<html>
    <head>
        <title>My first web page</title>
    </head>
    <body>
        <h1>Hello world</h1>
        <p>This is HTML!</p>
    </body>
</html>
```

Los documentos HTML son archivos de texto estructurados en forma de árbol. Cada elemento del árbol comienza con una etiqueta de apertura, como `<html>`, y finaliza con una etiqueta de cierre, como `</html>`. Entre estas etiquetas puede haber texto u otros elementos.

El elemento raíz de un documento HTML es siempre `<html>`, el cual incluye generalmente dos elementos: `<head>` y `<body>`.

- `<head>` contiene meta-información que no se suele mostrar al usuario, como el título del documento (`<title>`).
- `<body>` define el contenido visible del documento, el cual puede incluir encabezados (`<h1>`), párrafos (`<p>`), saltos de línea (`<br>`), enlaces a otras páginas (`<a>`), o formularios que permiten la interacción del usuario (`<form>`).

Además, las etiquetas pueden contener atributos con información adicional. Considera, por ejemplo, el siguiente formulario:

```
<form action="/search" method="get">
  <input type="text" name="q" autofocus></input>
</form>
```

CSS

Los archivos CSS (Hojas de estilo en cascada) permiten definir la apariencia de un documento HTML, separando la presentación del contenido.

Veamos un ejemplo de un archivo CSS:

```
.emphasized {
    font-size: 2rem;
    font-weight: bold;
}
```

Este archivo define la clase CSS `emphasized`, que permite resaltar elementos en HTML:

```
    <p class="emphasized">This is important</p>
```

HTTP

Para que un usuario pueda acceder a un sitio web, es necesario que se conecte a un servidor HTTP. Las conexiones HTTP operan sobre el protocolo TCP/IP, en el que el cliente (el programa que usa el usuario) envía solicitudes, que el servidor responde.

Veamos un ejemplo de una solicitud HTTP:

```
GET /index.html HTTP/1.1
Host: www.google.com
Referer: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/101.0.4951.67
Connection: keep-alive
[Línea en blanco]
```

El servidor puede responder a esta solicitud con una respuesta HTTP similar a la siguiente:

```
HTTP/1.1 200 OK
Date: Fri, 12 Feb 2022 16:21:12 GMT
Content-Type: text/html
Content-Length: 1221

<html>
<head>
<title>Google</title>[…]
</head>
<body>[…]</body>
</html>
```

El uso de HTTP plantea riesgos de seguridad, ya que los datos se transmiten en texto plano, lo que facilita su interceptación. En la práctica solemos preferir HTTPS, que cifra las comunicaciones HTTP con una capa adicional de seguridad.

El servidor HTTP

Para empezar esta práctica, instala la biblioteca `libmicrohttpd`:

```
vcpkg install libmicrohttpd
```

A continuación, compila el target `edahttpd` del starter code que te proporcionamos. Este target implementa un pequeño servidor HTTP que sirve los archivos de una carpeta `www`, e implementa la estructura básica del motor de búsqueda.

Para probar el servidor HTTP te suministramos una carpeta `www` con las 1000 páginas más visitadas de Wikipedia.

Puedes iniciar el servidor, especificando la ruta de la carpeta `www` con el parámetro `-h`. Por ejemplo:

```
edahttpd -h ../../www
```

El servidor responde en la URL http://localhost:8000.

El índice de búsqueda

Para implementar la búsqueda, deberás generar un índice de búsqueda con una base de datos SQLite.

Para hacer esto, instala primero la biblioteca `sqlite3`:

```
vcpkg install sqlite3
```

A continuación, compila el target `mkindex`. Este target ejemplifica el acceso a una base de datos SQLite.

Tu trabajo consiste en modificar `mkindex.cpp` para que genere el índice del motor de búsqueda. Deberá recibir por línea de comando la ruta de la carpeta `www` para la que se genera el índice. Para ello, aprovecha `CommandLineParser.h`.

Puedes usar **`std::filesystem`** (de C++17) para procesar todos los archivos que se encuentran en la carpeta `www/wiki`. Usa **`std::ifstream`** para leer los archivos. Para cada documento HTML, elimina las etiquetas HTML y conserva sólo el texto. Te sugerimos hacerlo manualmente, eliminando simplemente el contenido entre los signos **<** y **>**. No te preocupes por los acentos y demás caracteres especiales. Luego, añade las palabras al índice de búsqueda.

La búsqueda

A continuación, implementa el código del motor de búsqueda en `HttpRequestHandler`. El motor deberá ser capaz de encontrar todos los documentos cuyas palabras coincidan con las especificadas en la búsqueda.

La búsqueda `dominios anidados`, por ejemplo, debería devolver un enlace a `/wiki/Evolucion_biologica.html`.

Puedes usar `std::chrono` para medir el tiempo de búsqueda.

Tips

- **Piensa, luego actúa.** Diseña primero el esquema de tu base de datos; luego implementa el código.
- **Configura los parámetros del ejecutable en tu entorno de desarrollo.** Así podrás realizar el debugging cómodamente.
- **Optimiza.** No quieres que tus usuarios tengan que esperar mucho. ¡Sino no comprarán tu producto!
- **Securiza.** Asegúrate que tu motor de búsqueda sea inmune a inyecciones SQL. ¡No querrás que un hacker tome el control de tu base de datos!

Si el servidor no arranca en Windows, asegúrate de que el archivo `libmicrohttpd-dll.dll` se haya copiado en la carpeta de `edahttpd.exe`. En caso de que no se haya copiado, cópialo manualmente desde `dev/vcpkg/installed/x64-windows`.

Si el servidor devuelve siempre el mensaje **404 Not Found**, asegúrate de que la ruta a la carpeta `www` sea correcta.

Bonus points

- **Implementa operadores.** Permite que tu usuario pueda utilizar operadores como AND u OR en sus búsquedas. Puedes aprovechar FTS de sqlite3, que implementa los operadores por ti.