# Benemérita Universidad Autónoma de Puebla
# Facultad en Ciencias de la Computación

## Recuperacion de la informacion

## Manual de usuario
## Modelo Booleano


**Luis Francisco Matlalcuatzi Gonzalez**

**Julio Cesar Aca Toxtle**

**Emmanuel Flores Navarro**

**Hristo Cornejo Guzman**

**Osvaldo Mellado Lozano**


**12 Abril 2024**

# Índice

# 1. Metodologia

El presente documento se enfoca en el procesamiento de texto, con esto debemos tener presente los siguientes conocimientos, para la implementación de solución en la propuesta de la problemática de variedad de documentaci ón sobre ellos debemos procesar su frecuencia, ordenar por longitud de su carácter y saber en que documento pertenece, para así tener la finalidad de hacer consultas entre términos y saber donde pertenecen o donde se visualizan.

## 1.1. Diagrama de módulo

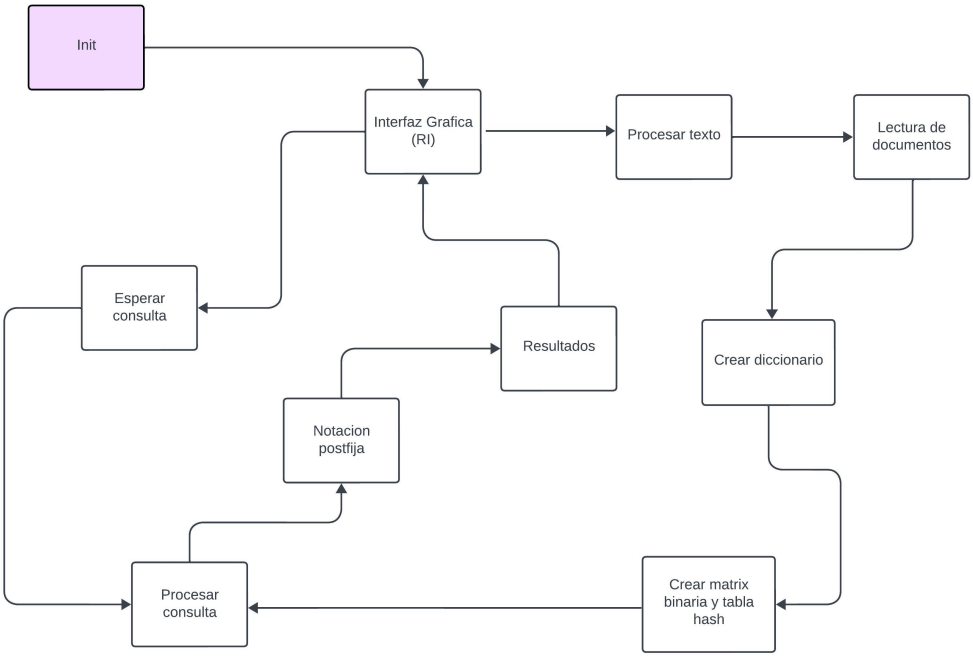


Figura 1: Diagrama del sistema booleano

## 1.2. Descripción de cada uno de los módulos del programa

- Init
  - init (self)
    El constructor de la clase Sistema RI inicializa varios atributos y configuraciones necesarios para el sistema de recuperación de información (RI). Esto incluye la inicialización de la interfaz gráfica en un hilo separado.
  - init gui thread(self)
    Este método inicializa la interfaz gráfica en un hilo separado utilizando PyQt5. También conecta la señal de consulta de la interfaz a una ranura (slot) que maneja la recepción de consultas.
- Interfaz Gráfica (RI)
  - init (self, sistema ri)
    El constructor de la clase InterfazRI inicializa la interfaz gráfica de usuario (GUI) y recibe una instancia de SistemaRI como argumento para establecer una comunicación entre la interfaz y el sistema de recuperación de información.

- initUI(self)
  Este método configura la interfaz de usuario utilizando la biblioteca PyQt5. Define la estructura de la ventana principal
- openDocument(self)
  Abre un cuadro de diálogo para seleccionar y abrir un documento de texto. Luego, muestra el contenido del documento en el área de visualización.
- update document displayed(self, item)
  Este método se llama cuando se hace clic en un elemento de la lista de resultados. Si el elemento hace referencia a un documento, muestra el contenido del documento seleccionado en el área de visualización.
- mostrar ayuda(self)
  Muestra un cuadro de diálogo modal (QDialog) con información de ayuda sobre cómo usar la aplicación.
- closeEvent(self, event)
  Este método maneja el evento de cierre de la ventana principal y asegura que la aplicación se cierre correctamente cuando se cierra la ventana.

- Procesar texto
  - procesar texto(self, contenido)
    Este método recibe un contenido de texto, realiza una serie de transformaciones como convertir a minúsculas, eliminar caracteres especiales, quitar acentos y números, y tokenizar el texto en palabras. Luego, devuelve una lista de palabras procesadas.
  - token stopw(self, string)
    Realiza la tokenización, elimina las stopwords y aplica stemming a una cadena de texto.
- Crear Diccionario
  - crear tabla(self, array, nombre)
    Este método crea una tabla a partir de un array y la almacena en un archivo de texto. Es utilizado para almacenar tablas de palabras, frecuencias, stopwords y stemming.
- Crear Matriz Binaria y Tabla Hash
  - crear matriz binaria(self, palabras, numero documentos)
    Este método crea una matriz binaria donde cada fila representa una palabra y cada columna representa un documento. El valor en cada celda de la matriz indica si la palabra aparece en el documento correspondiente.
  - computar hash(self, word)
    Calcula el valor hash MD5 de una palabra y lo devuelve como una cadena hexadecimal.
  - crear hash table(self, words, binary dict)
    Crea una tabla hash que asocia palabras a sus valores binarios en la matriz binaria.
- procesar consulta
  - process query(self, operator)
    Este método procesa una consulta booleana en notación infija y la convierte en notación postfija. Luego, ejecuta la consulta y muestra los resultados en la interfaz gráfica.
  - hash query(self, hash)
    Busca en la tabla hash un valor binario correspondiente a un hash de palabra.
  - ejecutar query(self, postfijo)
    Ejecuta una consulta booleana en notación postfija utilizando los arrays binarios y devuelve el resultado.

  - obtener docs(self, array docs)
    Imprime los documentos que satisfacen una consulta booleana en función del resultado de la evaluación.
  - clear(self)
    Limpia las variables y reinicia el sistema para una nueva consulta.

- Notacion posfija
  - procesar bool exp(self, query str)
    Toma una cadena de consulta booleana y la convierte en una lista de elementos que representan la expresión booleana.
  - procesar postfijo(self, bool exp list)
    Toma una lista de elementos que representan la expresión booleana en notación infija y la convierte en notación postfija.
  - or operator(self)
    Implementa la operación lógica OR en listas binarias y devuelve el resultado.
  - and operator(self)
    Implementa la operación lógica AND en listas binarias y devuelve el resultado.
  - not operator(self)
    Implementa la operación lógica NOT en una lista binaria y devuelve el resultado.

- Esperar consulta
  - esperar consulta(self)
    Bloquea hasta que se recibe una consulta desde la interfaz gráfica.
  - recibir consulta(self, consulta)
    Recibe una consulta desde la interfaz gráfica y la almacena para su posterior procesamiento.

- Resultados
  - enviar consulta(self)
    Este método se llama cuando se hace clic en el botón "Enviar"o se presiona la tecla Enter después de ingresar una consulta en el cuadro de texto. Toma la consulta ingresada por el usuario y la emite como una señal consulta signal para que el sistema de recuperación de información la procese.
  - update output(self, output)
    Este método agrega resultados a la lista de resultados en la interfaz gráfica.

# 2. Resultados

- Una prueba de ejecucion es la siguiente.
  Observando la tabla de excel generada podemos observar que contamos con 10 documentos, de los cuales ya tenemos su matris binaria para identificar que palabra esta en cada documento, y con ello poder realizar la busqueda de palabras clave en cada documento.

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Termino | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | |
| 2 | encontr | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 3 | mas | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 4 | perr | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 5 | gat | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 6 | man | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 7 | max | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 8 | pequenl̀f | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | |
| 9 | puebl | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 10 | pez | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 11 | raton | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 12 | habi | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 13 | junt | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | |
| 14 | explor | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 15 | llam | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 16 | siempr | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 17 | descubr | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 18 | encuentr | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | |
| 19 | bosqu | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 20 | lob | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 21 | lun | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 22 | cuid | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 23 | demostr | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |

Figura 2: Resultado

- Para este caso nos vamos a centrar en las palabras clave çachorroz .ªngel"para poder determinar el funcionamiento de nuestra aplicacion de busqueda por metodo booleano.

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 73 | mutu | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 74 | volvi | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 75 | bellez | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| 76 | aprend | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |
| 77 | alegri | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| 78 | ayud | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 79 | companl̀fi | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 80 | corazon | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 81 | callejer | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 82 | cos | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |
| 83 | camp | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 84 | angel | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 85 | cachorr | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 86 | decid | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 87 | bail | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 88 | aventur | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 89 | aunqu | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 90 | pes | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 91 | personal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 92 | principi | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 93 | profund | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 94 | prep | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 95 | provenient | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Figura 3: Resultado

- como podemos observar en la aplicacion tenemos un apartado donde podemos ingresar la exprecion booleana que deseamos buscar.

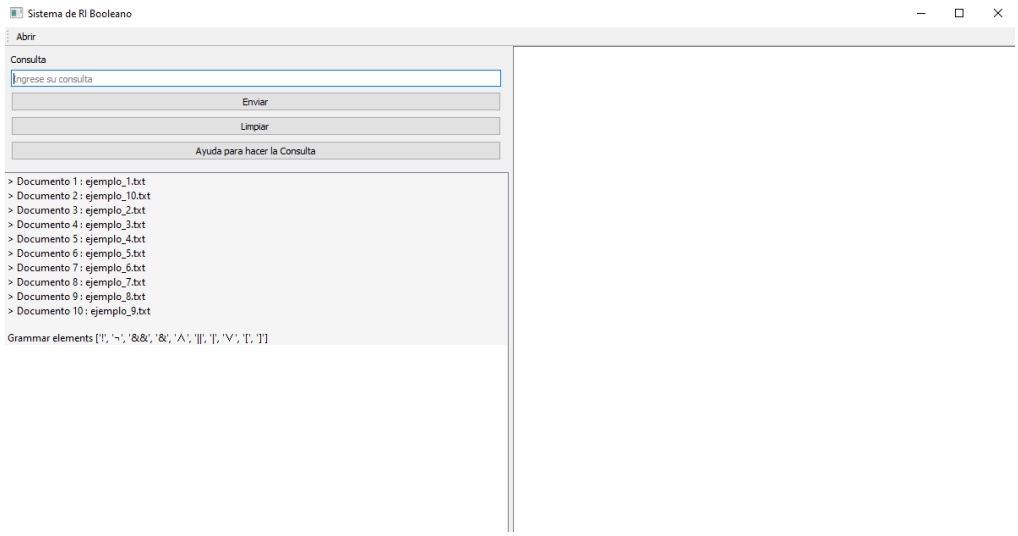


Figura 4: Resultado

- En este caso vamos a buscar la palabra çachorro.º .ªngel", para ello ingresamos la exprecion çachorro — angelz damos click en el boton de buscar, con ello nos mostrara los documentos que contienen la palabra çachorro.º .ªngel", en este caso solo el documento 1 y 2 contienen la palabra çachorro.º .ªngel".

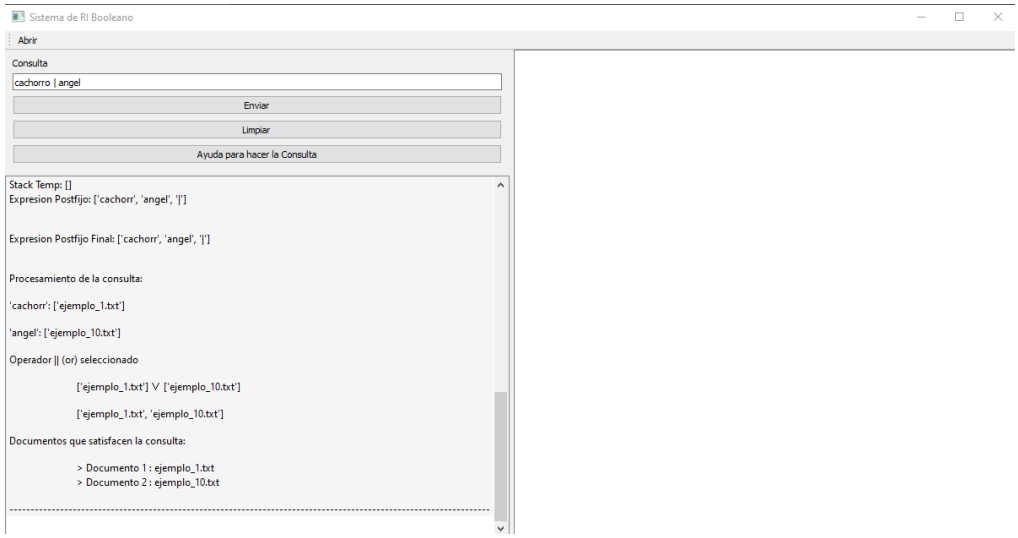


Figura 5: Resultado

- Ahora bien si deseamos buscar la palabra çachorroƶ .ªngel.en los documentos, ingresamos la exprecion çachorro & angelƶ damos click en el boton de buscar, con ello nos mostrara los documentos que contienen la palabra çachorroƶ .ªngel", en este caso no hay documento que cumpla con la condicion.

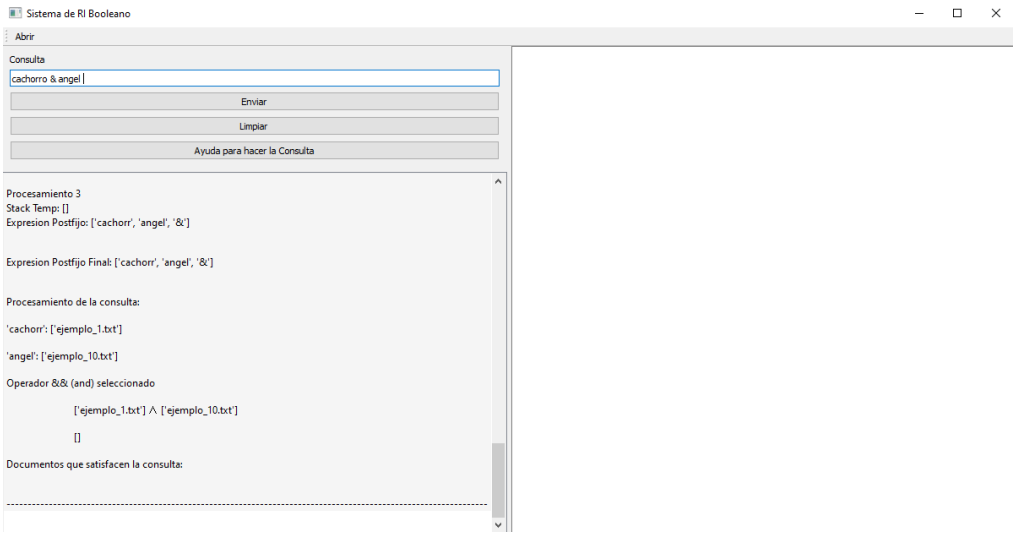


Figura 6: Resultado

- Pero que pasa si a toda la exprecion le agregamos un "not", en este caso vamos a buscar la palabra çachorroƶ .ªngel.en los documentos, ingresamos la exprecion " ( cachorro & angel )ƶ damos click en el boton de buscar, con ello nos mostrara los documentos que no contienen la palabra çachorroƶ .ªngel", en este caso son todos los documentos que no contienen la palabra çachorroƶ .ªngel.en el mismo documento.

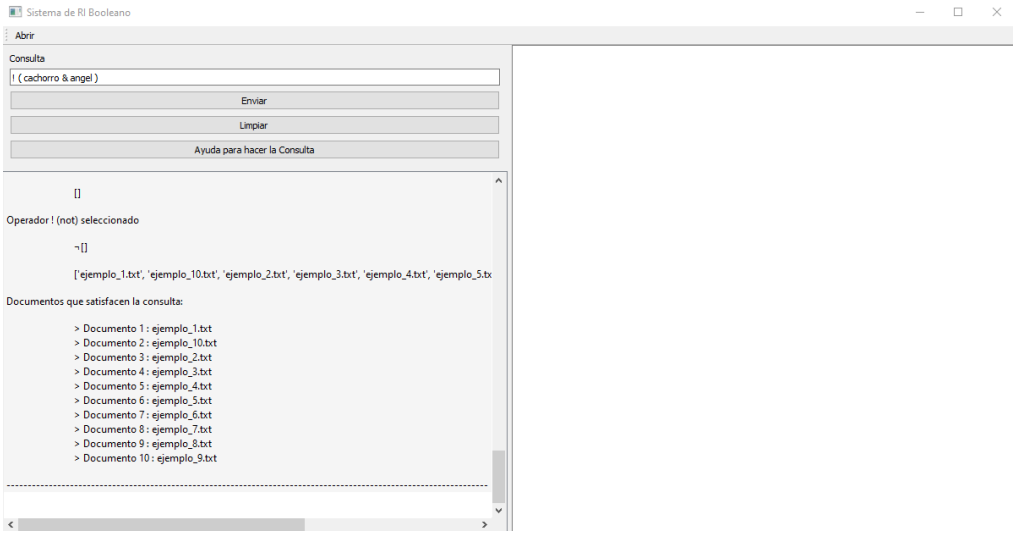


Figura 7: Resultado

# 3. Conclusiones

El objetivo central de esta práctica se orienta hacia el desarrollo de un Sistema de Recuperación de Información (RI) basado en un modelo booleano, el cual se divide en dos partes fundamentales que trabajan en conjunto para lograr su funcionalidad integral.

En la primera fase, se enfoca en la preparación de datos, un proceso esencial donde se realizan diversas operaciones sobre los documentos de texto para extraer información relevante. Esto implica la identificación y almacenamiento de palabras clave significativas, así como la creación de una matriz binaria que representa la presencia o ausencia de cada término en los documentos. Esta etapa establece los cimientos del sistema, proporcionando la estructura necesaria para la búsqueda y recuperación eficiente de información.

La segunda fase se concentra en la creación de una interfaz gráfica de usuario (GUI) que simplifica la interacción de los usuarios con el sistema de RI. A través de esta interfaz intuitiva, los usuarios tienen la capacidad de ingresar consultas booleanas de manera amigable, explorar los resultados de búsqueda de forma visualmente accesible y examinar los documentos recuperados de manera intuitiva. Esta GUI no solo mejora la experiencia del usuario al hacer que el sistema sea más accesible y fácil de usar, sino que también agrega una capa de interactividad que facilita la exploración y comprensión de los resultados.

En conjunto, estas dos partes del código conforman un sistema completo de RI que fusiona el procesamiento de datos subyacente con una interfaz de usuario intuitiva y amigable. Esta integración proporciona a los usuarios una herramienta poderosa y eficiente para buscar y recuperar información en documentos de texto, mejorando así la productividad y la experiencia general del usuario. La combinación de un sólido procesamiento de datos con una interfaz fácil de usar representa un avance significativo en la eficacia y accesibilidad de los sistemas de RI.

# 4. Conceptos básicos

## 4.1. Procesamiento de Texto

La manipulación y transformación de texto para prepararlo para su análisis y búsqueda. Esto incluye la conversión a minúsculas, la eliminación de caracteres especiales y acentos, así como la tokenización para dividir el texto en palabras o tokens.

## 4.2. Stemming

Un proceso que reduce las palabras a su forma raíz o base. Ayuda a simplificar las palabras para que se puedan buscar de manera más efectiva. En el código, se utilizan algoritmos de stemming como SnowballStemmer.

## 4.3. Índice Inverso

Una estructura de datos que almacena información sobre qué palabras clave aparecen en qué documentos. Facilita la búsqueda rápida de documentos que contienen ciertas palabras clave.

## 4.4. Consultas Booleanas

Consultas que utilizan operadores booleanos como AND, OR y NOT para combinar condiciones de búsqueda y recuperar documentos que cumplen con esas condiciones.

## 4.5. Notación Post-fija

Un método para representar y evaluar expresiones matemáticas o lógicas donde los operadores siguen a sus operandos. En el código, se utiliza para evaluar consultas booleanas de manera eficiente.

## 4.6. Matriz Binaria

Una representación en forma de matriz donde se indica la presencia (1) o ausencia (0) de palabras clave en los documentos. Se utiliza para acelerar la búsqueda de documentos relevantes.

## 4.7. Flexibilidad en el Idioma

La capacidad de procesar y comprender documentos en diferentes idiomas mediante el uso de algoritmos de procesamiento de lenguaje natural específicos para cada idioma.

## 4.8. Recuperación de Información

El proceso de buscar y obtener información relevante de una colección de documentos en función de criterios de búsqueda definidos por el usuario.

## 4.9. Normalización

La transformación de texto para llevarlo a una forma estándar o común, lo que facilita la comparación y búsqueda de palabras clave.

## 4.10. Stopwords

Palabras comunes que se filtran o eliminan del texto durante el procesamiento, ya que a menudo no aportan información relevante para la búsqueda.

## 4.11. Algoritmo MD5

(Message-Digest Algorithm 5) es un protocolo criptográfico que se utiliza para verificar la integridad de los datos. Aquí te explico cómo funciona:

### 4.11.1. Entrada de datos

El proceso comienza con la entrada de datos en forma de una cadena de texto. Esto puede ser cualquier tipo de información, desde un simple mensaje hasta archivos completos. La entrada de datos es el punto de partida del algoritmo MD5, donde se inicia el proceso de transformación de la información en un hash único.

### 4.11.2. División en bloques

Una vez que se ha proporcionado la entrada de datos, el algoritmo MD5 divide esta cadena en bloques de 512 bits. Esta división facilita el procesamiento de los datos y es una característica esencial del algoritmo. Los bloques de 512 bits son estándar para el algoritmo MD5 y permiten que el proceso de hashing se realice de manera eficiente.

### 4.11.3. Relleno de bits

Si la longitud de los datos de entrada no es un múltiplo de 512 bits, se agrega un proceso de relleno de bits. Esto implica agregar bits adicionales, generalmente ceros, para completar el último bloque de 512 bits. Además del relleno de bits, el algoritmo también añade la longitud original del mensaje al final del último bloque. Esta longitud original es crucial para garantizar la integridad de los datos durante el proceso de hashing.

### 4.11.4. Operaciones matemáticas

Una vez que los datos se han dividido en bloques y se ha realizado el relleno de bits necesario, el algoritmo MD5 realiza una serie de operaciones matemáticas en cada bloque de 512 bits. Estas operaciones incluyen combinaciones de rotaciones, adiciones módulo $2^{32}$, funciones booleanas y operaciones lógicas, diseñadas para mezclar y transformar los datos de manera específica.

#### 4.11.5. Generación del hash

Finalmente, después de completar todas las operaciones matemáticas en cada bloque de datos, el algoritmo MD5 genera un hash de salida de 128 bits. Este hash es una representación única y codificada de los datos de entrada. El proceso garantiza que, aunque se realicen pequeños cambios en los datos de entrada, el hash resultante será completamente diferente, lo que hace que el algoritmo MD5 sea valioso para la verificación de la integridad de los datos y la seguridad de la información.

### 4.12. NOTA

El resultado es una cadena de 32 caracteres que parece aleatoria, pero siempre será la misma para la misma entrada. Por ejemplo, la palabra "hola"siempre generará este hash: ç4ca4238a0b923820dcc509a6f75849b".

Es importante mencionar que aunque el MD5 es rápido y eficiente, no se considera seguro para funciones criptográficas debido a su vulnerabilidad a los ataques de colisión (dos entradas diferentes que producen el mismo hash). Por lo tanto, se utiliza principalmente para verificar la integridad de los archivos, en lugar de para almacenar contraseñas u otra información sensible.

## 5. Manual de usuario

Este manual esta diseñado para ayudar al usuario a utilizar el sistema de manera efectiva y realizar consultasbooleanas en tus documentos. El sistema permite buscar documentos que contienen palabras clave especificas utilizando operadores logicos como AND, OR y NOT.

### 5.1. Interfaz

En la parte de la izquierda, podemos ver una entrada de texto, para ingresar la consulta, a su vez, un boton para realizar las operaciones de la consulta y un boton de "ayuda"para ver como se debe ingresar la consulta. Mas abajo, podemos ver un cuadro de texto en donde se ven todos los documentos que se encuentran los documentos. En la parte de la derecha, habra un espacio de texto para ver el contenido de los documentos de texto que se encuentra en el archivo.

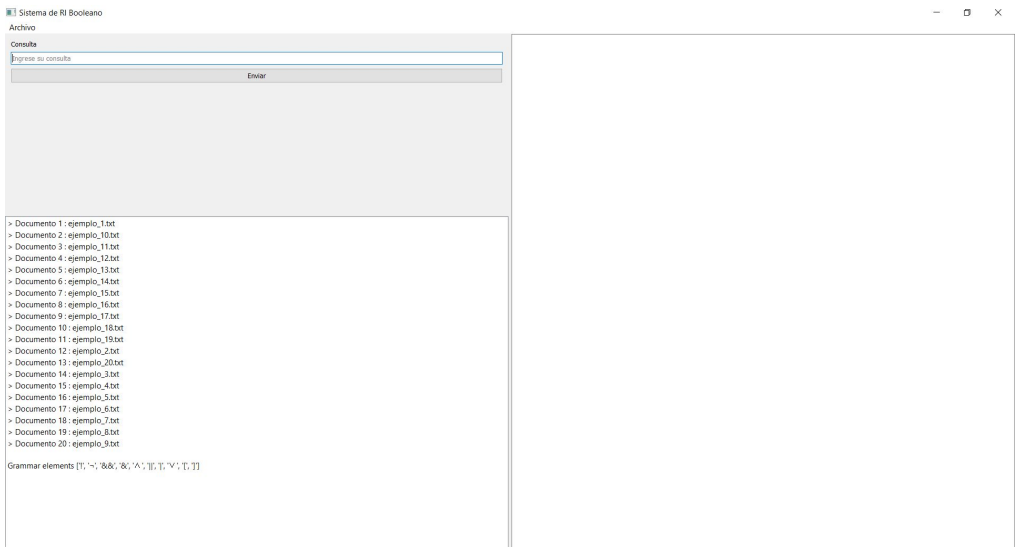


Figura 8: Interfaz del sistema

### Boton de ayuda

Con este boton, el programa nos lanzara una ventana para que nos aconseje como se debe de ingresar la consulta.

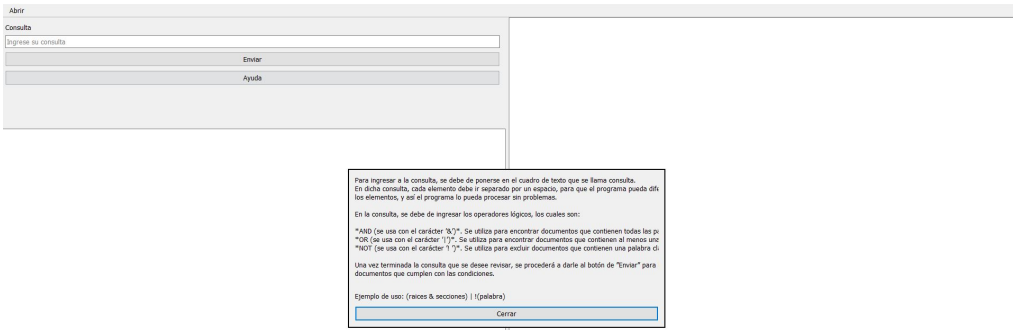


Figura 9: Interfaz del sistema

## 5.2. Visualizar documentos

Para poder el contenido de los documentos, simplemente se debe de dar clic a cualquiera de los documentos desplegados en la Interfaz, y en la parte derecha de la interfaz se desplegara el contenido de dicho documentos.

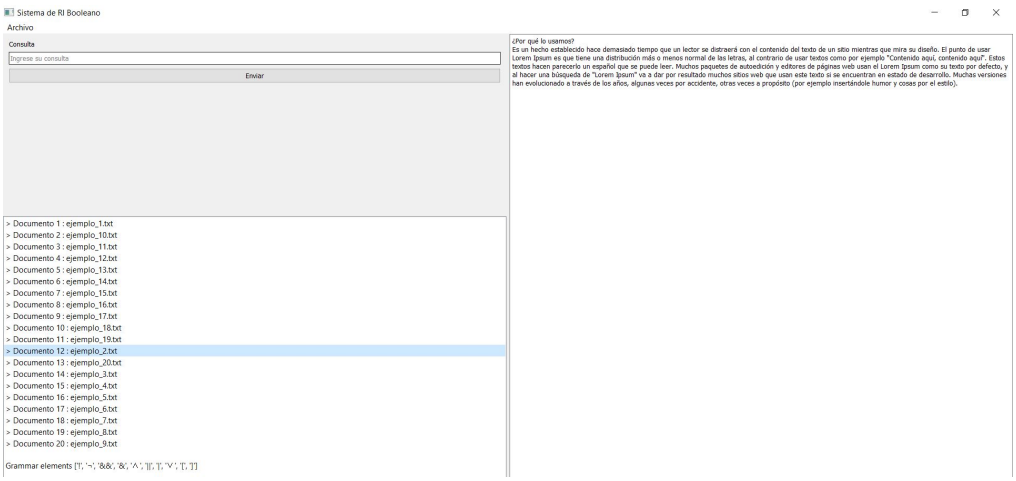


Figura 10: Interfaz del sistema

## 5.3. Realizar Consultas

Cada elemento de la consulta, debe ir separados por un espacio, para que el programa pueda diferenciar cada uno de los elementos de la consulta, y asi el programa lo pueda procesar sin problemas. En la consulta, se debe de usar los operadores logicos, los cuales son:

- De tipo AND (representado en el programa como '&'). Se utiliza para encontrar documentos que contienen todas las palabras clave.
- De tipo OR (representado en el programa como '||'): Se utiliza para encontrar documentos que contienen al menos una de las palabras clave.

- De tipo NOT (representado en el programa como ”): Se utiliza para excluir documentos que contienen una palabra clave especifica.

Una vez terminada la consulta que se desee revisar, se procedera a a darle al boton de ”Enviar”para proceder a hacer el metodo que permitira evaluar los documentos que cumplen con las condiciones de la consulta. Dicha evaluacion, se hara mediante la notacion Post-Fija. La notacion Post-Fija es importante debido a que permite evaluar expresiones booleanas de manera eficiente utilizando una pila (stack) en lugar de requerir una analisis de precedencia de operadores. Esto simplifica significativamente la logica de procesamiento de consultas booleanas y reduce la complejidad del codigo. Esto hace que la notacion no contenga ambig¨uedades debido a la secuencia de operadores y operandos dicta el orden de evaluacion. A su vez, la implementacion se puede implementar de manera sencilla mediante una pila, lo que simplifica la logica del programa. Por lo que cada operando se empuja a la pila, y cuando se encuentra un operador, se sacan los operandos necesarios de la pila para realizar la operacion. Esto hace que el codigo sea mas claro y menos propenso a errores. Algunos ejemplos para realizar la consulta son:

#### 5.3.1. consulta 1

( ( perro || gato ) & ( ( ! ( delfin || oso ) ) & lobo ) )
Query Expresion List: ['[', 'perr', '—', 'gat', ']', '&', ' [', '[', '!', '[', 'delfin', '—', 'oso', ']', ']', '&', 'lob', ']'] Stem Elements['perr', 'gat', 'delfin', 'oso', 'lob']
Procesamiento 1
Stack Temp: ['[']
Expresion Postfijo: []
Procesamiento 2
Stack Temp: ['[']
Expresion Postfijo: ['perr']
Procesamiento 3
Stack Temp: ['[', '||']
Expresion Postfijo: ['perr']
Procesamiento 4
Stack Temp: ['[', '||']
Expresion Postfijo: ['perr', 'gat']
Procesamiento 5
Stack Temp: []
Expresion Postfijo: ['perr', 'gat', '||']
Procesamiento 6
Stack Temp: ['&']
Expresion Postfijo: ['perr', 'gat', '||']
Procesamiento 7
Stack Temp: ['&', '[']
Expresion Postfijo: ['perr', 'gat', '||']
Procesamiento 8
Stack Temp: ['&', '[', '[']
Expresion Postfijo: ['perr', 'gat', '||']
Procesamiento 9
Stack Temp: ['&', '[', '[', ”]
Expresion Postfijo: ['perr', 'gat', '||']
Procesamiento 10
Stack Temp: ['&', '[', '[', ”, '[']
Expresion Postfijo: ['perr', 'gat', '||']
Procesamiento 11
Stack Temp: ['&', '[', '[', ”, '[']
Expresion Postfijo: ['perr', 'gat', '||', 'delfin']
Procesamiento 12
Stack Temp: ['&', '[', '[', ”, '[', '||']

Expresion Postfijo: ['perr', 'gat', '||', 'delfin']
Procesamiento 13
Stack Temp: ['&', '[', '[', '', '[', '||']
Expresion Postfijo: ['perr', 'gat', '||', 'delfin', 'oso']
Procesamiento 14
Stack Temp: ['&', '[', '[', '']
Expresion Postfijo: ['perr', 'gat', '||', 'delfin', 'oso', '||']
Procesamiento 15
Stack Temp: ['&', '[']
Expresion Postfijo: ['perr', 'gat', '||', 'delfin', 'oso', '||', '']
Procesamiento 16
Stack Temp: ['&', '[', '&']
Expresion Postfijo: ['perr', 'gat', '||', 'delfin', 'oso', '||', '']
Procesamiento 17
Stack Temp: ['&', '[', '&']
Expresion Postfijo: ['perr', 'gat', '||', 'delfin', 'oso', '||', '', 'lob']
Procesamiento 18
Stack Temp: []
Expresion Postfijo: ['perr', 'gat', '||', 'delfin', 'oso', '||', '', 'lob', '&', '&']
Expresion Postfijo Final: ['perr', 'gat', '||', 'delfin', 'oso', '||', '', 'lob', '&', '&']
Procesamiento de la consulta:
$'perr' : ['ejemplo_3.txt','ejemplo_4.txt','ejemplo_5.txt']$
'gat': ['$ejemplo_2.txt$', '$ejemplo_3.txt$', '$ejemplo_4.txt$']
Operador ||| (or) seleccionado
$['ejemplo_3.txt','ejemplo_4.txt','ejemplo_5.txt'] \vee ['ejemplo_2.txt','ejemplo_3.txt','ejemplo_4.txt']$
$['ejemplo_2.txt','ejemplo_3.txt','ejemplo_4.txt','ejemplo_5.txt']$
'delfin': ['$ejemplo_1.txt$', '$ejemplo_2.txt$', '$ejemplo_4.txt$']
'oso': ['$ejemplo_1.txt$', '$ejemplo_4.txt$', '$ejemplo_5.txt$']
Operador ||| (or) seleccionado
$['ejemplo_1.txt','ejemplo_2.txt','ejemplo_4.txt'] \vee ['ejemplo_1.txt','ejemplo_4.txt','ejemplo_5.txt']$
$['ejemplo_1.txt','ejemplo_2.txt','ejemplo_4.txt','ejemplo_5.txt']$
Operador ! (not) seleccionado
$\neg['ejemplo_1.txt','ejemplo_2.txt','ejemplo_4.txt','ejemplo_5.txt']$
$['ejemplo_3.txt']$
'lob': ['$ejemplo_1.txt$', '$ejemplo_2.txt$', '$ejemplo_3.txt$', '$ejemplo_4.txt$']
Operador && (and) seleccionado
$['ejemplo_3.txt'] \wedge ['ejemplo_1.txt','ejemplo_2.txt','ejemplo_3.txt','ejemplo_4.txt']$
$['ejemplo_3.txt']$
Operador && (and) seleccionado
$['ejemplo_2.txt','ejemplo_3.txt','ejemplo_4.txt','ejemplo_5.txt'] \wedge ['ejemplo_3.txt']$
$['ejemplo_3.txt']$
Documentos que satisfacen la consulta:
¿Documento 3 : $ejemplo_3.txt$

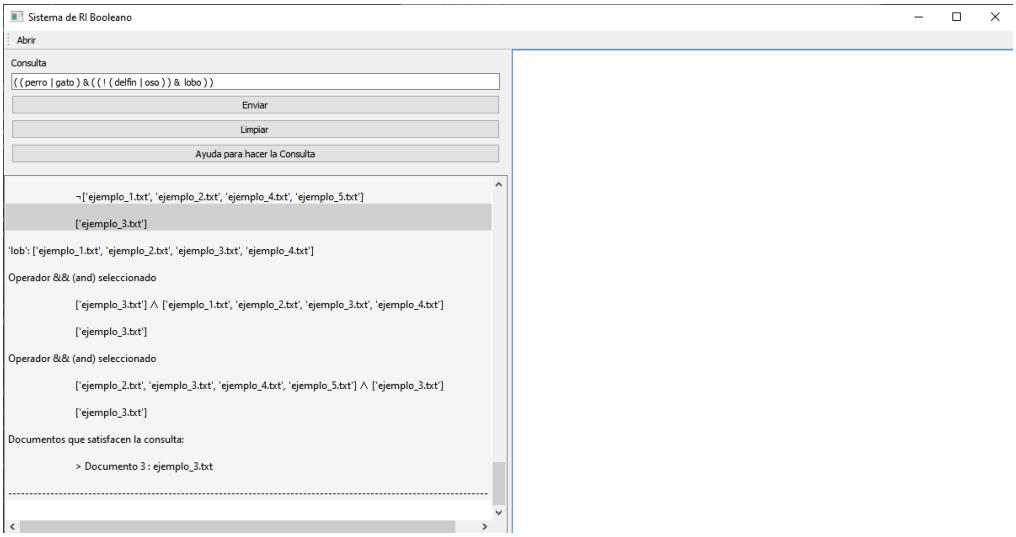


Figura 11: Interfaz del sistema

### 5.3.2. consulta 2

( ( perro || gato ) & ( ( ! ( delfin ) || oso ) & lobo ) )

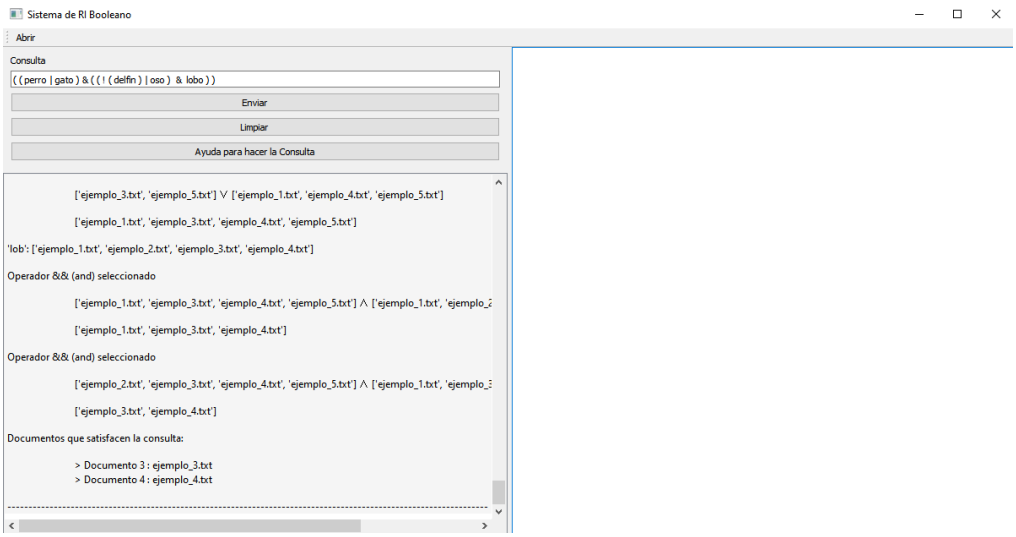


Figura 12: Interfaz del sistema

# 6. Manual tecnico

## 6.1. El objetivo del programa

Es desarrollar un sistema de recuperación de información que pueda realizar consultas booleanas en una colección de documentos de texto. Aquí hay algunos puntos clave sobre el objetivo del programa:

- Recuperación de Información: El programa está diseñado para permitir a los usuarios realizar consultas para buscar documentos que contengan ciertas palabras o combinaciones de palabras.

- Consultas Booleanas: El sistema admite consultas booleanas, lo que significa que los usuarios pueden combinar términos de búsqueda con operadores lógicos como AND, OR y NOT para refinar sus consultas y obtener resultados más relevantes.
- Procesamiento de Texto: Antes de ejecutar las consultas, el programa procesa el texto de los documentos y las consultas de varias maneras, como eliminación de caracteres especiales, conversión a minúsculas, eliminación de stopwords, stemming, etc.
- Interfaz Gráfica: Proporciona una interfaz gráfica de usuario (GUI) para que los usuarios interactúen con el sistema de manera intuitiva.
- Generación de Resultados: Después de ejecutar una consulta, el sistema devuelve una lista de documentos que coinciden con los criterios de búsqueda especificados por el usuario.
- Almacenamiento y Manejo de Datos: El sistema maneja la estructura de datos necesaria para almacenar la información sobre los documentos y las consultas, como matrices binarias, tablas hash, etc.

## 6.2. Instalacion

- os: Permite realizar operaciones relacionadas con archivos y directorios.
- re: Se utiliza para realizar operaciones con expresiones regulares. unicodedata.normalize: Ayuda a normalizar los caracteres Unicode en texto.
- nltk: Es la biblioteca de procesamiento de lenguaje natural. Se utiliza para tokenizar, eliminar palabras vacias y realizar stemming.
- numpy: Proporciona funcionalidades para operaciones con matrices y arreglos multidimensionales.
- pandas: Se utiliza para el manejo, analisis y procesamiento de datos en forma tabular.
- SnowballStemmer: Se utiliza para configurar stemmers para espanol e ingles utilizando Snowball Stemmer. Estos stemmers se utilizan para reducir las palabras a su forma raiz.
- hashlib: Permite calcular hash de cadenas de texto.
- pyparsing: Usada para definir y analizar gramaticas, especialmente en la definicion de consultas booleanas.
- PyQt5: Es una biblioteca de enlace de Python para la biblioteca de interfaz grafica multiplataforma Qt. Se utiliza para crear la interfaz grafica de usuario.
- sys: Proporciona funciones y variables que se utilizan para manipular diferentes partes del entorno de ejecucion de Python.

Teniendo esta libreras instaladas y posteriormente tener Python y Visual Code, teniendo los plugis de depuracion, se podra realizar la revision del programa.

## 6.3. Modulos

### 6.3.1. __init__

- Descripción general:
  - El código def __init__(self): es el método constructor de una clase en Python. Este método se ejecuta automáticamente cuando se crea una instancia de la clase. Su función principal es inicializar los atributos de la instancia y realizar cualquier configuración o tarea de inicialización necesaria.
- Explicación detallada:
  - Inicialización de atributos:

- self.consulta: Se inicializa a None, lo que indica que no hay una consulta actual.
- self.close: Se inicializa a False, lo que indica que el programa no debe cerrarse.

- Creación de carpetas:
  - Se verifica si existen las carpetas diccionario y consultas. Si no existen, se crean. Estas carpetas se utilizan para almacenar el diccionario y las consultas, respectivamente.
- Inicialización de variables:
  - self.numero_documentos: Se inicializa a 0, lo que indica que aún no se han procesado documentos.
  - self.directorio: Se establece la ruta del directorio que contiene los archivos de texto a procesar.
  - self.nombres_docs: Se crea una lista vacía para almacenar los nombres de los archivos de texto procesados.
- Definición de la gramática para los operadores y operandos:
  - Se utilizan expresiones regulares para definir la gramática de los operadores y operandos válidos en las consultas.
  - self.operand: Representa un operando, que puede ser una cadena entre comillas simples o una palabra formada por letras y números.
  - self.not_operator: Representa el operador de negación ( o ¬).
  - self.and_operator: Representa el operador AND (&&, &, o ∧).
  - self.or_operator: Representa el operador OR (|||, ||, o ∨).
  - self.grammar_elements: Se crea una lista con los símbolos gramaticales utilizados.
- Definición de la precedencia de los operadores:
  - Se establece la precedencia de los operadores utilizando una lista de tuplas. Cada tupla contiene el operador, su precedencia y su asociatividad (izquierda o derecha).
- Definición de la expresión lógica:
  - Se crea una expresión lógica utilizando la notación infija y la precedencia de los operadores.
  - self.expression: Representa la expresión lógica completa.
- Inicialización de variables relacionadas con la consulta:
  - self.matriz_binaria: Se crea una lista vacía para almacenar la matriz binaria de la consulta.
  - self.query_stem_elements: Se crea una lista vacía para almacenar los elementos de la consulta después de aplicar stemming.
  - self.binary_array_list: Se crea una lista vacía para almacenar los arrays binarios de cada elemento de la consulta.
  - self.hash_table: Se crea un diccionario vacío para almacenar la tabla hash de términos y documentos.
  - self.postfijo: Se crea una lista vacía para almacenar la notación postfija de la consulta.
  - self.impresion_postfijo: Se crea una lista vacía para almacenar la representación impresa de la notación postfija.
- Inicialización de mutex y condición:
  - self.mutex: Se crea un mutex para proteger el acceso a las variables compartidas.
  - self.condition: Se crea una condición para sincronizar el acceso a las variables compartidas.
  - self.consulta_recibida: Se inicializa a None, lo que indica que no hay una consulta recibida actualmente.
- Inicialización de la interfaz gráfica:
  - self.app_thread: Se crea un hilo para inicializar la interfaz gráfica.
  - self.app_thread.start(): Se inicia el hilo para ejecutar la interfaz gráfica.

- En Resumen:

  - El método def __init__(self): realiza la inicialización completa de la instancia de la clase, preparando el entorno para el procesamiento de consultas y la interacción con la interfaz gráfica.

#### 6.3.2. init_gui_thread

- Descripción general:
  - El código `def init_gui_thread(self):` se ejecuta en un hilo separado para inicializar la interfaz gráfica de usuario (GUI) de la aplicación. Su función principal es crear la ventana principal de la GUI, conectar las señales de la interfaz a los métodos de la clase y ejecutar el ciclo de eventos de la aplicación.
- Explicación detallada:
  - Inicialización de la aplicación Qt:
    - self.app = QApplication(sys.argv): Crea una instancia de la aplicación Qt utilizando la lista de argumentos de línea de comandos (sys.argv).
  - Creación de la interfaz gráfica:
    - self.interfaz = InterfazRI(self): Crea una instancia de la clase InterfazRI, que es la clase que representa la ventana principal de la GUI.
  - Conexión de señales:
    - self.interfaz.consulta_signal.connect(self.recibir_consulta): Conecta la señal `consulta_signal` de la interfaz al método `recibir_consulta` de la clase actual. Esto significa que cuando el usuario ingresa una consulta en la GUI, se emitirá la señal `consulta_signal` y se llamará al método `recibir_consulta` para procesarla.
  - Ejecución de la aplicación:
    - `self.app.exec_()`: Ejecuta el ciclo de eventos de la aplicación Qt. Esto significa que la GUI se mostrará en la pantalla y la aplicación responderá a los eventos del usuario, como clics en botones y entradas de teclado.
- Resumen:
  - El método `def init_gui_thread(self):` permite que la interfaz gráfica de usuario se ejecute en un hilo separado, lo que evita que bloquee el hilo principal de la aplicación. Esto permite que la aplicación continúe respondiendo a otros eventos mientras el usuario interactúa con la GUI.

#### 6.3.3. procesar_texto

- Descripción general:
  - El código `def procesar_texto(self, contenido):` se encarga de preprocesar un texto dado para su posterior uso en la búsqueda de información. Su función principal es convertir el texto a minúsculas, eliminar caracteres especiales, acentos y datos numéricos, y finalmente separar el texto en palabras individuales.
- Explicación detallada:
  - Conversión a minúsculas:
    - contenido = contenido.lower(): Convierte todo el texto en minúsculas. Esto asegura que la búsqueda no sea sensible a las mayúsculas y minúsculas.
  - Eliminación de caracteres especiales:
    - contenido = `re.sub(r'[^`<br>`w`<br>`s]', '' contenido)`: Utiliza la biblioteca re y una expresió regular para eliminar cualquier carácter que no sea un letra, número o espacio en blanco. Esto limpia el text de caracteres especiales que podrían interferir con e procesamiento.
  - Eliminación de acentos y normalización:

- contenido = `re.sub(r"([^n u0300 u036f]|n(? u0303(?![ u0300 u036f])))[ u0300 u036f]+", r" 1", normalize("NFD", contenido), 0, re.I)`: Esta línea utiliza expresiones regulares y la función normalize para eliminar los acentos de las palabras en español.

- Eliminación de datos numéricos:
  - contenido = re.sub(r'+', '', contenido): Elimina cualquier número del texto utilizando una expresión regular. Esto permite que la búsqueda se centre en el contenido textual y no en números.
- Separación de palabras:
  - contenido = nltk.word_tokenize(contenido): Utiliza la biblioteca nltk y la función word_tokenize para separar el texto en palabras individuales.
- Conversión a matriz:
  - contenido = np.asarray(contenido): Convierte la lista de palabras en un array de NumPy. Esto facilita el manejo posterior del texto procesado.

- Resumen:
  - El método `def procesar_texto(self, contenido):` realiza un preprocesamiento completo del texto de entrada, preparándolo para su uso en la búsqueda de información. Elimina ruido y normaliza el texto para mejorar la precisión de la búsqueda.

#### 6.3.4. crear_tabla

crear tabla(self, array, nombre). Este metodo crea una tabla a partir de un array y la almacena en un archivo de texto. Es utilizado para almacenar tablas de palabras, frecuencias, stopwords y stemming.

#### 6.3.5. crear_matriz_binaria

crear matriz binaria(self, palabras, numero documentos). Este metodo crea una matriz binaria donde cada fila representa una palabra y cada columna representa un documento. El valor en cada celda de la matriz indica si la palabra aparece en el documento correspondiente.

#### 6.3.6. computar_hash

computar hash(self, word). Calcula el valor hash MD5 de una palabra y lo devuelve como una cadena hexadecimal. crear hash table(self, words, binary dict). Crea una tabla hash que asocia palabras a sus valores binarios en la matriz binaria.

#### 6.3.7. token_snow

token stopw(self, string). Realiza la tokenizacion, elimina las stopwords y aplica stemming a una cadena de texto.

#### 6.3.8. procesar bool exp

procesar bool exp(self, query str). Toma una cadena de consulta booleana y la convierte en una lista de elementos que representan la expresion booleana.

#### 6.3.9. procesar postfijo

procesar postfijo(self, bool exp list). Toma una lista de elementos que representan la expresion booleana en notacion infija y la convierte en notacion postfija.

#### 6.3.10. operador or

or operator(self). Implementa la operacion logica OR en listas binarias y devuelve el resultado.

#### 6.3.11. operador and

and operator(self). Implementa la operacion logica AND en listas binarias y devuelve el resultado.

#### 6.3.12. operador not

not operator(self). Implementa la operacion logica NOT en una lista binaria y devuelve el resultado.

#### 6.3.13. esperar consulta

esperar consulta(self). Bloquea hasta que se recibe una consulta desde la interfaz grafica.

#### 6.3.14. recibir consulta

recibir consulta(self, consulta). Recibe una consulta desde la interfaz grafica y la almacena para su posterior procesamiento.

#### 6.3.15. procesar query

process query(self, operator). Ejecuta una operacion booleana (AND, OR o NOT) en los arrays binarios y devuelve el resultado.

#### 6.3.16. hash query

hash query(self, hash). Busca en la tabla hash un valor binario correspondiente a un hash de palabra.

#### 6.3.17. ejecutar query

ejecutar query(self, postfijo). Ejecuta una consulta booleana en notacion postfija utilizando los arrays binarios y devuelve el resultado.

#### 6.3.18. obtener docs

obtener docs(self, array docs). Imprime los documentos que satisfacen una consulta booleana en funcion del resultado de la evaluacion.

#### 6.3.19. clear

clear(self). Limpia las variables y reinicia el sistema para una nueva consulta.

#### 6.3.20. enviar consulta

enviar consulta(self). Este metodo se llama cuando se hace clic en el boton "Enviar"o se presiona la tecla Enter despues de ingresar una consulta en el cuadro de texto. Toma la consulta ingresada por el usuario y la emite como una se˜nal consulta signal para que el sistema de recuperacion de informacion la procese.

#### 6.3.21. update output

update output(self, output). Este metodo agrega resultados a la lista de resultados en la interfaz grafica.

#### 6.3.22. init interfaz

init (self, sistema ri). El constructor de la clase InterfazRI inicializa la interfaz grafica de usuario (GUI) y recibe una instancia de SistemaRI como argumento para establecer una comunicacion entre la interfaz y el sistema de recuperacion de informacion.

#### 6.3.23. initUI

initUI(self). Este metodo configura la interfaz de usuario utilizando la biblioteca PyQt5. Define la estructura de la ventana principal, que incluye un menu, un area para ingresar consultas, un boton para enviar consultas, una lista para mostrar resultados y un area para mostrar el contenido del documento seleccionado.

#### 6.3.24. openDocument

openDocument(self). Abre un cuadro de dialogo para seleccionar y abrir un documento de texto. Luego, muestra el contenido del documento en el area de visualizacion.

#### 6.3.25. update document displayed

update document displayed(self, item). Este metodo se llama cuando se hace clic en un elemento de la lista de resultados. Si el elemento hace referencia a un documento, muestra el contenido del documento seleccionado en el area de visualizacion.

#### 6.3.26. mostrar ayuda

mostrar ayuda(self). Muestra un cuadro de dialogo modal (QDialog) con informacion de ayuda sobre como usar la aplicacion.

#### 6.3.27. closeEvent

closeEvent(self, event). Este metodo maneja el evento de cierre de la ventana principal y asegura que la aplicacion se cierre correctamente cuando se cierra la ventana.

# 7. Codigos

Para poder ver el codigo completo y bien documentado de la aplicación puede dirigirse al siguiente enlace: `https://github.com/t4dokiary/RI-Programa1`.
No solo podra ver el codigo fuente de la aplicacion si no que tambien podra descargarlo para despues poder ejecutarlo en su computadora y ver el funcionamiento de la aplicacion en su totalidad.

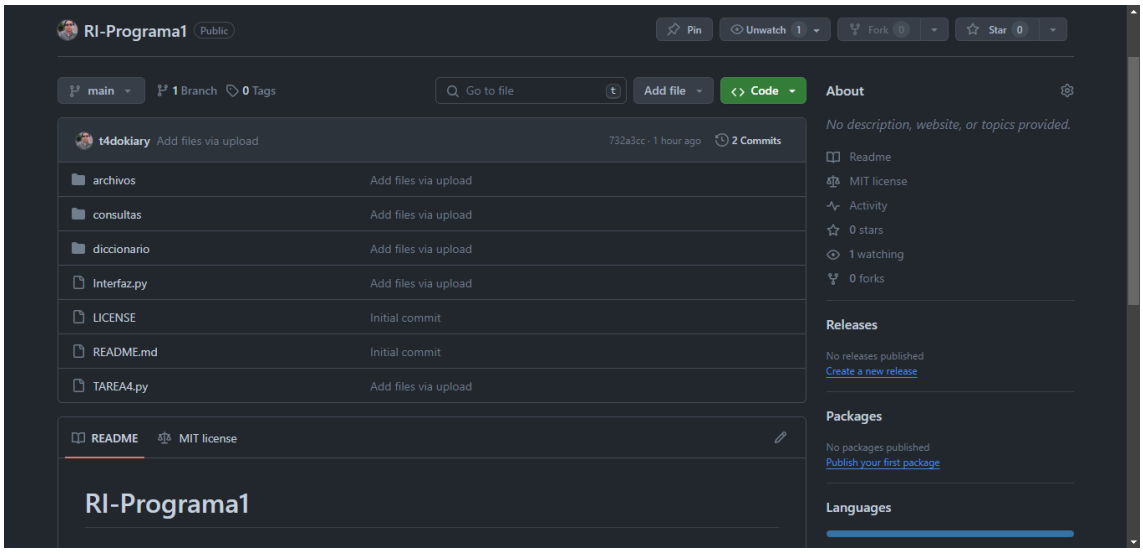


Figura 13: Código fuente de la aplicación