Sistema de Recomendación de Soluciones a Errores de Programación en Lenguaje C usando Minería de Datos

Christhian Guevara[1] Masun Nabhan Homsi[2]

[1]AUTOR
[2]TUTOR

Coordinación de Ingeniería de la Computación
Universidad Simón Bolívar

Sartenejas, Enero de 2019

Contenido

Síntesis

Desarrollar Software => Solucionar Errores

Bugs

Son errores de comprensión o lógica, e incluso comportamientos no definidos o inesperados.

19 % del tiempo se invierte en consultar fuentes externas.

Algunas fuentes externas son:

- Foros
- Listas de Correo Electrónico
- Sitios de Preguntas y Respuestas

Síntesis

Problemas del Proceso de Búsqueda:

- Falta de automatización
- Se requiere habilidad para formular la consulta
- La consulta no guarda relación con el código

"Los motores de búsqueda son ineficaces para realizar consultas empleando el código fuente".

Planteamiento del Problema

El proceso de solucionar errores es repetitivo y tedioso, en consecuencia:

- Requiere consultar fuentes externas con frecuencia
- No se emplea el código fuente en la consulta
- Se debe describir el problema e identificar su origen
- La documentación generalmente es escasa, incomprensible o inexistente.

=> Impide encontrar solución al problema en tiempo razonable y pospone la realización de la tarea asignada.

Antecedentes

Trabajos previos en el área apuntan a desarrollar un Sistema de Recomendación que se beneficie del conocimiento provisto por la comunidad de Stack Overflow.

Algunas de estas investigaciones son:

Debugging with the Crowd presenta un Sistema de Recomendación como una herramienta para *debugging*.

Prompter es un Sistema de Recomendación que se muestra como un *plug-in* para *Eclipse IDE*.

PDE4Java es un motor para detección de documentos similares en *Java*.

Justificación

Muchos de los problemas que afectan el desarrollo de software se relacionan directamente con el proceso de búsqueda. El programador invierte tiempo valioso y sus consultas no suelen ser exitosas.

"Queda en evidencia la falta de herramientas que faciliten el proceso de búsqueda al desarrollador".

Objetivos

Objetivo General

Desarrollar un sistema capaz de recomendar soluciones a errores cometidos por el programador en lenguaje C, apoyándose sobre algoritmos de Minería de Datos.

Objetivos

Objetivos Específicos

- Aplicar algoritmos de minería de datos para explotar los conocimientos de programación en lenguaje c de la comunidad de programadores del sitio web *Stack Overflow*.
- Implementar una herramienta que ayude al programador a identificar errores en el código, mediante una lista de recomendaciones de posibles soluciones proporcionada por la comunidad.
- Establecer un modelo de clasificación que incluya aspectos relacionados a la reputación de los usuarios de *Stack Overflow*.

Minería de Datos

El objetivo general del proceso de minería de datos consiste en extraer información de una fuente de datos e intentar convertir esa información en un conjunto aprovechable de conocimientos.

Proceso que consiste de 3 pasos:

- Preprocesamiento.
- Análisis.
- Interpretación.

Similitud entre Documentos

Es un subárea de minería de datos que aborda el problema de encontrar documentos similares.

Ampliamente utilizado en:

- Detección de Plagio.
- Detección de Clones.

A su vez, involucra el análisis del código fuente.

Similitud entre Documentos

Posibles variaciones en el código fuente:

- cambio de formato
- cambio de identificadores
- cambio en el orden de los operandos en las expresiones
- cambio del tipo de datos
- sustitución de expresiones por sus equivalentes
- introducción de declaraciones redundantes
- cambio en el orden de declaraciones
- cambio en la estructura de las instrucciones de iteración
- cambio en la estructura de las declaraciones de selección
- sustitución de las llamadas a función por el cuerpo
- introducción de declaraciones no estructuradas
- combinación del código original y la copia
- traducción del código de un lenguaje a otro

Similitud entre Documentos

Clasificación del código fuente por similitud:

Tipo I Idénticos exceptuando las variaciones en espacios, diseño y comentarios.

Tipo II Sintácticamente idénticos exceptuando las variaciones en identificadores, literales, tipos, diseño y comentarios.

Tipo III Con modificaciones adicionales. Expresiones alteradas y variaciones en identificadores, literales, tipos, diseño y comentarios.

Tipo IV Que realizan el mismo cálculo pero son implementados a través de diferentes variantes sintácticas.

Similitud entre Documentos

Clasificación general de los métodos desarrollados:

- Comparación basada en texto
- Comparación basada en lexemas
- Comparación basada en árboles de sintaxis
- Comparación basada en grafos de dependencia
- Comparación basada en métricas
- Comparación empleando enfoques híbridos

Document Fingerprinting

"Es una técnica para la detección precisa de copias completas y parciales entre documentos".

Consiste en dividir el texto en subsecuencias, calcular el *hash* para cada subsecuencia y crear la firma a partir de éstos.

Secuencia de ejemplo pablito clavó un clavito

Secuencia de *4-grams* pabl abli blit lito itoc toca ocav cavo avou voun ounc uncl ncla clav lavi avit vito

Secuencia de *hashes* 77 72 43 19 91 50 17 98 8 88 67 39 78 7 42 11 92

Document Fingerprinting

Similitud

Se puede medir la similitud empleando el coeficiente de similitud de *Jaccard*, para ello, los elementos del conjunto serán aquellos valores de *hash* que son calculados a partir del documento.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Document Fingerprinting

Firma

La firma se puede crear empleando la técnica de *MinHash*, que sirve para estimar de forma eficiente el coeficiente de similitud de *Jaccard*.

MinHash

Se selecciona un tamaño para la firma, se escoge igual cantidad de funciones de *hash* universales, se asocia una función a cada casilla y se escoge el mínimo valor obtenido al aplicar la función correspondiente al conjunto de valores.

Document Fingerprinting

Índice Invertido

El *Locality-Sensitive Hashing* es una técnica que se comporta de forma similar a un índice invertido, permite realizar búsquedas sobre las firmas generadas mediante el *MinHash*.

Locality-Sensitive Hashing

El enfoque general para el *LSH*, consiste en emplear diversas funciones de *hash* para asociar los elementos de un conjunto a distintas 'casillas'. De esta manera, los elementos similares entre sí, tienen mayores probabilidades de ser agrupados en las mismas 'casillas', que los demás elementos. Luego, se considera como 'par candidato' a cualquier par de elementos que se encuentre en la misma 'casilla'

Herramientas

IDE

Extensiones

Java
C/C++
Plug-in

Librerías

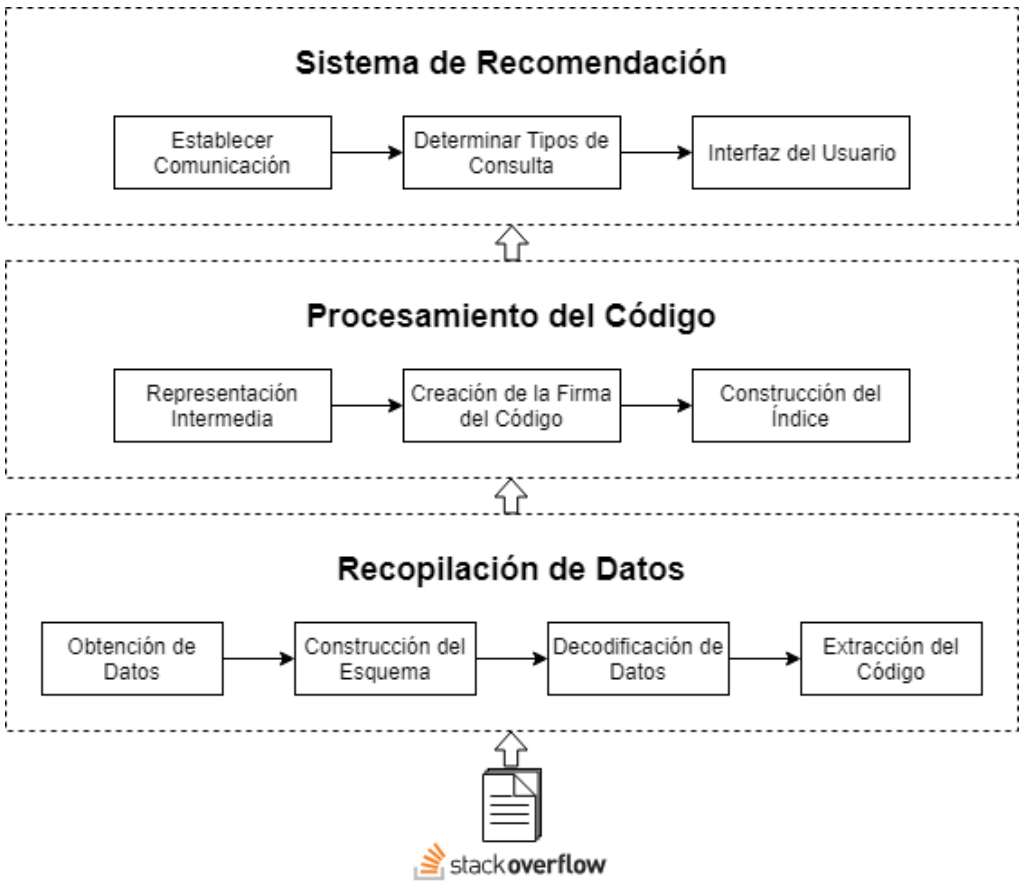
CDT: CORE
Apache
Commons Text

Base de Datos

Extensiones

PL/pgSQL

Esquema Metodológico



Recopilación de Datos

Tabla: Cantidad de tópicos por etiqueta.

Etiqueta	Preguntas	Respuestas	Respuestas Aceptadas	Total
todas	7,990,786	13,684,117	4,596,859	21,736,593
c/c++	442,450	978,214	284,297	1,420,664
c++	314,869	684,362	203,447	999,231
c	151,856	362,244	96,949	514,100

Recopilación de Datos

Tabla: Tópicos relacionados con c/c++.

Descripción	Preguntas	Respuestas	Total
todos	442,450	978,214	1,420,664
con código	314,247	459,073	773,320
relación	71,02 %	46,93 %	54,43 %

Tabla: Cantidad de fragmentos de código

Descripción	Preguntas	Respuestas	Total
fragmentos	598,775	778,940	1,377,716
relación	43,46 %	56,54 %	100,0 %

Recopilación de Datos

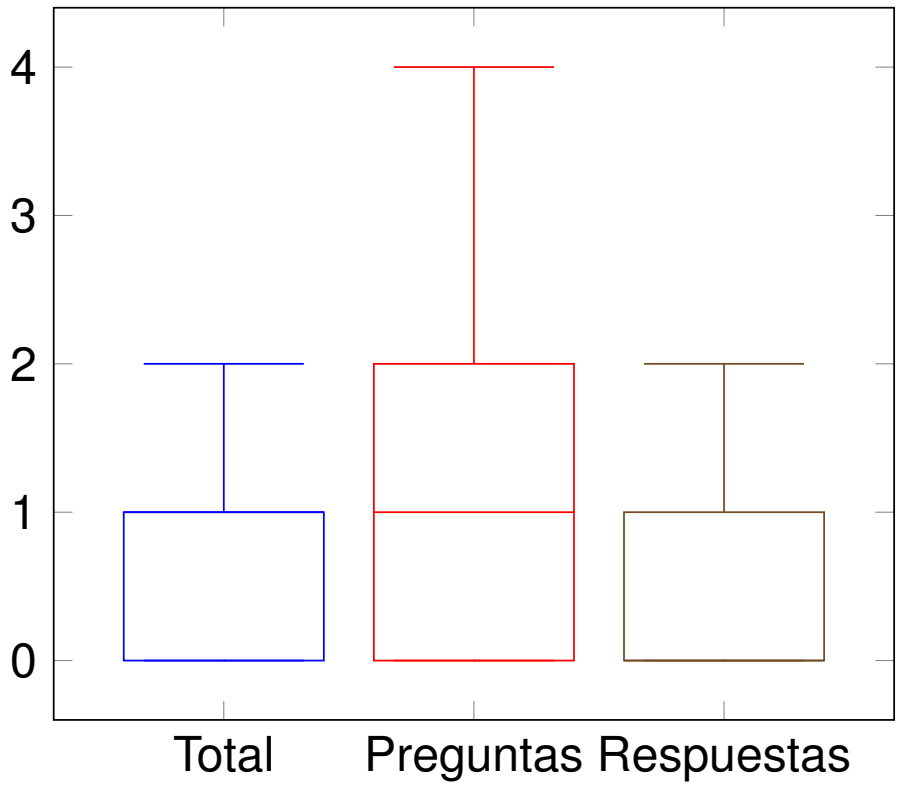


Figura: Estadísticas descriptivas de los fragmentos de código.

Recopilación de Datos

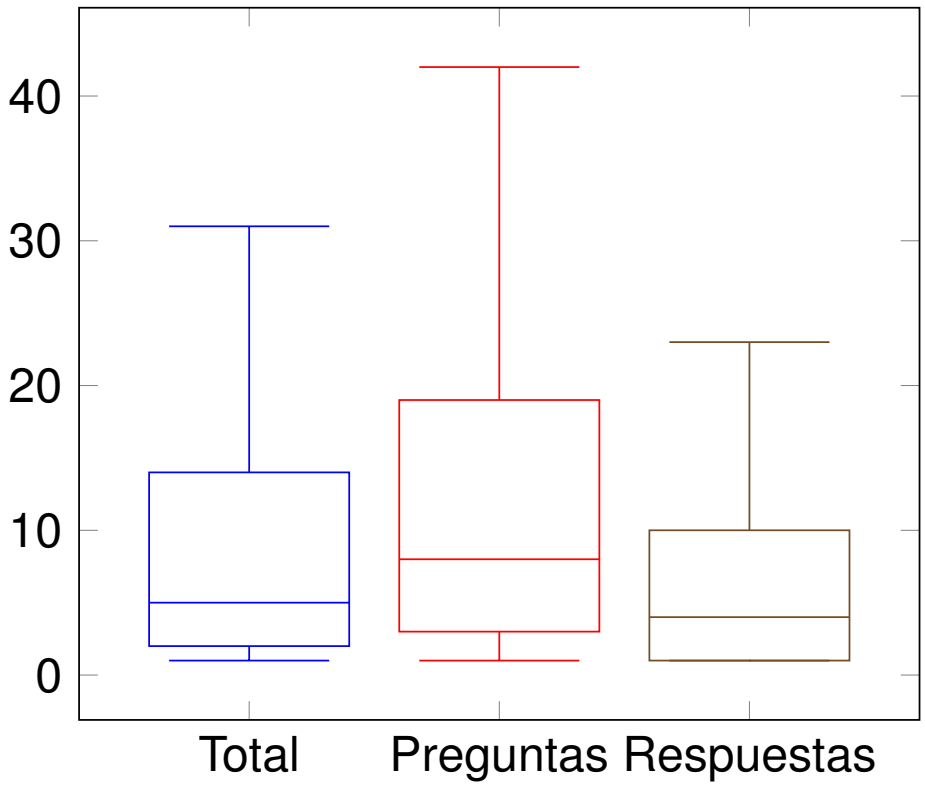


Figura: Estadísticas descriptivas de las líneas de código.

Procesamiento del Código

Tópico ejemplo:

```
1 decimal trans = trackBar1.Value / 5000;
2 this.Opacity = trans;
```

Fragmento de Código 1: Fragmento de código en c++

Lexemas identificados:

identifier, identifier, assign, identifier, dot, identifier, div, integer, semi, identifier, dot, identifier, assign, identifier, semi

Representación obtenida:

1, 1, 38, 1, 50, 1, 52, 2, 5, 1, 50, 1, 38, 1, 5

Procesamiento del Código

Secuencia de *4-grams*:

```
(1,1,38,1), (1,38,1,50), (38,1,50,1), (1,50,1,52), (50,1,52,2),
  (1,52,2,5), (52,2,5,1), (2,5,1,50), (5,1,50,1), (1,50,1,38),
                    (50,1,38,1), (1,38,1,5)
```

Secuencia de *hashes*:

```
5328, 767281, 110488464, 1016115, 146320561, 1057684, 152306496,
    3068977, 11950992, 1016101, 146318544, 767236, 110482127
```

$$firma(C_1) = (604227, 14244687, \ldots, 17027248)$$

Procesamiento del Código

Tabla: Configuración del *MinHash*.

Funciones	Error
100	10,0 %

Tabla: Configuración del *LSH*.

Bandas	Filas	Umbral
20	5	54,93 %

Procesamiento del Código

Conjunto 1:

Tabla: Posición de los fragmentos de código.

Top1	Top10	Otro
730	122	148

Tabla: Estadísticas generales.

T.Total	T.Consulta	P.Candidatos	E.Firma
194*ms*	185*ms*	500	0,28*ms*

Procesamiento del Código

Conjunto 2:

Tabla: Clasificación de Fragmentos de Código.

Rango 1	Rango 2	Rango 3	Rango 4	Sin rango
12	8	20	55	100

Rango 1 Estructuralmente iguales. Similitud >= 95 %.

Rango 2 Variaciones en los *n-grams*. Similitud >= 85 %.

Rango 3 Cambios estructurales básicos. Similitud >= 70 %.

Rango 4 Cambios estructurales notables. Similitud >= 50 %.

Sin rango Poca o ninguna similitud.

Procesamiento del Código

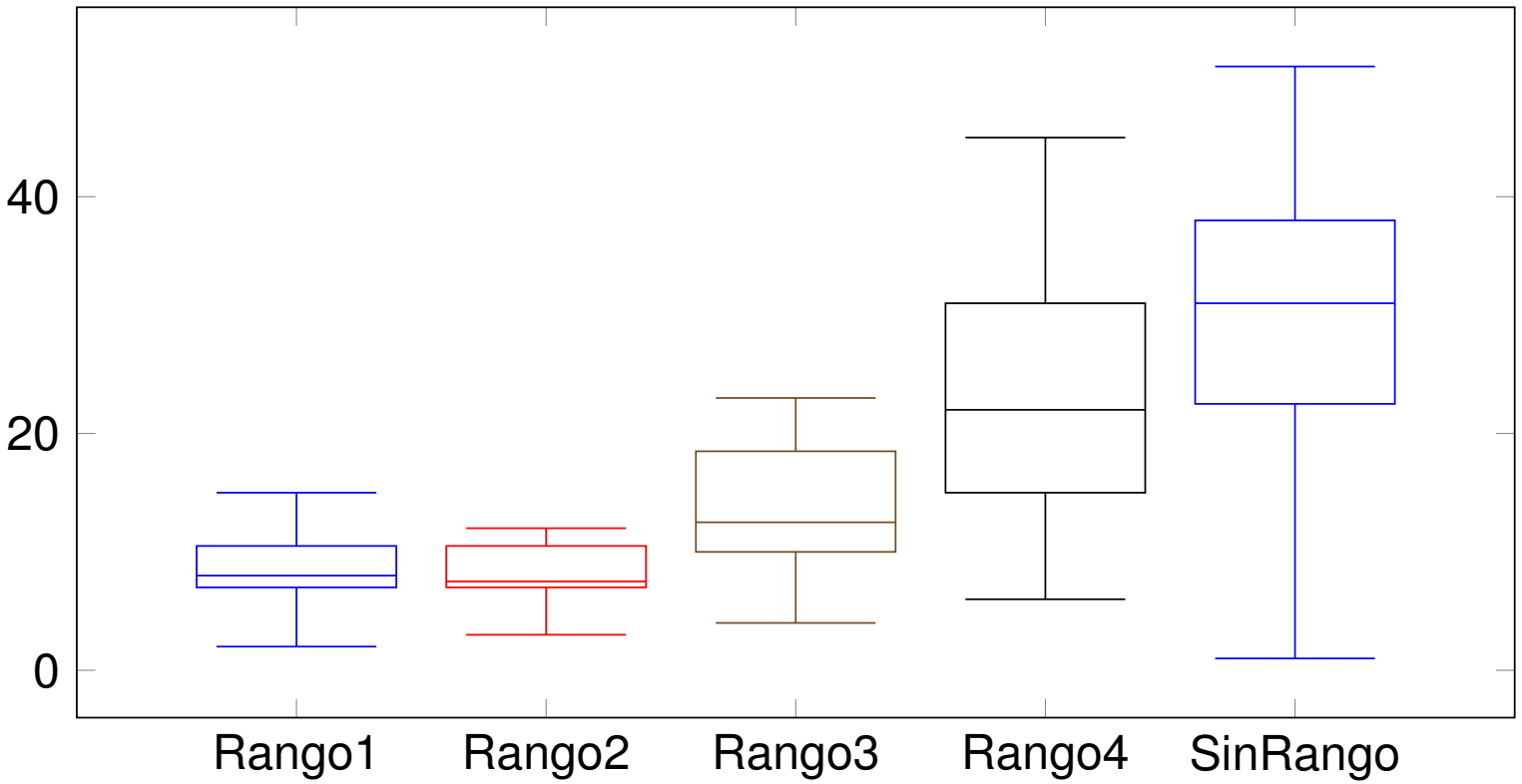


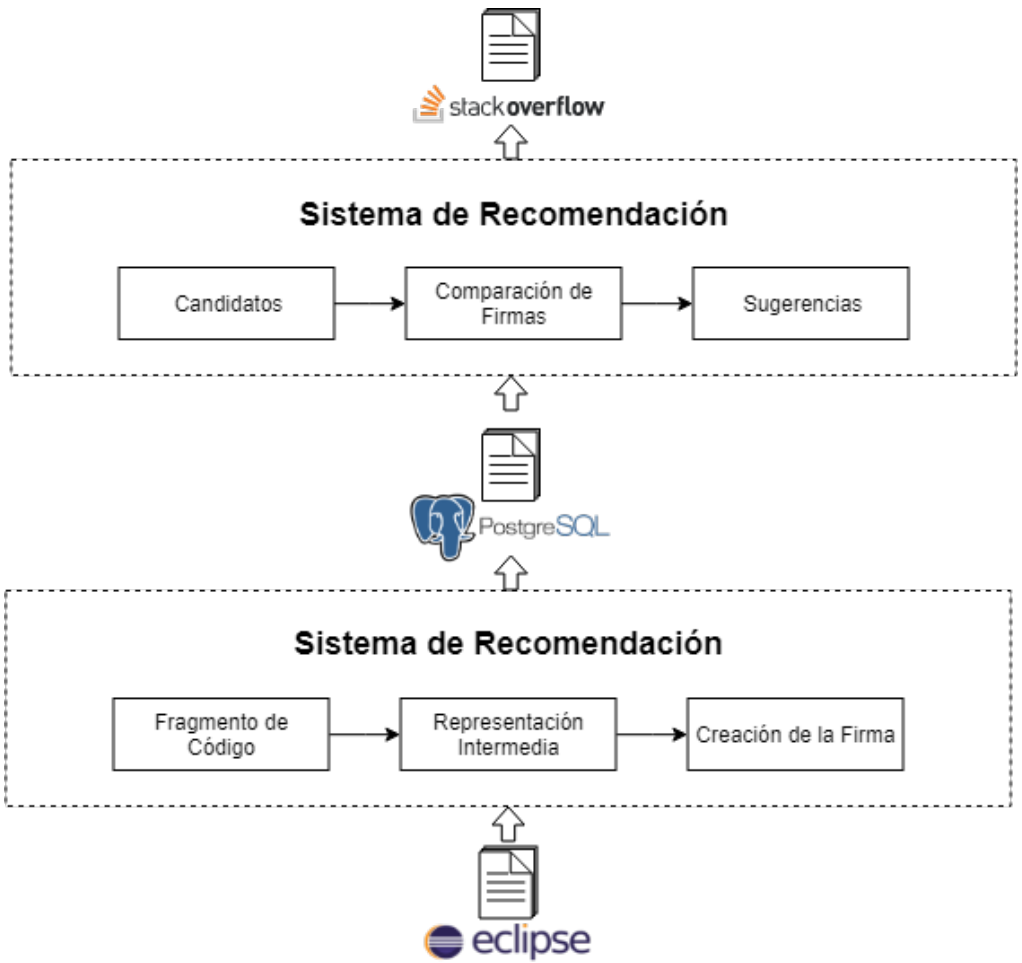
Figura: Estadísticas descriptivas de las líneas de código.

Procesamiento del Código

Tabla: Clasificación de Fragmentos de Código en el rango 8 ± 1.

Rango 1	Rango 2	Rango 3	Rango 4	Sin rango
6	4	3	1	2
37,5 %	25,0 %	18,75 %	6,25 %	12,5 %

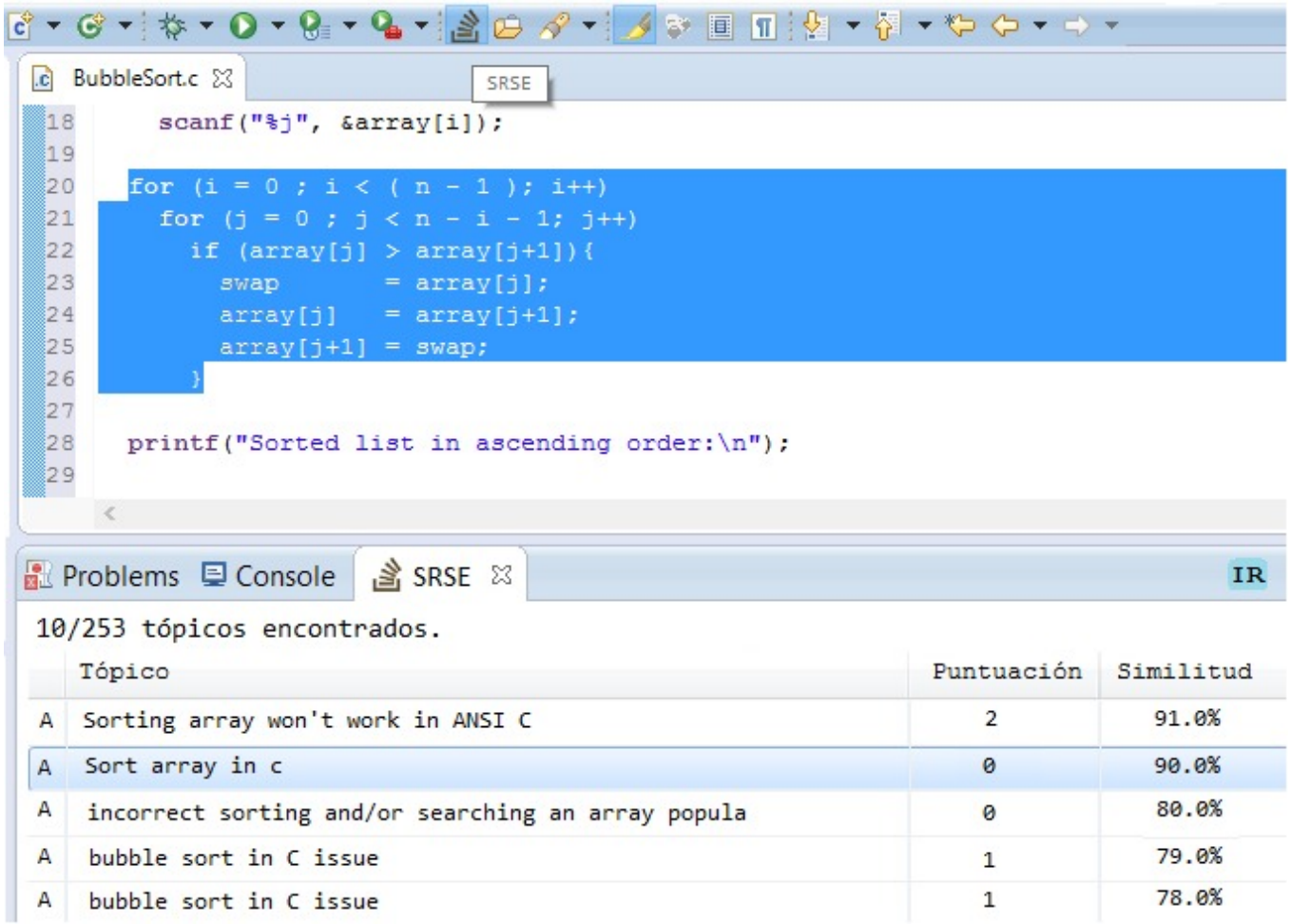
Estructura de Comunicación



Filtros de Consulta

Tipo	Respuesta Aceptada	Rango
-	-	-
Pregunta Respuesta	SI	1 2 3 4

Interfaz Gráfica



Evaluación del Sistema de Recomendación

Conjunto 3:

Tabla: Resultados generales.

Algoritmo	Resultado	Líneas	Candidatos	Porcentaje
Alexander Bogomolny	*Encontrado*	36	3	83 %
Bin Packing	*Noencontrado*	30	6	55 %
Binary Search Tree	*Encontrado*	163	47	64 %
Fibonacci Search	*NoEncontrado*	48	0	0 %
Fisher Yates	*Encontrado*	33	2	63 %
Hash Table	*NoEncontrado*	184	0	0 %
0-1 Knapsack Problem	*Encontrado*	78	1	94 %
Naor Reigngold	*NoEncontrado*	17	0	0 %
Selection Sort	*Encontrado*	41	6	68 %
Sieve Eratosthenes	*Encontrado*	22	2	78 %

Evaluación del Sistema de Recomendación

Tabla: Síntesis de Resultados

Resultado	Cantidad	Porcentaje
Encontrado	6	60,0 %
NoEncontrado	4	40,0 %
Relacionado	0	0,0 %

Conclusiones

"El sistema implementado demostró funcionar correctamente al sugerir tópicos con fragmentos de código, iguales o similares al proporcionado por el usuario"

Resultados parciales:

- El índice desarrollado es eficaz.
- Baja especificidad en los fragmentos de código.

Recomendaciones

- Expandir el preprocesamiento.
- Ampliar el análisis sintáctico.
- Emplear función de *hash* en el índice.
- Clasificación automática de los resultados.
- Algoritmos para evaluar contención de conjuntos.