Instituto Politécnico Nacional

Centro de Investigación en Computación

Detección automática de similitud entre programas del lenguaje KAREL

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

Lic. Martín Ibarra Romero

DIRECTORES DE TESIS

Dr. GRIGORI SIDOROV

Dr. RAFAEL GUZMÁN CABRERA

México, D. F., diciembre de 2014

SIP-14bis

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 16:00 horas del día 25 del mes de noviembre de 2014 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Detección automática de similitud entre programas del Lenguaje KAREL"

Presentada por el alumno:

IBARRA	**ROMERO**	**MARTÍN**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: B 1 2 1 0 5 0

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA
Directores de Tesis

Dr. Grigori Sidorov

Dr. Rafael Guzmán Cabrera

Dr. Alexander Gelbukh

Dr. Marco Antonio Moreno Ibarra

Dra. Sofia Galicia Haro

Dr. Miguel Jesús Torres Ruiz

PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACIÓN
EN COMPUTACIÓN
DIRECCIÓN

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 4 del mes Diciembre del año 2014, el (la) que suscribe Martín Ibarra Romero alumno (a) del Programa de Maestría en ciencias de la computación con número de registro B121050, adscrito a Centro de investigación en computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Grigori Sidorov y Rafel Guzmán Cabrera y cede los derechos del trabajo intitulado Detección automática de similitud entre programas del lenguaje Karel al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección maibarraromero@yahoo.com.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Martín Ibarra Romero

Resumen

Estamos interesados para calcular la similitud entre programas fuentes (misma que puede ser considerada como plagio, bajo ciertas circunstancias). Este interés nació durante mi desempeño como entrenador de Olimpiada de Informática, para encontrar programas y/o ideas de soluciones similares entre los competidores.

Para determinar la similitud entre programas utilizo un enfoque basado en técnicas de recuperación de información, utilizando la representación vectorial de un documento como una bolsa de palabras y el uso de una herramienta de aprendizaje automático (WEKA) para clasificar y evaluar los resultados a través de la validación cruzada de 10 repeticiones.

Presento cómo obtuvimos el corpus de programas de lenguaje Karel y cómo lo preparamos. Posteriormente describo los experimentos teniendo en cuenta los siguientes elementos:

- Preprocesamiento del corpus,
- La representación de los programas,
- La ponderación de términos y la creación de las dimensiones en un Modelo de Espacio Vectorial,
- La normalización y ponderación de términos (características),
- La aplicación del análisis semántico latente.

Los resultados de los experimento demuestran que el trabajo con 3-gramas y utilizando sistema de ponderación tf-idf nos da una mejor clasificación. Así, el principal resultado de este trabajo es que el uso de los 3-gramas mejora el rendimiento de nuestra tarea, debido a la forma secuencial de las instrucciones que se encuentran en la mayoría de los códigos los programas siguen el orden de ejecución de arriba a abajo y de izquierda a derecha.

Abstract

We are interested to calculate similarity between programs (it can be considered as plagiarism as well in certain circumstances). This interest was born during the Olympiad in Informatics as a coach (trainer), to determine similar programs or ideas with similar solution.

To find out the similar programs I use the approach based on the information retrieval techniques, using vector representation of a document as a bag of terms and using a machine learning tool (WEKA) to classify and evaluate the results through the 10 fold cross validation.

I present how we obtained the corpus of programs in Karel and how we prepared it. Then I describe the experiments: how I put it together and conducted the experiments, considering the following elements:

- Preprocessing of the corpus,
- Representing of the programs,
- Weight and creation of the dimensions in the Vector Space Model,
- Normalization and weighting of terms (features),
- Application of Latent semantically analysis.

The experiment results show that working with 3-grams and using tf-idf weighting scheme gives us a better classification. So, the main result of this work is that using the 3-grams improves the performance for our task. It is due to the sequential form of the instructions that is found in most of the codes, and the programs follow the execution order up to down and left to right.

DEDICATORIA

A Natalia, hija no importa la edad que tengas espero nunca te rindas y siempre logres la superación personal y profesional acompañada de la hermosa sonrisa que Dios te dio.

AGRADECIMIENTOS

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico recibido durante mis estudios.

Al Instituto Politécnico Nacional (IPN), programa PIFI y al Centro de Investigación en Computación (CIC), por darme la oportunidad de estudiar en tan honorable institución.

A mis directores de tesis por el apoyo recibido.

Contenido

Capítulo 1. Introducción

La inquietud de este trabajo surge como motor principal de apoyo a los docentes implicados en la difícil tarea de enseñar las técnicas de programación, principalmente la descomposición de un programa en partes y la idea de lo que es un algoritmo. Ya que desde mi punto de vista es importante que los alumnos creen sus propias soluciones y sobre todo que escriban programas de autoría y de esta manera obtener la habilidad de programación.

En muchas instituciones educativas, la programación de computadoras requiere, desarrollar la habilidad y destreza necesaria en los estudiantes tanto de carreras técnicas como licenciaturas e ingenierías. En donde muchas veces no es suficiente comprender "que es la programación" sino ir un paso adelante en la solución de problemas utilizando un lenguaje de programación.

Una problemática que acompaña al profesor es la verificación de las diferentes tareas y/o exámenes de programación que realizan sus alumnos. Sobre todo si nos damos cuenta que el docente tiene a su cargo en los niveles de secundaria, preparatoria y universitario poco más de 30 alumnos por clase, esto hace que la revisión de ejercicios y prácticas consuma mucho tiempo y mucho veces sea incompleta y/o no equitativa (mismo criterio de evaluación).

Durante los cursos de iniciación en la programación es común utilizar herramientas y lenguajes de programación educacionales que permitan en un entorno agradable comprender los fundamentos de programación.

Una herramienta popular en este medio es el robot Karel desarrollado por Richard E. Pattis (1). Esta herramienta es utilizada en el curso de **Metodología de programación de la universidad de Stanford CS106A** y actualmente se utilizaa en la olimpiada Mexicana de informática. (2). En sus faces estatales y nacional.

Este lenguaje Karel y su herramienta de programación por su sencillez es utilizable en alumnos de nivel secundaria, de nivel bachillerato y en cursos introductorios universitarios de programación.

Esta tesis utilizará el lenguaje de programación Karel utilizado en la olimpiada Mexicana de informática.

Definición del problema

Dado un conjunto de programas clasificados por idea de solución o problema que soluciona, poder determinar en nuevos programas a que idea de solución o problema corresponde. Esto Lo realizo basado en técnicas de recuperación de la información y en la utilización de una herramienta de clasificación de la información.

Nota importante: Un programa "P" se define por un conjuntos de elementos (rutinas, procedimiento, funciones, métodos) p_1, p_2, …p_n, decimos que un programa "P" es un conjunto { p_1, p_2, …p_n} de la misma manera existe otro programa "Q" con su conjunto {q_1, q_2, q_3…q_m}.

La Ilustración 1 ejemplifica dos programas "P" y "Q" que contiene elementos similares unidos por una línea.

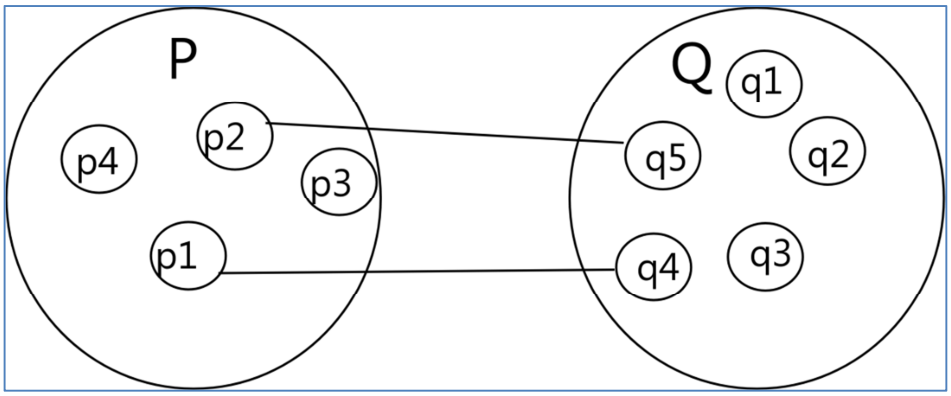


Ilustración 1. Representación gráfica de similitud de programas.

Quisiera recalcar en este punto que la similitud entre programas se puede dar total o parcial, esta parcialidad la podemos ver cuando los procedimientos y/o rutinas entre dos o más programas presentan algún grado de similitud por ejemplo cuando tenemos dos programas "A" con 9 procedimientos y "B" con 8 procedimientos pero el grado de similitud entre 3 procedimientos en ambos programas es alto.

Objetivo general

El objetivo de este trabajo es determinar el grado de similitud entre programas desarrollados en lenguaje Karel-pascal.

Para lograr este objetivo debemos tomar en cuenta los siguientes supuestos:

- Se tiene un problema dado por un docente.
- Se tiene un conjunto de programas (respuestas del alumno) que solucionan el problema y que compilan sin errores.
- Las respuestas (programas) de los alumnos que resuelven el problema se mantiene en un mismo almacén y pueden ser de diferentes generaciones (ejemplo 2015, 2014, 2013, 2012, 2011…), Esto con la finalidad de detectar similitud en código entre alumnos de diferentes generaciones.
- Los programas también pueden ser de alumnos de diferentes profesores.

Este proceso de similitud ayudará a determinar los siguientes aspectos:

- Cuando dos programas sean similares, en base a información adicional, es posible detectar indicios de:
    - Plagio de programas y/o Compartición de un código de programa de un alumno hacia otro.
    - Compartición de ideas de solución entre alumnos.
    - Si dos o más alumnos de diferentes generaciones muestran alguna similitud.
    - Si dos o más profesores desarrollan en sus alumnos técnicas de programación similares.
- Cuando los programas tiendan a ser diferentes, es posible:
    - Determinar las ideas diferentes de solución de un problema.
    - Originalidad en la creación de programas y en ideas de solución de problemas.

Nota especial.- En caso de sistemas de más de un programa el análisis de similitud serviría para revisar procedimientos que al ser similares permitan mejorar el proceso de la ingeniería de software ya que el programador los podría revisar y simplificarlas de alguna manera.

Objetivos particulares

Los objetivos particulares de trabajo son:

- Creación de un corpus de 100 problemas y por cada problema 100 programas, estos problemas deben resolver el problema satisfactoriamente y compilar libres de errores.
- Modelar los atributos que utilizaremos para determinar la similitud.
- Definir la representación de los programas.
- Desarrollar los programas y/o algoritmos para clasificar los problemas.
- Aplicar el algoritmo al corpus para recuperar la información de cada programa.
- Realizar las pruebas de validación y el análisis de resultados.

Capítulo 2. Estado del arte

La mayoría de los trabajos iniciales de detección de similitud se habla de plagio sin embargo en este trabajo hablamos de similitud ya que el plagio refiere a una situación de dolo que no siempre es posible detectar por ejemplo sabemos que los hindúes y mayas cada uno por su cuenta inventaron y/o descubrieron el cero y no había forma de que se lo plagiaran. Por tal motivo aunque hago referencias a trabajos sobre plagio, en este documento utilizaremos el término de similitud entre programas.

Vamos a determinar que dos programas son similares siempre y cuando uno pueda transformarse en el otro aplicando un número pequeño de transformaciones. Las transformaciones pueden ir desde simples cambios de comentarios hasta cambios en la lógica de control del mismo.

En 1987 J.A Faidhi y S.K. Robinson (3) propusieron 6 niveles de modificaciones a un programa el cual fue llamado "**espectro de plagio**" de software que a continuación describo:

Tabla 1. Espectro de plagio (también llamado "similitud" en este trabajo)

NIVEL	DESCRIPCIÓN
0 – Sin cambios	Sin cambios o trasformaciones
1 – Comentarios	Se modifican los comentarios
2 – Identificadores	Se modificaron los identificadores de un programa
3 – Posición de variables[1]	Se modificaron las posiciones de la variables
4 – Combinación de procedimientos	Cuando los procedimientos se combinas o se

[1] Como pascal Karel solo utiliza una variable en nuestro trabajo esta opción no tiene sentido

	dividen
5 – Modificación orden de instrucciones	Cuando se modifica el orden de instrucciones independientes
6 – Lógica de control	Cuando se modifica la lógica de control del programa

Métricas

Los primeros programas para detección de similitud o plagio aparecieron en los años 70 utilizando técnicas de conteo de atributos también llamadas métricas como las utilizadas por Halsted. (4) y McCabe (5). En la Tabla 2 mostramos un cuadro resumen de las principales métricas.

Tabla 2. Aquí se muestran las diferentes métricas.

Métricas	**Descripción**	**Referencia**
Volumen	Utilización de métricas de Halstead	(4)
Flujo de control	Calculo de la complejidad ciclo matica del número de caminos ejecutables	(5)
Dependencia de datos	De manera similar a medir el flujo de un programa, se mide la dependencia de los datos	(6)
Profundidad de anidación	Utilizando la métrica del promedio de profundidad de anidación de un programa o módulo	(7)
Estructura de control	Estas métricas asignan pesos a las estructuras de control y esta suma de pesos describe la complejidad de un programa	(8) y (9)

Mantener datos estadísticos de los caracteres del programa fuente (llamado correlación de caracteres), realizar esta comparación con todos los programas y por último realizar un resumen ordenando de mayor a menor en base a su valor estadístico.

Comparación de cadenas de caracteres

Esta técnica consiste en convertir los diferentes elementos de un programa en atributos que nos ayuden a su identificación por ejemplo un identificador de una variable puede ser <VAR> y el de un valor numérico <VALOR> y se podría escribir x=x+5; como <VAR> = <VAR> +><VALUE> para las estructuras de control podemos usar <BEGIN LOOP>…<END LOOP>

Esto suele ser el primer proceso y posteriormente se le aplican diversos algoritmos de comparación.

Analizadores de arboles

David Gitchell y Nicholas Tran (10) en los años 90 trabajaron en base al análisis del árbol que genera un programa, los reducen a su representación árbol utilizando un analizador léxico estándar, desde un enfoque de cadenas de caracteres de estos árboles los alinearon para obtener la sub secuencia mayor y utilizando un valor ente 0 y 1 para reportar el grado de similitud.

Comparación de cadenas

En los años 80 se utilizó una combinación de métricas y de algoritmos de comparación de cadena. Lo programa Plague, Sim y YAP son representativos de esta época, estos trabajos consistían en ignorar los comentarios líneas en blanco, espacios y tabulaciones. Utilizar un programa comparador de cadenas (diff, grep, etc.) que se basan en una comparación línea a línea usando la distancia de Levenshtein.

Posteriormente El algoritmo "Running –Karp-Rabin Greedy-String-Tiling" (RKSGST) es utilizado en la herramienta YAP3 de Michel Wise (11), este algoritmo básicamente busca todas las coincidencias de un tamaño "X" a un tamaño "1" utilizando el algoritmo de Rabin-Karp y compara todos los valores que coincidieron.

Comparación semántica

Quiero puntualizar que estos métodos utilizan información semántica y nos referimos al término semántico como a la información contenida en el código del programa que se liga con el problema (dominio de la solución)

Se ha utilizado la similitud del coseno, la cual consiste en ver a los programas como una entidad de caracteres y crear una representación vectorial normalizada en base a su frecuencia de n-gramas (normalmente bigramas o trigramas) y posteriormente entre cada par de programas calcular su estimación de similitud del coseno.

Trabajos más recientes (12) utilizan análisis semántico latente para determinar la dimensional dad de los vectores que representan los programas.

A fechas recientes el uso de relacionar la semántica del código fuente con la de la documentación, definición de variables nombres de procedimientos y/o comentarios del mismo ha surgido para ayuda y/o apoyo en mejorar el entendimiento de datos, en los requerimientos y diseño de los códigos fuentes. (13)

A continuación describiré las herramientas de software utilizadas para la detección de similitud y/o plagio entre programas.

JPlag

Creado por Guido Malpohl en la universidad de Karlsruhe, este software convierte el programa en una secuencia de cadenas de caracteres y utiliza un algoritmo de “Running –Karp-Rabin Greedy-String-Tiling” propuesto por Michel Wise, pero le agrega algunas optimizaciones para mejorar la eficiencia. Soporta los siguientes lenguajes Java, C#, C, C++ Scheme y proceso de texto en lenguaje natural.

Marble

Desarrollado en la universidad de Utrecht por Jurriaan Hage este sistema convierte los programas encadenas de caracteres y realiza dos normalizaciones de cada programa una simple y otra agrupando cadenas de caracteres, posteriormente utiliza la herramienta “diff” y el score es calculado tomando en

cuenta el número de líneas en que difieren, soporta los siguientes lenguajes Java, Perl PHP y XSLT

MOSS

Moss es un acrónimo de “Measure for Software Similarity” fue desarrollado en 1994 en la universidad de Stanford por Aiken este software se basa en un algoritmo llamado “winnowing” (14) que realiza lo siguiente, divide un programa en cadenas continuas llamadas k-gramas, cada k grama es convertida a una llave de “hash” y el subconjunto de todos las llaves diferentes de “hash” son lo que conforman la huella del documento. Los lenguajes que soporta esta herramienta son: C, C++, Java, C#, Python, Visual Basic, Java Script, Fortran, ML, Haskell, Lisp, Scheme, Pascal, Modul2, Ada, Perl, TCL, Mathlab, VHDL, Verilog, Spice, ensamblador MIPS , ensamblador 80x86, HCL2.

SIM

Este software fue desarrollado por Dick Grune de la Universidad UV de Amsterdam

Soporta comparaciones entre programas en C, Java, Pascal, Modula-2, Lisp Miranda y textos en lenguaje natural, reduce el programa a 80 términos aproximadamente convirtiendo los identificadores a cadenas, como <tidf> los string a <string> y los caracteres a <char> dividiendo cada palabra del lenguaje como termino y permanecen sin cambio los caracteres como punto y coma comas, etc. Para encontrar la similitud de programas compara la cantidad de términos que existen entre las representación de dos programas.

Plaggie

Programa desarrollado en el 2002 en la Universidad Tecnológica de Helsenky, esta aplicación es muy similar a la de JPlag ya que crea lcadenas de caracteres y posteriormente usa el algoritmo de “Running –Karp-Rabin Greedy-String-Tiling”, este software solo soporta comparar programas escritos en lenguaje Java.

Capítulo 3. Desarrollo

Cuando un alumno realiza una práctica de programación utiliza diversas técnicas de desarrollo. Sin embargo la creatividad siempre va asociada al modelo que representa la solución y este a su vez al programa que escribe, ya que la idea(s) que se tienen para la resolución de un problema las escribe en su programa, por tanto podemos pensar que el programa encapsula su "idea de solución". De los programas voy a extraer en base a la ocurrencia de los diferentes términos el problema y/o idea de solución utilizando LSA (Análisis Semántico Latente).

Creación del corpus

Para realizar las mediciones de similitud entre programas creamos 3 bancos de datos de prueba "corpus" todos basados en el sitio de Karelotitlan (www.cmirg.com/Karelotitlan), Este es un sitio en donde aprenden a programar jóvenes en su mayoría menores de 18 años, utilizando el lenguaje de Karel que los acerca de una manera agradable a abstraer los elementos principales de la programación como son el funcionamiento de un lenguaje de programación así como el pensamiento algorítmico en la solución de problemas. De este corpus se obtuvieron aproximadamente 100 códigos de diferentes alumnos por cada problema.

La selección fue hecha de la siguiente manera de cada problema se cuantificó la cantidad de programas que diferentes personas habían enviado, este valor se dividió entre 100 dando un factor "X" posteriormente en base a la lista de programas se tomaron 100 (siempre que fuera posible) 1 programa cada "X" programas (ósea se tomaba 1 y nos saltábamos "X", tomábamos otro y nos saltábamos "X", etc.).

De esta madera conformamos los siguientes corpus de datos:

Corpus "Karel_clasificado_solución".- Este corpus consta de 374 programas aproximadamente 34 por problema, clasificados por problema y por idea de solución. Se tomaron 11 problemas y en cada problema se agruparon por idea de

solución, veamos un ejemplo en el problema de la isla tenemos dos ideas de solución etiquetadas como idea 2 e idea 3. (Ver anexo problema de la isla)

En la idea 2 agrupamos todos los programas que realizan lo siguiente:

Para buscar la salida llegan a una esquina y de cada esquina, marcándola con zumbadores intentan salir en caso de lograrlo se ubican en la casilla 1,1. Como se puede apreciar en la Ilustración 2.

En La idea 3 agrupamos todos los programas que rodean la isla con zumbadores de manera que hace un recorrido alrededor de la isla buscando salir perpendicularmente y si se encuentran con una pared sin zumbadores significa que están fuera de la isla y basta con dirigirse a la casilla 1,1 como se puede apreciar en la Ilustración 3.

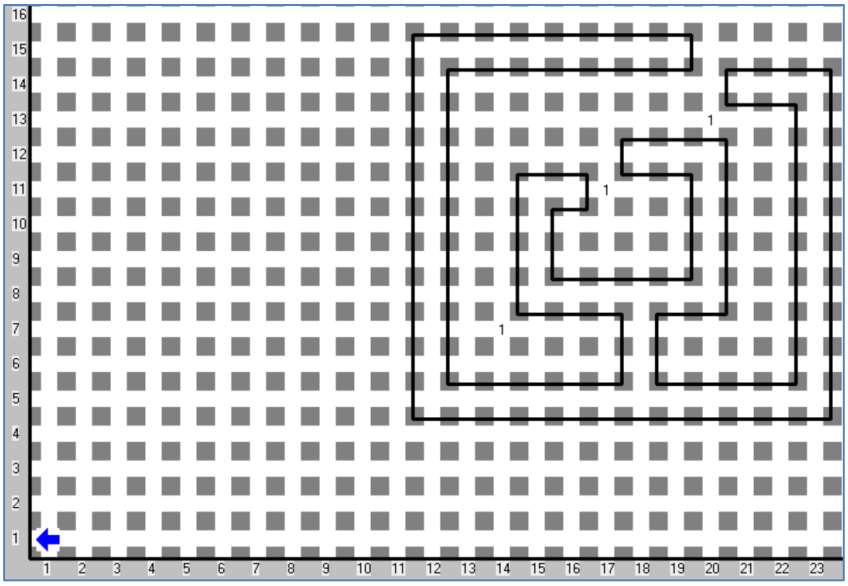


Ilustración 2. Idea de solución 2 (problema Isla).

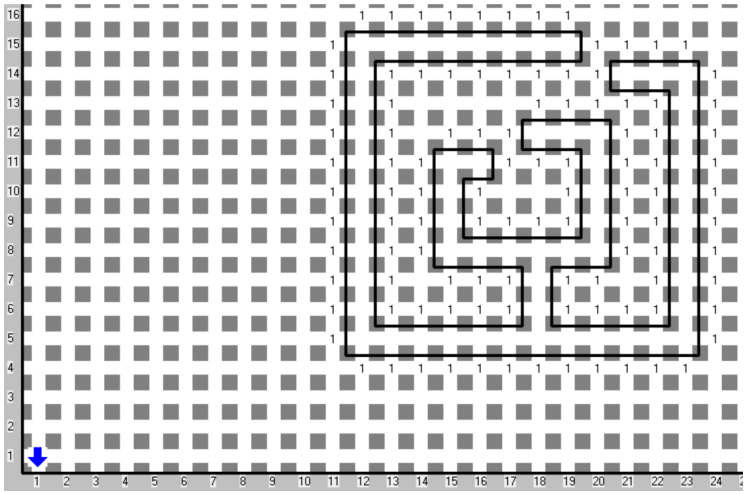
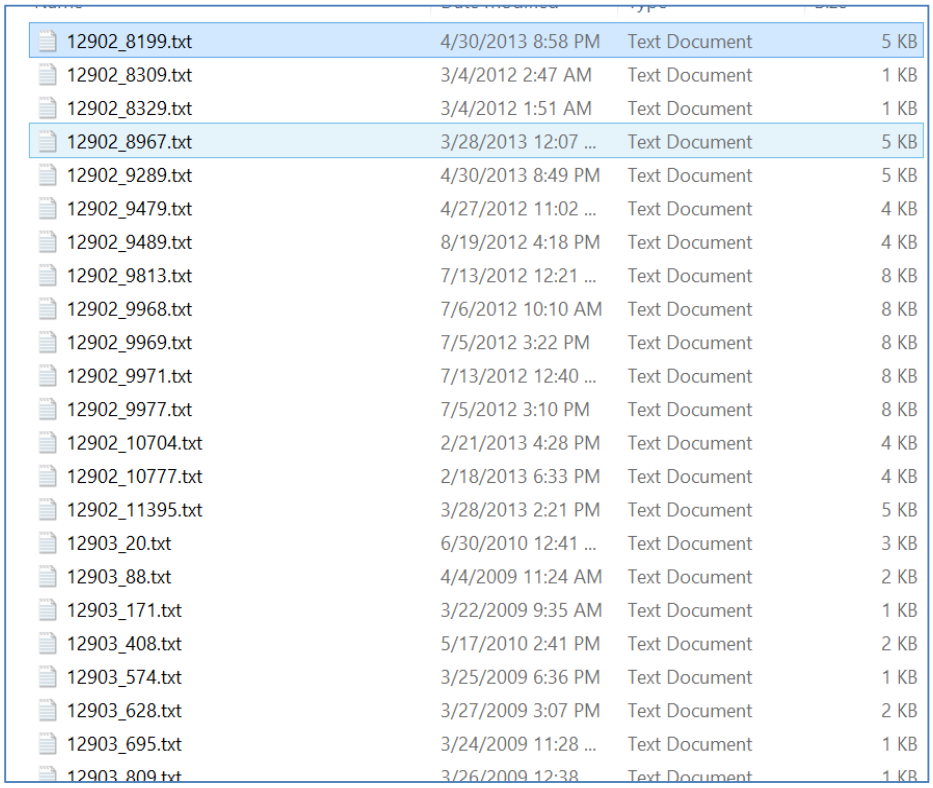


Ilustración 3. Idea de solución 3 (problema Isla).

La clasificación de este corpus se hizo manualmente.

Corpus "Karel 100".- Este consta de 100 problemas y de aproximadamente 100 programas por problema dando un total de 10,000 programas, aquí solo están clasificados por problema. Los programas dentro del sitio web de "Karelotitlan" se encuentran agrupados por persona/alumno y problema de aquí tomamos 100 problemas dando un total de 10,735 programas, todos estos programas están contenidos en una carpeta, como se pude apreciar en la Ilustración 4. En donde observamos que el nombre está separado en dos números el primero representa el número de problema y el segundo representa el número

del competidor/alumno que elaboró el programa. Los problemas que mostramos son 12902 llamado “El Malefico chuzpa” y el 12903 llamado “Un beeper en casa”. Por favor revise en anexos la lista de problemas del sitio de Karelotitlan.



12902_8199.txt	4/30/2013 8:58 PM	Text Document	5 KB
12902_8309.txt	3/4/2012 2:47 AM	Text Document	1 KB
12902_8329.txt	3/4/2012 1:51 AM	Text Document	1 KB
12902_8967.txt	3/28/2013 12:07 ...	Text Document	5 KB
12902_9289.txt	4/30/2013 8:49 PM	Text Document	5 KB
12902_9479.txt	4/27/2012 11:02 ...	Text Document	4 KB
12902_9489.txt	8/19/2012 4:18 PM	Text Document	4 KB
12902_9813.txt	7/13/2012 12:21 ...	Text Document	8 KB
12902_9968.txt	7/6/2012 10:10 AM	Text Document	8 KB
12902_9969.txt	7/5/2012 3:22 PM	Text Document	8 KB
12902_9971.txt	7/13/2012 12:40 ...	Text Document	8 KB
12902_9977.txt	7/5/2012 3:10 PM	Text Document	8 KB
12902_10704.txt	2/21/2013 4:28 PM	Text Document	4 KB
12902_10777.txt	2/18/2013 6:33 PM	Text Document	4 KB
12902_11395.txt	3/28/2013 2:21 PM	Text Document	5 KB
12903_20.txt	6/30/2010 12:41 ...	Text Document	3 KB
12903_88.txt	4/4/2009 11:24 AM	Text Document	2 KB
12903_171.txt	3/22/2009 9:35 AM	Text Document	1 KB
12903_408.txt	5/17/2010 2:41 PM	Text Document	2 KB
12903_574.txt	3/25/2009 6:36 PM	Text Document	1 KB
12903_628.txt	3/27/2009 3:07 PM	Text Document	2 KB
12903_695.txt	3/24/2009 11:28 ...	Text Document	1 KB
12903_809.txt	3/26/2009 12:38 ...	Text Document	1 KB

Ilustración 4. Directorio parcial de Corpus Karel 100, el primer número identifica el problema.

Corpus “Karel_100_50”.- Este corpus está constituido por un subconjunto de 4,598 programas de los 100 problemas seleccionados anteriormente con la única finalidad de poder realizar los cálculos con mayor velocidad.

Experimentos

Para realizar los experimentos tome en cuenta estas consideraciones:

a) **Pre procesamiento.-** Para estos experimentos consideramos varios elementos y así mostrar su efecto en la detección de similitud de código fuente.

b) **Representación**.- La representación de los programas en espacio vectorial y bolsa de palabras.

c) **Ponderación.-** La ponderación de los términos que utilizamos son el termino de frecuencia (tf) y el tf-idf (termino de frecuencia y termino de frecuencia inversa del documento)

c) **Normalización**.- Normalización de los dimensiones acorde al documento.

c) **LSA (análisis semántico latente)**.- Una vez que tenemos la representación vectorial en tf y/o tf-idf, le aplicaremos la reducción de dimensiones así como la búsqueda de "relaciones latentes" para lo cual utilizamos LSA como lo indica Scott Deerwester en (15)

d) **Evaluación.**- Para evaluar los resultados estamos utilizando WEKA con Naive Bayes y SMO (la implementación de WEKA de máquina de soporte de vectores).

Preprocesamiento del corpus

El preprocesamiento de los programas del corpus que estamos considerando son los siguientes:

- **Eliminación de comentarios**.- Consiste en eliminar los comentarios de todos los programas basados en que estoy detectando la similitud entre código fuente y los comentarios no representan información ejecutable del código.
- **Termino numéricos.-** Mis experimentos consisten en eliminar o no los términos numéricos en la Tabla 3 podemos observar el código después de eliminar el término numérico.

Tabla 3. Izquierda programa original, derecha programa eliminando el término numérico.

```
inicia-ejecucion
   mientras no-orientado-al-norte inicio
              hacer gira-izquierda;
   fin;
   mientras frente-libre hacer inicio
              cuenta-compara(0);
   fin;
   apagate;
termina-ejecucion
```

```
inicia-ejecucion
   mientras no-orientado-al-norte inicio
              hacer gira-izquierda;
   fin;
   mientras frente-libre hacer inicio
              cuenta-compara;
   fin;
  apagate;
termina-ejecucion
```

- **Términos de longitud 1.-** Experimente eliminando o no términos de longitud 1, la justificación de este preprocesamiento es debido a que los términos de longitud 1 carecen de valor semántico dentro de un programa.
- **Términos únicos**.- Consiste en eliminar o no los términos únicos.
- **Términos sintácticos.-** Consideramos los términos en incluirlos o eliminarlos estos términos son "(", ")" ";"
- **Términos sin valor.-** También consideramos los términos sin valor ("stop Words" en inglés) estos son los términos más usados y que normalmente no aportan nada de valor en el idioma español un ejemplo de estos términos son "el", "la" "los" "las".

Términos reservados sin valor.- De manera similar hicimos un tratamiento de las palabras reservados y consideramos como términos sin valor, los siguientes "entonces", "veces", "hacer", "como", "define-nueva-instrucción", "iniciar-programa", "finalizar-programa".

Representación vectorial del documento y bolsa de palabras

A continuación describo la representación vectorial de acuerdo a lo que nos dice Manning en (16). Utilizaremos la representación vectorial de un programa en donde cada término representa una dimensión y que debemos cuantificar de alguna manera, debemos tomar en cuenta que pueden existir términos no útiles a los que llamamos en inglés "stop words"; estos términos debemos suponer que no aportan valor cualitativo y/o cuantitativo en los supuestos de algún experimento. Como no existe ningún criterio para seleccionar los términos carentes de

significado, sin embargo sugiero que de cada término siempre se trate de justificar su inclusión o exclusión.

En la tabla 3 mostramos dos programas y una posible representación vectorial de los mismos considerando las siguientes características:

- Los términos “iniciar-programa” y “finalizar-programa” son considerados sin valor (“stop words”) debido a que pertenecen a la plantilla del formato de programa, todos los programas tienen estos 2 términos en las mismas posiciones en la primera línea y en la última línea significativa.
- La dimensión es binaria contiene un valor 1 si existe el término y un valor de cero si no existe el término.

Tabla 4. Tenemos 2 programas que para poder hacer operaciones con ellos debemos darles la misma dimensionalidad.

Programa 1	**Programa 1** **Dimensión** **Valor**	**Programa 2**	**Programa 2** **Dimensión** **Valor**
iniciar-programa inicia-ejecucion avanza; avanza; gira-izquierda; gira-izquierda; gira-izquierda; deja-zumbador; apagate; termina-ejecucion finalizar-programa	Avanza 1 gira-izquierda 1 deja-zumbador 1 apágate 1 inicia-ejecucion 1 termina-ejecucion 1	iniciar-programa inicia-ejecucion avanza; avanza; deja-zumbador; apagate; termina-ejecucion finalizar-programa	avanza 1 gira-izquierda 1 deja-zumbador 1 apágate 1 inicia-ejecucion 1 termina-ejecucion 1

En esta representación todos los términos (dimensiones) tienen la misma ponderación o valor con esto podríamos valorar solamente si ese término existe o no, si observamos algunos términos aparecen más veces que otros y es aquí donde les podemos dar alguna ponderación que aproveche esta mejor esta observación, para los cual también nos apoyamos del concepto de “bolsa de palabras” en cual consiste en dar como valor cuantitativo a la dimensión, el programa de nuestro ejemplo de la tabla 3 quedaría con la representación de bolsa de palabras que presentamos en la tabla 4.

Tabla 5. Bolsa de palabras, asigna o pondera algunos los términos de acuerdo a su cantidad de apariciones.

Programa 1	**Programa 1** **Dimensión** **Valor**	**Programa 2**	**Programa 2** **Dimensión** **Valor**
iniciar-programa inicia-ejecucion avanza;	Avanza 2 gira-izquierda 3	iniciar-programa inicia-ejecucion avanza;	avanza 2 gira-izquierda 1

avanza; gira-izquierda; gira-izquierda; gira-izquierda; deja-zumbador; apagate; termina-ejecucion finalizar-programa	deja-zumbador 1 apágate 1 inicia-ejecucion 1 termina-ejecucion 1	avanza; deja-zumbador; apagate; termina-ejecucion finalizar-programa	deja-zumbador 0 apágate 1 inicia-ejecucion 1 termina-ejecucion 1

La representación de bolsa de palabras ignora el orden exacto de los términos dentro del documento.

Ponderación de los términos y/o dimensiones

A continuación describo el proceso de pesado de dimensiones acuerde a lo que nos dice Manning en (16). Para este modelo de bolsa, las siguientes dos frases tendrían el mismo valor "Natalia es más rápida que Martín" y "Martín es más rápido que Natalia", intuitivamente podemos ver que si dos documentos o programas con bolsa de palabras similares representan contenidos similares. Debe notar que no todos los términos tiene la misma importancia en el Documento/programa. Por tal motivo debemos usar una manera de darles una ponderación (valor) esta ponderación estará en función de que tanto se utiliza el término dentro del documento para nuestros experimentos utilizamos frecuencia de termino (tf por sus siglas en inglés) que escribiremos de la forma $tf_{t,d}$ que significa frecuencia del término "t" en el documento "d" ósea el número de veces que aparece un término dentro de un documento. Por ejemplo si un documento "A" contiene 12 veces el término "calcula" y el documento "B" aparece 18 veces, tendremos que; $tf_{calcula,A} = 12$ y $tf_{calcula,b} = 18$.

Por otro lado usamos "frecuencia de término y frecuencia del documento inversa" (tf-idf por sus siglas en ingles). Con este valor pretendemos que un término sea ponderado acorde con la cantidad de documentos del corpus en donde aparece. Para esto nos apoyamos en el valor de la frecuencia del documento inversa (idf por sus siglas en ingles) el cual nos indica que tan común es un término en la colección de documentos, que denotaremos como idf_t y la "t" representa el término. La fórmula es:

$$\mathbf{idf_t = \log \frac{N}{df_t}}$$

Donde “N” representa la cantidad de documentos del corpus y “df” el número de documentos en donde aparece el término “t” el logaritmo puede ser base 10 o 2 como lo explica Manning en (16) y la frecuencia de documentos (df por sus siglas en inglés) representa el número de documentos en donde aparece el término “t”.

El valor de ponderación tf-idf está dado por:

$$\text{tf\_idf}_t = \text{tf}_{t,d} \text{ x } \text{idf}_t$$

Por tanto podemos ver que $\text{tf\_idf}_{t,d} = \text{tf}_{t,d}$ x idf_t. Asigna un valor muy alto cuando el término aparece muchas veces en un número pequeño de documentos, un valor bajo cuando el término aparece algunas veces en un documento o aparece en muchos documentos y un valor muy bajo cuando aparece en pocos documentos.

Normalización de términos del documento

A continuación describo el proceso de la normalización de acuerdo a lo que nos dice Manning en (16). Podrían existir dos documentos que sean similares en contenido y su conteo de términos podría ser diferente debido a que un documento puede ser muy grande y el otro pequeño, uno podría describir varias veces el mismo proceso y el otro solo una vez, por tal motivo será de gran utilidad normalizar la ponderación de los términos del documento (tf) en relación al tamaño del documento, que permitirá una mejor distribución, en este caso utilizaremos la normalización euclidiana, que se encarga de dar un valor a cada término entre 0 y 1 acorde al número de apariciones dentro del documento.

Tabla 6. Izquierda fórmula de Longitud Euclidiana, derecha un vector normalizado.

$\sqrt{\sum_{i=1}^{n} \overline{\overline{V}}_i^2(d)}$	**Ejemplo:** longitud Euclidiana =6.56

Termino	**Valor normalizado**	**Vector**
Gira-izquierda	3	.46
Avanza	5	.76
Mientras	3	.46

El procedimiento es el siguiente:

- Se calcula la longitud euclidiana a partir de la fórmula de la tabla
- Se calcula para cada valor del vector su porcentaje con respecto a la longitud euclidiana.

Análisis semántico latente

Dentro de nuestros experimentos incluimos la técnica de “análisis semántico latente” para encontrar relaciones latentes entre los términos, como lo menciona Deerwester, Dumais, Furnas y Landauer en (17).

La idea básica de esta técnica es encontrar relaciones “ocultas” (latentes) lo cual mostraré con un ejemplo descrito por Fatema Al-YaZeedi, Annette Payne and Timoothy Cribbon en (18), vamos a realizar una búsqueda de los términos “dies” y “dagger” en los siguientes documentos:

D1 : Romeo and Juliet.

D2 : Juliet: O happy dagger!

D3 : Romeo died by dagger.

D4 : “Live free or die”, that’s the New-Hampshire’s motto.

D5 : Did you know, New-Hampshire is in New-England.

No hay duda que D3, D2 y D4 se deben considerar ya que representan sin duda similitud, ahora veamos a D1 y D5; Como humanos sabemos que D1 está relacionado con la búsqueda y que D5 está menos relacionado con la búsqueda, por tanto sería bueno que pudiéramos encontrar esta relación. LSA debe considerar que el término “dagger” está más relacionado a D1, porque se encuentra junto al término “Romeo” y “Julieta” en D2 y D3 respectivamente. El término “Dies” está relacionado con D1 , porque se encuentra relacionado con Romeo(D3) y con D5 porque se encuentra. A continuación mostramos en conteo de términos.

	D1	**D2**	**D3**	**D4**	**D5**
romero	1	0	1	0	0
juliet	1	1	0	0	0
happy	0	1	0	0	0
dagger	0	1	1	0	0
live	0	0	0	1	0
die	0	0	1	1	0

free 0 0 0 1 0

new-hamspire 0 0 0 1 1

Aplicando SVD tenemos las siguientes 3 matrices:

MATRIZ T

-0.396 0.280 -0.571 0.450 -0.102

-0.314 0.450 0.411 0.513 0.204

-0.178 0.269 0.497 -0.257 0.043

-0.438 0.369 0.013 -0.577 -0.220

-0.264 -0.346 0.146 0.047 0.417

-0.524 -0.246 -0.339 -0.273 0.155

-0.264 -0.346 0.146 0.047 0.417

-0.326 -0.460 0.317 0.237 -0.725

MATRIZ S

2.285 0.000 0.000 0.000 0.000

0.000 2.010 0.000 0.000 0.000

0.000 0.000 1.361 0.000 0.000

0.000 0.000 0.000 1.118 0.000

0.000 0.000 0.000 0.000 0.797

MATRIZ D

-0.311 -0.407 -0.594 -0.603 -0.143

0.363 0.541 0.200 -0.695 -0.229

-0.118 0.677 -0.659 0.198 0.233

0.861 -0.287 -0.358 0.053 0.212

0.128 0.034 -0.209 0.333 -0.910

Elegimos K=2 para eliminar dimensiones, debemos considerar las siguientes matrices.

T 2

Romero	−0.396	0.280

Juliet	−0.314	0.450
happy	−0.178	0.269
Dagger	−0.438	0.369
Live	−0.264	−0.346
Die	−0.524	−0.246
free	−0.264	−0.346
New-hampire	−0.326	−0.460

S 2

2.285	0
0	2.01

D 2

-0.311 -0.407 -0.594 -0.603 -0.143

0.363 0.541 0.200 -0.695 -0.229

A continuación calculamos la matriz para realizar operaciones de comparación:

Matriz de los términos T2*S2

Romero	-0.905	0.563
Juliet	-0.717	0.905
happy	-0.407	0.541
Dagger	-1.001	0.742
Live	-0.603	-0.695
Die	-1.197	-0.494
free	-0.603	-0.695
New-hampire	-0.745	-0.925

Matriz de documentos S2*T2'

D1	D2	D3	D4	D5

-0.711	-0.930	-1.357	-1.378	-0.327
0.730	1.087	0.402	-1.397	-0.460

Para comparar utilizaremos la similitud de coseno. Ejemplo: de la búsqueda de “dies” y “dagger” debemos hacer su representación vectorial

Q=(dies + dagger)/2 = ([-1.197 -0.494]+[-1.001 0.742])/2=**[-1.099 0.124]**

Calcularemos la similitud del coseno y tendremos los siguientes datos:

D1	D2	D3	D4	D5
0.77364153	0.73119433	0.98461306	0.61800043	0.48435793

Como se puede observar el orden de similitud está dado por D3, D2, D4, D1 y D5, a continuación muestro el cálculo de similitud del coseno sin haber utilizado LSA.

d1	d2	d3	d4	d5
-	0.4082	0.8165	0.3536	-

En donde podemos observar que le documento d1 y d5 no presentan ningún tipo de similitud.

Modelo estadístico de clasificación

En este apartado daré una descripción breve de la herramienta WEKA, misma que he utilizado para realizar el trabajo de clasificación de programas. La información que describo está basada en el libro llamado “Data Mining” escrito por Ian H. Witten, Eibe Frank y Mark A. Hall (19).

```
%  Atributos WEKA
@attribute ( numeric
@attribute ) numeric
@attribute - numeric
@attribute A numeric
@attribute B; numeric
  …
@attribute y numeric
@attribute                                clase                                {
12921,12920,12923,12922,12925,12924,12927,12926,12929,12928,12932,12933,12930,12931,12936,12937,12934,12935,
12938,12939,12909,12908,12907,12906,12905,12904,12903,12902,12901,12900,12859,12858,12851,12850,12853,12852,
12855,12854,12857,12856,12952,12953,12918,12919,12910,12911,12912,12913,12914,12915,12916,12917,12849,12846,
12847,12877,12876,12875,12874,12873,…… ,12881 }


%  Vectores TFIDF para WEKA
@data
{0 8, 1 8, 2 19, 3 1, 4 18, 5 44, 6 4, 7 32, 8 11, 9 49, 10 12, 11 5, 12 7, 13 64, 14 3, 15 8, 16 5, 17 54, 18 34, 19 12, 20 1, 21
26, 22 24, 23 19, 24 14, 25 5, 26 1, 27 6, 66 12846}
{0 8, 1 8, 2 51, 3 1, 4 28, 5 88, 6 27, 7 55, 8 45, 9 130, 10 24, 11 8, 12 10, 13 121, 14 6, 15 26, 16 16, 17 98, 18 82, 19 7, 20
14, 21 76, 22 46, 23 45, 24 35, 25 8, 26 9, 27 26, 28 1, 29 1, 30 1, 66 12846}
```

Ilustración 5. Representación de datos para Weka.

WEKA es un sistema desarrollado en la universidad de Waikato en Nueva Zelandia, en un software de aprendizaje de máquina, el cual utilizamos para poder evaluar la representación vectorial de nuestro corpus por medio de dos algoritmos el de “Bayes” y el de “SMO” la utilización de WEKA consiste de lo siguiente, en base al preprocesamiento descrito anteriormente creo la representación vectorial del programa y a este le asigno una categoría basada en el nombre del problema que resuelve, en la sección de anexos busque en “Relación de problemas de Karelotilan” para conocer las diversos problemas. En la Ilustración 5 muestro un extracto de un archivo, donde podemos observar que se tomaron los caracteres como atributos, y en la clase podemos observar el nombre del problema a resolver, y también podemos ver los primeros 3 vectores en representación “tf” en donde el primer valor representa el atributo y/o dimensión y el segundo representa su “tf”, el último valor del vector representa la clase.

Para realizar la clasificación de los resultados utilizamos el método llamado “Cross-Fold-Validation” (validación cruzada) con un parámetro d 10, la cual

consiste en que el corpus se divide en 10 partes de las cuales se utilizan 9 para entrenar el modelo y una para probarlo, estos lo hace 10 veces por el valor del parámetro que le di y el promedio de los 10 experimentos es el resultado que arroja. A continuación detallo este proceso.

Del corpus total de programas se divide en 10 partes de la 1 a la 10 aleatoriamente, después repite este proceso 10 veces:

- Toma la parte 1 para aplicar la clasificación y con las otras 9 partes realiza el entrenamiento.
- Toma la parte 2 para aplicar la clasificación y con las otras 9 partes realiza el entrenamiento.
- ……
- Toma la parte 10 para aplicar la clasificación y con las otras 9 partes realiza el entrenamiento.
- Al final entrega como resultado el promedio de los 10 experimentos realizados.

Métodos de clasificación

Para propósitos de mi tesis utilice los métodos de “Bayes” y “SMO” que describo a continuación:

Naive Bayes.- Lo considere porque es un método estadístico que ha mostrado a lo largo del tiempo fortaleza.

SMO.- Es la implementación de WEKA al método conocido como máquina de soporte de vectores (SVM) fue inventado por Vladimir N. Vapnik, básicamente este método representa los programas como puntos en el espacio y trata de separar las clases y/o categorías lo más posible. SVM busca un hiperplano que permite realizar la separación óptima de las diferentes clases.

Capítulo 4. Resultados de los experimentos

En esta sección mostraré los resultados arrojados por mis experimentos así como la descripción de los mismos. Cabe hacer mención acerca de que todos los preprocesamiento incluían eliminar los comentarios de los programas. En la Tabla 7 describo las características de preprocesamiento de cada experimento, en donde la columna llamada nombre se refiere al nombre dado al experimento, si significa que se incluyó y no que no se incluyó.

Tabla 7. Características de preprocesamiento.

Nombre	n-Grama	Pesado	Termino	Términos Sintácticos	Normalizado	Incluye Números	Términos Longitud 1	Términos Únicos	Stop word Programa	Stops Word
Karel 00	1	TF	caracter	Si	No	Si	Si	Si	Si	Si
Karel 01	3	TF	caracter	Si	No	Si	Si	Si	Si	Si
Karel 03	1	TF	token	Si	No	Si	Si	Si	Si	Si
Karel 04	3	TF	token	Si	No	Si	Si	Si	Si	Si
Karel 05	2	TF	token	Si	No	Si	Si	Si	Si	Si
Karel 06	3	TF	token	No	No	No	No	No	No	No
Karel 07	4	TF	token	Si	No	Si	Si	Si	Si	Si
Karel 08	1	TF-IDF	caracter	Si	No	Si	Si	Si	Si	Si
Karel 09	3	TF-IDF	caracter	Si	No	Si	Si	Si	Si	Si
Karel 10	1	TF-IDF	Token	Si	No	Si	Si	Si	Si	Si
Karel 11	3	TF-IDF	Token	Si	No	Si	Si	Si	Si	Si
Karel 12	2	TF-IDF	Token	Si	No	Si	Si	Si	Si	Si
Karel 13	3	TF-IDF	Token	No	No	No	No	No	No	No
Karel 14	4	TF-IDF	Token	Si	No	Si	Si	Si	Si	Si
Karel 15	3	TF-IDF	Token	No	Si	No	Si	Si	No	No
Karel 16	2	TF	Token	No	Si	Si	Si	Si	No	No
Karel 17	3	TF-IDF	Token	Si	Si	Si	Si	Si	No	No

Mis experimentos iniciaron con el corpus más pequeño llamado "Karel_clasificado_solución" recuerde que este corpus consta de 374 programas, los experimentos fueron realizados en WEKA utilizando un pesado de términos "tf" y "tf_idf", la Tabla 8 muestra los resultados en

donde la columna llamada "descripción" define un valor "KarelXX" indica el preproceso realizado al corpus, la columna "algoritmo" indica el nombre del algoritmo utilizado para su clasificación, la siguiente columna indica el método de ponderación que se le dio a los términos, La columna "Todas las dimensiones" indica que algoritmo se utilizó únicamente con la ponderación de términos, la columna llamada "todas las dimensiones" significa que se utilizaran todas los términos, las columnas LSA 50, LSA 100, …. LSA 600 indica el resultado obtenido aplicando LSA y el número indica el número de dimensiones utilizada. la Tabla 8, la

Tabla 9 y en la

Tabla 10 mostramos los resultados respetivamente de los corpus "Karel_clasificado_solución", "Karel100-50" y del corpus llamado "Karel100", En cada renglón el valor en cursiva y negrilla representa el mayor porcentaje obtenido por ese experimento.

Tabla 8. Corpus Karel_clasificado_solución.

Descripción	Algoritmo	Ponderación	Todas las dimensiones	LSA 50	LSA 100	LSA 150	LSA 200	LSA 300	LSA 400	LSA 500	LSA 600
Karel00	Bayes	tf	***47.23***	44.72	44.61	44.50	44.61	44.50	44.61	44.72	44.45
Karel00	SMO (SVM)	tf	37.06	39.20	39.25	38.98	39.15	39.30	***39.36***	39.30	39.15
Karel01	Bayes	tf	***50.40***	49.63	44.14	41.59	38.26	38.26	37.86	49.63	37.99
Karel01	SMO (SVM)	tf	***55.62***	45.44	48.00	45.46	30.89	30.89	31.28	45.44	31.02
Karel03	Bayes	tf	***50.54***	47.07	46.53	44.28	42.13	36.53	36.93	36.53	36.53
Karel03	SMO (SVM)	tf	40.37	***39.43***	46.13	44.65	41.71	32.76	33.28	32.89	32.49
Karel04	Bayes	tf	***51.75***	44.90	46.13	46.66	43.05	39.03	39.30	39.43	39.43
Karel04	SMO (SVM)	tf	***51.72***	41.30	***50.41***	48.81	45.20	36.24	36.10	36.23	35.97
Karel05	Bayes	tf	***49.88***	46.67	45.60	46.26	42.27	37.17	36.90	36.90	38.10
Karel05	SMO (SVM)	tf	51.21	44.25	***53.70***	50.53	45.06	34.63	34.37	34.90	34.63
Karel06	Bayes	tf	***47.20***	42.12	42.40	42.64	40.27	34.78	34.24	34.24	34.78
Karel06	SMO (SVM)	tf	***45.70***	35.81	44.50	44.25	42.90	37.56	37.70	37.03	37.57
Karel07	Bayes	tf	***50.12***	45.32	47.86	47.46	43.32	42.38	41.72	42.25	41.72
Karel07	SMO (SVM)	tf	***51.73***	40.36	50.53	47.20	43.59	35.59	35.32	35.58	35.45
Karel08	Bayes	tf-idf	25.52	26.20	25.39	26.20	26.20	25.53	25.39	26.34	***26.34***
Karel08	SMO (SVM)	tf-idf	29.41	29.54	29.68	29.81	29.81	***29.94***	29.54	29.54	29.81
Karel09	Bayes	tf-idf	***50.92***	45.60	40.66	38.11	35.16	33.16	33.43	33.43	32.89
Karel09	SMO (SVM)	tf-idf	***55.91***	50.28	50.53	49.46	48.13	40.53	40.80	40.80	40.53

Descripción	Algoritmo	Ponderación	Todas las dimensiones	LSA 50	LSA 100	LSA 150	LSA 200	LSA 300	LSA 400	LSA 500	LSA 600
Karel00	Bayes	tf	***47.23***	44.72	44.61	44.50	44.61	44.50	44.61	44.72	44.45
Karel10	Bayes	tf-idf	***49.05***	39.05	38.00	35.31	34.64	33.96	33.56	33.96	34.10
Karel10	SMO (SVM)	tf-idf	***45.33***	42.50	46.40	43.18	41.31	39.85	39.71	39.58	39.58
Karel11	Bayes	tf-idf	***52.81***	47.20	45.34	42.54	38.90	38.37	37.97	37.97	37.83
Karel11	SMO (SVM)	tf-idf	53.62	51.20	***53.08***	50.27	45.59	38.39	38.25	38.39	38.12
Karel12	Bayes	tf-idf	***52.14***	45.47	43.88	40.54	37.60	37.04	37.18	37.44	37.18
Karel12	SMO (SVM)	tf-idf	***54.55***	52.01	51.74	49.86	48.12	42.53	42.40	42.53	42.27
Karel13	Bayes	tf-idf	***45.85***	39.86	39.04	35.96	35.54	33.71	33.57	33.45	33.84
Karel13	SMO (SVM)	tf-idf	***47.98***	43.17	45.83	44.90	44.24	39.72	40.25	40.12	40.12
Karel14	Bayes	tf-idf	***49.47***	43.59	45.99	41.59	39.32	38.36	38.36	38.63	38.63
Karel14	SMO (SVM)	tf-idf	51.60	51.07	***53.63***	50.15	46.53	41.45	41.72	41.71	41.45
Karel15	Bayes	tf-idf	***44.65***	40.67	39.59	37.33	35.31	35.71	35.18	35.85	35.98
Karel15	SMO (SVM)	tf-idf	***47.72***	44.24	46.67	43.98	41.18	37.83	37.83	37.43	37.70
Karel16	Bayes	tf-idf	***47.86***	43.61	44.54	43.20	41.07	37.17	37.17	37.03	37.17
Karel16	SMO (SVM)	tf-idf	***47.99***	34.22	45.31	45.18	42.51	37.97	37.97	37.97	37.84
Karel17	Bayes	tf-idf	***49.99***	45.73	45.21	41.74	39.60	38.79	39.05	38.51	38.65
Karel17	SMO (SVM)	tf-idf	***50.95***	49.47	51.35	47.46	44.79	39.73	39.73	39.73	39.86

Tabla 9. Resultados Corpus Karel100_50 (5,000 programas).

Descripción	Algoritmo	Ponderación	Todas las dimensiones	LSA 50	LSA 100	LSA 150	LSA 200	LSA 300	LSA 400	LSA 500	LSA 600
Karel01	Bayes	tf	***50.91***	34.17	38.93	40.43	39.89	38.23	35.84	33.86	32.51
Karel01	SMO (SVM)	tf	65.42	35.38	56.39	61.59	64.53	65.88	***65.96***	64.64	63.25
Karel04	Bayes	tf	33.93	37.34	38.17	38.12	38.10	29.43	37.10	36.58	***62.27***
Karel04	SMO (SVM)	tf	39.08	50.76	60.20	67.36	69.42	21.27	***69.77***	68.99	64.92
Karel07	Bayes	tf	***62.27***	29.43	33.93	37.34	38.17	38.12	38.10	37.10	36.58
Karel07	SMO (SVM)	tf	64.92	21.27	39.08	50.76	60.20	67.36	69.42	69.77	***68.99***
Karel09	Bayes	tf-idf	***52.50***	33.65	42.41	44.74	46.43	46.61	45.30	43.39	42.34
Karel09	SMO (SVM)	tf-idf	***71.77***	46.19	58.40	63.18	64.20	65.42	65.55	65.46	65.68
Karel11	Bayes	tf-idf	***65.66***	27.16	41.43	45.15	47.80	49.35	48.17	47.87	46.65
Karel11	SMO (SVM)	tf-idf	***74.14***	44.80	63.11	69.81	72.12	***73.60***	73.34	73.31	72.40
Karel12	Bayes	tf-idf	***57.33***	33.32	42.56	45.48	46.48	45.35	44.19	41.74	38.87
Karel12	SMO (SVM)	tf-idf	68.96	45.87	61.57	66.81	68.40	***69.62***	69.55	68.03	67.68
Karel14	Bayes	tf-idf	***63.72***	24.45	35.52	40.93	43.69	45.24	45.54	44.59	43.80
Karel14	SMO (SVM)	tf-idf	72.55	35.17	58.79	66.64	70.75	73.38	***73.88***	72.99	72.16

Tabla 10. Corpus Karel100 (10,000 programas).

Descripción	Algoritmo		Pesado	LSA 50	LSA 100	LSA 150	LSA 200	LSA 300	LSA 400	LSA 500	LSA 600
Karel_00	Bayes	tf	13.07	***21.86***	19.20	19.20	19.19	19.19	19.19	19.19	19.19
Karel_00	SMO (SVM)	tf	4.58	8.25	9.27	9.35	9.24	9.35	9.28	9.32	***9.39***
Karel_01	Bayes	tf	***49.05***	29.93	32.40	33.91	34.55	34.34	34.22	33.51	32.48
Karel_01	SMO (SVM)	tf	58.59	11.18	21.34	32.70	40.32	52.07	56.21	58.27	***59.42***
Karel_03	Bayes	tf	***39.27***	24.24	23.71	21.51	20.99	21.99	22.50	22.93	22.06
Karel_03	SMO (SVM)	tf	***43.25***	6.97	12.44	15.02	17.93	23.33	27.31	31.66	34.35
Karel_04	Bayes	tf	***55.36***	24.76	30.34	32.77	33.81	35.00	35.93	36.32	36.29
Karel_04	SMO (SVM)	tf	55.69	5.99	16.71	29.07	38.08	51.00	56.65	61.25	***63.89***
Karel_11	Bayes	tf-idf	***58.81***	21.96	30.84	36.53	39.54	43.04	44.74	45.29	45.17
Karel_11	SMO (SVM)	tf-idf	68.56	34.50	52.23	60.35	65.03	68.42	69.85	***69.94***	69.88

A continuación describo el análisis que realizo de mis resultados:

El corpus llamado “Karel_clasificado_solución” hago las siguientes observaciones:

- El algoritmo SVM muestra mejores resultados de clasificación los 3 más altos valores están dados por los experimentos Karel05 (53.70%), Karel09 (55.91%) y Karel14 (53.63).
- En el experimento Karel09 podemos observar el mejor resultado sin hacer reducción de dimensiones y realizando el pesado de términos de manera “tf_idf” sobre caracteres en 3-gramas, Sin embargo los resultados de los experimentos de Karel 05 y Karel14 son muy cercanos a menos de 2.3 % utilizando token y 3-grama y 4-grama respectivamente.

Del corpus llamado Karel100_50 podemos observar lo siguiente:

- En los experimentos Karel11 (74.14% y 73.60%) y Karel14 (73.80%) se observan en Karel11 algoritmo SVM utilizando 3-gramas obtuvo los mejores resultado utilizando un pesado de caracteres “tf_idf”. Los

resultados más cercanos dados por LSA en Karel11(73.60) con dimensiones= 300 y Karel14 con 400 dimensiones.

Del corpus llamado Karel100 podemos observar lo siguiente:

- En el experimento Karel11 (69.94%) tenemos nuestro mejor resultado con el algoritmo de SVM, usando 3-gramas en pesado “tf_idf”, utilizando 500 dimensiones.

Capítulo 5. Conclusiones

De las conclusiones de los resultados obtenidos, puedo dar la siguiente aportación:

El manejo de trigramas, pesado de términos “tf_idf”, junto con la utilización de LSA con un parámetro de operación de 500 dimensiones nos da mejor porcentaje de clasificación y por tanto de similitud entre programas Karel.

Observemos que este resultado no dice es el lenguaje natural quien domina el proceso de similitud entre programas ya que el lenguaje Karel posee un reducido número de palabra reservadas, sin embargo los identificadores (nombre de rutinas y procedimientos) suelen ser ilimitados y únicos. Algunos identificadores nos ocasionaron problemas por ejemplo si un procedimiento se llamaba “rodear” y su función principal significaba “rodear un contorno” el programador siempre tiene la opción de nombrarlo de cualquier forma por ejemplo “xxx” o “gatito”, Esta situación la podemos observar en el ámbito escolar ya que en el ambiente laboral tiende a disminuir.

También quiero recalcar que los elementos de un lenguaje formal siempre tiene un significado semántico como son los paréntesis, el punto y coma. El LSA fue de gran utilidad al reducir las dimensiones a 500 (curiosamente el número “mágico” que la mayoría de los autores reconoce como los apropiados del LSA considerando definiciones que alumnos universitarios pueden proporcionar).

A continuación menciono otras conclusiones que obtuve:

- La representación de programas en bolsa de palabras nos permitió el manejo práctico del espacio vectorial de los documentos y la utilización de n-gramas, permitieron dar un aspecto de seriación que en diversas rutinas de programación suelen ser útiles.
- Se desarrollaron herramientas para análisis de sintaxis de programas Karel en lenguaje Python.
- Es importante determinar las características a modelar de los vectores.
- Se tiene la creación de 3 corpus los cuales pueden servir para realizar investigaciones futuras.

- Se desarrollaron herramientas para análisis de sintaxis de programas Karel.
- Se hicieron las pruebas de validación apoyadas en Cross-Fold-Validation y se verificaron sus resultados; siendo WEKA de gran ayuda.
- En base a este trabajo se realizó una estancia de investigación en la Universidad Politécnica de Valencia la cual obtuvo como fruto un artículo titulado “Modelos de recuperación de información basados en n-gramas aplicados a la reutilización de código fuente” elaborado conjuntamente por Enrique Flores, Lidia Moreno, Grigori Sidorov, and Paolo Rosso y un servidor. Que se publicó en la tercera conferencia Española de recuperación de la información, el 19 y 20 de junio de 2014. En la faculta de informática de la Universidad de Coruña, España. Este trabajo consistió en evaluar un corpus de programas en lenguaje “C” aplicando preprocesamiento similar al de mi trabajo de tesis, así como creación de modelo vectorial basado aplicando LSA, utilizando la similitud de coseno para dar medida de igualdad entre programas.

Capítulo 6. Trabajos futuros

A quien lea esta tesis le dejo estos elementos que pudieran dar dirección a la continuidad de este trabajo.

- Trabajar con la similitud suave de Coseno. Como la similitud de coseno resulta que ha sido ampliamente utilizado, sin embargo mi propuesta es que incorporemos la similitud suave de coseno implementada por el Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno y David Pinto Avendaño la cual calcula en el peor caso, el mismo valor que el de similitud de coseno para más detalles lea (20).
- Si bien el lenguaje formal que se utiliza en la elaboración de programas cada elemento es indispensable para determinar la dimensionalidad correcta, es importante hacer notar que se requieren de heurísticas para mejorar el proceso de similitud enfocándonos al trabajo de interpretación de los identificadores tanto de procedimientos métodos o rutinas , como, de los identificadores de las diversas variables, por ejemplo; si alguien escribiera el nombre de un procedimiento como “Leer_Datos” tendríamos al menos dos posibilidades , dejar un término al que llamaríamos “LeerDatos” bajo la premisa que “LeerDatos” en sí mismo denota un concepto o quizá dividirlo en dos términos uno sería “Leer” y el otro “Datos”. Una vía que sugiero es la adición de métricas similares a las de los años setenta y noventas a la representación vectorial de un programa.
- Implementar proceso de lenguaje natural en el tratamiento de identificadores, raíces, lemas, etc. esto permitiría unificar flexiones de términos por ejemplo; si tenemos varias rutinas llamadas “calcula”, “calc” “calculando” debemos tener presente que engloban el mismo concepto y una situación parecida podríamos suceder con los sinónimos como podría ser la palabra “computa” y “calcula”.
- Sugiero seguir la línea de disminuir la dimensionalidad de la representación de los programas y para eso el LSA es útil ya que para el corpus de 10,000 programas con un total de 600 dimensiones De la clasificación hecha por WEKA se computa en un 2% del obteniendo resultados similares y en varias ocasiones mejores a los utilizados sin la reducción de dimensiones.

Anexos

Relación de problemas de Karelotilan

Aquí se muestran las listas de problemas utilizados en nuestros Corpus

Clave	Nombre	Origen	Categoria
12846	El apagón	Concentración 2007, 1a OCyMI	Recursión con
12847	Midiendo la distancia	8.5a OMI, Piedras Negras 2004	Recursión
12849	Cactus	11a OMI, examen estatal del DF y Estado de México	Básico
12850	Búnkeres	Concentración 2007, 1a OCyMI	Búsquedas
12851	Camino a Torreón	Concentración 2007, 1a OCyMI	Recursión con
12852	Camino a Torreón Reloaded	Concentración 2007, 1a OCyMI	Problemario
12853	Chuzpa regresa	1a OCyMI, Distrito Federal 2007	Búsquedas
12854	El sitio	Concentración 2008, 2a OCyMI (DF, EDOMEX y OAXACA)	Recursión con
12855	La benévola Chizpa	Concentración 2008, 2a OCyMI	Problemario
12856	La ciudad de Karelotitlán	1a OCyMI, Distrito Federal 2007	Recursión
12857	Los bloques perdidos	1a OCyMI, Distrito Federal 2007	Problemario
12858	Los ratones Karel	Propuesta Competencia PUMINET 2008	Recursión con
12859	Reconstruyendo la pirámide	1a OCyMI, Distrito Federal 2007	Recursión con
12860	Sumas Karelianas	Concentración 2007, 1a OCyMI	Recursión con
12861	Travesía escolar	Competencia PUMINET 2007	Recursión
12862	Torneos	Competencia PUMINET 2008	Recursión con
12863	Arquitectos Karelianos	2a OCyMI, Distrito Federal 2008	Problemario
12864	Escalando la montaña	2a OCyMI, Distrito Federal 2008	Recursión con
12865	El hijo del cartero	2a OCyMI, Distrito Federal 2008	Problemario
12866	Matriuskas	2a OCyMI, Distrito Federal 2008	Recursión
12867	Bacterias infecciosas	Concentración 2008, 2a OCyMI	Búsquedas
12868	Los cubos de Karel	Concentración 2008, 2a OCyMI	Recursión con
12869	Matriuskas Reloaded	Concentración 2008, 2a OCyMI	Problemario
12870	Delegaciones	Concentración 2008, 2a OCyMI (DF, EDOMEX y OAXACA)	Problemario
12871	Monte Albán	Concentración 2008, 2a OCyMI (DF, EDOMEX y OAXACA)	Problemario
12872	Carrera	11a OMI, examen estatal del DF y Estado de México	Recursión
12873	KarelGol	11a OMI, examen estatal del DF y Estado de México	Recursión
12874	Tendedero	11a OMI, examen estatal del DF y Estado de México	Recursión con
12875	La balanza	10a OMI, examen estatal del DF y Estado de México	Recursión con
12876	Bardando el terreno	10a OMI, examen estatal del DF y Estado de México	Recursión con
12877	Comprando	10a OMI, repechaje del DF y Estado de México	Recursión con
12878	Cuenta pasos	10a OMI, examen estatal del DF y Estado de México	Problemario
12879	Karel-perímetro	10a OMI, repechaje del DF y Estado de México	Recursión

12880	Sembrando	10a OMI, repechaje del DF y Estado de México	Introducción
12881	Los canales del lago	Concentración 2008, 2a OCyMI	Introducción
12882	Detrás del arcoíris	9a OMI, examen estatal oficial	Básico
12883	El mapa del tesoro	10a OMI, Durango 2005	Básico
12884	Después del recreo	Karelotitlán 2009	Recursión
12885	Alacranes zumbadores	10a OMI, Durango 2005	Básico
12886	Paredes	8.5a OMI, Piedras Negras 2004	Introducción
12887	Leyendo el periódico	Curso básico de recursividad, Karel	Recursión
12888	Amontonar zumbadores	9a OMI, Morelia 2004	Introducción
12889	La marcha	9a OMI, Morelia 2004	Recursión
12890	El estadio de fútbol	9a OMI, Morelia 2004	Recursión con
12891	Sumar	9a OMI, examen estatal oficial	Básico
12892	Recoge zumbadores	8.5a OMI, Piedras Negras 2004	Introducción
12893	Pasa los números al otro lado	8.5a OMI, Piedras Negras 2004	Introducción
12894	Pasando los números Reloaded	Karelotitlán 2009	Recursión
12895	Multiplicación y división	8.5a OMI, Piedras Negras 2004	Recursión con
12896	Autopista	10a OMI, Durango 2005	Recursión
12897	Medias pirámides	Karelotitlán 2009	Introducción
12898	Sedimárip saidem	9002 Náltitolerak	Introducción
12899	Los ratones Karel, Episodio I	Karelotitlán 2009	Básico
12900	Buscaminas	11a OMI, San Luis Potosí 2006	Recursión
12901	Áreas	11a OMI, San Luis Potosí 2006	Búsquedas
12902	El maléfico Chuzpa	11a OMI, San Luis Potosí 2006	Problemario
12903	Un beeper en la casa	12a OMI, Torreón 2007	Básico
12904	Vigas	12a OMI, Torreón 2007	Básico
12905	Costo mínimo	12a OMI, Torreón 2007	Problemario
12906	Erosión	13a OMI, Puebla 2008	Básico
12907	Súper-Karelman	13a OMI, Puebla 2008	Problemario
12908	Camino de primos	13a OMI, Puebla 2008	Búsquedas
12909	La expedición de Karelotl	Karelotitlán 2009	Problemario
12910	Calcetines desemparejados	3a OCyMI, Distrito Federal 2009	Básico
12911	La fila de las tortillas	3a OCyMI, Distrito Federal 2009	Básico
12912	Vigías	3a OCyMI, Distrito Federal 2009	Recursión con
12913	Pasteles	3a OCyMI, Distrito Federal 2009	Problemario
12914	La fila de las tortillas Reloaded	Concentración 2009, 3a OCyMI	Recursión con
12915	TlaKarelel, y la piedra de la maldición	Concentración 2009, 3a OCyMI	Búsquedas
12916	Burocracia interminable	Problema perdido en el 2007	Recursión
12917	La fila	14a OMI, examen estatal de Morelos	Recursión
12918	Campamento	14 OMI, examen estatal de Morelos	Problemario
12919	Piedras	14a OMI, examen estatal de Morelos	Recursión con
12920	Rutas	Examen por Internet, Septiembre 2009	Búsquedas
12921	Ríos	Examen por Internet, Septiembre 2009	Problemario
12922	Isla	14a OMI, Colima 2009	Problemario
12923	Influenza	14a OMI, Colima 2009	Recursión

12924	Divisores	14a OMI, Colima 2009	Búsquedas
12925	Clases de manejo	4a Olimpiada de Informática del DF y Edo Méx 2010	Problemario
12926	Comida buena y mala	4a Olimpiada de Informática del DF y Edo Méx 2010	Recursión con
12927	El cochecito	4a Olimpiada de Informática del DF y Edo Méx 2010	Problemario
12928	Calorías	Olimpiada de Informática de Morelos 2010	Búsquedas
12929	El cine	Olimpiada de Informática de Morelos 2010	Problemario
12930	Indentaciones	Olimpiada de Informática de Morelos 2010	Problemario
12931	¡Tache!	14° OMI en Aguascalientes, 2009	Básico
12932	Kábola	14° OMI en Aguascalientes, 2009	Básico
12933	Kontando en el Kallejón	14° OMI en Aguascalientes, 2009	Recursión
12934	Kamión	14° OMI en Aguascalientes, 2009	Recursión con
12935	Klaveles Kompany	14° OMI en Aguascalientes, 2009	Problemario
12936	Hombre-Puma	Concentración 2010, 4a OMI DF-EDOMEX	Búsquedas
12937	San Bombazo	Concentración 2010, 4a OMI DF-EDOMEX	Problemario
12938	Karel, el aprendiz de pintor	Concentración 2010, 4a OMI DF-EDOMEX	Problemario
12939	Almohada	15a OMI, Mérida 2010	Recursión con
12940	Rombo	15a OMI, Mérida 2010	Problemario
12941	Recuerda de dónde saliste	15a OMI, Mérida 2010	Problemario
12942	Rebelde	15a OMI, Mérida 2010	Búsquedas
12943	Comedor	5a Olimpiada de Informática del DF y Edo Méx 2011	Problemario
12944	Galletas	5a Olimpiada de Informática del DF y Edo Méx 2011	Problemario
12945	Zapatos	5a Olimpiada de Informática del DF y Edo Méx 2011	Problemario
12946	Comprando Reloaded	Concentración 2011, 5a OMI DF-EDOMEX	Problemario
12947	Dulces	Concentración 2011, 5a OMI DF-EDOMEX	Problemario
12948	Durmiendo	Concentración 2011, 5a OMI DF-EDOMEX	Problemario
12949	Edificios	Concentración 2011, 5a OMI DF-EDOMEX	Recursión con
12950	Riesgo	5a Olimpiada de Informática del DF y Edo Méx 2011	Búsquedas
12951	La casa	16a OMI, Delegación Yucatán	Recursión con
12952	Esquinas	16a OMI, Delegación Yucatán	Básico
12953	El encuentro	16a OMI, Delegación Yucatán	Recursión
12954	El mural	16a OMI, Delegación Yucatán	Recursión con
12955	La pared	16a OMI, Delegación Yucatán	Recursión
12956	Los Aluxes	16a OMI, Delegación Yucatán	Problemario
12957	Karel Mosby, arquitecto	16a OMI, Cuernavaca 2011	Básico
12958	Kareleseo	16a OMI, Cuernavaca 2011	Recursión
12959	Kareloscopio	16a OMI, Cuernavaca 2011	Recursión con
12960	Kill Karel	16a OMI, Cuernavaca 2011	Problemario
12961	Hansel & Gretel	16a OMI, Primera Fase Estatal San Luis Potosí	Problemario
12962	Karelevy	16a OMI, Primera Fase Estatal San Luis Potosí	Básico
12963	Cuadro mágico	15a OMI, Primera Fase Estatal San Luis Potosí	Problemario
12964	Asterisco	Olimpiada Potosina de Informática 2012	Básico
12965	La novia de Karel	Olimpiada Potosina de Informática 2012	Básico
12966	Saludos	Olimpiada Potosina de Informática 2012	Problemario
12967	Karel Digital	Olimpiada Potosina de Informática 2012, fase 2	Problemario

12968	Rehilete	Propuesto por Elias Augusto Tuyu Alamilla	Problemario
12969	Tiro con arco	17a OMI, Hermosillo 2012	Básico
12970	Sombras del desierto	17a OMI, Hermosillo 2012	Problemario
12971	Simetría	17a OMI, Hermosillo 2012	Problemario
12972	Patrones	17a OMI, Hermosillo 2012	Problemario
12973	Fix it Karel	18a OMI, Toluca 2013	Problemario
12974	Karel Attractio	18a OMI, Toluca 2013	Problemario
12975	Capitán Karel	18a OMI, Toluca 2013	Problemario
12976	Mr. Karel-tastic	18a OMI, Toluca 2013	Problemario
12977	Ofrendas	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12978	Burbujas	6a Olimpiada de Informática del DF y Edo Méx 2012	Recursión con
12979	Copas	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12980	Plagas	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12981	Estacionamiento de Alichis	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12982	Bicicleta	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12983	Cueva del tesoro	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12984	Sindicatos	6a Olimpiada de Informática del DF y Edo Méx 2012	Problemario
12985	Fuga	7a Olimpiada de Informática del DF y Edo Méx 2013	Problemario
12986	Trackmania OMI	7a Olimpiada de Informática del DF y Edo Méx 2013	Problemario
12987	Buscando un OXXO	7a Olimpiada de Informática del DF y Edo Méx 2013	Problemario
12988	Simón dice	7a Olimpiada de Informática del DF y Edo Méx 2013	Introducción

Ejemplo de un problema de programación

Problema burbujas

DESCRIPCIÓN

En el laboratorio de química, se tiene dentro de un tubo de ensayo una sustancia viscosa que después de ser agitada vigorosamente forma una hilera de grumos y una burbuja, cada grumo o burbuja está representado por un montón de zumbadores que a su vez indican la densidad del mismo. La burbuja a diferencia de los grumos, siempre tiende a subir de tal manera que se va intercambiando con el grumo inmediato superior siempre y cuando la densidad del grumo sea mayor a la densidad de la burbuja.

PROBLEMA
Escribe un programa, que ayude a simular el ascenso de la burbuja y la deje en la altura que le corresponde.

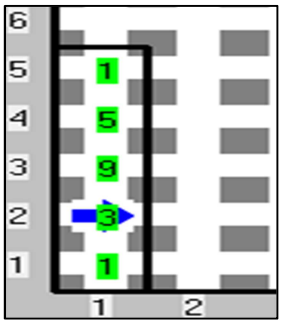
ENTRADA
El tubo de ensayo está representado por un rectángulo de ancho 1 y altura 'x' en donde 'x' varía desde el valor 1 hasta el valor 20, el tubo de ensayo siempre se encuentra en posición vertical, cada celda dentro del tubo representa un grumo o a la burbuja con un montón de zumbadores que van desde 0 hasta 100 mismo número que indica la densidad del grumo o de la burbuja, solo existe una burbuja dentro del tubo y la puedes distinguir de los grumos porque Karel siempre inicia en la posición de la burbuja.

SALIDA
Tu programa deberá dejar la burbuja en la posición que le corresponde dentro del tubo de ensayo y solo se te calificará que tanto Karel como la burbuja terminen en la posición correcta, tanto Karel como la burbuja deberán terminar en la misma posición

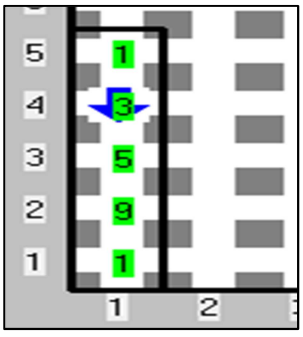
CONSIDERACIONES

Tu programa deberá funcionar para varios mundos de prueba.

MUNDO DE EJEMPLO



SOLUCIÓN



Ejemplo del Problema de isla

DESCRIPCIÓN

Karel está vacacionando en la playa de una isla que se encuentra en algún lugar de su mundo. Sin embargo sus vacaciones terminaron y debe regresar a su casa la cual se encuentra en la esquina inferior izquierda del mundo.

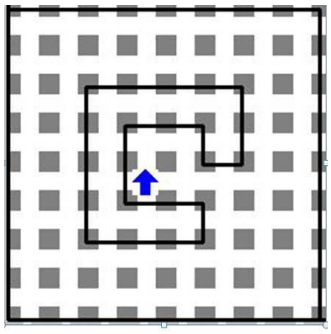
PROBLEMA

Ayuda a Karel a encontrar el camino desde el lugar donde se encuentra hasta su casa que se encuentra en la esquina inferior izquierda del mundo.

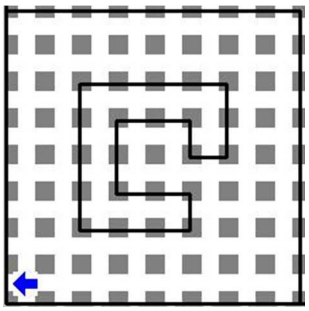
CONSIDERACIONES

La orilla de la isla está representada por paredes que forman un polígono de forma arbitraria.
La orilla siempre está separada de las paredes del mundo por una distancia de al menos 2 casillas.
Cualesquiera 2 segmentos de la orilla se encuentran separados por una distancia de al menos dos casillas, ya sea en dirección vertical u horizontal.
El mundo contiene únicamente una isla, además de las paredes que limitan la isla, no hay ninguna otra pared dentro del mundo.
No hay zumbadores en ninguna parte del mundo.
Karel siempre iniciará en una casilla adyacente a una de las orillas de la isla.
Karel lleva un número infinito de zumbadores en la mochila.
Karel siempre puede llegar desde su posición inicial hasta la esquina inferior izquierda del mundo.
Para obtener los puntos de este problema basta con que Karel se apague estando en la esquina inferior izquierda del mundo, no importa su orientación.

EJEMPLO



Mundo de ejemplo



Solución de ejemplo

BNF del lenguaje Karel-pascal

El compilador no distingue de minúsculas y mayúsculas

```
DeclaracionDePrograma  ::=   "iniciar-programa"
                                    [ DeclaracionDeProcedimiento ";" ]...
                                    "inicia-ejecucion"
                                          ExpresionGeneral [ ";"
                                          ExpresionGeneral ]...
                                    "termina-ejecucion"
                             "finalizar-programa"
                              EOF
 DeclaraProcedimiento  ::= "define-nueva-instruccion" Identificador [ "(" Identificador ")" ] "como"
     Expresion
ExpresionGeneral  ::=        { Expresion | ExpresionVacia }
Expresion  ::=               {"apagate"| "gira-izquierda"| "avanza"| "coge-zumbador"
                             | "deja-zumbador"| "sal-de-instruccion"| ExpresionLlamada
                             | ExpresionSi| ExpresionMientras| ExpresionPara
                             | "inicio"
                                    ExpresionGeneral [ ";"
                                    ExpresionGeneral ]...
                              "fin" }
ExpresionLlamada  ::=  Identificador [ "(" ExpresionEntera ")" ]
ExpresionSi  ::=       "si" Termino "entonces"
                              Expresion
                       ["sino"
                              Expresion ]
ExpresionMientras  ::= "mientras" Termino "hacer"
                              Expresion
ExpresionPara  ::=     "repetir" ExpresionEntera "veces"
                              Expresion
ExpresionVacia  ::=
Termino  ::=           ClausulaY [ "o" ClausulaY ]...
ClausulaY  ::=         ClausulaNo [ "y" ClausulaNo ]...
ClausulaNo  ::=        [ "no" ] ClausulaAtomica
ClausulaAtomica  ::=   { "si-es-cero" "(" ExpresionEntera ")" | FuncionBooleana | "(" Termino ")" }
ExpresionEntera  ::=   { Decimal | Identificador | "precede" "(" ExpresionEntera ")"
                        | "sucede" "(" ExpresionEntera ")" }
Identificador  ::=     Letra [ Letra | Digito | "-" ] ...
Decimal  ::=           { {"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"} [ Digito ] ...| "0" }
Digito  ::=            {"0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"}
Cadena  ::=            Cualquier cosa entre comillas en una única línea.
FuncionBooleana  ::=   {"frente-libre"| "frente-bloqueado"| "izquierda-libre"|
                       "izquierda-bloqueada"| "derecha-libre"| "derecha-bloqueada"
                       | "junto-a-zumbador"| "no-junto-a-zumbador"
                       | "algun-zumbador-en-la mochila"| "ningun-zumbador-en-la mochila"
```

| "orientado-al-norte"| "orientado-al-sur"| "orientado-al-este"
| "orientado-al-oeste"| "no-orientado-al-norte"| "no-orientado-al-sur"
| "no-orientado-al-este"| "no-orientado-al-oeste"}

Letra ::= Una letra del alfabeto

EOF ::= Marca de final de fichero.

Comentarios

Están soportados los dos estilos de comentario en Pascal:

{ Comentario en la misma línea }
(* Comentario en la misma línea *)

Están soportados los dos estilos de comentario en Java/C++:

/* Comentario en la misma línea */
// Comentario para el resto de línea

Relación de ilustraciones

Relación de tablas

Referencias

1. **Richard E. Pattis, Jim Roberts, Mark Stehlik.** *Karel the Robot: Gentle Introduction to the Art of Programming.* s.l. : John Wiley & Sons; Edición: 2nd Revised edition (11 de octubre de 1994, 1994.

2. http://www.olimpiadadeinformatica.org.mx/. [En línea]

3. **Robinson, J.A Faidhi y S.K.** *An Empirical aproach for detecting program similarity and plagiarism within a university programming environment.* s.l. : Cumputing in Education Vol 11, PP (11-19), 1987.

4. **Halstead, M.H.** *Elements of software science.* North Holland, New York : s.n., 1977.

5. **McCabe, T.J.** *A complexity measure.* s.l. : IEEE transaction on software Engineering SE-"(4), pp(308-320), 1976.

6. **Debnath, J.M. Bieman and N.C.** *an analisis of software structure using a generalized programa graph.* s.l. : COMPSAC 85 pp(254-259), 1985.

7. **Dunsmore.** *Software metrics: an overview of an envolving methodology.* s.l. : Informatión processing and Managament 20 pp (183-192), 1984.

8. **Jeyaprakash.** *MEBOW: A comprehensive measure of control flow complexity, pp(283-244).* s.l. : COMPSAC, 1987.

9. **Nejmeh.** *NPATH: A mesuare of execution path complexity and its applications.* s.l. : Communications ACM 31 pp(188-200), 1988.

10. **Tran, D. Gitchell and N.** *Sim: a utility for detecting similarity in computer programs.* s.l. : SIGCSE Bulletin, 31(1) pp(266–270), 1999.

11. **Wise, M.J.** *YAP3: improved detection of similarities in computer program and other texts. In Proc. of.* s.l. : SIGCSE '96 Technical Symposium, Philadelphia, USA, pp. 130–134., 1996.

12. **Cosma, Georgina.** *An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis.* s.l. : A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science, University of Warwick, Department of Computer Science, Julio 2008.

13. **Lin, Sheng- Kuei Hsu y Shi-Jen.** *A Block-Structures Model for Source code Retrieval.* Taiwan, ROC : s.n.

14. **Saul Schleimer, Daniel S. Wilkerson y alex Aiken.** *Winnowing: LOcal Algortihms for document fingerprinting.* s.l. : In Preceeding of the ACM SIGMOD, 2003.

15. *Indexing by Latent Semantic Analysis.* **Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer.** 1990, Journal of the American society for information science.

16. **Christopher D. Manning, Prabhakar Raghavan,Hinrich Schütze.** *An Introduction to Information Retrieval.* s.l. : Online edition (c) 2009 Cambridge Up, 2009.

17. *Indexing by Latent Semantic Analysis.* **Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman.** 6, 1990, JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, Vol. 41.

18. *Auto Grading And Providing Formative Feedback to Studenst on Document submitted Electronically.* **Fatema Ai-Yazeedi, Annette Payne and Timothy Cribbon.** s.l. : Universidad de Groningen The Netherlands, 2012. the Proceedings of The 11th European Conference on e-Learning.

19. **Ian H. Witten, Eibe Frank y Mark A. Hall.** *Data Mining Practical Machine Learning Tools and Techniques.* s.l. : Morgan Kauffmann Publishers, 2011.

20. *Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model.* **Grigori Sidorov, Alexander Gelbuckh, Helena Gómez-Adorno, David Pinto Avendaño.** 3, s.l. : Computación y Sistemas, 2014, Vol. 18.

21. **Clough, Paul.** *Plagiarism in natural and programming lenguages: an overviw of current tools and technologies.* Universidad of Sheffield : s.n., 2000.

22. **Mozgovoy, M.** *Desktop tools for offline plagiarism detection in computer programs.* s.l. : Informatics in Education, 5(1) pp(897–112), 2006.

23. **L. Prechelt, G. Malpohl, and M. Philippsen.** *Finding plagiarisms among a set of.* s.l. : Journal of Universal Computer Science, 8(11) pp(1016–1038), 2002.

24. **Whale, G.** *Identification of program similarity in large populations.* s.l. : The Computer Journal, 33(2) pp(140–146), 1990.

25. **Wise, M.** *Yap3: improved detection of similarities in computer program and other texts.* s.l. : SIGCSE Bulletin, 28(1) pp(130–134), 1996.

26. *A comparison of plagiarism detection tools.* Utrecht, The Netherlands : s.n., JUNIO 2012.

27. **Christopher D. Manning, Prabhakar Raghavan y Hinrich Schütze.** *An Introduction to Information Retrival.* s.l. : Online edition (c) 2009 Cambridge UP., 2008.