Articulo **Foro de Estudios sobre Guerrero**
CIENCIAS EXACTAS E INGENIERÍA Mayo 2013 Abril 2014 Vol.1 No.1 160-163

Análisis del algoritmo de levenshtein una aplicación sencilla sobre la web

SALGADO-LINARES, Irving Jaime*†, PEÑA-GALEANA, Ricardo y ROSARIO-CRUZ, Roberto

Unidad Académica De Ciencias Y Tecnologías De La Información - UAGro. Av. De las Colinas No. 37-A, Frac. Las Playas, Acapulco, Gro. México

Recibido Junio 4, 2014; Aceptado Octubre 13, 2014

Resumen

La coincidencia de patrones o Pattern Matching es un problema básico dentro de la ciencia de la computación, el rendimiento de muchos programas es determinado por el trabajo que se requiere para que coincida con los patrones, sobre todo en las áreas de procesamiento de texto, reconocimiento de voz, la recuperación de información y la biología computacional. La coincidencia de cadenas es un caso especial de coincidencia de patrones, donde el patrón se describe por una secuencia finita de símbolos. Esto consiste en encontrar uno o más generalmente todas las ocurrencias de un patrón x = x0x1… xm-1 de longitud m dentro de un texto y = y0 y1 … yn-1 de longitud n. Tanto x como y se construyen sobre el mismo alfabeto ∑.

Análisis, algoritmo, levenshtein.

Abstract

Pattern Matching and Pattern Matching is a basic problem within computer science, the performance of many programs is determined by the labor required to match patterns, especially in the areas of word processing, recognition speech, information retrieval and computational biology. The string matching is a special case of pattern matching, where the pattern is described by a finite sequence of symbols. This is to identify one or more generally all occurrences of a pattern x = x0x1 ... xm-1 of length m in a text y = y0 y1 ... yn-1 of length n. Both x and y are built on the same alphabet Σ.

Analysis, algorithm, levenshtein.

Citación: SALGADO-LINARES, Irving Jaime, PEÑA-GALEANA, Ricardo y ROSARIO-CRUZ, Roberto. Análisis del algoritmo de levenshtein una aplicación sencilla sobre la web. Foro de Estudios sobre Guerrero. Mayo 2013 Abril 2014, 1-1: 160-163

* Correspondencia al Autor (Correo Electrónico: irjasali@hotmail.com)
† Investigador contribuyendo como primer autor.

Introducción

Las aplicaciones sobre coincidencia de cadenas van desde la simple tarea de buscar un texto único para una cadena de caracteres, a la búsqueda de una base de datos de ocurrencias aproximadas de un patrón complejo. El problema de la coincidencia de cadenas puede ser entendido como el problema de encontrar un determinado patrón con alguna propiedad dentro de una secuencia determinada de símbolos, el caso más simple es el de encontrar una determinada cadena dentro de la secuencia dada.

En el presente trabajo se describe el análisis, desarrollo y la implementación de una aplicación basada en la Web del algoritmo de Levenshtein. Algoritmo por excelencia utilizado para la búsqueda de coincidencia de cadenas a través del concepto de distancia y que ha sido utilizado como punta de lanza para el desarrollo de la gran mayoría de los algoritmos sobre coincidencia de patrones de la actualidad.

Objetivos

El objetivo de este trabajo es presentar el análisis, desarrollo e integración de una aplicación web que muestra la aplicación del algoritmo de Levenshtein para la búsqueda de coincidencia de cadenas, utilizando las nuevas tecnologías de desarrollo, frameworks de trabajo y sobre todo los nuevos lenguajes de programación para la Web como es el estándar HTML5, CSS y Jquery

Metodología

Para el desarrollo del presente trabajo se realizaron cuatro fases o etapas: Se analizaron las dos técnicas principales de correspondencia de cadenas: a) la correspondencia de cadenas exacta y b) la correspondencia aproximada de cadenas haciendo uso del concepto matemático de distancia entre dos secuencias de caracteres de un mismo alfabeto.

Se adoptó la definición de distancia de Levenshtein como la medida de la similitud entre dos cadenas (s) y (t), la cual se calcula como el número de eliminaciones, inserciones, o sustituciones necesarias para transformar s en t. Cuanto mayor sea la distancia Levenshtein, más diferentes son las cadenas. Esta métrica también se le llama veces distancia de edición.

Se analizaron y seleccionaron las herramientas, tecnologías y lenguajes de programación basados en la Web a utilizar para desarrollar una demostración "en vivo" de una búsqueda de una cadena dentro de un texto dado, determinándose en utilizar lenguajes de programación: Jquery, JavaScript, CSS y HTML5.

Se desarrolló y probó la aplicación web misma que incluyó la codificación del algoritmo de Levenshtein

Resultados

La aplicación sencilla basada en la Web está constituida por 3 archivos de manera independiente, pero que todos en conjunto le dan la fuerza y la fortaleza a la aplicación:

Nombre del Archivo	Descripción.
Archivo buscapalabra.html	Archivo desarrollado en el estándar HTML5
Archivo funciones.js	Archivo desarrollado en el lenguaje JavaScript y Jquery
Archivo style.css	Archivo de Hoja de cascada de estilos.

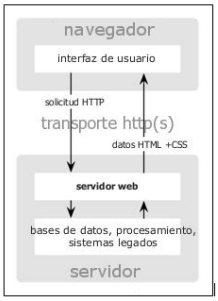

Figura 1 Modelo Clásico de una Aplicación Web.

El archivo llamado funciones.js representa el núcleo de la aplicación, por lo que debido a su importancia se muestra su código en JQUERY a continuación.

ISSN 2007-882X

```
// Modificacion del prototipo prototipo String
String.prototype.endsWith = function(suffix)  {
 return this.indexOf(suffix, this.length - suffix.length) !== -1 };
function Levenshtein(a, b) {
var n = a.length;
var m = b.length;
// Creación de matriz de cambios mínimos
var d = [];
// si una de las dos está vacía,
// la distancia es insertar todas las otras
if(n == 0)  return m;
if(m == 0)  return n;
for(var i = 0; i <= n; i++) {d[i] = []}[0] = i;
for(var j = 0; j <= m; j++)  d[0][j] = j;
for(var i = 1, l = 0; i <= n; i++, l++)
  for(var j = 1, J = 0; j <= m; j++, J++)
    if(b[J] == a[l])  d[i][j] = d[l][J];
      else
        d[i][j] = Math.min(d[l][j], d[i][J], d[l][J]) + 1;
// el menor número de operaciones
                return d[n][m];
}

function cambiarProcesado() {
    $("#cajaTexto textarea").hide();   /*desaparezco el textarea*/
    $("#cajaTexto #procesado").css("display","inline-block");/* display ya no sera none, sera inline-block*/
    $("#cajaTexto #procesado").hide().fadeIn();   /*aparezco el contenedor de texto procesado, el article*/
}
function cambiarNuevo() {
    $("#cajaTexto #procesado").hide();/*desaparezco el article con id procesado*/
    $("#cajaTexto textarea").hide().fadeIn();/*aparezco el text area*/
}
function EncontrarConKeyUp(evt){//cargar al dar enter en el campo de texto de la pista
var e = window.event || evt; // for trans-browser compatibility
var tecla = e.which || e.keyCode;
  if(tecla == 13){Encontrar();}
}
function Encontrar() {
  if($("#findWord").val().length > 0) {  //si el campo de pista no está vacío
      var shortest = -1;
      var matchlev = null;
      var sm = " ";
      var i = 0;
      var count = 0;
      var pista = $("#findWord").val(); //Obtengo la pista
      var texto = $("#nw_textarea").val(); //Obtengo el texto en el textarea
      var salida = ""; //Aquí voy a poner el texto que luego colocare en el article
      var coin = 0; //coincidencias
       var aprox = 0; //aproximaciones
```

```
       var sabrox = "";
       var comit = ""; //caracter omitido
       var palabra = ""; //palabra actual
       var palabras = texto.split(" "); //convertir texto del textarea en arreglo de palabras String, y por eso todos
tienen ya la funcion endsWith
       for (var i=0; i <= palabras.length-1; i++) {//hago un recorrido del arreglo de palabras
          comit = ""; //caracter omitido
          palabra = ""; //palabra actual
          var temp = palabras[i];  //necesito quitarle antes puntos y comas
          if(temp.endsWith(".") || temp.endsWith(",")) { //si .|coma acaba en punto o coma, hay que quitarcelo
              palabra = temp.substring(0,temp.length-1); //obtengo la palabra
              comit = temp.substring(temp.length-1); //extraigo y deposito aquí el carácter sobrante
           }
          else
              palabra = temp; //si no hay nada que le sobre a la palabra la deposito en la variable de palabra actual
          var lev = Levenshtein(palabra, pista);//utilizo el algoritmo
          if(lev ==0 ) {//si la distancia es 0
              matchlev = pista; //matchlev es igual a pista
              shortest = 0; //pongo esta bandera en 0
              coin++; //le digo que hay una coincidencia mas
              salida+="<div class='coincidencia'>" + palabra + "</div>" + comit + " ";
// en rojo  palabra antes de agregar a la cadena de salida, si existe algún carácter omitido extraído se concatena
           }
       else
          salida +=palabra + comit + " " ;//si la distancia es diferente de 0, concatenar palabra y carácter omitido
            if (lev <= shortest || shortest < 0) {//bueno esto lo deje igual
                matchlev = pista;
                shortest = lev;
                sm = palabra;
            }
       if(shortest == 0)   //si encontré la palabra exacta le digo que encontré el patrón exacto
            $("#aproximacion").text("patron exacto encontrado");
         else  //sino le digo cual fue la que casi se parece
            $("#aproximacion").text("Quizo decir "" + sm + ""?");
          }

        $("#cajaTexto #procesado").html(salida); // todo el texto de salida con todo estilos dentro de article procesado
        $("#patron").text("" + pista + ""); //pongo en el área de información que el patrón es la pista
        $("#cantidad").text(coin); // veces que coincidio la palabra,  en la primera hace uso html, en las otras dos text
// html respeta el significado de las tags de html y el text lo agarra como si fuera texto simplemente
//en la primera ocupo las tags, así que conviene usar html
      cambiarProcesado(); //desaparezco el text area y aparezco el article con todo y el contenido que acabo de hacer
      }
        else { // Si no había nada en el cambio de pista
          $("#findWord").focus(); // Mando el foco a el campo de pista
        }
}
function nuevo() {//este método borra el contenido del campo findword (en el que se pone la pista)
        $("#findWord").val("");
```

La parte principal del código mostrado radica en las primeras líneas de código del 1 al 3,ya que como sabemos JavaScript no contiene funciones para trabajar con cadenas que permitan eliminar ciertos caracteres.

Por ello aprovechando que jQuery es un lenguaje orientado a prototipos se modificó el prototipo String, y se le agregó a todo objeto String la función endsWith.

La especificación HTML5 trae muchos nuevos elementos a los desarrolladores web, permitiéndoles describir la estructura de un documento web con semántica estandarizada, es así como se codifico el archivo antes mencionado.

Figura 2 Ejecución del algoritmo de Levenshtein. La aplicación no encuentra el patrón buscado (Nivada), pero indica que el usuario quiso decir "Nevada".

Figura 3 Ejecución del algoritmo de Levenshtein. La aplicación encuentra el patrón buscado "Nevada" y 10 coincidencias más.

Discusión

Si pensamos por un minuto la infinidad de formas en que las personas comenten errores tipográficos, podemos llegar a una enorme lista, en ciertas ocasiones omitimos letras (como por ejemplo crecimiento por crecmiento), a veces agregamos demasiadas letras (Viirreinato por Virreinato) y algunas veces sustituimos una letra por otra.

ISSN 2007-882X

Por todo esto tiene sentido tomar la distancia entre dos palabras para determinar el número más pequeño de tales transformaciones necesarias para convertir una palabra en otra y así conocer la métrica.

La métrica de Levenshtein es una herramienta que permite conocer cuáles fueron los cambios que tuvo que sufrir una cadena para convertirse en otra. Comparar secuencias extremadamente grandes como las secuencias del genoma humano requiere tratamientos especiales a los algoritmos utilizados. Esto ocurre por el crecimiento cuadrático del tiempo de ejecución asintótico del algoritmo para obtener la distancia de Levenshtein.

El problema que representan las secuencias sumamente grandes, implica un problema de espacio (al guardarlas en la memoria de una computadora, o al escribirlas en un papel), de modo que se requiere diseñar modificaciones a esta métrica, con el fin de determinar la distancias parciales de una subsecuencia con otra, ambas de longitud manejable.

Conclusión

La distancia de Levenshtein es de fundamental importancia en diversos campos tales como la biología computacional y la búsqueda o procesamiento de texto, y en consecuencia, todos los problemas que de alguna manera involucran la distancia de edición están siendo ampliamente estudiados hoy en día.

En el presente trabajo, se mostró a través de una aplicación web, el uso potencial que tiene el algoritmo de LEVENSHTEIN y los conceptos asociados de distancias o métricas.

Actualmente grandes volúmenes de información estructurada y no estructurada se derivan de la Web, toda esta información se encuentra totalmente imprecisa, por lo que la integración de los diferentes lenguajes, frameworks y tecnologías de programación.

Como son HTML5, JQuery y CSS, hacen de este algoritmo sea una poderosa herramienta para el análisis de patrones sobre la web.

Referencias

Navarro G., Raffinot Mathieu, (2002). Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences, Cambridge University Press (221 pages).

Cormen T., Leirserson Ch., Rivest R.,(1990). Introduction to Algorithms Second Edition. The MIT Press Cambridge, Massachusetts London, England.

Jones N., Pevzner P. (2004) , An Introduction to Bioinformatics Algorithms, A Bradford Book The MIT Press, Cambridge, Massachusetts London, England.2004.

Yates B., Navarro (2012). Approximate string matching for spam filtering. , Second International Conference on Innovative Computing Technology (INTECH),pp. 16-20.

ISSN 2007-882X