novática
Revista de la Asociación de Técnicos de Informática

Novática, revista de aparición trimestral fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Consejo Editorial
Guillem Alsina González, Rafael Fernández Calvo (presidente del Consejo), Jaime Fernández Martínez, Luis Fernández Sanz, José Antonio Gutiérrez de Mesa, Silvia Leal Martín, Dídac López Viñas, Francesc Noguera Puig, Joan Antoni Pastor Collado, Viktu Pons i Colomer, Moisés Robles Gener, Cristina Vigil Díaz, Juan Carlos Vigo López

Coordinación Editorial
Llorenç Pagés Casas <pages@ati.es>
Composición y autoedición
Jorge Llácer Gil de Ramales
Traducciones
Grupo de Lengua e Informática de ATI <http://www.ati.es/gt/lengua-informatica/>
Administración
Tomás Brunete, María José Fernández, Enric Camarero

Secciones Técnicas - Coordinadores
Acceso y recuperación de la Información
José María Gómez Hidalgo (Optenet), <jmgomezh@yahoo.es>
Enrique Puertas Sanz (Universidad Europea de Madrid), <enrique.puertas@uem.es>
Administración Pública electrónica
Francisco López Crespo (MAE), <flc@ati.es>
Sebastià Justicia Pérez (Diputación de Barcelona) <sjusticia@ati.es>
Arquitecturas
Enrique F. Torres Moreno (Universidad de Zaragoza), <enrique.torres@unizar.es>
José Flich Cardo (Universidad Politécnica de Valencia, <jflich@disca.upv.es>
Auditoría SITIC
Marina Touriño Troitiño, <marinatourino@marinatourino.com>
Sergio Gómez-Landero Pérez (Endesa), <sergio.gomezlandero@endesa.es>
Derecho y tecnologías
Isabel Hernando Collazos (Fac. Derecho de Donostia, UPV), <isabel.hernando@ehu.es>
Elena Davara Fernández de Marcos (Davara & Davara), <edavara@davara.com>
Enseñanza Universitaria de la Informática
Cristóbal Pareja Flores (DSIP-UCM), <cpareja@sip.ucm.es>
J. Ángel Velázquez Iturbide (DLSI I, URJC), angel.velazquez@urjc.es>
Entorno digital personal
Andrés Marín López (Univ. Carlos III), <amarin@it.uc3m.es>
Diego Gachet Páez (Universidad Europea de Madrid), <gachet@uem.es>
Estándares Web
Encarna Quesada Ruiz (Virati), <encarna.quesada@virati.com>
José Carlos del Arco Prieto (TCP Sistemas e Ingeniería), <jcarco@gmail.com>
Gestión del Conocimiento
Joan Baiget Solé (Cap Gemini Ernst & Young), <joan.baiget@ati.es>
Gobierno corporativo de las TI
Manuel Palao García-Suelto (ATI), <manuel@palao.com>,
Miguel García-Menéndez (iTTi) <mgarciamenendez@ittrendsinstitute.org>
Informática y Filosofía
José Angel Olivas Varela (Escuela Superior de Informática, UCLM), <joseangel.olivas@uclm.es>
Roberto Feltrero Oreja (UNED), <rfeltrero@gmail.com>
Informática Gráfica
Miguel Chover Sellés (Universitat Jaume I de Castellón), <chover@lsi.uji.es>
Roberto Vivó Hernando (Eurographics, sección española), <rvivo@dsic.upv.es>
Ingeniería del Software
Luis Fernández Sanz, Daniel Rodríguez García (Universidad de Alcalá),<{luis.fernandezs, daniel.rodriguez}@uah.es>
Inteligencia Artificial
Vicente Botti Navarro, Vicente Julián Inglada (DSIC-UPV), <{vbotti,vinglada}@dsic.upv.es>
Interacción Persona-Computador
Pedro M. Latorre Andrés (Universidad de Zaragoza, AIPO), <platorre@unizar.es>
Francisco L. Gutiérrez Vela (Universidad de Granada, AIPO), <fgutierr@ugr.es>
Lengua e Informática
M. del Carmen Ugarte García (ATI), <cugarte@ati.es>
Lenguajes informáticos
Óscar Belmonte Fernández (Univ. Jaime I de Castellón), <belfern@lsi.uji.es>
Inmaculada Coma Tatay (Univ. de Valencia), <Inmaculada.Coma@uv.es>
Lingüística computacional
Xavier Gómez Guinovart (Univ. de Vigo), <xgg@uvigo.es>
Manuel Palomar (Univ. de Alicante), <mpalomar@dlsi.ua.es>
Modelado de software
Jesus García Molina (DIS-UM), <jmolina@um.es>
Gustavo Rossi (LIFIA-UNLP, Argentina), <gustavo@sol.info.unlp.edu.ar>
Mundo estudiantil y jóvenes profesionales
Federico G. Mon Trotti (RITSI), <gnu.fede@gmail.com>
Mikel Salazar Peña (Área de Jovenes Profesionales, Junta de ATI Madrid), <mikeltxo_uni@yahoo.es>
Profesión informática
Rafael Fernández Calvo (ATI), <rfcalvo@ati.es>
Miquel Sárries Griñó (ATI), <miquel@sarries.net>
Redes y servicios telemáticos
Juan Carlos López López (UCLM), <juancarlos.lopez@uclm.es>
Ana Pont Sanjuán (UPV), <apont@disca.upv.es>
Robótica
José Cortés Arenas (Sopra Group), <joscorare@gmail.com>
Juan González Gómez (Universidad CARLOS III), <juan@iearobotics.com
Seguridad
Javier Areitio Bertolín (Univ. de Deusto), <jareitio@deusto.es>
Javier López Muñoz (ETSI Informática-UMA), <jlm@lcc.uma.es>
Sistemas de Tiempo Real
Alejandro Alonso Muñoz, Juan Antonio de la Puente Alfaro (DIT-UPM), <{aalonso,jpuente}@dit.upm.es>
Software Libre
Jesús M. González Barahona (GSYC - URJC), <jgb@gsyc.es>
Israel Herráiz Tabernero (Universidad Politécnica de Madrid), <isra@herraiz.org>
Tecnologías para la Educación
Juan Manuel Dodero Beardo (UC3M), <dodero@inf.uc3m.es>
César Pablo Córcoles Briongo (UOC), <ccorcoles@uoc.edu>.
Tecnologías y Empresa
Didac López Viñas (Universitat de Girona), <didac.lopez@ati.es>
Alonso Álvarez García (TID) <aag@tid.es>
Tendencias tecnológicas
Gabriel Martí Fuentes (Interbits), <gabi@atinet.es>
Juan Carlos Vigo (ATI) <juancarlosvigo@atinet.es>
TIC y Turismo
Andrés Aguayo Maldonado, Antonio Guevara Plaza (Univ. de Málaga), <{aguayo, guevara}@lcc.uma.es>

Las opiniones expresadas por los autores son responsabilidad exclusiva de losmismos. ***Novática*** permite la reproducción, sin ánimo de lucro, de todos los artículos, a menos que lo impida la modalidad de © o copyright elegida por el autor, debiéndose en todo caso citar su procedencia y enviar a ***Novática*** un ejemplar de la publicación.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Plaza de España 6, 2ª planta, 28008 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn. 963740173 <novatica_prod@ati.es>
Administración y Redacción ATI Cataluña
Calle Ávila 48-50, 3a planta, local 9, 08005 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia <secregal@ati.es>
Suscripción y Ventas <novatica.subscripciones@atinet.es>
Publicidad Plaza de España 6, 2ª planta, 28008 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: "Del juego y la vida" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

sumario

Nº 230, octubre-diciembre 2014, año XL

Tema del próximo número: "La mujer en la Informática: historia, actualidad y retos para el futuro"

Facturación de SMS

Julio Javier Castillo, Diego Javier Serrano, Marina Elizabeth Cárdenas

Laboratorio de Investigación de Software MsLabs, Dpto. Ing. en Sistemas de Información, Facultad Regional Córdoba - Universidad Tecnológica Nacional (Argentina)

<jotacastillo@gmail.com>, <diegojserrano@gmail.com>, <ing.marinacardenas@gmail.com>

El enunciado de este problema apareció en el número 229 de ***Novática*** (julio-septiembre 2014, p. 106).

El problema planteado puede ser resuelto sin dificultad con un algoritmo de fuerza bruta. Para el cálculo del precio de cada mensaje solamente se requiere la longitud del mismo más el costo fijo de 10 centavos. Finalmente debe restarse el total de los descuentos otorgados por las abreviaturas que la empresa ofrece en forma promocional.

El eje del problema se encuentra en la identificación de todas las apariciones en cada mensaje de las abreviaturas que la empresa promociona.

1. Fuerza bruta

Un algoritmo de fuerza bruta es sencillo de implementar tomando cada una de las abreviaturas disponibles y buscándola en cada uno de los mensajes con algún método de búsqueda de subcadenas. En el caso de Java el método `int indexOf(String target, int fromIndex)` de la clase `String` es suficiente para lograr este objetivo.

Sin embargo, este algoritmo no es muy rápido. Por cada mensaje debe recorrer la lista de abreviaturas, y por cada una de ellas iniciar una búsqueda de subcadenas, la cual en el mejor de los casos es lineal a la cantidad de caracteres del mensaje. Por lo tanto este algoritmo presenta una complejidad temporal de O(n*m), siendo n la cantidad de abreviaturas y m la longitud de todos los mensajes. Y dicha complejidad empeora en el caso de que el método de búsqueda de subcadenas no fuera lineal.

2. Mejora propuesta

Dado que la lista de abreviaturas no tiene ambigüedades, no deberían encontrarse pares de abreviaturas en los que una de ellas esté incluida dentro de la otra. Es decir, no existirá una abreviatura "sa" y otra "a", ya que eso no permitiría determinar con certeza cuál de los descuentos se debe aplicar.

En ese caso se puede plantear una estructura de datos que almacene todas las abreviaturas permitiendo un recorrido lineal, de forma que cada carácter del mensaje sea recorrido una única vez independientemente de la cantidad de abreviaturas disponibles.

La estructura de datos más adecuada para esta situación es un árbol de búsqueda digital, también llamado en la literatura especializada *trie*.

Un *trie* es un árbol que permite almacenar un conjunto de cadenas de forma tal que para aquellas cadenas que comparten un mismo prefijo se almacena una única vez dicho prefijo y solo se dividen en ramas del árbol las diferencias.

De esta manera, si se almacenan las palabras "aro" y "árbol", existirá un nodo para la letra ‘a’, con un hijo para la letra ‘r’ y de allí dos hijos, uno para la letra ‘o’ y otro para la letra ‘b’ (ver **figura 1**). Por este motivo también se conoce a estos árboles como *prefix trees*.

Estos árboles permiten realizar búsquedas muy rápidas sobre cadenas mediante la estrategia de recorrer simultáneamente la cadena en donde se busca y el árbol que la representa. Para ello se plantea la existencia de una referencia o puntero que en todo momento esté apuntando a un nodo del árbol, el cual es asignado inicialmente a la raíz y por comodidad lo llamaremos nodo actual.

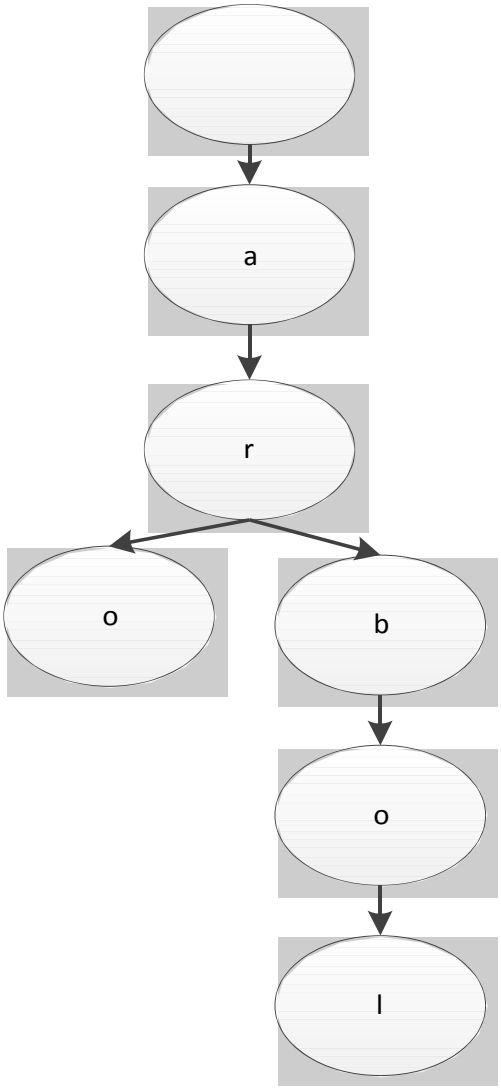

Figura 1. Árbol digital o *trie*.

Durante el recorrido de la cadena donde se realiza la búsqueda, por cada carácter de la misma se verifica si en el nodo actual existe un hijo para el mismo carácter. En tal caso, se asigna el nodo actual al nodo hijo correspondiente y se continúa el recorrido desde allí.

Cuando el recorrido finaliza en un nodo hoja, es decir un nodo sin hijos, se concluye que se acaba de encontrar la cadena correspondiente a la palabra. Y si, por otra parte, ocurre que no existe ningún hijo para continuar el recorrido, entonces se asigna como actual al nodo raíz, concluyendo así que no existe ninguna de las palabras del árbol en esa posición de la cadena.

3. Solución propuesta

En la solución presentada se desarrolla la estructura de búsqueda *Trie* en las clases `Trie` y `Nodo`. La clase `Nodo` posee un arreglo de referencias a `Nodo` para representar los posibles hijos de cada nodo. Dado que en el problema se indica que las abreviaturas sólo pueden estar compuestas de letras y dígitos, el arreglo plantea la posible existencia de 36 hijos por cada nodo, 26 para las letras sin acentos ni ñ y 10 para los caracteres de dígitos, del '0' al '9'. El arreglo de hijos se crea en el constructor del nodo, pero se mantiene su contenido sin datos, es decir con todas las referencias nulas.

La clase `Trie` está compuesta por dos referencias a nodos, siendo la más importante la referencia raíz que posee la raíz del árbol y es creada durante su construcción. Por otro lado la referencia llamada actual se utiliza en el recorrido del árbol durante las búsquedas. En la clase `Trie` el comportamiento principal se encuentra en los métodos `insertar` y en el método `descuento`. El método `insertar` inserta una abreviatura dentro del árbol, creando nos nodos necesarios. En el caso de encontrar una abreviatura cargada previamente que comparta los primeros caracteres con la que se esté insertando, el algoritmo recorre el árbol hasta encontrar la primera diferencia y a partir de allí crea nodos nuevos.

Finalmente el método `descuento` efectúa el recorrido concreto en el árbol, a razón de un carácter por ejecución del método. Si durante el recorrido se llega a un nodo hoja, el método retorna el descuento que se debe efectuar ya que, en ese caso, se encontró una abreviatura en el mensaje actual, y en cualquier otro caso retorna 0. Además puede observarse que cuando se encuentra una abreviatura se reinicia el recorrido.

Con respecto a la complejidad temporal de la solución, la carga de los datos naturalmente es lineal a la cantidad de los mismos, por lo tanto el análisis debe centrarse en los algoritmos que manipulan el árbol. La inserción en este árbol es lineal a la longitud de cada abreviatura. La complejidad temporal de la búsqueda es en función de la longitud de los mensajes de texto, y dado que la misma se realiza con una operación constante por cada carácter e independiente a la cantidad de abreviaturas, el recorrido de cada mensaje será lineal a su longitud.

En consecuencia, la complejidad será igual a O(n+m), con n igual a la longitud de todas las abreviaturas y m igual a la longitud de todos los mensajes. Mientras que cualquier algoritmo que requiera recorrer la lista de abreviaturas durante el procesamiento de los mensajes será al menos cuadrático.

A continuación se expone la solución propuesta en lenguaje Java:

```
import java.util.Scanner;

public class SMS {

      public static void main(String[] args) {

           Scanner sc = new Scanner(System.in);
          Trie arbol = new Trie();
           int bonificaciones = sc.nextInt();

          for (int i = 0; i < bonificaciones; i++) {
               String abreviatura = sc.next();
               int descuento = sc.nextInt();
                arbol.insertar(abreviatura, descuento);
          }

           int mensajes = sc.nextInt();
           sc.nextLine(); // consume el retorno de carro

           arbol.iniciar();
          int acumulador = 0;

          for (int i = 0; i < mensajes; i++) {

               String linea = sc.nextLine();
               int precio = 10 + linea.length();

               for (char c : linea.toCharArray()) {
                    precio -= arbol.descuento(c);
              }
               if (precio < 0) precio = 0;

                System.out.println(precio);
               acumulador += precio;
          }
             System.out.println(acumulador);
     }
}

class Trie {
     Nodo raiz;
     Nodo actual;

     Trie() {
          raiz = new Nodo();
     }

      void insertar(String palabra, int descuento) {
          Nodo p = raiz;
           for (char c : palabra.toCharArray()) {
               int pos = posicion(c);
               if (p.hijos[pos] == null) {
```

```
                    p.hijos[pos] = new Nodo();
                }
                p = p.hijos[pos];
            }
            p.descuento = descuento;
            p.hoja = true;
        }

        void iniciar() {
            actual = raiz;
        }

        int descuento(char l) {
            actual = actual.hijos[posicion(l)];
            if (actual == null) {
                iniciar();
            } else if (actual.hoja) {
                int d = actual.descuento;
                iniciar();
                return d;
            }
            return 0;
        }

        int posicion(char l)
        {
            if (l >= 'a' && l <= 'z') return l - 'a';
            if (l >= 'A' && l <= 'Z') return l - 'A';
            if (l >= '0' && l <= '9') return l - '0' + 26;
            return 36;
        }
    }

    class Nodo {
        int descuento;
        boolean hoja;
        Nodo[] hijos = new Nodo[37];
    }
```