
Algoritmo 1: División Greedy

```
Var Global: ispellList // Palabras de ispell + Diccionario
Var Global: abrevList // Abreviaciones conocidas
Var Global: stopList // Palabras Excluyentes
Entrada   : idHarword // identificador a dividir
Salida    : softwordDiv // id separado con espacios

1 softwordDiv  $\leftarrow$  ""
2 softwordDiv  $\leftarrow$  dividirCaracteresEspecialesDigitos(idHarword)
3 softwordDiv  $\leftarrow$  dividirCamelCase(softwordDiv)
4 forall the (Para cada substring s separado por ' ' en softwordDiv)
  do
5   if (s no pertenece a (stopList  $\cup$  abrevList  $\cup$  ispellList)) then
6     sPrefijo  $\leftarrow$  buscarPrefijo(s, "")
7     sSufijo  $\leftarrow$  buscarSufijo(s, "")
8     // Se elije la división que mayor particiones hizo.
9     s  $\leftarrow$  maxDivisión(sPrefijo, sSufijo)
9 return softwordDiv // Retorna el id dividido por espacios.
```

Función buscarPrefjjo

Entrada: s // Abreviaturas a dividir

Salida : *abrevSeparada* // Abreviaturas separadas

// Punto de parada de la recursión.

1 if ($length(s) = 0$) then

2 return *abrevSeparada*

3 if (s pertenece a ($stopList \cup abrevList \cup ispellList$)) then

4 return ($s + ' ' + buscarPrefjjo(abrevSeparada, "")$)

// Se extrae y se guarda el último caracter de s .

5 $abrevSeparada \leftarrow s[length(s) - 1] + abrevSeparada$

// Llamar nuevamente a la función sin el último caracter.

6 $s \leftarrow s[0, length(s) - 1]$

7 return $buscarPrefjjo(s, abrevSeparada)$

Función buscarSufijo

Entrada: s // Abreviaturas a dividir

Salida : *abrevSeparada* // Abreviaturas separadas

// Punto de parada de la recursión.

1 if ($length(s) = 0$) then

2 return *abrevSeparada*

3 if (s pertenece a ($stopList \cup abrevList \cup ispellList$)) then

4 return ($buscarSufijo(abrevSeparada, "") + ' ' + s$)

// Se extrae y se guarda el primer caracter de s .

5 $abrevSeparada \leftarrow abrevSeparada + s[0]$

// Llamar nuevamente a la función sin el primer caracter.

6 $s \leftarrow s[1, length(s)]$

7 return $buscarSufijo(s, abrevSeparada)$

Algoritmo 2: divisiónHardWord

Entrada: *token* // *token a dividir*

Salida : *tokenSep* // *token separado con espacios*

```
1 token ← dividirCaracteresEspecialesDigitos(token)
2 token ← dividirMinusSeguidoMayus(token)
3 tokenSep ← ""
4 forall the (Para cada substring s separado por ' ' en token) do
5     if (  $\exists \{i | esMayus(s[i]) \wedge esMinus(s[i + 1])\}$  ) then
6         n ← length(s) - 1
7         // se determina con la función score si es del tipo
8         // camelcase u otra alternativa
9         scoreCamel ← score(s[i,n])
10        scoreAlter ← score(s[i+1,n])
11        if (scoreCamel >  $\sqrt{scoreAlter}$ ) then
12            if (i > 0) then
13                s ← s[0,i - 1] + ' ' + s[i,n] // GP Sstate
14            else
15                s ← s[0,i] + ' ' + s[i + 1,n] // GPS state
16        tokenSep ← tokenSep + ' ' + s
17 token ← tokenSep
18 tokenSep ← ' '
19 forall the (Para cada substring s separado por ' ' en token) do
20     tokenSep ← tokenSep + ' ' + divisiónSoftWord(s,score(s))
21 return tokenSep
```

Algoritmo 3: divisiónSoftWord

Entrada: s // *softword string*

Entrada: $score_{sd}$ // *puntaje de s sin dividir*

Salida : $tokenSep$ // *token separado con espacios*

```
1  $tokenSep \leftarrow s$ ,  $n \leftarrow \text{length}(s) - 1$ 
2  $i \leftarrow 0$ ,  $maxScore \leftarrow -1$ 
3 while ( $i < n$ ) do
4    $score_{izq} \leftarrow \text{score}(s[0,i])$ 
5    $score_{der} \leftarrow \text{score}(s[i+1,n])$ 
6    $preSuf \leftarrow \text{esPrefijo}(s[0,i]) \vee \text{esSufijo}(s[i+1,n])$ 
7    $split_{izq} \leftarrow \sqrt{score_{izq}} > \max(\text{score}(s), score_{sd})$ 
8    $split_{der} \leftarrow \sqrt{score_{der}} > \max(\text{score}(s), score_{sd})$ 
9   if ( $\neg preSuf \wedge split_{izq} \wedge split_{der}$ ) then
10     if ( $(split_{izq} + split_{der}) > maxScore$ ) then
11        $maxScore \leftarrow (split_{izq} + split_{der})$ 
12        $tokenSep \leftarrow s[0,i] + ' ' + s[i+1,n]$ 
13     else if ( $\neg preSuf \wedge split_{izq}$ ) then
14        $temp \leftarrow \text{divisiónSoftWord}(s[i+1,n], score_{sd})$ 
15       if ( $temp$  se dividió?) then
16          $tokenSep \leftarrow s[0,i] + ' ' + temp$ 
17    $i \leftarrow i+1$ 
18 return  $tokenSep$ 
```

Algoritmo 4: Expansión Básica

Entrada: *abrev* // Abreviatura a expandir
Entrada: *wordList* // Palabras extraídas del código
Entrada: *phraseList* // Frases extraídas del código
Entrada: *stopList* // Palabras Excluyentes
Entrada: *dicc* // Diccionario en Inglés
Salida : *únicaExpansión* // Abreviatura expandida, o null

```
1 if (abrev ∈ stopList) then
2   | return null
3 listaExpansión ← [ ]
   // Buscar coincidencia de acrónimo.
4 forall the (Para cada frase phrase en phraseList) do
5   | if ( ∃{phrase | abrev es un acrónimo de phrase} ) then
6   |   | listaExpansión.add(phrase)
   // Buscar abreviatura común.
7 forall the ( Para cada palabra word en wordList ) do
8   | if ( ∃{word | abrev es una abreviatura de word} ) then
9   |   | listaExpansión.add(word)
   // Si no hay éxito, buscar en el diccionario.
10 if (isEmpty(listaExpansión)) then
11   | listaCandidatos ← buscarDiccionario(abrev,dicc)
11   | listaExpansión.add(listaCandidatos)
12 únicaExpansión ← null
   // Debe haber un solo resultado, sino no retorna nada.
13 if (length(listaExpansión) = 1) then
14   | únicaExpansión ← listaExpansión[0]
15 return únicaExpansión
```

Algoritmo 5: Búsqueda por Palabras Singulares

Entrada: *pa* // *Palabra Abreviada*
Entrada: *patrón* // *Expresión regular*
Entrada: Cuerpo y Comentarios del Método
Entrada: Comentarios de la Clase
Salida : Palabras largas candidatas, o null si no hay

// Las expresiones regulares están entre comillas

```
1 if (patrón prefijo  $\vee$  pa coincide "[a-z][aeiou]+"  $\vee$  length(pa) > 3)  $\wedge$ 
   (pa no coincide con "[a-z][aeiou][aeiou]+") then
    // Si alguna de las siguientes búsquedas encuentra un
    // único resultado, el algoritmo lo retorna
    // finalizando la ejecución
2   Buscar en Comentarios JavaDoc con "@param pa patrón"
3   Buscar en Nombres de Tipos y la correspondiente Variable
   declarada con "patrón pa"
4   Buscar en el Nombre del Método con "patrón"
5   Buscar en las Sentencias con "patrón pa" y "pa patrón"
6   if (length(pa  $\neq$  2)) then
7       Buscar en palabras del Método con "patrón"
8       Buscar en palabras que están en los Comentarios del Método
       con "patrón"
9   if (length(pa > 1)  $\wedge$  (patrón prefijo) then
       // Solo se busca con patrones prefijos
10  Buscar en palabras que están en los Comentarios de la Clase
    con "patrón"
```

Algoritmo 6: Búsqueda por Multi Palabras

Entrada: *pa* // Palabra Abreviada

Entrada: *patrón* // Expresión regular

Entrada: Cuerpo y Comentarios del Método

Entrada: Comentarios de la Clase

Salida : Palabras largas candidatas, o null si no hay

// Las expresiones regulares están entre comillas

```
1 if (patrón acrónimo  $\vee$  length(pa) > 3) then
    // Si alguna de las siguientes búsquedas encuentra un
    // único resultado, el algoritmo lo retorna
    // finalizando la ejecución
2   Buscar en Comentarios JavaDoc con “@param pa patrón”
3   Buscar en Nombres de Tipos y la correspondiente Variable
   declarada con “patrón pa”
4   Buscar en el Nombre del Método con “patrón”
5   Buscar en todos los ids (y sus tipos) dentro del Método con
   “patrón”
6   Buscar en Literales String con “patrón”
   // En este punto se buscó en todos los lugares
   // posibles dentro del método
7   Buscar en palabras que están en los Comentarios del Método con
   “patrón”
8   if (patrón acrónimo) then
       // Solo se busca con patrones Acrónimos
9       Buscar en palabras que están en los Comentarios de la Clase
       con “patrón”
```
