Algoritmo 1: División Greedy

In : id a dividir, ***idHarword***

In : Diccionario Ingles + Palabras Linux **ispell**

In : Palabras Excluyentes, ***stopList***

Out: id separado con espacios

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

Función buscarPrefjjo(string ***s***, string ***abrevSeparada***)

In : abreviaturas a dividir, ***s***

Out: abreviaturas separadas con espacios, ***abrevSeparada***

```
// Punto de parada de la recursión.
if (length(s) = 0) then
    return abrevSeparada
if (s pertenece a stopList ∨ s pertenece a (Dicc + ispell)) then
    return (s + ' ' + buscarPrefjjo(abrevSeparada,"") )
// Se extrae y se guarda el último caracter de s.
abrevSeparada ← s[length(s) - 1] + abrevSeparada
// Llamar nuevamente a la función sin el último caracter.
s ← s[0 , length(s) - 1]
return buscarPrefjjo(s,abrevSeparada)
```

Función buscarSufijo(string ***s***, string ***abrevSeparada***)

In : abreviaturas a dividir, ***s***

Out: abreviaturas separadas con espacios, ***abrevSeparada***

```
// Punto de parada de la recursión.
if (length(s) = 0) then
    return abrevSeparada
if (s pertenece a stopList ∨ s pertenece a (Dicc + ispell)) then
    return (buscarSufijo(abrevSeparada,"") + ' ' + s)
// Se extrae y se guarda el primer caracter de s.
abrevSeparada ← abrevSeparada + s[0]
// Llamar nuevamente a la función sin el primer caracter.
s ← s[1 , length(s)]
return buscarSufijo(s,abrevSeparada)
```

Algoritmo 2: divisiónHardWord(***token***)

In : token a dividir, ***token***
Out: token separado con espacios, ***tokenSep***

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

Algoritmo 3: divisiónSoftWord($\boldsymbol{s}$,$\boldsymbol{score}_{sd}$)

Input : softword string, $\boldsymbol{s}$

Input : score sin dividir, $\boldsymbol{score}_{sd}$

Output: token separado con espacios, ***tokenSep***

```
1  tokenSep ← s, n ← length(s) - 1
2  i ← 0, maxScore ← − 1
3  while (i < n) do
4      score_izq ← score(s[0,i])
5      score_der ← score(s[i+1,n])
6      preSuf ← esPrefijo(s[0,i]) ∨ esSufijo(s[i+1,n])
7      split_izq ← √score_izq > max(score(s),score_sd)
8      split_der ← √score_der > max(score(s),score_sd)
9      if (!presuf ∧ split_izq ∧ split_der) then
10         if ((split_izq + split_der) > maxScore) then
11             maxScore ← (split_izq+split_der)
12             tokenSep ← s[0,i] + ' ' + s[i+1,n]
13     else if (!presuf ∧ split_izq) then
14         temp ← divisiónSoftWord(s[i+1,n],score_sd)
15         if (temp se dividió?) then
16             tokenSep ← s[0,i] + ' ' + temp
17     i ← i+1
18 return tokenSep
```

Algoritmo 4: Expansión Básica

In : Abreviatura a expandir, ***abrev***
In : Lista de Palabras extraídas del código, ***wordList***
In : Lista de Frases extraídas del código, ***phraseList***
In : Palabras Excluyentes, ***stopList***
In : Diccionario Ingles, ***dicc***
Out: Abreviatura expandida, o null si no encontró una

```
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)
       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

In : palabra abreviada, ***pa***
In : expresión regular se encarga de buscar la palabra larga, ***patrón***
In : cuerpo y comentarios del método
In : comentarios de la clase
Out: 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

In : palabra abreviada potencial, ***pa***
In : expresión regular se encarga de buscar la palabra larga, ***patrón***
In : cuerpo y comentarios del método
In : comentarios de la clase
Out: 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***"