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*** ← “”
2 ***softwordDiv*** ← dividirCaracteresEspecialesDigitos(***idHarword***)
3 ***softwordDiv*** ← dividirCamelCase(***softwordDiv***)
4 **para todo** (***s*** | ***s*** *es un substring separado por ‘ ’ en* ***softwordDiv***) **hacer**
5 **si** (***s*** *no pertenece a* (***stopList*** ∪ ***abrevList*** ∪ ***ispellList***)) **entonces**
6 resultadoPrefijo ← buscarPrefijo(***s***,“”)
7 resultadoSufijo ← buscarSufijo(***s***,“”)
`// Se elije la división que mayor particiones hizo.`
8 ***s*** ← maxDivisión(resultadoPrefijo,resultadoSufijo)
9 **devolver** ***softwordDiv*** `// Retorna el id dividido por ‘ ’`

Función buscarPrefjjo

Entrada: ***s*** *// cadena a dividir*
Salida : ***sDiv*** *// Cadena dividida con espacios*

```
1 si (length(s) = 0) entonces
      // Punto de parada de la recursión, retorna el
         resultado de la división.
2     devolver sDiv

3 si (s pertenece a (stopList ∪ abrevList ∪ ispellList )) entonces
      // Se separa el prefijo encontrado y el resto de la
         cadena se sigue procesando.
4     devolver (s + ' ' + buscarPrefjjo(sDiv,"") )

  // Se extrae y se guarda el último caracter de s.
5 sDiv ← s[length(s) - 1] + sDiv

  // Llamar nuevamente a la función sin el último caracter.
6 s ← s[0 , length(s) - 1]
7 devolver buscarPrefjjo(s,sDiv)
```

Función buscarSufijo

Entrada: ***s*** *// cadena a dividir*
Salida : ***sDiv*** *// Cadena separada con espacios*

```
1 si (length(s) = 0) entonces
      // Punto de parada de la recursión, retorna el
         resultado de la división.
2     devolver sDiv

3 si (s pertenece a (stopList ∪ abrevList ∪ ispellList )) entonces
      // Se separa el sufijo encontrado y el resto de la
         cadena se sigue procesando.
4     devolver (buscarSufijo(sDiv,"") + ' ' + s)

  // Se extrae y se guarda el primer caracter de s.
5 sDiv ← sDiv + s[0]

  // Llamar nuevamente a la función sin el primer caracter.
6 s ← s[1 , length(s)]
7 devolver buscarSufijo(s,sDiv)
```

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  para todo (s | s es un substring separado por ' ' en token) hacer
5      si ( ∃{i|esMayus(s[i]) ∧ esMinus(s[i + 1])} ) entonces
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          si (scoreCamel > √scoreAlter) entonces
10             si (i > 0) entonces
11                 s ← s[0,i − 1] + ' ' + s[i,n] // GP Sstate
12         en otro caso
13             s ← s[0,i] + ' ' + s[i + 1,n] // GPS state
14     tokenSep ← tokenSep + ' ' +s
15 token ← tokenSep
16 tokenSep ← ' '
17 para todo (s | s es un substring separado por ' ' en token) hacer
18     tokenSep ← tokenSep + ' ' + divisiónSoftWord(s,score(s))
19 devolver tokenSep
```

Algoritmo 3: divisiónSoftWord

Entrada: s `// softword string`
Entrada: $score_{sd}$ `// puntaje de s sin dividir`
Salida : $tokenSep$ `// token separado con espacios`

```
1  tokenSep ← s, n ← length(s) - 1
2  i ← 0, maxScore ← − 1
3  mientras (i < n) hacer
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      si (!presuf ∧ split_izq ∧ split_der) entonces
10         si ((split_izq + split_der) > maxScore) entonces
11             maxScore ← (split_izq+split_der)
12             tokenSep ← s[0,i] + ' ' + s[i+1,n]
13     sinó, si (!presuf ∧ split_izq) entonces
14         temp ← divisiónSoftWord(s[i+1,n],score_sd)
15         si (temp se dividió?) entonces
16             tokenSep ← s[0,i] + ' ' + temp
17     i ← i+1
18 devolver tokenSep
```

Algoritmo 4: Expansión Básica

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

```
si (abrev pertenece stopList) entonces
    devolver null
listaExpansión ← [ ]
// Buscar coincidencia de acrónimo.
para todo (Frase | Frase es una frase en listaFrases) hacer
    si (abrev es un acrónimo de Frase) entonces
        // Se prioriza aquella Frase que está en el mismo
           método que abrev
        devolver Frase
// Buscar abreviatura común.
para todo (Pal | Pal es una palabra en listaPalabras ) hacer
    si (abrev es una abreviatura de Pal) entonces
        // Se prioriza aquella Pal que está en el mismo
           método que abrev
        devolver Pal
// Si no hay éxito, buscar en el diccionario.
listaCandidatos ← buscarDiccionario(abrev,dicc)
listaExpansión.add(listaCandidatos)
únicaExpansión ← null
// Debe haber un solo resultado, sino no retorna nada.
si (length(listaExpansión) = 1) entonces
    únicaExpansión ← listaExpansión[0]
devolver únicaExpansión
```

Algoritmo 5: Búsqueda por Palabras Singulares

Entrada: ***pa*** `// Palabra Abreviada`
Entrada: ***patrón*** `// Expresión regular`
Salida : ***palCand*** `// Palabras candidatas, o null si no hay`

`// Las expresiones regulares están entre comillas`

1 **si** (***patrón*** *prefijo* $\vee$ ***pa*** *coincide "[a-z][^aeiou]+"* $\vee$ *length(**pa**)* > 3) $\wedge$ (***pa*** *no coincide con "[a-z][aeiou][aeiou]+"*) **entonces**

`// 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***", si el resultado es único **devolver** en ***palCand***

3 Buscar en Nombres de Tipos y la correspondiente Variable declarada con "***patrón pa***", si el resultado es único **devolver** en ***palCand***

4 Buscar en el Nombre del Método con "***patrón***", si el resultado es único **devolver** en ***palCand***

5 Buscar en las Sentencias con "***patrón pa***" y "***pa patrón***", si el resultado es único **devolver** en ***palCand***

6 **si** (*length(**pa**)* $\neq 2$) **entonces**

7 Buscar en palabras del Método con "***patrón***", si el resultado es único **devolver** en ***palCand***

8 Buscar en palabras que están en los Comentarios del Método con "***patrón***", si el resultado es único **devolver** en ***palCand***

9 **si** (*length(**pa**)* > 1) $\wedge$ (***patrón*** *prefijo*) **entonces**

`// Solo se busca con patrones prefijos`

10 Buscar en palabras que están en los Comentarios de la Clase con "***patrón***", si el resultado es único **devolver** en ***palCand***

Algoritmo 6: Búsqueda por Multi Palabras

Entrada: ***pa*** *// Palabra Abreviada*
Entrada: ***patrón*** *// Expresión regular*
Salida : ***palCand*** *// Palabras candidatas, o null si no hay*

// Las expresiones regulares están entre comillas

1 **si** (***patrón*** *acrónimo* $\vee$ *length(**pa**)* $>$ *3*) **entonces**

    // 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***", si el resultado es único **devolver** en ***palCand***

3 Buscar en Nombres de Tipos y la correspondiente Variable declarada con "***patrón pa***", si el resultado es único **devolver** en ***palCand***

4 Buscar en el Nombre del Método con "***patrón***", si el resultado es único **devolver** en ***palCand***

5 Buscar en todos los ids (y sus tipos) dentro del Método con "***patrón***", si el resultado es único **devolver** en ***palCand***

6 Buscar en Literales String con "***patrón***", si el resultado es único **devolver** en ***palCand***

    // 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***", si el resultado es único **devolver** en ***palCand***

8 **si** (***patrón*** *acrónimo*) **entonces**

        // Solo se busca con patrones Acrónimos

9 Buscar en palabras que están en los Comentarios de la Clase con "***patrón***", si el resultado es único **devolver** en ***palCand***