Algoritmo 1 Alta mediante bits sufijos.

```
int HashingExtensibleSufijo::insertar(registro, tabla, bloques):
        int posTabla = fhash(registro.clave)
        int numeroBloque = tabla[posTabla]
        Bloque bloque = bloques[numeroBloque]

        if (bloque.contiene(registro.clave))
                return RES_REGISTRO_DUPLICADO

        if (bloque.cantRegsLibres < tabla.cantRegsPorBloque)
                bloque.insertar(registro)
                return RES_OK

        // La clave no esta en el bloque y no se puede insertar en el
        // El bloque aparece referenciado en la tabla solo 1 vez
        if (bloque.td == tabla.tt)
                tabla.duplicar()
                tabla.tt *= 2

                Bloque nuevoBloque // Uno nuevo o el primero de bloquesLibres
                bloques.agregar(nuevoBloque) // Si cree uno nuevo

                bloque.td *= 2
                nuevoBloque.td = bloque.td

                // Actualizo la referencia en la tabla
                tabla[posTabla] = numNuevoBloque

        // La clave no esta en el bloque y no se puede insertar en el
        // El bloque aparece referenciado en la tabla mas de 1 vez
        else:
                Bloque nuevoBloque // Uno nuevo o el primero de bloquesLibres
                bloques.agregar(nuevoBloque) // Si cree uno nuevo

                bloque.td *= 2
                nuevoBloque.td = bloque.td

                // Actualizo las referencias en la tabla
                int punteroInicial = numBloque
                while (tabla[punteroInicial] == numBloque)
                        tabla[punteroInicial] = numNuevoBloque
                        punteroInicial = (punteroInicial + bloque.td) mod (tabla.tt)

        bloque.redispersar()

        insertar(registro, tabla, bloques)
```

Algoritmo 2 Baja mediante bits sufijos

```
int HashingExtensibleSufijo::baja (registro, tabla, bloques, bloquesLibres):
        int posTabla = funcionMod(registro.clave)
        int numeroBloque = tabla[posTabla]
        Bloque bloque = bloques[numeroBloque]

        if (! bloque.contiene(registro.clave))
                return RES_REGISTRO_NO_EXISTE

        if (bloque.cantRegsLibres > 1)
                bloque.eliminar(registro)
                return RES_OK

        // El bloque queda vacio
        else:
                int posTabla_atras = (posTabla - bloque.td / 2) mod (tabla.tt)
                int posTabla_adelante = (posTabla - bloque.td / 2) mod (tabla.tt)

                if (tabla[posTabla_atras] == tabla[posTabla_adelante):
                        // El bloque se lo puede agregar a "bloques libres"

                        bloquesLibres.agregar(numBloque)
                        numBloqueReemplazo = tabla[posTabla_atras]
                        Bloque reemplazo = bloques[numBloqueReemplazo]

                        // Se reemplazan sus referencias en la tabla por otro bloque
                        int punteroInicial = numBLoque
                        while (tabla[punteroInicial] != numBloqueReemplazo)
                                tabla[punteroInicial] = numBloqueReemplazo
                                punteroInicial = (punteroInicial + reemplazo.td) mod

                        reemplazo.td /= 2
                        if (tabla.mitadesIguales())
                                tabla.cortarPorMitad()

                else:
                        // No se puede marcar como libre, queda vacio
                        return RES_OK
```