Módulo DiccionarioTrie(α)

Interfaz

parámetros formales
géneros $string, \alpha$
se explica con: $Diccionario(\alpha)$
géneros: $diccTrie(\alpha)$

Operaciones básicas de DiccTrie(α)

CrearDicc() $\rightarrow$ res:diccTrie(α)
Pre $\equiv$ {true}
Post $\equiv$ {res$=_{obs}$ vacio}
Complejidad: O(1)
Descripcion: Crea un diccionario vacio

Definir(**in/out** d: diccTrie(α), **in** c: string, **in** s:conj(α))
Pre $\equiv$ { d$=_{obs}$ $d_0 \wedge \neg$definido?(d,c)}
Post $\equiv$ { d $=_{obs}$ definir(d_0,c,s) }
Complejidad: O(Longitud(c))
Descripcion: define la clave c con el significado en el diccionario.

Definido?(**in** d: diccTrie(α), **in** c:string) $\rightarrow$ res:bool
Pre $\equiv$ {true}
Post $\equiv$ { res $=_{obs}$ def?(c,d) }
Complejidad: O(Longitud(c))
Descripcion: devuelve true si y solo si c esta en el diccionario

Significado(**in** d: diccTrie(α), **in** c:string) $\rightarrow$ res:α
Pre $\equiv$ { def?(c,d) }
Post $\equiv$ { res $=_{obs}$ obtener(c,d) }
Complejidad: O(Longitud(c))
Descripcion: devuelve el significado con clave c
Aliasing: No se devuelve una copia del alpha en res, se devuelva una referencia a la original.

TodosLosSignificados(**in/out** d:diccTrie(α)) $\rightarrow$ res:conj(α)
Pre $\equiv$ { true }
Post $\equiv$ { ($\forall$ a:α) a $\in$ res $\Rightarrow$ ($\exists$ c: clave) c $\in$ claves(d) $\wedge_L$ a = obtener(d,c) }
Complejidad: O()
Descripcion: devuelve todos los significados guardados en el diccionario d
Aliasing: res no es modificable

Representacion

diccTrie(α) se representa con **dic**
donde **dic** es arreglo_dimensionable de tuplas
donde tuplas es <ptrDic:puntero(dic), ptrSignificado: puntero(α) >

Rep: dic d $\rightarrow$ bool
Rep(d) $\equiv$ tam(d) = 27

Abs(d) $\equiv$ d':diccTrie(α) |
($\forall$ c:clave) def?(c,d') $=_{obs}$ (Significado(c,d) != NULL) $\wedge$ obtener(c,d') $=_{obs}$ Significado(c,d)

Algoritmos

iCrear() → *res*: dic

```
d: arreglo(tuplas)
j:nat
d ← CrearArreglo[27]
j ← 0
while j < 27
    d[j].ptrTrie ← NULL
    d[j].ptrSignificado ← NULL
    j++
endwhile
```

iDefinir(**in/out** *d*:dic, **in** *c*:string, **in** *s*: α)

```
j:nat
p:dic
i:nat
j ← 0
p ← d
while j < Longitud(c) - 1
    i ← ord(c[j])
    if p[i].ptrTrie = NULL then
        p[i].ptrTrie ← &iCrearDicc()
    fi
    p ← *(p[i].ptrTrie)
    j++
endwhile
i ← ord(c[j])
p[i].ptrSignificado ← &s
```

iSignificado(**in** *d*:dic, **in** *c*:string) → *res*: (α)

```
j:nat
p:dic
i:nat
j ← 0
p ← d
while j < Longitud(c) - 1
    i ← ord(c[j])
    p ← *(p[i].ptrTrie)
    j++
endwhile
i ← ord(c[j])
res ← *(p[i].ptrSignificado)
```

iDefinido?(**in** *d*:dic, **in** *c*:string) → *res*: bool

```
j:nat
p:dic
j ← 0
p ← d
while j < Longitud(c) - 1
    i ← ord(c[j])
    p ← *(p[i].ptrTrie)
    j++
endwhile
i ← ord(c[j])
```

```
    res → (p[i].ptrSignificado != NULL)

SignificadosHijos(in/out d:dic, in s: string) → conj(α)
    i:nat
    j:nat
    p:dic
    j ← 0
    p ← d
    while j < Longitud(s) - 2 ∧ p != vacia()
        i ← ord(s[j])
        if p[i].ptrDic = NULL then
            p ← vacia()
        else
            p ← *(p[i].ptrDic)
            j++
    endwhile
    if p = vacia() then
        res ← vacia()
    else
        res ← todosLosSignificados(p)
    fi

TodosLosSignificados(in/out d:dic) → conj(α)
    j:nat
    p:dic
    res ← iVacia()
    j ← 0
    p ← d
    while j < 27
        if p[j].ptrSignificado != NULL then
            it ← Agregar(res, *(p[j].ptrSignificado))
                if p[j].ptrDic != NULL then
                    res ← res ∪ TodosLosSignificados(*(p[j].ptrdic))
                fi
        fi
        j++
    endwhile

    ∪(in/out c1: conj(α), in c2: conj(α)) → res: conj(α)
        it: itConj(α)
        it ← CrearIt(c2)
        while haySiguiente?(c2)
            Agregar(c1, siguiente(c2))
            avanzas(c2)
        endwhile
```