1. Módulo DiccClavesRapidas

Interfaz

parámetros formales
género *significado*
función COPIAR(**in** a: *significado*) $\rightarrow$ *res* : *significado*
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} a$}
Complejidad: $\Theta(copy(a))$
Descripción: función de copia de *significado*

se explica con: DICCIONARIO(STRING, SIGNIFICADO)

géneros: `dcr`

El modulo funciona como un diccionario, pero solo se utiliza con claves del tipo *string*.

Operaciones básicas de DiccClavesRapidas

VACIO() $\rightarrow$ *res* : `dcr`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ vacio}
Complejidad: $O(1)$
Descripción: Crea un diccionario vacio.

DEFINIR(**in** n: `string`, **in** s: `significado`, **in/out** d: `dcr`)
Pre $\equiv$ {$d =_{\text{obs}} d_0$}
Post $\equiv$ {$d =_{\text{obs}}$ definir(n, s, d_0)}
Complejidad: $O(Longitud(n))$
Descripción: Define la clave n con significado s en el diccionario d.

DEF?(**in** n: `string`, **in** d: `dcr`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ def?(n, d)}
Complejidad: $O(Longitud(n))$
Descripción: Devuelve *true* si el string n esta definido en el diccionario d.

OBTENER(**in** s: `string`, **in** d: `dcr`) $\rightarrow$ *res* : `significado`
Pre $\equiv$ {def?(n, d)}
Post $\equiv$ {$res =_{\text{obs}}$ obtener$(p, m) \wedge$ alias$(res, obtener(p, n))$}
Complejidad: $O(Longitud(n))$
Descripción: Retorna el *significado* de la clave n.
Aliasing: res devuelve el significado por referencia.

BORRAR(**in** s: `string`, **in/out** d: `dcr`)
Pre $\equiv$ {def?$(p, m) \wedge d =_{\text{obs}} d_0$}
Post $\equiv$ {$d =_{\text{obs}}$ borrar(p, d_0)}
Complejidad: $O(Longitud(n))$
Descripción: Elimina la clave n y su *significado* del diccionario d.

CLAVES(**in** d: `dcr`) $\rightarrow$ *res* : `conj(string)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ claves$(d) \wedge$ alias$(res =_{\text{obs}} claves(d))$}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de claves del diccionario d
Aliasing: *res* no es modificable. Si se definen nuevas claves, *res* se invalida.

Representación

Representación de DiccClavesRapidas

`dcr` se representa con `estr`

donde `estr` es `tupla`(*dicc*: `arreglo(puntero(nodo))` , *claves*: `conj(string)`)
donde `nodo` es `tupla`(*definido*: `bool` , *dato*: `significado` , *sig*: `arreglo(puntero(nodo))`, *it*: `itConj(string)`)

Invariante de representacion en castellano:

1. Toda clave de *claves* esta definida en *dicc*, es decir, *dicc* posee un nodo con la misma clave.
2. Todo nodo donde su .definido es *true*, su .clave esta contenida en *claves*.
3. El arreglo *dicc* posee longitud igual a 256
4. Todo nodo contenido en la estructura posee el arreglo *sig* con tamaño igual a 256

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

1. ($\forall$ s : string) $s \in$ e.claves $\Rightarrow$ (hayCamino(s, e.dicc) $\wedge_{\text{L}}$ Π_1(tuplaEnDicc(s, e.dicc))) $\wedge$
2. ($\forall$ s : string) (hayCamino(s, e.dicc) $\wedge_{\text{L}}$ Π_1(tuplaEnDicc(s, e.dicc)) $\Rightarrow$ $s \in$ e.claves $\wedge$
3. tam(e.dicc) = 256 $\wedge$
4. ($\forall$ s : string) (hayCamino(s, e.dicc) $\wedge_{\text{L}}$ tam(Π_3(tuplaEnDicc(s, e.dicc))) = 256

Abs : estr e $\longrightarrow$ dicc($string, significado$) {Rep(e)}
Abs(e) $\equiv$ d : dicc($string, significado$) /
($\forall$ s : string) En(s, e.claves) $=_{\text{obs}}$ def?(p, d) $\wedge_{\text{L}}$
($\forall$ s : string) En(s, e.claves) $\Rightarrow_{\text{L}}$ Π_2(tuplaEnDicc(s, e.dicc)) $=_{\text{obs}}$ obtener(s, d))

Operaciones auxiliares de TAD:

hayCamino : string s $\times$ arreglo $array$ $\longrightarrow$ bool {tam(array) = 256}
hayCamino(s, array) $\equiv$ **if** Longitud(s) = 0 **then**
true
else
if definido(ord(prim(s)), array) **then**
hayCamino(fin(s), *array[ord(prim(s))])
else
false
fi
fi

tuplaEnDicc : string s $\times$ arreglo $array$ $\longrightarrow$ tupla($\alpha_1 ... \alpha_n$) {hayCamino($s, array$) $\wedge$ tam(array) = 256}
tuplaEnDicc(s, array) $\equiv$ **if** Longitud(s) = 1 **then**
*array[ord(prim(s))]
else
tuplaEnDicc(fin(s), array[ord(prim(s))]
fi

Algoritmos

Lista de algoritmos

iVacio() → res: estr
begin
  res.dicc ← CrearArreglo(256) //**O(1)**
  res.claves ← Vacio() //**O(1)**
  return *res*
end
Complejidad: $O(1)$

Algoritmo 1: Vacio

```
iDefinir(in n: string, in s: significado, in/out e: estr)
begin
    var t : string; p : puntero(nodo) tuplaVacia : nodo; ittr : itConj(string)
    t ← fin(n)                                                          //O(1)
    ittr ← CrearIt(e.claves) tuplaVacia ← <false, s, CrearArreglo(256), ittr>   //O(1)
    if ¬Def?(n, e.claves) then                                          //O(Longitud(n))
        if ¬Definido(e.dicc, Ord(Prim(t)) ) then                        //O(1)
            e.dicc[Ord(Prim(t))] ← tuplaVacia
        end
                                                                        //O(1)
        p ← e.dicc[Ord(Prim(t))]                                        //O(1)
        while Longitud(t) >0 do                                         //O(Longitud(n))
            if ¬Definido(p.sig, Ord(Prim(t)) ) then                     //O(1)
                p.sig[Ord(Prim(t))] ← tuplaVacia
            end
                                                                        //O(1)
            p ← p.sig[Ord(Prim(t))]                                     //O(1)
            t ← fin(t)                                                  //O(1)
        end
        p.definido ← true                                               //O(1)
        p.dato ← s                                                      //O(1)
        ittr ← AgregarRapido(n, e.claves)                               //O(1)
        p.it ← ittr                                                     //O(1)
        else
            while Longitud(t) >0 do                                     //O(Longitud(n))
                p ← p[Ord(Prim(t))]                                     //O(1)
                t ← fin(t)                                              //O(1)
            end
            p.dato ← s                                                  //O(1)
        end
    end
end
```

Complejidad: O(Longitud(n))

Comentarios: *AgregarRapido*$(n, e.claves)$ es O(1) y se puede utilizar dado que se sabe por el if que no esta definido. El it se guarda para luego poder eliminar del conjunto en O(1). En la rama else, donde la clave ya había sido definida anteriormente, solo actualizamos el significado.

Algoritmo 2: Definir

```
iDef?(in n: string, in e: estr) → res: bool
begin
    var t : string; p : puntero(nodo)
    t ← fin(n)                                                          //O(1)
    p ← e.dicc[prim(n)]                                                 //O(1)
    while Longitud(t) >0 do                                             //O(Longitud(n))
        p ← p.sig[Ord(Prim(t))]                                         //O(1)
        t ← fin(t)                                                      //O(1)
    end
    res ← p.definido                                                    //O(1)
    return res
end
```

Complejidad: O(Longitud(n))

Algoritmo 3: Def?

```
iObtener(in n: string, in e: estr) → res: significado
begin
    var t : string; p : puntero(nodo)
    t ← fin(n)                                                    //O(1)
    p ← e.dicc[Ord(prim(n))]                                      //O(1)
    while Longitud(t) >0 do                                       //O(Longitud(n))
        p ← p.sig[Ord(Prim(t))]                                   //O(1)
        t ← fin(t)                                                //O(1)
    end
    res ← p.dato                                                  //O(1)
    return res
end
Complejidad: O(Longitud(n))
```

Algoritmo 4: Obtener

```
iBorrar(in s: string, in/out d: dcr)
begin
    var t : string; p : puntero(nodo)
    t ← fin(n)                                                    //O(1)
    p ← e.dicc[Ord(prim(n))]                                      //O(1)
    while Longitud(t) >0 do                                       //O(Longitud(n))
        p ← p.sig[Ord(Prim(t))]                                   //O(1)
        t ← fin(t)                                                //O(1)
    end
    p.definido ← false
end
Complejidad: O(Longitud(n))
```

Algoritmo 5: Borrar

```
iClaves(in e: estr) → res: conj(string)
begin
    res ← e.claves
end
Complejidad: O(1)
```

Algoritmo 6: Claves