

1. Modulo DiccRapido(clave, significado)

Interfaz

se explica con: DICCIONARIO(CLAVE, SIGNIFICADO)

generos: diccRapido(clave, significado)

Operaciones basicas de conjunto

clave debe ser de tipo nat, sino la funcion de hash no funciona.

VACIO() $\rightarrow res : \text{diccRapido}(\text{clave}, \text{significado})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{vacio}()\}$

Complejidad: $O(N)$ donde N es la cantidad de claves en c

Descripción: Crea un nuevo diccionario vacio.

DEFINIR(**in** $k : \text{clave}$ **in** $s : \text{significado}$, **in/out** $d : \text{diccRapido}(\text{clave}, \text{significado})$)

Pre $\equiv \{d =_{\text{obs}} d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{definir}(k, s, d_0)\}$

Complejidad: $\Theta(1)$

Descripción: Define el elemento k , con significado s , en el diccionario d

DEF?(**in** $c : \text{clave}$, **in** $d : \text{diccRapido}(\text{clave}, \text{significado})$) $\rightarrow res : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$

Complejidad: $O(N)$, con N siendo la cantidad de claves

Descripción: Devuelve *true* si la clave k esta definida

OBTENER(**in** $c : \text{clave}$, **in** $d : \text{diccRapido}(\text{clave}, \text{significado})$) $\rightarrow res : \text{significado}$

Pre $\equiv \{\text{def?}(c, d)\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{obtener}(c, d))\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve el significado de la clave c

Aliasing: res es una referencia al significado de c en el diccionario d . Si se modifica, se modificara el significado dentro del diccionario. Si se borra una clave, o se define alguna clave, la referencia queda invalidada.

CLAVES(**in** $d : \text{diccRapido}(\text{clave}, \text{significado})$) $\rightarrow res : \text{conj}(\text{clave})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{dameUno}(c)\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve el conjunto de claves del diccionario d

Aliasing: res es una referencia constante a un $\text{conj}(\text{clave})$.

Representación

Representacion del DiccRapido

$\text{diccRapido}(\text{clave}, \text{significado})$ se representa con *estr*

donde *estr* es $\text{tupla}(\text{defs: arreglo}(\text{tuplaSignificado}), \text{claves: conj}(\text{clave}))$

donde tuplaSignificado es $\text{tupla}(\text{key: clave}, \text{def: significado})$

Invariante de representacion en castellano:

1. Todas las claves estan definidas en el arreglo y todas las cosas definidas estan en claves.
2. La funcion Hash manda a cada clave al lugar donde esta definida.

Rep : estr \rightarrow bool

Rep(d) \equiv true \iff

1. ($(\forall c : \text{clave}) \text{En}(c, d.\text{claves}) \Rightarrow ((\exists! i : \text{nat}) \text{Definido?}(i, d.\text{defs}) \wedge_{\text{L}} d.\text{defs}[i].\text{key} = c)) \wedge$
 $(\forall i : \text{nat}) \text{Definido?}(i, d.\text{defs}) \Rightarrow_{\text{L}} \text{En}(d.\text{defs}[i].\text{key}, d.\text{claves})) \wedge_{\text{L}}$
2. ($(\forall c : \text{clave}) \text{En}(c, d.\text{claves}) \Rightarrow (\text{Definido?}(\text{Hash}(c, c, d), d.\text{defs}) \wedge_{\text{L}} d.\text{defs}[\text{Hash}(c, c, d)].\text{key} = c)$
 $)$

Hash(k, c, d) \equiv **if** (Definido?(fHash) \wedge_{L} d.defs[fHash].key == c) \vee (\neg Definido?(fHash)) **then**
 fHash

else

Hash($k + 1, c, d$)

fi

//Donde fHash es ($k \% \text{Cantidad}(d.\text{claves})$)

//La clave k es el valor sobre el cual aplico el Hash, y la clave c es la clave que busco ubicar.

Abs : estr $e \rightarrow$ Diccionario(clave, significado)

{Rep(e)}

Abs(e) $\equiv d : \text{Diccionario}(\text{clave}, \text{significado}) /$

$(\forall c : \text{clave}) \text{En}(c, e.\text{claves}) \iff \text{def?}(c, d) \wedge_{\text{L}}$

$(\forall c : \text{clave}) \text{def?}(c, d) \Rightarrow_{\text{L}} \text{obtener}(c, d) =_{\text{obs}} e.\text{defs}[\text{Hash}(c, c, e)].\text{significado}$

Algoritmos

Algoritmos de Agentes

Lista de algoritmos

1.	Vacio	2
2.	Definir	3
3.	Def?	3
4.	Obtener	3
5.	Claves	3
6.	nombre	4

iVacio() \rightarrow res: estr

begin

 res.claves \leftarrow Vacio()

//O(1)

 res.defs \leftarrow CrearArreglo(1)

//O(1)

end

Complejidad: O(1)

Algoritmo 1: Vacio

```

iDefinir(in k: clave, in s: significado, in/out d: estr)
begin
  var
    i: nat
    arregloAux: Arreglo
    i ← 0
    if Def?(k, d) then
      //Entonces es redefinir
      d.defs[Hash(k, k, d)].def ← s
      d.defs[Hash(k, k, d)].key ← k
    else
      //Entonces es definir algo nuevo, asi que debo ampliar el arreglo
      Agregar(s, d.claves)
      arregloAux ← CrearArreglo(Tam(d.defs) + 1)
      while i < Tam(d.defs) do
        //Re-Hasheo con la nueva cantidad de keys
        arregloAux[Hash(d.defs[i].key, d.defs[i].key, d)].key ← d.defs[i].key
        arregloAux[Hash(d.defs[i].key, d.defs[i].key, d)].def ← d.defs[i].def
        i ← i + 1
      end
      //Y defino a k donde corresponde
      arregloAux[Hash(k, k, d)].def ← s
      arregloAux[Hash(k, k, d)].key ← k
      d.defs ← arregloAux
    end
  end
  //While: O(N)
  //O(1)
  //O(1)
  //O(1)
  //O(N)
end
Complejidad: O(N)

```

Algoritmo 2: Definir

```

iDef?(in k: clave, in d: estr) → res: bool
begin
  | res ← En(k, d.claves)
end
Complejidad: O(N)

```

Algoritmo 3: Def?

```

iObtener(in k: clave, in d: estr) → res: significado
begin
  | res ← &(d.defs[Hash(k, k, d)].significado)
end
Complejidad: O(1)

```

Algoritmo 4: Obtener

```

iClaves(in d: estr) → res: conj(clave)
begin
  | res ← d.claves
end
Complejidad: O(1)

```

Algoritmo 5: Claves

```

Hash(in  $c$ : clave, in  $d$ : estr)  $\rightarrow$  res: nat
begin
  var
    k : nat
    desvio : nat
    desvio  $\leftarrow$  0
    k  $\leftarrow$   $c \% \text{Cardinal}(d.claves)$ 
    while  $\text{Definido?}(k, d.defs) \wedge_{\text{L}} \neg(d.defs[k].key = c)$  do
      | desvio  $\leftarrow$  desvio + 1
      | k  $\leftarrow$   $(c + desvio) \% \text{Cardinal}(d.claves)$ 
    end
    //While: O(1) en caso promedio. O(N) peor caso. res  $\leftarrow$  k
end

```

Complejidad: O(1) en caso Promedio con buena distribucion. O(N) peor caso.

Comentarios: La idea es que intento poner a c en el resto de c en la division por la cantidad de claves. Si fallo, intento ponerlo en el siguiente lugar, y asi hasta poder colocarlo.

Algoritmo 6: nombre