DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 2

Algoritmos y Estructuras de Datos II

Grupo: 21

Integrante	LU	Correo electrónico
Langberg, Andrés	249/14	andreslangberg@gmail.com
Walter, Nicolás	272/14	nicowalter25@gmail.com
Sticco, Patricio Bernardo	337/14	pbsticco@hotmail.com
Len, Julián	467/14	julianlen@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. **TAD** POSICION **ES** TUPLA(X:NAT, Y:NAT)
2. **TAD** DIRECCION **ES** ENUM{ IZQ,DER,ARRIBA,ABAJO}
3. **TAD** AGENTE **ES** NAT
4. **TAD** NOMBRE **ES** STRING
5. Suponemos que contamos con el TAD DiccionarioM, donde la funcion vacio() toma como parámetro un 'k', cuyo valor acota superiormente a la cantidad de claves.

1. Diseño del Tipo DiccionarioProm

1.1. Especificación

Se usa el Tad DiccionarioM (Nota al corrector: leer observaciones).

1.2. Aspectos de la interfaz

1.2.1. Interfaz

Se explica con especificación de DiccionarioM(κ, σ)

Género `diccProm(`κ`,`σ`)`

Operaciones básicas de diccionario

Definido?(**in** d: $diccProm(\kappa, \sigma)$, **in** k: κ) $\longrightarrow res : bool$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} def?(d, k)$ }
Complejidad: $\mathcal{O}(n)$ *n es la cantidad de claves.*
Descripción: Devuelve true si y sólo si k está definido en el diccionario.

Obtener(**in** d: $diccProm(\kappa, \sigma)$, **in** k: κ) $\longrightarrow res : \sigma$
Pre $\equiv$ { $def?(d, k)$ }
Post $\equiv$ { $alias(res =_{\text{obs}} obtener(d, k))$ }
Complejidad: $\mathcal{O}(n)$ *n es la cantidad de claves.*
Descripción: Devuelve el significado de la clave k en d.
Aliasing: se devuelve una referencia al significado de la clave.

Claves(**in** d: $diccProm(\kappa, \sigma)$) $\longrightarrow res : itConj(\kappa)$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} claves(d)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: *Devuelve el conjunto con las claves definidas en d.*

Vacio(**in** n: nat) $\longrightarrow res : diccProm(\kappa, \sigma)$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} vacio(n)$ }
Complejidad: $\mathcal{O}(n)$
Descripción: *Genera un diccionario vacío, donde n acota superiormente a la cantidad de claves.*

Definir(**in/out** d: $diccProm(\kappa, \sigma)$, **in** k: κ, **in** s: σ)
Pre $\equiv$ { $d =_{\text{obs}} d_0$ }
Post $\equiv$ { $d =_{\text{obs}} definir(k, s, d_0)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: *Define la clave k con el significado s en el diccionario.*

1.3. Pautas de implementación

1.3.1. Estructura de representación

$diccProm(\kappa, \sigma)$ **se representa con** $estr$
donde $estr$ **es**
tupla(
Cclaves: ***conjLineal(***κ***)*** $\times$
clavesMax: ***nat*** $\times$
tabla: ***arreglo*** *de* $lista(datos)$
)
donde $datos$ **es**
tupla(
clave: κ $\times$

significado: σ
)

1.3.2. Justificación

1.3.3. Invariante de Representación

Informal

- *clavesMax es mayor que cero.*
- *La longitud del arreglo es igual a clavesMax.*
- *Todas las posiciones del arreglo estan definidas.*
- *Todos los elementos de Cclaves estan definidos en la tabla y viceversa.*
- *Todas las claves de la tabla estan definidos en Cclaves.*

Formal

$Rep : estr \longrightarrow$ boolean
$(\forall$ e $: estr)$
$Rep($e$) \equiv (true \iff$
$(1)\ e.clavesMax > 0 \wedge_{\text{L}}$
$(2)\ longitud(e.tabla) == e.clavesMax \wedge$
$(3)\ (\forall\ i{:}\ nat)(i \leq e.clavesMax \Rightarrow_{\text{L}} definido?(e.tabla,i)) \wedge$
$(3)\ (\forall\ k{:}\ \kappa)(k \in e.Cclaves\ \Rightarrow\ (\exists\ j : nat)(estaEn?(e.tabla[j],k))) \wedge$
$(4)\ (\forall\ i{:}\ nat)(\forall\ k{:}\ \kappa)(i\ \ e.clavesMax \wedge_{\text{L}} estaEn?(e.tabla[i],k)\ \Rightarrow\ k \in e.Ccclaves\ )\ )$

Funciones Auxiliares

$estaEn?\ :\ lista(datos) \times \kappa\ \longrightarrow\ bool$
$estaEn?(l,k)\ \equiv\ (\exists\ i : nat)(i<longitud(l) \Rightarrow_{\text{L}} l[i].clave == k)$

1.3.4. Función de Abstracción

$Abs : estr\ e \longrightarrow DiccionarioProm(\kappa, \sigma)$ $\{Rep(e)\}$
$(\forall\ e{:}estr)\ Abs(e) =_{\text{obs}} d : DiccionarioProm(\kappa, \sigma)\ /$
cla

Funciones Auxiliares

1.3.5. Algoritmos

```
1: function IVACIO(in n: nat )⟶ res : estr                          ▷ O(clavesMax)
2:     var arreglo(lista(datos)) tabla ← crearArreglo[n]            ▷ O(clavesMax)
3:     for i ← 0 to n do                                            ▷ O(clavesMax)
4:     tabla[i] ← Vacia()                                           ▷ O(1)
5:     end for
6:     res ← <n,tabla>                                              ▷ O(1)
7: end function
```

```
1: function IDEFINIR(in/out d: estr, in k: nat, in s: σ)            ▷ O(1)
2:     nat i ← fHash(k, e.clavesMax)                                ▷ O(1)
3:     e.tabla[i] ← AgregarAtras(e.tabla[i],<k,s>)                  ▷ O(1)
4: end function
```

```
1: function IOBTENER(in d: estr, in k: nat)⟶ res : σ                ▷ O(longitud(tabla[i]))
2:     nat i ← fHash(k, e.clavesMax)                                ▷ O(1)
3:     var itLista(datos) it ← crearIt(tabla[i])
4:     while haySiguiente(it) do
5:         if siguiente(it).clave = k then
6:             res ← siguiente(it).significado
7:         end if
8:     end while
9: end function
```

```
1: function IDEFINIDO?(in d: estr, in k: nat)⟶ res : bool            ▷ O(longitud(tabla[i]))
2:     nat i ← fHash(k, e.clavesMax)                                ▷ O(1)
3:     var itLista(datos) it ← crearIt(tabla[i])
4:     bool aux ← false
5:     while haySiguiente(it) do
6:         if siguiente(it).clave = k then
7:             aux ← true
8:         end if
9:     end while
10:    res ← aux
11: end function
```

```
1: function FHASH(in k: nat, in clavesMax: nat)⟶ res : nat           ▷ O(1)
2:     res ← k mod clavesMax                                        ▷ O(1)
3: end function
```

```
1: function ICLAVES(in d: estr )⟶ res : itConj(κ)                   ▷ O(clavesMax)
2:     res ← crearIt(e.Cclaves)
3: end function
```

1.4. Servicios Usados