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. Diseño del Tipo DiccionarioProm(σ)

1.1. Especificación

Se usa el Tad Diccionario(κ, σ) especificado en el apunte de Tads básicos.

1.2. Aspectos de la interfaz

1.2.1. Interfaz

parámetros formales

género κ, σ

función $\bullet = \bullet$(**in** a_1: κ, **in** a_2: κ) $\longrightarrow res : bool$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} (a_1 = a_2)$ }
Complejidad: $\Theta(equals(a_1, a_2))$
Descripción: función de igualdad de κ's

función **COPIAR**(**in** k: κ) $\longrightarrow res : \kappa$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} k$ }
Complejidad: $\Theta(copy(k))$
Descripción: función de copia de κ's

función **COPIAR**(**in** s: σ) $\longrightarrow res : \sigma$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} s$ }
Complejidad: $\Theta(copy(s))$
Descripción: función de copia de σ's

Se explica con especificación de Diccionario(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: $\mathcal{O}(Na)$ *Na es la cantidad de agentes.*
Descripción: Devuelve true si y sólo si k está definido en el diccionario.

Obtener(**in** d: $diccString(\kappa, \sigma)$, **in** k: κ) $\longrightarrow res : \sigma$
Pre $\equiv$ { $def?(d, k)$ }
Post $\equiv$ { $alias(res =_{\text{obs}} obtener(d, k))$ }
Complejidad: $\mathcal{O}(Na)$ *Na es la cantidad de agentes.*
Descripción: Devuelve el significado de la clave k en d.
Aliasing: res no es modificable.

Vacio(**in** $cantClaves$: nat) $\longrightarrow res : diccString(\kappa, \sigma)$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} vacio()$ }
Complejidad: $\mathcal{O}(Na)$ *Na es la cantidad de agentes.*
Descripción: Genera un diccionario vacío.

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(
CantClaves: ***nat*** $\times$
tabla: ***arreglo[CantClaves]*** $de\ lista(datos)$
)
donde $datos$ **es**
tupla(
clave: κ $\times$
significado: σ
)

1.3.2. Justificación

1.3.3. Invariante de Representación

Informal

- ***Todas las posiciones del arreglo de caracteres estÃ¡n definidas.***
- ***No hay claves de 0 caracteres. El significado de la raÃz es NULL.***
- ***No hay ciclos en la estructura. Es decir, existe una cota superior sobre la cantidad de niveles posibles del Ã¡rbol.***
- ***Dado un nodo cualquiera del trie, existe un Ãºnico camino desde la raÃz hasta el nodo.***

Formal

Rep : estr $\longrightarrow$ **boolean**
($\forall$ **e** ***: estr)***
Rep(**e****)*** $\equiv$ ***(***$true \iff$
(1)($\forall i : nat)(i < 256 \Rightarrow$ ***definido?(e*** $\rightarrow$ ***caracteres,i))*** $\wedge_{\text{L}}$
(2)(e $\rightarrow$ $significado = NULL) \wedge_{\text{L}}$
$(2)(\exists$ ***n:nat)(finaliza(e,n))*** $\wedge_{\text{L}}$
(3)($\forall$ ***p,q: puntero(nodo))(p*** $\in$ ***punteros(e)*** $\wedge q \in (punteros(e) - \{p\}) \Rightarrow$ ***p***$\neq q) \wedge_{\text{L}}$
)

1.3.4. Función de Abstracción

Abs : roseTree(estrDato) $r \longrightarrow$ ***dicc_trie(***σ***)*** ***{Rep(***r***)}***
($\forall$ r***:roseTree(estrDato)) Abs(***r***)*** $=_{\text{obs}}$ d ***: dicc_trie(***σ***) /***
($\forall$ k ***: secu(***$letra$***))(def?(k, d)*** $=_{\text{obs}}$ ***esta?(k, r))*** $\wedge$ ***(def?(c, d)*** $\Rightarrow$ ***(obtener(k, d)*** $=_{\text{obs}}$ ***buscar(k, r)))***

Funciones Auxiliares

1.3.5. Algoritmos

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

```
1: function iDefinir(in/out d: estr, in k: nat, in s: σ)               ▷ O(1)
2:     nat i ← fHash(k, e.cantClaves)                                   ▷ O(1)
3:     e.tabla[i] ← AgAtras(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.cantClaves)                                   ▷ O(1)
3:     bool flag ← false                                                ▷ O(1)
4:     nat j ← 0                                                        ▷ O(1)
5:     while ¬ flag ∧ j < longitud(tabla[i]) do                         ▷ O(longitud(tabla[i]))
6:         if datos.clave.tabla[i][j]=k then                            ▷ O(1)
7:             res ← datos.significado.tabla[i][j]                      ▷ O(1)
8:             aux ← true                                               ▷ O(1)
9:         end if
10:        j ← j + 1
11:    end while
12: end function
```

```
1: function IDEFINIDO?(in d: estr, in k: nat)⟶ res : bool                ▷ O(longitud(tabla[i]))
2:     nat i ← fHash(k, e.cantClaves)                                     ▷ O(1)
3:     nat j ← 0                                                          ▷ O(1)
4:     bool def ← false                                                   ▷ O(1)
5:     while (j < longitud(tabla[i]) and ¬ def) do                        ▷ O(longitud(tabla[i]))
6:         if datos.clave.tabla[i][j]=k then                              ▷ O(1)
7:             def ← true                                                 ▷ O(1)
8:         end if
9:     end while
10:    res ← def                                                          ▷ O(1)
11: end function
```

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

1.4. Servicios Usados

Requerimientos sobre el Tipo

- ***La función $|x|$ debe tener complejidad $\mathcal{O}(1)$ en el caso peor.***
- ***La función $|x|$ debe tener complejidad $\mathcal{O}(1)$ en el caso peor.***
- ***Las operaciones deben realizarse por referencia.***
- ***Debe proveer una operación Copia que devuelve una nueva instancia de la secuencia pero que es independiente de la actual, con complejidad $\mathcal{O}(n)$ en el caso peor.***
- ***Debe proveer un iterador para avanzar que comienza en el primero elemento de la secuencia.***
- ***Debe proveer un iterador para retroceder que comienza en el Ãºltimo elemento de la secuencia.***
- ***Las operaciones CrearIt, Siguiente, Anterior, TieneSiguiente, TieneAnterior deben tener complejidad $\mathcal{O}(1)$ en el caso peor.***

Donde n es la longitud de la palabra.