DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2016 Algoritmos y Estructuras de Datos II

Grupo 22

Integrante	LU	Correo electrónico
BENZO, Mariano	198/14	marianobenzo@gmail.com
FARIAS, Mauro	821/13	farias.mauro@hotmail.com
GUTTMAN, Martin	686/14	mdg_92@yahoo.com.ar

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

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) 4576-3300
http://www.exactas.uba.ar

Índice

1 Modulo Diccionario Lexicografico(*String*, σ)

1.1 Interfaz

se explica con DICCIONARIO(STRING, σ)

géneros `diccString(String, σ)`, `itDiccString(String, σ)`, `itClavesString`

Operaciones del diccionario

VACIO() $\longrightarrow$ *res* : `diccString(String, σ)`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} vacio()\}$
Descripción: Crea un nuevo diccionario
Complejidad: $O(1)$

DEFINIDO?(**in** *d* : `diccString(String, σ)` *in n* : `String`) $\longrightarrow$ *res* : `Bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} def?(n, d)\}$
Descripción: Indica si la clave esta definida
Complejidad: $O(Longitud(n))$

DEFINIR(**in/out** *d* : `diccString(String, σ)` *in n* : `String`, *in s* : σ)
Pre $\equiv \{\neg def?(n, d) \land d = d_0\}$
Post $\equiv \{d =_{\text{obs}} Definir(n, s, d_0)\}$
Descripción: Se define s en el diccionario
Complejidad: $O(Longitud(n) + copy(s))$
Aliasing: *s* se define por referencia

BORRAR(**in/out** *d* : `diccString(String, σ)` *in n* : `String`)
Pre $\equiv \{d =_{\text{obs}} d_0 \land def?(n, d)\}$
Post $\equiv \{d =_{\text{obs}} Borrar(n, d_0)\}$
Descripción: Elimina el elemento n
Complejidad: $O(Longitud(n))$

SIGNIFICADO(**in** *d* : `diccString(String, σ)` *in n* : `String`) $\longrightarrow$ *res* : σ
Pre $\equiv \{def?(n, d)\}$
Post $\equiv \{res =_{\text{obs}} obtener(n, d)\}$
Descripción: Se retorna el significado de n
Complejidad: $O(Longitud(n))$

DICCCLAVES(**in** *d* : `diccString(String σ)`) $\longrightarrow$ *res* : `conj`
Pre $\equiv \{TRUE\}$
Post $\equiv \{res =_{\text{obs}} claves(d)\}$
Descripción: Se retorna el conjunto de claves del diccionario
Complejidad: $O(1)$
Aliasing: Res un conjunto devuelto por referencia no modificable.

MAXIMO(**in** *d* : `diccString(String σ)`) $\longrightarrow$ *res* : `String`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} max(claves(d))\}$
Descripción: Se retorna la clave maxima en orden lexicográfico
Complejidad: $O(longitud(k))$ donde k es la aplabra más larga del diccionario

MINIMO(**in** d : `diccString (String` σ`)`) $\longrightarrow$ res : `String`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} min(claves(d))\}$
Descripción: Se retorna la clave minima en orden lexicográfico
Complejidad: $O(longitud(k))$ donde k es la aplabra más larga del diccionario

1.2 Operaciones del iterador

El iterador que presentamos permite modificar el diccionario recorrido. Sin embargo, cuando el diccionario es no modificable, no se pueden utilizar las funciones de eliminacion. Ademas, las claves de los elementos iterados no pueden modificarse nunca, por cuestiones de implementacion. Cuando d es modificable, decimos que it es modificable.

Para simplificar la notacion, vamos a utilizar clave y significado en lugar de Π_1 y Π_2 cuando utilicemos una tupla($String, \sigma$). CREARIT(**in** d : `diccString(String,` σ`)` *in* n : `String`) $\longrightarrow$ res : `itDiccString(`$String, \sigma$`)`
Pre $\equiv$ $\{true\}$
Post $\equiv$ {alias(esPermutacion(SecuSuby(res), d)) $\wedge$ vacia?(Anteriores(res))}
Descripción: crea un iterador bidireccional del diccionario, que apunta al primer elemento del mismo en orden lexicografico.
Complejidad: $O(CL * long(k))$ Donde CL es la cantidad de claves de d y k la palabra mas larga de d
Aliasing: hay aliasing entre los significados en el iterador y los del diccionario

HAYSIGUIENTE(**in** it : `itDiccString(String,` σ`)` *in* n : `String`) $\longrightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}}$ haySiguiente?(it)$\}$
Descripción: devuelve `true` si y solo si en el iterador todavia quedan elementos para avanzar.
Complejidad: $O(1)$

HAYANTERIOR(**in** it : `itDiccString(String,` σ`)` *in* n : `String`) $\longrightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}}$ hayAnterior?(it)$\}$
Descripción: devuelve `true` si y solo si en el iterador todavia quedan elementos para retroceder.
Complejidad: $O(1)$

SIGUIENTE(**in** it : `itDiccString(String,` σ`)` *in* n : `String`) $\longrightarrow$ res : `tupla(String,` σ`)`
Pre $\equiv$ {HaySiguiente?(it)}
Post $\equiv$ {alias($res =_{\text{obs}}$ Siguiente(it))}
Descripción: devuelve el elemento siguiente del iterador.
Complejidad: $O(1)$
Aliasing: res.significado es modificable si y solo si it es modificable, por aliasing. En cambio, res.clave no es modificable.

SIGUIENTECLAVE(**in** it : `itDiccString(String,` σ`)` *in* n : `String`) $\longrightarrow$ res : `String`
Pre $\equiv$ {HaySiguiente?(it)}
Post $\equiv$ {alias($res =_{\text{obs}}$ Siguiente(it).clave)}
Descripción: devuelve la clave del elemento siguiente del iterador.
Complejidad: $O(1)$
Aliasing: res no es modficable.

SIGUIENTESIGNIFICADO(**in** it : `itDiccString(String,` σ`)` *in* n : `String`) $\longrightarrow$ res : σ
Pre $\equiv$ {HaySiguiente?(it)}
Post $\equiv$ {alias($res =_{\text{obs}}$ Siguiente(it).significado)}
Descripción: devuelve el significado del elemento siguiente del iterador.

Complejidad: $O(1)$
Aliasing: *res* es modificable si y solo si *it* es modficable, por aliasing.

ANTERIOR(**in** *it* : `itDiccString(String,` σ`)` *in* *n* : `String`) $\longrightarrow$ *tupla(clave* : *String*, `significado:` σ`)`
Pre $\equiv$ {HayAnterior?(*it*)}
Post $\equiv$ {alias(*res* $=_{\text{obs}}$ Anterior(*it*))}
Descripción: devuelve el elemento anterior del iterador.
Complejidad: $O(1)$
Aliasing: *res*.significado es modificable si y solo si *it* es modificable, por aliasing. En cambio, *res*.clave no es modificable.

ANTERIORCLAVE(**in** *it* : `itDiccString(String,` σ`)` *in* *n* : `String`) $\longrightarrow$ *res* : *String*
Pre $\equiv$ {HayAnterior?(*it*)}
Post $\equiv$ {alias(*res* $=_{\text{obs}}$ Anterior(*it*).clave)}
Descripción: devuelve la clave del elemento anterior del iterador.
Complejidad: $O(1)$
Aliasing: *res* no es modficable.

ANTERIORSIGNIFICADO(**in** *it* : `itDiccString(String,` σ`)` *in* *n* : `String`) $\longrightarrow$ *res* : σ
Pre $\equiv$ {HayAnterior?(*it*)}
Post $\equiv$ {alias(*res* $=_{\text{obs}}$ Anterior(*it*).significado)}
Descripción: devuelve el significado del elemento anterior del iterador.
Complejidad: $O(1)$
Aliasing: *res* es modificable si y solo si *it* es modficable, por aliasing.

AVANZAR(**inout** *it* : `itDiccString(String,` σ`)` *in* *n* : `String`)
Pre $\equiv$ {$it = it_0 \wedge$ HaySiguiente?(*it*)}
Post $\equiv$ {*it* $=_{\text{obs}}$ Avanzar(it_0)}
Descripción: avanza a la posicion siguiente del iterador.
Complejidad: $O(1)$

RETROCEDER(**inout** *it* : `itDiccString(String,` σ`)` *in* *n* : `String`)
Pre $\equiv$ {$it = it_0 \wedge$ HayAnterior?(*it*)}
Post $\equiv$ {*it* $=_{\text{obs}}$ Retroceder(it_0)}
Descripción: retrocede a la posicion anterior del iterador.
Complejidad: $O(1)$

1.3 Representacion

diccString

se representa con `dLex)`

donde `dLex` **es** `tupla`⟨raiz : `puntero(nodo)`,
claves : `conj(string`⟩

nodo

se representa con `enodo)`

donde `enodo` **es** `tupla`⟨dato : σ,
esSig? : `Bool`,
claveEnConj : `itConj(String)`,
continuaciones : `puntero(char)` $\sqsubset 256 \sqsupset$⟩

1. conj(string) esel Conjunto Lineal de los modulos Básicos. itConj(string) es el iterador del mismo

Invariante de representación

1. Todo Nodo, si sus continuaciones son todos punteros a null, es porque es significado.
2. No hay ciclos, ni nodos con dos padres.
3. En el conjunto claves sólo se encuentran definidas las claves del diccionario y se encuentran todas ellas

Función de abstracción

Respresentación del iterador

El iterador del diccionario lo recorre en orden lexicográfico. Los significados están por referencia.Se explica con el iterador bidireccional no modificable. itDiccString($String$, σ)

se representa con `itdLex)`

donde `dLex` **es** `tupla`⟨claves : `Lista(`$String$`)`,
significados : `Lista(`σ`)`⟩

1.4 Algoritmos

1.4.1 Algoritmos del Diccionario

iVacio() $\longrightarrow$ res : `diccString`

$res.raiz \leftarrow NULL$	O(1)
$res.claves \leftarrow Vacio$	O(1)
	O(1)

iDefinir(**in/out** d : `diccString`, *in* n : `String`, *in* $s : \sigma$)

$aux \leftarrow d.raiz$	O(1)
$i \leftarrow 0$	O(1)
while $i < Longitud(n)$ **do**	O(Longitud(n))
if $aux == NULL$ **then**	
$nNodo.esSig? \leftarrow false$	O(1)
$j \leftarrow 0$	
while $j < 256$ **do**	O(256)=O(1)
$nNodo.continuaciones[j] \leftarrow NULL$	O(1)
end while	
$aux \leftarrow \&nNodo$	O(1)
end if	
$aux \leftarrow aux.continuaciones[ord(n[i])]$	O(1)
end while	
$aux * .esSig? \leftarrow True$	O(1)
if $aux * .esSig? \neq True$ **then**	
$aux * .claveEnConj \leftarrow AgregarRapido(d.claves, n)$	O(long(n))

end if
$aux * .esSig? \leftarrow True$ O(1)
$aux * .dato \leftarrow s$ O(1))

O(Longitud(n)), el último paso se realiza p

iDefinido?(**in/out** d : `diccString`, $in\ n$: `String`) $\longrightarrow$ res : `bool`

$aux \leftarrow d.raiz$ O(1)
$i \leftarrow 0$ O(1)
while $i < (Longitud(n) - 1) \wedge aux \neq NULL$ **do** O(Longitud(n))
  $aux \leftarrow aux.continuaciones[ord(n[i])]$ O(1)
end while
if $aux \neq NULL$ **then**
  $res \leftarrow aux * .esSig?$ O(1)
else
  $res \leftarrow false$ O(1)
end if

O(Longitud(n))

iSignificado(**in** d : `diccString`, $in\ n$: `String`) $\longrightarrow$ res : σ

$aux \leftarrow d.raiz$ O(1)
$i \leftarrow 0$ O(1)
while $i < Longitud(n)$ **do** O(Longitud(n))
  $aux \leftarrow aux.continuaciones[ord(n[i])]$ O(1)
  $i \leftarrow i + 1$
end while
$res \leftarrow aux * .dato$ O(1)(es una referencia)

O(Longitud(n))

iBorrar(**in/out** d : `diccString`, $in\ n$: `String`)

$aux \leftarrow d.raiz$ O(1)
$i \leftarrow 0$ O(1)
$pila(puntero(nodo))p \leftarrow Vacia()$ O(1)
while $i < Longitud(n)$ **do** O(Longitud(n))
  Apilar(p,aux) O(1)
  $aux \leftarrow aux.continuaciones[ord(n[i])]$ O(1)
  $i \leftarrow i + 1$
end while
$aux * .esSig? \leftarrow false$
$BorrarSiguiente(aux * .claveEnConj)$
$aux * esSig? \leftarrow false$
$i \leftarrow i - 1$
$j \leftarrow 0$
while $aux * .continuaciones[j] = NULL \wedge j < 256$ **do** O(256)=O(1)
  $j \leftarrow j + 1$
end while
if $j < 256$ **then**
  $p \leftarrow Vacia()$
end if
while $\neg EsVacia?(p)$ **do**
  $j \leftarrow 0$
  $Tope(p) * .continuaciones[ord(n[i])] \leftarrow NULL$
  while $Tope(p) * .continuaciones[j] = NULL \wedge j < 256$ **do** O(256)=O(1)

```
        j ← j + 1
      end while
      if j < 256 then
        p ← Vacia()
      else
        Desapilar(p)
        i ← i − 1
      end if
    end while
```

O(Longitud(n))

iDiccClaves(**in/out** d : diccString(String σ)) $\longrightarrow$ res : $conj(String)$

```
  res ← d.claves
```

O(1), d.claves se pasa por referencia

Maximo(**in/out** d : diccString(String σ)) $\longrightarrow$ res : $String$

```
  aux ← d.raiz
  res ← Vacia()
  Boolf ← True
  while f do
    nati ← 256
    while aux ∗ .continuaciones[i − 1] = NULL ∧ i > 0 do
      i ← i + 1
    end while
    if i = 0 then
      f ← false
    else
      AgregarAtras(res, ord⁻1(i − 1))
      i ← 0
    end if
  end while
```

O(1)

O(1) d.claves se pasa por referencia

Minimo(**in/out** d : diccString(String σ)) $\longrightarrow$ res : $String$

```
  aux ← d.raiz
  res ← Vacia()
  Boolf ← True
  while f do
    nati ← 0
    while aux ∗ .continuaciones[i] = NULL ∧ i < 256 do
      i ← i + 1
    end while
    if i = 256 then
      f ← false
    else
      AgregarAtras(res, ord⁻1(i))
      i ← 0
    end if
  end while
```

O(1)

O(1) d.claves se pasa por referencia

1.4.2 Algoritmos del iterador

iCrearIt(**in/out** d : `diccString`, *in* n : `String`) $\longrightarrow$ *res* : `itDiccString`

$lista(String)cs \leftarrow Vacia()$	
$lista(\sigma)ss \leftarrow Vacia()$	
$Stringn \leftarrow Vacio()$	
$auxXrIt(cs, ss, n, d.raiz)$	O(Longitud(k)*tam(d))
$res.claves \leftarrow cs$	
$res.significados \leftarrow ss$	
	O(Longitud(k)*tam(d))

1. k es la palabra más larga definida en d y tam(d) es la cantidad de palabras definidas que hay [1].

auxCrIt(**in/out** cs : `Lista(string) in/out ss:Lista(`σ`) in/out n:String in p:puntero(nodo)`)

if $p \neq NULL$ **then**	
if $p*.esSig?$ **then**	
$AgregarAtras(cs, n)$	
$AgregarAtras(ss, p*.dato)$	O(1)
end if	
$i \leftarrow 0$	
while $i < 256$ **do**	
$AgregarAtras(n, ord^{-}1(i))$	
$auxCrIt(cs, ss, n, p*.continuaciones[i])$	
$TirarUltimos(n, 1)$	
end while	
end if	
	Tn desconocido

iHaySiguiente(**in/out** it : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : *bool*

$res \leftarrow it.claves.siguiente \neq NULL$	
	O(1)

iHayAnterior(**in/out** it : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : *bool*

$res \leftarrow it.claves.anterior \neq NULL$	
	O(1)

iSiguiente(**in/out** it : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : `tupla(string,`σ`)`

$res.clave \leftarrow it.claves.siguiente*.dato$	O(1)(es una referencia)
$res.significado \leftarrow it.dignificados.siguiente*.dato$	O(1)(es una referencia)
	O(1)

iSiguienteclave(**in/out** it : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : *String*

$res \leftarrow it.claves.siguiente*.dato$	O(1)(es una referencia)
	O(1)

iSiguienteSignificado(**in/out** it : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : σ

[1] Si bien no nos fue posible realizar el calculo de complejidad correspondiente, el algoritmo recursivo recorre una vez cada nodo, y el total de nodos esta acotado por la sumatoria del largo de cada palabra, que a su vez está acotada por la cantidad de palabras multiplicada por la longitud de la palabra más larga, por lo que la cota propuesta a la complejidad es razonable

$res \leftarrow it.significados.siguiente * .dato$	O(1)(es una referencia)
	O(1)

iAnterior(**in/out** *it* : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : `tupla(string,`σ`)`

$res.clave \leftarrow it.claves.anterior * .dato$	O(1)(es una referencia)
$res.significado \leftarrow it.significados.anterior * .dato$	O(1)(es una referencia)
	O(1)

iAnteriorClave(**in/out** *it* : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : *String*

$res \leftarrow it.claves.anterior * .dato$	O(1)(es una referencia)
	O(1)

iAnteriorSignificado(**in/out** *it* : `itDiccString`, *in* n : `String`) $\longrightarrow$ *res* : σ

$res \leftarrow it.significados.anterior * .dato$	O(1)(es una referencia)
	O(1)

iAvanzar(**in/out** *it* : `itDiccString`, *in* n : `String`)

$it.claves \leftarrow it.claves.siguiente$	O(1)(es una referencia)
$it.significados \leftarrow it.significados.siguiente$	O(1)(es una referencia)
	O(1)

iRetroceder(**in/out** *it* : `itDiccString`, *in* n : `String`)

$it.claves \leftarrow it.claves.anterior$	O(1)(es una referencia)
$it.significados \leftarrow it.significados.anterior$	O(1)(es una referencia)
	O(1)