2. Módulo Tabla

El módulo Tabla provee una tabla con filas y columnas. Las columnas están representadas por los campos, identificados con un nombre, y las filas por los registros, que contiene valores asociados a los campos de la tabla.

Uno de los campos es el campo "clave" de la tabla. La función de este campo es poder identificar unívocamente a los registros, ya que no puede haber dos registros que tengan el mismo valor en el campo clave.

El módulo provee operaciones para crear una tabla nueva (vacía) indicando los nombres de sus campos y el del campo clave (en tiempo proporcional a la longitud de los nombres de los campos), insertar un registro, borrar un registro indicando el valor de su campo clave, obtener el conjunto de campos de la tabla, obtener el nombre del campo clave y obtener el conjunto de registros de la tabla (por referencia, en tiempo constante).

Además, es posible averiguar si un existe un registro con un determinado valor clave y obtenerlo en tiempo proporcional a la longitud del *string* valor. También se puede acceder a cada campo o columna de la tabla a partir de su nombre y obtener los valores que hay en esa columna para todos los registros. Finalmente, se provee una operación para obtener el conjunto de valores clave de los registros incluidos en la tabla.

Interfaz

se explica con: TAD Tabla

géneros: `tabla`.

Operaciones básicas de Tabla

Nueva(**in** cs : `conj(nombre_campo)`, **in** c : `nombre_campo`) $\rightarrow$ res : `tabla`
Pre $\equiv$ $\{c \in cs\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{nueva}(cs, c)\}$
Complejidad: $\Theta(|c|)$
Descripción: genera una tabla vacía con los campos cuyos nombres están contenidos en cs e identifica al campo c como el campo clave de la tabla.

Insertar(**in/out** t : `tabla`, **in** r : `registro`)
Pre $\equiv$ $\{t = t_0 \wedge \text{campos}(t) =_{\text{obs}} \text{campos}(r)\}$
Post $\equiv$ $\{t =_{\text{obs}} \text{insertar}(t_0, r)\}$
Complejidad: $\Theta(|v| + |c| + n + size(r))$, donde n es la cantidad de registros de la tabla.
Descripción: inserta el registro r en la tabla t. Si ya existe un registro con el mismo valor clave, lo sobreescribe.
Aliasing: si ya existe un registro con el mismo valor clave, el anterior se borra y se libera el espacio en memoria.

Borrar(**in/out** t : `tabla`, **in** v : `valor`)
Pre $\equiv$ $\{t = t_0\}$
Post $\equiv$ $\{t =_{\text{obs}} \text{borrar}(t, v)\}$
Complejidad: $\Theta(|v| + |c| + n + size(r))$, donde n es la cantidad de registros de la tabla.
Descripción: borra el registro que tiene el valor v en el campo clave de la tabla t.
Aliasing: se libera el espacio en memoria.

Campos(**in** t : `tabla`) $\rightarrow$ res : `conj(nombre_campo)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{campos}(t)\}$
Complejidad: $\Theta(1)$
Descripción: devuelve un conjunto con los nombres de los campos de la tabla t.
Aliasing: el conjunto se devuelve por referencia y no es modificable.

Clave(**in** t : `tabla`) $\rightarrow$ res : `nombre_campo`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{clave}(t)\}$
Complejidad: $\Theta(1)$
Descripción: devuelve el nombre del campo clave de la tabla t.

Registros(**in** t : `tabla`) $\rightarrow$ res : `conj(registro)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{registros}(t)\}$
Complejidad: $\Theta(1)$

Descripción: devuelve el conjunto de registros definidos en la tabla t.
Aliasing: el conjunto se devuelve por referencia y no es modificable.

ExisteRegConClave(**in** v: `valor`, **in** t: `tabla`) → res : `bool`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}}$ ($\exists r$: *registro*)($r \in$ registros(t) $\wedge_{\text{L}}$ r[clave(t)] = v}
Complejidad: $\Theta(|v|)$, donde v es la longitud máxima de valor.
Descripción: devuelve *true* si y solo si existe un registro en la tabla que tenga el valor v en el campo clave.

RegPorClave(**in** v: `valor`, **in** t: `tabla`) → res : `registro`
Pre ≡ {($\exists r$: registro)(r ∈ *registros*(t) $\wedge_{\text{L}}$ r[*clave*(t)]= v)}
Post≡ {$res \in$ registros(t) $\wedge_{\text{L}}$ res[clave(t)] = v}
Complejidad: $\Theta(|v|)$, donde v es la longitud máxima de valor.
Descripción: devuelve el registro que tiene el valor v en el campo clave.
Aliasing: el registro se devuelve por referencia y no es modificable.

ValoresClave(**in** t: `tabla`) → res : `conj(valor)`
Pre ≡ {true}
Post ≡ {#res = #registros(t) $\wedge_{\text{L}}$ ($\forall$ v: valor)($v \in res \Leftrightarrow$ ($\exists$ r: registro)($r \in$ registros(t) $\wedge_{\text{L}}$ r[clave(t)]=v))}
Complejidad: $\Theta(1)$
Descripción: devuelve un conjunto con los valores del campo clave de la tabla t.
Aliasing: el conjunto se devuelve por referencia y no es modificable.

ObtenerColumna(**in** c: `nombre_campo`, **in** t: `tabla`) → res : `dicc(itConj(registro), valor)`
Pre ≡ {$c \in$ campos(t)}
Post ≡ {El diccionario contiene iteradores a todos los registros de t y el valor asociado a cada registro es el valor que tiene el campo c en dicho registro en la tabla.}
Complejidad: $\Theta(|c|)$
Descripción: devuelve un diccionario lineal cuyas claves son iteradores a los registros del conjunto de registros de la tabla t y cuyos significados son los valores asociados al campo c en cada registro.
Aliasing: el diccionario se devuelve por referencia y no es modificable.

Representación

Representación de la tabla

La tabla se representa con una estructura que incluye:

- un string que corresponde al campo principal de la tabla.
- un conjunto de los registros incluidos en la tabla.
- un diccionario sobre Trie de valores del campo clave (que identifican unívocamente a los registros) y registros (a los que se accede mediante un iterador al conjunto de registros de la estructura).
- un diccionario sobre Trie de campos, cuyos significados son diccionarios lineales de iterador a registro y valor (el valor asociado al campo (clave del diccionario Trie) en dicho registro). La inclusión de este diccionario tiene como objetivo recorrer las columnas de la tabla sin que sea necesario iterar sobre el conjunto de campos.

`tabla` se representa con `tab`

donde `tab` es `tupla`(*campoClave*: `nombre_campo`, *registros*: `conj(registro)`, *diccValorClave*: `diccTrie(valor_clave, itConj(registro))`, *diccColumnas*: `diccTrie(nombre_campo, diccLineal(itConj(registro), valor))`)

Invariante de representación

1. Todos los registros del conjunto *registros* tienen los mismos campos, uno de los cuales es el campo clave.
2. La cantidad de registros del conjunto *registros* es igual a la cantidad de valores clave definidos en *diccValorClave* y el valor clave de cada entrada coincide con el valor del campo clave que tiene el registro al que apunta el iterador en *registros*.

3. El conjunto de claves del *diccColumnas* debe ser igual al conjunto de campos de los registros de *registros*.

4. La cantidad de claves de cada diccionario lineal (significados de *diccColumnas*) debe ser igual a la cantidad de registros en *registros*. Además, para cada iterador a registro definido en cada uno de esos diccionarios, el valor asociado debe ser igual al valor que tiene ese registro en el campo correspondiente (es decir, el campo que es la clave asociada a ese diccionario lineal) en *registros*.

Función de abstracción

1. El campo clave de la tabla será el campo contenido en el elemento *campoClave* de la estructura.

2. Sus campos serán los campos contenidos en el conjunto de claves de *diccColumnas*.

3. Tendrá todos los registros de *registros* y no tendrá ninguno que no esté en dicho conjunto.

Algoritmos

---

iNueva(**in** cs : `conj(nombre_campo)`, **in** c : `nombre_campo`)) $\rightarrow$ res : tab
  $res \leftarrow \langle c$, Vacío(), Vacío(), Vacío()$\rangle$
  $it \leftarrow$ CrearIt(cs)
  while (HaySiguiente(it)) **do**
    Definir($t.diccColumnas$, Siguiente(it), DiccLineal(itConj(registro), valor)::Nuevo())
    Avanzar(it)
  end while

  Complejidad: $\Theta(|c|)$
  Justificación: la cantidad de campos se considera constante.

---

iInsertar(**in/out** t : `tab`, **in** r : `registro`)
  $vc \leftarrow$ r[$t.campoClave$]
  if Definido?($t.diccValorClave$, vc) **then**
    $itReg \leftarrow$ Significado($t.diccValorClave$, vc)
    $itCampos \leftarrow$ CrearIt(Campos(r))
    while HaySiguiente($itCampos$) **do**
      Borrar(Significado($t.diccColumnas$, Siguiente($itCampos$)), $itReg$)
      Avanzar($itCampos$)
    end while
    Borrar($t.diccValorClave$, vc)
    EliminarSiguiente($itReg$)
  end if
  $itNuevoReg \leftarrow$ AgregarRapido($t.registros$, r)
  Definir($t.diccValorClave$, vc, Siguiente($itNuevoReg$))
  $itCampos \leftarrow$ CrearIt(Campos(r))
  while HaySiguiente($itCampos$) **do**
    Definir(Significado($t.diccColumnas$, Siguiente($itCampos$)), itNuevoReg, r[Siguiente($itCampos$)])
    Avanzar($itCampos$)
  end while

  Complejidad: $\Theta(|v| + |c| + n + size(r))$
  Justificación: el peor caso se da cuando ya existe un registro con el mismo valor clave en la tabla. En ese caso, n es la cantidad de registros de la tabla (por la iteración en el diccionario lineal dentro de *diccColumnas*) y $size$(r) corresponde al costo de borrar el registro liberando la memoria. Si no, el costo de insertar un registro es $\Theta(|v| + |c|)$.

---

iBorrar(**in/out** t : `tab`, **in** v : `valor`)

$itReg \leftarrow$ Significado($t.diccValorClave$, v)
$itCampos \leftarrow$ CrearIt(Campos(r))
while HaySiguiente($itCampos$) **do**
  Borrar(Significado($t.diccColumnas$, Siguiente($itCampos$)), $itReg$)
  Avanzar($itCampos$)
end while
Borrar($t.diccValorClave$, v)
EliminarSiguiente($itReg$)

Complejidad: $\Theta(|v| + |c| + n + size(r))$
Justificación: n es la cantidad de registros de la tabla (por la iteración en el diccionario lineal dentro de $diccColumnas$) y $size$(r) corresponde al costo de borrar el registro liberando la memoria.

---

iCampos(**in** t : `tab`) $\rightarrow$ res : conj(nombre_campo)

$res \leftarrow$ Claves($t.diccColumnas$)

Complejidad: $\Theta(1)$
Justificación: La operación $Claves$ del módulo Diccionario sobre Trie tiene complejidad constante en peor caso.

---

iClave(**in** t : `tab`) $\rightarrow$ res : nombre_campo

$res \leftarrow t.campoClave$

Complejidad: $\Theta(1)$
Justificación: Se devuelve el nombre del campo clave de la tabla por referencia.

---

iRegistros(**in** t : `tab`) $\rightarrow$ res : conj(registro)

res $\leftarrow t.registros$

Complejidad: $\Theta(1)$
Justificación: Se devuelve el conjunto por referencia.

---

iExisteRegConClave(**in** v : `valor`, **in** t : `tab`) $\rightarrow$ res : bool

$res \leftarrow$ Definido?($t.diccValorClave$, v)

Complejidad: $\Theta(|v|)$
Justificación: La operación $Definido?$ del módulo Diccionario sobre Trie tiene complejidad constante en peor caso.

---

iRegPorClave(**in** v : `valor`, **in** t : `tab`) $\rightarrow$ res : registro

$it \leftarrow$ Significado($t.diccValorClave$, v)
$res \leftarrow$ Siguiente(it)

Complejidad: $\Theta(|v|)$
Justificación: Se devuelve el registro cuyo campo clave tiene el valor v por referencia.

---

iValoresClave(**in** t : `tab`) $\rightarrow$ res : conj(valor)

$res \leftarrow$ Claves($t.diccValorClave$, v)

Complejidad: $\Theta(1)$
Justificación: La operación $Claves$ del módulo Diccionario sobre Trie tiene complejidad constante en peor caso.

---

iObtenerColumna(**in** c : `nombre_campo`, **in** t : `tab`) $\rightarrow$ res : dicc(itConj(registro), valor)

$res \leftarrow$ Significado($t.diccColumnas$, c)

Complejidad: $\Theta(|c|)$

Justificación: La operación $Significado$ del módulo Diccionario sobre Trie se realiza en tiempo lineal en relación a la longitud de la clave.