1. Módulo Matriz

Interfaz

parámetros formales
género $significado$
función COPIAR(**in** a : $significado$) $\rightarrow$ res : $significado$
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} a$}
Complejidad: $\Theta(copy(a))$
Descripción: función de copia de $significado$

se explica con: DICCIONARIO(POS, SIGNIFICADO)

géneros: `matriz`

El modulo funciona como un diccionario, pero solo se utiliza con claves del tipo *pos*. Extiende el TAD para contemplar que la creacion de una nueva matriz requiere dos parametros, el *alto* y el *ancho*.

Operaciones básicas de matriz

NUEVAMATRIZ(**in** al : `nat`, **in** an : `nat`) $\rightarrow$ res : `matriz`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ vacio}
Complejidad: $O(n * m)$
Descripción: Crea una nueva matriz de alto al x ancho an.

DEFINIR(**in** p: `pos`, **in** s: `significado`, **in/out** m: `matriz`)
Pre $\equiv$ {$m =_{\text{obs}} m_0$}
Post $\equiv$ {$m =_{\text{obs}}$ definir(p, s, m_0)}
Complejidad: $O(1)$
Descripción: Define el significado s en la posicion p de la matriz m.

DEF?(**in** p: `pos`, **in** m: `matriz`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ def?(p, m)}
Complejidad: $O(1)$
Descripción: Devuelve *true* si la posicion p esta ocupada.

OBTENER(**in** p: `pos`, **in** m: `matriz`) $\rightarrow$ res : `significado`
Pre $\equiv$ {def?(p, m)}
Post $\equiv$ {$res =_{\text{obs}}$ obtener(p, m)}
Complejidad: $O(1)$
Descripción: Retorna el $significado$ almacenado en la posicion p.

ELIMINAR(**in** p: `pos`, **in/out** m: `matriz`)
Pre $\equiv$ {def?$(p, m) \wedge m =_{\text{obs}} m_0$}
Post $\equiv$ {$m =_{\text{obs}}$ borrar(p, m_0)}
Complejidad: $O(n)$
Descripción: Elimina el contenido de la posicion p de la matriz m.

CLAVES(**in** m: `matriz`) $\rightarrow$ res : `conj(pos)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ claves(m)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de posiciones ocupadas en la matriz m

Representación

Representación de Matriz

`matriz` se representa con `estr`

donde `estr` es `tupla`(*alto*: `nat` , *ancho*: `nat` , *claves*: `conj(pos)`, *tablero*: `vector(vector(info))`)
donde `info` es `tupla`(*definido*: `bool`, *dato*: `significado`, *it*: `itConj(pos)`)

Utilizamos un definido de tipo bool para los algoritmos y un dato de tipo significado para admitir que se utilice este módulo como un diccionario de posiciones con otros significado; más allá de que en campus se utiliza unicamente para significados de tipo bool. El .it se utiliza para poder acceder y eliminar en O(1) una clave de .claves.

donde `pos` es `tupla`(*fila*: `nat`, *columna*: `nat`)

Invariante de representacion en castellano:

1. La longitud de tablero es *alto*
2. Toda clave contenida en .claves esta en el rango de la matriz.
3. Para toda posicion de tablero, el vector que contiene posee longitud *ancho*
4. Para toda clave p en el rango de la matriz, p contenida en *claves* implica que las componentes de c (.fila, .columna) en *tablero* dan una tupla $info$ donde .definido es *true*.
5. Analogo al anterior, pero para toda p que este en el rango de la matriz y no este contenida en *claves*, la tupla $info$ posee .definido igual a $false$

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

1. Longitud(tablero) $=_{\text{obs}}$ e.alto $\wedge$
2. ($\forall$ p : pos) En(p, e.claves) $\Rightarrow_{\text{L}}$ p.fila $\geq 0 \wedge$ p.fila $<$e.alto $\wedge$ p.col $\geq 0 \wedge$ p.col $<$e.ancho $\wedge$
3. ($\forall$ i : int) ($i <$Longitud(e.tablero)) $\Rightarrow_{\text{L}}$ Longitud(e.tablero[i]) $=_{\text{obs}}$ e.ancho $\wedge_{\text{L}}$
4. ($\forall$ p : pos) (p.fila $\leq$ e.alto $\wedge$ p.columna $\leq$ e.ancho $\wedge$ $p \in$ e.claves) $\Rightarrow_{\text{L}}$
   (e.tablero[p.fila][p.columna].definido $=_{\text{obs}}$ $true$)
5. ($\forall$ p : pos) (p.fila $\leq$ e.alto $\wedge$ p.columna $\leq$ e.ancho $\wedge$ $p \notin$ e.claves) $\Rightarrow_{\text{L}}$
   (e.tablero[p.fila][p.columna].definido $=_{\text{obs}}$ $false$)

Abs : estr e $\longrightarrow$ dicc($pos, significado$) {Rep(e)}
Abs(e) $\equiv$ m : dicc($pos, significado$) /
($\forall$ p : pos) En?(p, e.claves) $=_{\text{obs}}$ def?(p, m) $\wedge$
($\forall$ p : pos) En?(p, e.claves) $\Rightarrow_{\text{L}}$ Π_2(e.tablero[p.fila][p.col]) $=_{\text{obs}}$ obtener(p, m))

Las claves definidas y sus significados son iguales. La segunda parte indica que para toda clave definida, tablero en esa posición posee un info donde la segunda componente (.dato) es igual al significado del diccionario resultante.

Algoritmos

Lista de algoritmos

```
iNuevaMatriz(in al : nat, in an : nat) → res: estr
begin
    res.alto ← al                                                    //O(1)
    res.ancho ← an                                                   //O(1)
    res.claves ← Vacio()                                             //O(1)
    res.tablero ← CrearArreglo(al)                                   //O(al)
    for i ← 0..(al − 1) do                                           //O(al)
        res.tablero[i] ← CrearArreglo(an)                            //O(an)
    end
end
```

Complejidad: $O(al * an)$

Algoritmo 1: NuevaMatriz

```
iDefinir(in p : pos, in s : significado, in/out e : estr)
begin
    if p.fila <e.alto ∨ p.columna <e.ancho then                      //O(1)
        if ¬ Π1(e.tablero[p.fila][p.columna]) then                   //O(1)
            var temp: itConj
            temp ← AgregarRapido(p, e.claves)                        //O(1)
            e.tablero[p.fila][p.columna] ← <true, s, temp>           //O(1)
            else
                var temp: itConj
                temp ← Π3(e.tablero[p.fila][p.columna])              //O(1)
                e.tablero[p.fila][p.columna] ← <true, s, temp>       //O(1)
            end
        end
    end
end
```

Complejidad: O(1)

Comentarios: El algoritmo descarta las ejecuciones con posiciones no válidas en el primer If. Luego, utilizando el .definido de la tupla $info$, puede saber en O(1) si el elemento se encuentra anteriormente definido. Si no está definido, lo define de forma simple, utilizando el agregado rápido del módulo basico conjunto y utilizando el iterador que devuelve para agregarlo en la tupla. Si está definido, tal y como un diccionario convencional lo actualiza. Para eso, tiene en cuenta que es necesario actualizar solo el significado, por ende, en vez de tener un iterador nuevo, lo obtiene de la definición actual.

Algoritmo 2: Definir

```
iDef?(in p : pos, in e : estr) → res: bool
begin
    if p.fila >e.alto ∨ p.columna >e.ancho then                      //O(1)
        res ← false                                                  //O(1)
        else
            res ← Π1(e.tablero[p.fila][p.columna])                   //O(1)
        end
    end
    return res
end
```

Complejidad: O(1)

Algoritmo 3: Def?

iObtener(**in** p: `pos`, **in** e: `estr`) → res: significado
begin
  res ← Π_2(e.tablero[p.fila][p.columna]) //**O(1)**
  return *res*
end
Complejidad: O(1)

Algoritmo 4: Obtener

---

iEliminar(**in** p: `pos`, **in/out** e: `estr`)
begin
  var temp: itConj
  temp ← Π_3(e.tablero[p.fila][p.columna]) //**O(1)**
  EliminarSiguiente(temp) //**O(1)**
  Π_1(e.tablero[p.alto][p.columna]) ← $false$ //**O(1)**
end
Complejidad: O(1)
Comentarios: La función eliminar hace uso de la precondición, que dice que p está definido y el iterador que guardamos en la estructura, para poder eliminar la clave de .claves en O(1). Para ello, primero obtiene el iterador, elimina el elemento y después procede a setear la tupla como no definida.

Algoritmo 5: Eliminar

---

iClaves(**in** e: `estr`) → res: conj(pos)
begin
  res ← e.claves //**O(1)**
  return *res*
end
Complejidad: O(1)

Algoritmo 6: Claves
