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.

COLOCAR(**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: Coloca ($define$) el significado s en la posicion p de la matriz m.

OCUPADA?(**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: $O(1)$
Descripción: Elimina el contenido de la posicion p de la matriz m.

POSICIONESOCUPADAS(**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*: `conjRapido(pos)`, *tablero*: `vector(vector(info))`)
donde `info` es `tupla`(*definido*: `bool`, *dato*: `significado`)
donde `pos` es `tupla`(*fila*: `nat`, *columna*: `nat`)

Invariante de representacion en castellano:

1. La longitud de tablero es *alto*
2. Para toda posicion de tablero, el vector que contiene posee longitud *ancho*
3. 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*.
4. 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) $=_{\text{obs}}$ alto $\wedge$
2. ($\forall$ i : int) (i <Longitud(tablero)) $\Rightarrow_{\text{L}}$ Longitud(tablero[i]) $=_{\text{obs}}$ ancho $\wedge_{\text{L}}$
3. ($\forall$ p : pos) (p.fila $\leq$ alto $\wedge$ p.columna $\leq$ ancho $\wedge$ $p \in$ e.claves) $\Rightarrow_{\text{L}}$
   (talbero[p.fila][p.columna].definido $=_{\text{obs}}$ $true$)
4. ($\forall$ p : pos) (p.fila $\leq$ alto $\wedge$ p.columna $\leq$ ancho $\wedge$ $p \notin$ e.claves) $\Rightarrow_{\text{L}}$
   (talbero[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) def?(p, e.claves) $=_{\text{obs}}$ def?(p, m) $\wedge$
($\forall$ p : pos) def?(p, e.claves) $\Rightarrow_{\text{L}}$ Π_2(obtener(p, e)) $=_{\text{obs}}$ obtener(p, m))
Las claves definidas y sus significados son iguales

Algoritmos

Lista de algoritmos

1. nombre 3

*i*NuevoCampus(**in** *al* : `nat`, **in** *an* : `nat`) → res: estr
begin
  res.alto ← al **//O(1)**
  res.ancho ← an **//O(1)**
  res.obstaculos ← vacio() **//O(al * an)**
end
Complejidad: $O(al * an)$

Algoritmo 1: nombre