1. Modulo Campus Seguro

Interfaz

se explica con: Campus Seguro

generos: `campseg`

Operaciones basicas de Campus Seguro

NuevoCampus(**in** C: `campus`, **in** dA: `dicc(Ag, pos)`) $\rightarrow$ res : `campseg`
Pre $\equiv$ {($\forall$ a $\in$ Agentes) (def?(a,dA) $\Rightarrow_{\text{L}}$ (PosValida?(obtener(a,dA)) $\wedge$ $\neg$ ocupada?(obtener(a,dA),c) $\wedge$ ($\forall$ a_0 , a_1 $\in$ Agente) (def?a_0,dA) $\wedge$ def?(a_1,dA)$\Rightarrow_{\text{L}}$ obtener((a_0,dA)) != obtener(a_1,dA))}
Post $\equiv$ {res $=_{\text{obs}}$ comenzarRastrillaje(c,dA)}
Complejidad: O(Na + (alto * ancho))
Descripción: Crea un nuevo campus seguro con un campus y un diccionario de agentes, posicion

ingresarEstudiante(**in** e: `nombre`,**in** p: `pos` , **in/out** c: `campseg`)
Pre $\equiv$ {$\neg$ (e $\in$ (estudiantes(c) U hippies(c))) $\wedge$ esIngreso?(p, $campus(c)$) $\wedge$ $\neg$ estaOcupada?(p,c) $\wedge$ $c =_{\text{obs}} c_0$ }
Post $\equiv$ {$c =_{\text{obs}}$ ingresarEstudiante(e, p, c_0)}
Complejidad: O(|n$_m$|)
Descripción: Ingresa un estudiante en una posicion valida

ingresarHippie(**in** e: `nombre`,**in** p: `pos` , **in/out** c: `campseg`)
Pre $\equiv$ {$\neg$ (e $\in$ (estudiantes(c) U hippies(c))) $\wedge$ esIngreso?(p, $campus(c)$) $\wedge$ $\neg$ estaOcupada?(p,c) $\wedge$ $c =_{\text{obs}} c_0$ }
Post $\equiv$ {$c =_{\text{obs}}$ ingresarHippie(e, p, c_0)}
Complejidad: O(|n$_m$|) Ingresa un hippie en una posicion valida

moverEstudiantes(**in** e: `nombre`,**in** d: `dir`,**in/out** c: `campseg`)
Pre $\equiv$ {e $\in$ (estudiantes(c) $\wedge$ (seRetira(e,d,c) $\vee$
posValida?(proxPosicion(posHippieYEstudiante(e,c),d,campus(c)),campus(c)) $\wedge$
$\neg$ estaOcupada?(proxPosicion(posHippieYEstudiante(e,c),d,campus(c)),campus(c),c) $\wedge$ $c =_{\text{obs}} c_0$ }
Post $\equiv$ {$c =_{\text{obs}}$ moverEstudiante(e,d,c_0)}
Complejidad: O(|n$_m$|)
Descripción: Muevo un estudiante!

moverHippie(**in** h: `nombre`,**in/out** c: `campseg`)
Pre $\equiv$ {h $\in$ (hippie(c) $\wedge$ $\neg$ todasOcupadas?(vecinos(posEstudianteHippie(h,c),campus(c),c)) $\wedge$ $c =_{\text{obs}} c_0$}
Post $\equiv$ {$c =_{\text{obs}}$ moverHippie(h,c_0}
Complejidad: O((cant(estudiantes)) + |n$_m$|)
Descripción: Muevo un hippie

moverAgente(**in** a: `agente`,**in/out** c: `campseg`)
Pre $\equiv$ {a $\in$ (agentes(c) $\Rightarrow_{\text{L}}$ cantSanciones(a,c) $\leq$ 3 $\wedge$ $\neg$ todasOcupadas?(vecinos(posAgente(a,c),campus(c),c)) $\wedge$ $c =_{\text{obs}} c_0$}
Post $\equiv$ {$c =_{\text{obs}}$ moverAgente(a, c_0)}
Complejidad: O(cant(Hip) + |n$_m$|)
Descripción: Muevo un agente que no esta con mas de 3 sanciones

campus(**in** c: `campseg`) $\rightarrow$ res : `campus`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ campus(c)}
Complejidad: $\Theta(1)$
Descripción: Devuelvo una referencia a campus

estudiantes(**in** c: `campseg`) $\rightarrow$ res : `Itconj`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearIT(estudiantes(c))}
Complejidad: $\Theta(1)$
Descripción: Devuelvo un iterador a conjunto.

hippies(**in** c: `campseg`) $\rightarrow$ res : `Itconj`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearIT(hippies(c))}
Complejidad: $\Theta(1)$

Descripción: Devuelvo un iterador a conjunto.

AGENTES(**in** c: `campseg`) $\rightarrow$ res : `Itconj`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearIT(agentes(c))}
Complejidad: $\Theta(1)$
Descripción: Devuelvo un iterador a conjunto

POSHIPPIEYESTUDIANTES(**in** n: `nombre`,**in** c: `campseg`) $\rightarrow$ res : `pos`
Pre $\equiv$ {(n $\in$ (estudiantes(c) U hippies(c))}
Post $\equiv$ {$res =_{\text{obs}}$ posHippieYEstudiantes(n,c)}
Complejidad: $\Theta(longituddelinput)$
Descripción: Pregunto por la posicion de un estudiante o hippie

POSAGENTE(**in** a: `agente`,**in** c: `campseg`) $\rightarrow$ res : `pos`
Pre $\equiv$ {(a $\in$ agentes(c)}
Post $\equiv$ {$res =_{\text{obs}}$ posAgente(a,c)}
Complejidad: $\Theta(1)$ en caso promedio
Descripción: Pregunto por la posicion de un agente

CANTSANCIONES(**in** a: `agente`,**in** c: `campseg`) $\rightarrow$ res : `nat`
Pre $\equiv$ {(a $\in$ agentes(c)}
Post $\equiv$ {$res =_{\text{obs}}$ cantSanciones(a,c)}
Complejidad: $\Theta(1)$ en caso promedio
Descripción: Pregunto por la cant de sanciones de un agente

CANTHIPPIESATRAPADOS(**in** a: `agente`,**in** c: `campseg`) $\rightarrow$ res : `nat`
Pre $\equiv$ {(a $\in$ agentes(c)}
Post $\equiv$ {$res =_{\text{obs}}$ cantHippiesAtrapados(a,c)}
Complejidad: $\Theta(1)$ en caso promedio
Descripción: Pregunto por la cant de hippies atrapados de un agente

CANTHIPPIES(**in** c: `campseg`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ cantHippies(c)}
Complejidad: O(1)
Descripción: Pregunto por la cant de hippies

CANTESTUDIANTES(**in** c: `campseg`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ cantEstudiantes(c)}
Complejidad: O(1)
Descripción: Pregunto por la cant de estudiantes

MASVIGILANTE(**in** c: `campseg`) $\rightarrow$ res : `agente`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ masVigilante(c)}
Complejidad: O(1)
Descripción: Pregunto por la cant de estudiantes

CONMISMASSANCIONES(**in** a: `agente`,**in** c: `campseg`) $\rightarrow$ res : `conj(agentes)`
Pre $\equiv$ {(a $\in$ agentes(c)}
Post $\equiv$ {$res =_{\text{obs}}$ conMismasSanciones(c)}
Complejidad: O(1) en caso promedio
Descripción: Devuelve al conj de agentes con la misma cantidad de sanciones

CONKSANCIONES(**in** k: `nat`,**in** c: `campseg`) $\rightarrow$ res : `conj(agentes)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ conKSanciones(c)}
Complejidad: O(cant Agentes) la primera vez que se llama y O(log(cant Agentes) los otras veces que se llama mientras no ocurran sanciones)

Descripción: Devuelve al conj de agentes con k sanciones

Representación

Representacion del Campus Seguro

`Campus Seguro` **se representa con** `estr`

donde `estr` es `tupla`(*campus*: `Campus`, *HipYEst*: `HippiesYEstudiantes`, *agentes*: `Agentes`, *posOcupadasHippies*: `Matriz(nombre)`, *posOcupadasEstudiantes*: `Matriz(nombre)`, *posOcupadasAgentes*: `Matriz(placa)`)

// Invariante de representacion en castellano:

1. Una posicion en posOcupada no puede estar ocupada por el resto de las matrices de posOcupadas al mismo tiempo (En otras palabras la interseccion del conj de pos definidas en posOcupadasHippies, posOcupadasEstudiantes y posOcupadasAgentes es vacia.
2. La dimension de las matrices posOcupadasHippies/Estudiantes/Agentes tiene que coincidir con el *alto* y *ancho* del *campus*.
3. Para todo nombre que este definido en HipYEst la posicion de ese nombre tiene que estar definida en posOcupadasHippies ó(excluyente) en posOcupadasEstudiantes.
4. Para toda placa que este definida en Agentes la posicion de esa placa tiene que estar definida en posOcupadasAgentes.
5. Para toda pos definida en posOcupadasEstudiantes o posOcupadasHippies existe un nombre tal que si esta definido darme la pos de ese nombre es igual a la primera
6. Para toda pos definida en posOcupadasAgentes Existe una placa tal que posAgente de esa placa es igual a la pos definida en posOcupas(es decir todas las pos en poscupadasAgentes tienen un placa asociada en Agentes).
7. Los obstaculos del campus no pueden estar definidos en posOcupadasHippie/Estudiante/Agente.
8. No existe una pos definida en posOcupadasHippie tal que los vecinos de esa pos esten todos definidos en posOcupadaEstudiante.
9. No existe una pos definida en posOcupadasHippie tal que que este bloqueada por al menos un agente.

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$

1. (($\forall$ p : pos) def?(p,e.posOcupadaHippie) $\Rightarrow$ $\neg$ def?(p,e.posOcupadaEstudiante) $\wedge$ $\neg$ def?(p,e.posOcupadaAgente)) $\wedge$ (($\forall$ p : pos) def?(p,e.posOcupadaEstudiante) $\Rightarrow$ $\neg$ def?(p,e.posOcupadaHippie) $\wedge$ $\neg$ def?(p,e.posOcupadaAgente)) (($\forall$ p : pos) def?(p,e.posOcupadaAgente) $\Rightarrow$ $\neg$ def?(p,e.posOcupadaEstudiante) $\wedge$ $\neg$ def?(p,e.posOcupadaHippie))
2. Alto(*campus*) $=_{\text{obs}}$ alto(e.posOcupadasHippie) $\wedge$ Alto(e.*campus*) $=_{\text{obs}}$ alto(e.posOcupadasEstudiante) $\wedge$ Alto(e.*campus*) $=_{\text{obs}}$ alto(e.posOcupadasAgente) $\wedge$ Ancho(e.*campus*) $=_{\text{obs}}$ ancho(e.posOcupadasHippie) $\wedge$ Ancho(e.*campus*) $=_{\text{obs}}$ ancho(e.posOcupadasEstudiante) $\wedge$ Ancho(e.*campus*) $=_{\text{obs}}$ ancho(e.posOcupadasAgente)
3. ($\forall$ n : nombre) def?(n,e.HipYEst) $\Rightarrow_{\text{L}}$ def?(posHippieYEstudiante(obtener(n,e.HipYEst)),e.posOcupadaHip $\oplus$ def?(posHippieYEstudiante(obtener(n,e.HipYEst)),e.posOcupadaEstudiante)
4. ($\forall$ pl : placa) def?(pl,e.agentes) $\Rightarrow_{\text{L}}$ def?(posAgente(obtener(n,e.agente)),e.posOcupadaAgente)
5. ($\forall$ p : pos) (def?(p,e.posOcupadaHippie) $\vee$ def?(p,e.posOcupadaEstudiante)) $\Rightarrow$ (($\exists$ n : nombre) def?(n,e.HipYEst) $\Rightarrow_{\text{L}}$ posEstudianteHippie(obtener(n,e,e.HipYEst)) = p)
6. ($\forall$ p : pos) (def?(p,e.posOcupadaAgente) $\Rightarrow$ (($\exists$ pl : placa) def?(pl,e.agentes) $\Rightarrow_{\text{L}}$ posAgente(obtener(pl,e.agentes)) = p)
7. ($\forall$ p : pos) p $\in$ obstaculos(e.campus) $\Rightarrow$ $\neg$ def?(p,e.posOcupadasHippie) $\wedge$ $\neg$ def?(p,e.posOcupadasEstudiante) $\wedge$ $\neg$ def?(p,e.posOcupadasAgente)
8. ($\forall$ p : pos) def?(p,e.posOcupadasHippie) $\Rightarrow_{\text{L}}$ $\neg$ (($\forall$ po : pos) vecinos(p,e.campus) $\Rightarrow$ def?(po,e.posOcupadasEstudiante))
9. ($\forall$ p : pos) def?(p,e.posOcupadasHippie) $\Rightarrow_{\text{L}}$ $\neg$ ((($\forall$ po : pos) po $\in$ vecinos(p,e.campus) $\wedge$ estaOcuapada(po,e)) $\Rightarrow$ ($\exists$ p1: pos) def?(p1,e.posOcupadasAgentes) $\wedge$ p1 = p0))

Abs : estr e $\longrightarrow$ campusSeguro {Rep(e)}

Abs(e) $\equiv$ c : campusSeguro /
e.campus $=_{\text{obs}}$ campus(c) $\wedge$
Estudiantes(e.HipYEst) $=_{\text{obs}}$ estudiantes(c) $\wedge$
Hippies(e.HipYEst) $=_{\text{obs}}$ hippies(c) $\wedge$
Agentes(e.agentes) $=_{\text{obs}}$ agentes(c) $\wedge$
($\forall$ n : nombre) (Esta?(n, e.HipYEst)) $\Rightarrow_{\text{L}}$ (posHippieYEstudiante(n, c) $=_{\text{obs}}$ posHippieYEstudiante(n, e.HipYEst)) $\wedge$
($\forall$ a : placa) (a $\in$ Agentes?(e.agentes)) $\Rightarrow_{\text{L}}$ (
(PosAgente(a, e.agentes) $=_{\text{obs}}$ posAgente(a, c)) $\wedge$
(SancionesAgente(a, e.agentes) $=_{\text{obs}}$ cantSanciones(a, c)) $\wedge$
(CapturasAgente(a, e.agentes) $=_{\text{obs}}$ cantHippiesAtrapados(a, c)))

Algoritmos **Algoritmos de Campus Seguro**

Lista de algoritmos

iNuevoCampus(**in** c: `campus`, **in** dA: `dicc(agente,pos)`) → res: estr
begin
    res.campus ← c //**O(1)**
    res.agentes ← nuevoAgentes(dA) //**O(nA)**
    matrizAgentes ← nuevaMatriz(c.alto,c.ancho) //**O(c.alto *c.ancho)**
    ITagentes ← crearIT(agentes?(res.agentes)) //**O(1)**
    while *haymas(ITagentes)* **do** //**O(1)**
        definir(actual(ITagentes),posAgentes(actual(ITagentes),res.agentes),matrizAgentes) //**O(1)** + **O(1)(caso Promedio)**
        siguiente(ITagente) //**O(1)**
    end
    res.PosOcupadasAgente ← matrizAgente //**O(1)**
    res.PosOcupadasHippie ← crearMatriz(c.alto,c.ancho) //**O(c.alto *c.ancho))**
    res.PosOcupadasEstudiante ← crearMatriz(c.alto,c.ancho) //**O(c.alto *c.ancho)**
    res.HipYEst ← nuevoHipYEst() //**O(1)**
end
Complejidad: O(nA)+ 3 * O(alto * ancho) = O(nA+ alto* ancho)) nA= Cantiadd de agentes y Alto y ancho las dimensiones del campus.

Algoritmo 1: NuevoCampus

```
iingresarEstudiante(in n: nombre,in p: pos, in/out e: estr)
begin
    if iEstudianteHippieficado(p,n) then                                          //O(1)
        definirHippie(n,p,e.HipYEst)                                              //O(|n_m|)
        definir(p,n,e.posOcupadasHippie)                                          //O(1)
        cjtoVec ← vecinos(p,e.campus)                                             //O(1)
        Itvec ← crearIt(cjtoVec)                                                  //O(1)
        while haymas(Itvec) do                                                    //O(1)
            if def?(actual(Itvec),e.posOcupadasHippie) then                       //O(1)
                if hippieCapturado(actual(Itvec),e)) then                         //O(1)
                    borrar(obtener(actual(Itvec),e.posOcupadasHippie),e.HipYEst)  //O(|n_m|)
                end
            end
            if def?(actual(Itvec),e.posOcupadasEstudiante) then                   //O(1))
                if estudianteHippieficado(actual(Itvec),e)) then                  //O(1)
                    hippificar(actual(Itvec),e)                                   //O(|n_m|))
                end
                if estudianteSancionar(acltual(Itvec),e) then                     //O(1)
                    sancionar(actual(Itvec),e)                                    //O(1) en caso promedio
                end
            end
            siguiente(ItVec)                                                      //O(1)
        end
    else
        definirEstudiante(e,p,e.HipYEst)                                          //O(|n_m|)
        definir(p,e,e.posOcupadasEstudiante)                                      //O(1)
        cjtoVec ← vecinos(p,e.campus)                                             //O(1)
        Itvec ← crearIt(cjtoVec)                                                  //O(1)
        while haymas(Itvec) do                                                    //O(1)
            if def?(actual(Itvec),e.posOcupadasHippie)) then                      //O(1))
                if hippieRodeadoEst(actual(Itvec),e) then                         //O(1)
                    transformarHippie(actual(Itvec),e)                            //O(|n_m|)
                end
                if hippieCapturado(actual(Itvec),e) then                          //O(1)
                    eliminarHippie(actual(Itvec),e)                               //O(|n_m|)
                end
            end
            if def?(actual(Itvec),e.posOcupadasEstudiante)) then                  //O(1))
                if estudianteSancionar(actual(Itvec),e) then                      //O(1)
                    sancionar(actual(Itvec),e)                                    //O(1) En caso promedio!
                end
            end
            siguiente(Itvec)                                                      //O(1)
        end
    end
    if estudianteSancionar(p,e) then                                              //O(1)
        cjtoVec ← vecinos(p,cs.campus)                                            //O(1)
        Itvec ← crearIt(cjtoVec)                                                  //O(1)
        while haymas(Itvec) do                                                    //O(1)
            if def?(actual(Itvec),e.posOcupadasAgente)) then                      //O(1))
                agregarSancion(obtener((actual(Itvec),e),e.posOcupadasAgente),e.Agente)   //O(1) En caso
                promedio!
            end
            siguiente(Itvec)                                                      //O(1)
        end
    end
end
```

Complejidad: Teniendo en cuenta el caso promedio que involucra todas las operaciones con agentes la complejidad que me queda es entrando por cualquier rama del if en el peor caso 3*$|n_m|$ = O($|n_m|$)

Algoritmo 2: IngresarEstudiante

```
iingresarHippie(in h : nombre,in p : pos, in/out e : estr)
begin
    if hippieCapturado(p,cs) then                                                    //O(1)
        sumarCapturas(p,e)                                                  //O(1) En caso promedio
    end
    if hippieRodeadoEst(p,e) then                                                    //O(1)
        definirEstudiante(h,p,e.HipYEst)                                          //O(|n_m|)
        definir(p,h,e.posOcupadasEstudiante)                                         //O(1)
        cjtoVec ← vecinos(p,e.campus)                                                //O(1)
        Itvec ← crearIt(cjtoVec)                                                     //O(1)
        while hayMas(Itvec) do                                                       //O(1)
            if estudianteSancionar((actual(Itvec),e)) then                           //O(1)
                sancionar(actual(Itvec),e)                                  //O(1) En caso promedio
            end
            siguiente(Itvec)                                                         //O(1)
        end
    else
        definirHippie(h,p,e.HipYEst)                                              //O(|n_m|))
        definir(p,h,e.posOcupadasHippie)                                             //O(1)
        cjtoVec ← vecinos(p,e.campus)                                                //O(1)
        Itvec ← crearIt(cjtoVec)                                                     //O(1)
        while hayMas(Itvec) do                                                       //O(1)
            if def?(actual(Itvec),e.posOcupadasHippie)) then                         //O(1)
                if hippieCapturado(actual(Itvec),e) then                             //O(1)
                    eliminarHippie (actual(Itvec),e)                              //O(|n_m|))
                end
            end
            if def?(actual(Itvec),e.posOcupadasEstudiante)) then                     //O(1))
                if estudianteHippieficado(actual(Itvec),e) ∧ hippieCapturado(actual(Itvec),e) then    //O(1) +
                O(1)
                    sumarCapturas(actual(Itvec),e)                          //O(1) En caso Promedio
                    borrar(obtener((actual(Itvec),e),e.posOcupadasEstudiante),e.HipYEst)      //O(|n_m|)
                    borrar((actual(Itvec),e),e.posOcupadasEstudiante)                //O(1)
                end
                if estudianteHippieficado(actual(Itvec),e) then                      //O(1)
                    hippificar(actual(Itvec),e)                               //O(|n_m|))
                end
                if estudianteSancionar(actual(Itvec),e) then                         //O(1)
                    sancionar(actual(Itvec),e)                          //O(1) En caso Promedio!
                end
            end
            siguiente(Itvec)                                                         //O(1)
        end
    end
end
```

Complejidad: Teniendo en cuenta el caso promedio que involucra todas las operaciones con agentes la complejidad que me queda es de:
si defino como estudiante tengo O($|n_m|$) mas las posibles sanciones a los agentes pero en el peor caso si lo defino como hippie tengo que verificar todos mis vecinos para ver como los afecto si tengo que transformarlos a hippies o si provoco que un hippie sea eliminado por lo tanto tengo 3*$|n_m|$ mas el acceso a los agentes = O($|n_m|$)
Comentarios: Ingreso un hippie teniendo en cuenta todos los casos que pueden pasar.

Algoritmo 3: ingresarHippie

```
imoverEstudiante(in n: nombre,in d: dir, in/out e: estr)
begin
    pose ← posHippieYEstudiante(n,e)                                          //O(|n_m|)
    proxpos ← proxPosicion(pose,d)                                            //O(1)
    if meVoy(pose,d,e.campus) then                                            //O(1)
        borrar(n,e,HipYEst)                                                   //O(|n_m|)
        borrar(pose,e.PosOcupadasEstudiante)                                  //O(1)
    else
        muevoEstudiante(n,pose,dir,e)                                         //O(|n_m|)
        posNueva ← posHippieYEstudiante(n,e)                                  //O(|n_m|)
        if estudianteHippieficado(posNueva,e) then                            //O(1)
            hippificar(posNueva,e)                                            //O(|n_m|)
            cjtovec ← vecinos(posNueva,e)                                     //O(1)
            Itvec ← crearIT(cjtovec)                                          //O(1)
            while hayMas(Itvec) do                                            //O(1)
                if def?(actual(Itvec),e.posOcupadasHippie)) then              //O(1)
                    if hippieCapturado(actual(Itvec),e) then                  //O(1)
                        eliminarHippie(actual(Itvec),e)                       //O(|n_m|)
                    end
                end
                if def?(actual(Itvec),e.posOcupadasEstudiante)) then          //O(1)
                    if estudianteHippieficado(actual(Itvec),e) ∧ hippieCapturado(actual(Itvec),e) then   //O(1)
                    + O(1)
                        sumarCapturas(actual(Itvec),e)                        //O(1) En caso promedio
                        borrar(obtener((actual(Itvec),e),e.posOcupadasEstudiante),e.HipYEst)   //O(|n_m|)
                        borrar((actual(Itvec),e),e.posOcupadasEstudiante)     //O(1)
                    end
                    if estudianteSancionar(actual(Itvec),e) then              //O(1)
                        sancionar(actual(Itvec),e)                            //O(1) En caso promedio
                    end
                end
                siguiente(Itvec)                                              //O(1)
            end
        else
            cjtovec ← vecinos(posNueva,e)                                     //O(1)
            Itvec ← crearIT(cjtovec)                                          //O(1)
            while hayMas(Itvec) do                                            //O(1)
                if def?(actual(Itvec),e.posOcupadasHippie)) then              //O(1))
                    if hippieRodeadoEst(actual(Itvec),e) then                 //O(1)
                        transformarHippie((actual(Itvec),e)                   //O(|n_m|)
                    end
                    if hippieCapturado(actual(Itvec),e) then                  //O(1)
                        eliminarHippie(actual(Itvec),e)                       //O(|n_m|)
                    end
                end
                if def?(actual(Itvec),e.posOcupadasEstudiante)) then          //O(1)
                    if estudianteSancionar(actual(Itvec),e) then              //O(1)
                        sancionar(actual(Itvec),e)                            //O(1) En caso promedio
                    end
                end
                siguiente(ItVec)                                              //O(1)
            end
        end
    end
end
```

Complejidad: Teniendo en cuenta el caso promedio que involucra todas las operaciones con agentes la complejidad que me queda es de: En el peor de los casos si el estudiante que muevo no se va del campus y ademas se convierte en hippie tengo 2*$|n_m|$ + 4*$|n_m|$(de transformarlo en hippie mas que en el peor de los casos tengo que eliminar hippies) = 6* $|n_m|$ = O($|n_m|$) las otras ramas del if no contemplarian el peor caso.

Comentarios:

Algoritmo 4: moverEstudiante

```
imoverHippie(in h: nombre, in/out e: estr)
begin
    posh ← posHippieYEstudiante(h,e)                                                    //O(|n_m|))
    posDestino ← estMasCercano(posh,e)                                                  //O(cantestudiantes)
    muevoHippie(posh,posDestino,e)                                                      //O(|n_m|)
    posNueva ← posHippieYEstudiante(h,e)        //O(|n_m|) cjtovec ← vecinos(posNueva,e)   //O(1)
    Itvec ← crearIT(cjtovec)                                                            //O(1)
    while hayMas(Itvec) do                                                              //O(1)
        if def?(actual(Itvec),e.posOcupadasHippie)) then                                //O(1)
            if hippieCapturado(actual(Itvec),e) then                                    //O(1)
                eliminarHippie(actual(Itvec),e)                                         //O(|n_m|)
            end
        end
        if def?(actual(Itvec),e.posOcupadasEstudiante)) then                            //O(1))
            if estudianteHippieficado(actual(Itvec),e) ∧ hippieCapturado(actual(Itvec),e) then  //O(1) + O(1)
                sumarCapturas(actual(Itvec),e)                                  //O(1) En caso promedio
                borrarEstudiante(quineEs?((actual(Itvec),e),e.posOcupadasEstudiante),e.HipYEst)   //O(|n_m|)
                borrar((actual(Itvec),e),e.posOcupadasEstudiante)                       //O(1)
            end
            if estudianteSancionar(actual(Itvec),e) then                                //O(1)
                sancionar(actual(Itvec),e)                                      //O(1) En caso promedio
            end
        end
        siguiente(Itvec)                                                                //O(1)
    end
end
```

Complejidad: Teniendo en cuenta el caso promedio que involucra todas las operaciones con agentes la complejidad que me queda es de: ya un O(*cantestudiantes*) para saber a donde me tengo que mover o(k*|n_m|) con k cte, que involucra cambiar mi poscion y/o elimiar,modificar a mis vecinos por mi movimiento, entonces tengo O(cant est + k*|n_m|) sacando las cte me queda O(cant est + |n_m|)

Comentarios:

Algoritmo 5: moverHippie

```
imoverAgente(in a : agente, in/out e : estr)
begin
    posa ← posAgente(a,e.agente)                                          //O(1) En caso promedio
    posDestino ← hipMasCercano(posh,e)                                    //O(cant hippies)
    muevoAgente(posh,posDestino,e)                                        //O(1) En caso promedio
    posNueva ← posAgente(a,e.agente)  //O(1) En caso promedio cjtovec ← vecinos(posNueva,e)  //O(1)
    Itvec ← crearIT(cjtovec)                                              //O(1)
    while hayMas(Itvec) do                                                //O(1)
        if def?(actual(Itvec),e.posOcupadasHippie)) then                  //O(1)
            if hippieCapturado(actual(Itvec),e) then                      //O(1)
                eliminarHippie(actual(Itvec),e)                           //O(|n_m|)
            end
        end
        if def?(actual(Itvec),e.posOcupadasEstudiante)) then              //O(1)
            if estudianteSancionar(actual(Itvec),e) then                  //O(1)
                sancionar(actual(Itvec),e)                                //O(1) en caso promedio
            end
        end
        siguiente(Itvec)                                                  //O(1)
    end
end
```

Complejidad: Teniendo en cuenta el caso promedio que involucra todas las operaciones con agentes la complejidad que me queda es de: O(cant hip) (para saber a donde me tengo que mover) + 3*O(|n_m|) que involucra borrar a los hippies que encerre que a lo sumo pueden ser 3. Entonces la complejidad me queda sacando todas las constantes O((cant hip)+ |n_m|)

Comentarios:

Algoritmo 6: moverAgente

```
icampus(in e : estr) → c: &campus
begin
    res ← e.campus return res
end
```

Complejidad: $O(1)$

Comentarios: Devuelvo un campus por referencia

Algoritmo 7: campus

```
iestudiantes(in e : estr) → it: Itconj
begin
    res ← crearIt(*Estudiantes(e.HipYEst))                                //O(1)
    return res
end
```

Complejidad: $O(1)$

Comentarios: Devuelvo un iterador

Algoritmo 8: estudiantes

```
iagentes(in e : estr) → it: Itconj
begin
    res ← crearIt(*Agentes(e.agentes))                                    //O(1)
    return res
end
```

Complejidad: $O(1)$

Comentarios: Devuelvo un iterador

Algoritmo 9: agentes

---

*i*hippies(**in** e: `estr`) → it: Itconj
begin
  res ← crearIt(*Hippies(e.HipYEst)) //**O(1)**
  return *res*
end
Complejidad: $O(1)$
Comentarios: Devuelvo un iterador

Algoritmo 10: hippie

---

*i*posEstundianteYHippie(**in** n: `nombre`,**in** e: `estr`) → p: pos
begin
  res ← posEstundianteHippie(n,e.HipYEst) //**O($|n_m|$)**
  return *res*
end
Complejidad: $O(|n_m|)$
Comentarios: Devuelvo una poss

Algoritmo 11: posEstundianteYHippie

---

*i*posAgente(**in** a: `agente`,**in** e: `estr`) → p: pos
begin
  res ← posAgente(a,e.Agentes) //**O(1) en caso Promedio**
  return *res*
end
Complejidad: $O(1)$ en caso promedio
Comentarios: Devuelvo una poss de un agente

Algoritmo 12: posAgente

---

*i*cantSanciones(**in** a: `agente`,**in** e: `estr`) → k: nat
begin
  res ← sancionesAgentes(a,e.Agentes) //**O(1) en caso promedio**
  return *res*
end
Complejidad: $O(1)$ en caso promedio
Comentarios: Cantidad de sanciones de un agente

Algoritmo 13: cantSanciones

---

*i*masVigilante(**in** e: `estr`) → a: agente
begin
  res ← masVigilante(e.Agentes) //**O(1)**
  return *res*
end
Complejidad: $O(1)$
Comentarios: Devulvo a un agente donde agenete es un placa, del mas vigilante

Algoritmo 14: masVigilante

---

*i*conMismasSancioens(**in** a: `agente`,**in** e: `estr`) → cA: conj(agente)
begin
  res ← conMismasSanciones(a,e.Agentes) //**O(1) en caso promedio**
  return *res*
end
Complejidad: $O(1)$ en caso promedio
Comentarios: Devulvo a un conj(agente) donde agente es un placa, y todos los del cjto tienen las mismas sanciones

Algoritmo 15: conMismasSanciones

*i*conKsanciones(**in** k: `nat`,**in** e: `estr`) → cA: conj(agente)
begin
| res ← conKSanciones(k,e.Agentes) **return** *res*
end
Complejidad: $O(cantAgentes)$ la primera vez que se llama y $O(Log(cantAgentes))$ las siguientes veces que se llama mientras no se produzcan sanciones)
Comentarios: Devulvo a un conj(agente) donde agenete es un placa, y el conj son todos los agentes con k sanciones

Algoritmo 16: conkSanciones

*i*cantHippiesAtrapados(**in** a: `agente`,**in** e: `estr`) → n: nat
begin
| res ← cantHippiesAtrapados(a,e.Agentes) //**O(1) en caso promedio**
| **return** *res*
end
Complejidad: $O(1)$ En caso promedio
Comentarios: Devuelvo la cantidad de hippies atrapados por un agente en particular

Algoritmo 17: cantHippiesAtrapados

*i*cantHippies(**in** e: `estr`) → n: nat
begin
| res ← cardinal(*Hippies(e.HipYEst)) //**O(1)**
| **return** *res*
end
Complejidad: $O(1)$
Comentarios: Devuelvo la cantidad de hippies

Algoritmo 18: cantHippies

*i*cantEstudiantes(**in** e: `estr`) → n: nat
begin
| res ← cardinal(*Estudiantes(e.HipYEst)) //**O(1)**
| **return** *res*
end
Complejidad: $O(1)$
Comentarios: Devuelvo la cantidad de estudiantes

Algoritmo 19: cantEstudiantes

*i*hippieRodeadoEst(**in** p: `pos`,**in** e: `estr`) → b: bool
begin
| cjtoVec ← vecinos(p,e.campus) //**O(1)**
| cantEst ← 0 //**O(1)**
| Itvec ← crearIT(cjtoVec) //**O(1)**
| **while** *hayMas(Itvec)* **do** //**O(1)**
| | **if** *def?(actual(Itvec),e.posOcupadasEstudiante))* **then** //**O(1)**
| | | cantEst++ //**O(1)**
| | **end**
| | siguiente(Itvec) //**O(1)**
| **end**
| res ← cantEst = cardinal(cjtoVec) //**O(cant elementos)**
| **return** *res*
end
Complejidad: Es $O(1)$ porque la cantidad de elementos en cjtoVec es cte.
Comentarios: me fijo si estoy rodeado por estudiantes

Algoritmo 20: hippieRodeadoEst

*i*hippieCapturado(**in** p: pos,**in** e: estr) → b: bool
begin
  cjtoVec ← vecinos(p,e.campus) //**O(1)**
  alMenos1Ag ← false //**O(1)**
  Itvec ← crearIT(cjtoVec) //**O(1)**
  while *hayMas(Itvec)* **do** //**O(1)**
    if *def?(actual(Itvec),e.posOcupadasAgente))* **then** //**O(1)**
      alMenos1Ag ← true //**O(1)**
    end
    siguiente(Itvec) //**O(1)**
  end
  res ← alMenos1Ag ∧ bloqueado(p,e) //**O(1)**
  return *res*
end
Complejidad: $O(1)$
Comentarios: me fijo si me pueden capturar

Algoritmo 21: hippieCapturado

*i*estudianteHippieficado(**in** p: pos,**in** e: estr) → b: bool
begin
  cjtoVec ← vecinos(p,e.campus) //**O(1)**
  cantHip ← 0 //**O(1)**
  Itvec ← crearIT(cjtoVec) //**O(1)**
  while *hayMas(Itvec)* **do** //**O(1)**
    if *ocupada?(actual(Itvec),e.posOcupadasHippie))* **then** //**O(1)**
      cantHip++ //**O(1)**
    end
    siguiente(Itvec) //**O(1)**
  end
  res ← cantHip ≥ 2 //**O(1)**
  return *res*
end
Complejidad: $O(1)$ Como la cantaidad de pos a chequear es constante, es decir a lo sumo puedo tener 4 vecinos, la complejidad queda O(1)
Comentarios: me fijo si 2 vecinos mios son hippies como para atraerme al hippismo

Algoritmo 22: estudianteHippieficado

*i*estudianteSancionar(**in** p: pos,**in** e: estr) → b: bool
begin
  cjtoVec ← vecinos(p,e.campus) //**O(1)**
  alMenos1Ag ← false //**O(1)**
  Itvec ← crearIT(cjtoVec) //**O(1)**
  while *hayMas(Itvec)* **do** //**O(1)**
    if *definida?(actual(Itvec),e.posOcupadasAgente))* **then** //**O(1)**
      alMenos1Ag ← true //**O(1)**
    end
    siguiente(Itvec) //**O(1)**
  end
  res ← alMenos1Ag ∧ bloqueado(p,e) //**O(1)**
  return *res*
end
Complejidad: $O(1)$ Me fijo si uno de mis vecinos me estan bloquenado y al menos tengo un agente como solo accedo una cantidad finita a una matriz es $O(1)$
Comentarios: me fijo si un agente me esta bloqueando

Algoritmo 23: estudianteSancionar

```
ibloqueado(in p: pos,in e: estr) → b: bool
begin
    cjtoVec ← vecinos(p,e.campus)                                                    //O(1)
    cosasBloqueando ← 0                                                              //O(1)
    Itvec ← crearIT(cjtoVec)                                                         //O(1)
    while hayMas(Itvec) do                                                           //O(1)
        if (def?(actual(Itvec),e.posOcupadasAgente) ∨ def?(actual(Itvec),e.posOcupadasHippie)) ∨
        def?(actual(Itvec),e.posOcupadasEstudiante)) ∨ def?(actual(Itvec),obstaculos(e.campus))) then //O(1)
            cosasBloqueado++                                                         //O(1)
        end
        siguiente(Itvec)                                                             //O(1)
    end
    res ← cosasBloqueado = cardinal(cjtoVec)                                         //O(1)
    return res
end
```

Complejidad: $O(1)$
Comentarios: me fijo si estoy bloqueado

Algoritmo 24: bloqueado

```
ihippificar(in p: pos,in/out e: estr)
begin
    definirHippie(obener(pos,e.posOcupadasEstudiante)),pos,e.HipYEst)   //O(|n_m|)
    definir(pos,obener(pos,e.posOcupadasEstudiante)),e.posOcupadasHippie) //O(1) + O(1)
    borrar(obener(pos,e.posOcupadasEstudiante)),e.HipYEst)              //O(|n_m| + O(1)
    borrar(pos,e.posOcupadasEstudiante)                                 //O(1)
end
```

Complejidad: 2*O(|n_m|) = O(|n_m|)
Comentarios: transformo un estudiante a hippie

Algoritmo 25: hippificar

```
isancionar(in p: pos,in/out e: estr)
begin
    cjtoVec ← vecinos(p,e.campus)                                                    //O(1)
    Itvec ← crearIT(cjtoVec)                                                         //O(1)
    while hayMas(Itvec) do                                                           //O(1)
        if def?(actual(Itvec),e.posOcupadasAgente)) then                             //O(1)
            agregarSancion(obtener(actual(Itvec),e.posOcupadasAgente),e.agentes) //O(1) En caso promedio
        end
    end
end
```

Complejidad: $O(1)$ En caso promedio ya que sabemos que las placas de los agentes estan distribuidas uniformemente
Comentarios: sanciono agentes

Algoritmo 26: sancionar

```
itransformarHippie(in p: pos,in/out e: estr)
begin
    definirEstudiante(obener(pos,e.posOcupadasHippie)),pos,e.HipYEst)   //O(|n_m|)
    definir(pos,obener(pos,e.posOcupadasHippie)),e.posOcupadasEstudiante) //O(1)
    borrar(obener(pos,e.posOcupadasHippie)),e.HipYEst)                  //O(|n_m|)
    eliminar(pos,e.posOcupadasHippie)                                   //O(1)
end
```

Complejidad: $2 * O(|n_m|) = O(|n_m|)$
Comentarios: transformo un hippie a estudiante

Algoritmo 27: transformarHippie

```
isumarCapturas(in p: pos,in/out e: estr)
begin
    cjtoVec ← vecinos(p,e.campus)                                          //O(1)
    Itvec ← crearIT(cjtoVec)                                               //O(1)
    while hayMas(Itvec) do                                                 //O(1)
        if def?(actual(Itvec),e.posOcupadasAgente)) then                   //O(1)
            agregarCaptura(obtener(actual(Itvec),e.posOcupadasAgente),e.agentes)   //O(1) En caso
            Promedio
        end
        siguiente(Itvec)                                                   //O(1)
    end
end
```

Complejidad: $O(1)$ En caso promedio ya que sabemos que las placas estan distribuidas uniformemente y ademas como el cjtovec es cte el ciclo tambien es O(1)

Comentarios: Agrego las capturas a los agentes que participaron

Algoritmo 28: sumarCapturas

```
ieliminarHippie(in p: pos,in/out e: estr)
begin
    sumarCapturas(p,e)                                                     //O(1) En caso promedio
    borrar(obener(pos,e.posOcupadasHippie)),e.HipYEst)                     //O(n_m)
    eliminar(pos,e.posOcupadasHippie)                                      //O(1)
end
```

Complejidad: Tomando el caso promedio de agregar sancion en peor caso me quedaria O(n_m)

Comentarios: Elimino un hippie y agrego las capturas a los agentes que participaron de la matanza

Algoritmo 29: eliminarHippie

```
imeVoy(in p: pos,in d: dir,in e: estr)
begin
    res ← ((ingresoSuperior(p,e.campus) ∧ d = Arriba) ∨ (ingresoInferior(p,e.campus) ∧ d = Abajo))
    //O(1)+O(1)
end
```

Complejidad: O(1)

Comentarios: Me fijo si me voy del campus!

Algoritmo 30: meVoy

```
iestMasCercano(in p: pos,in e: estr) → res: pos
begin
    if cantEst(e) !=0 then                                                 //O(1)
        cjtoPosEst ← claves(e.posOcupadasEstudiante)                       //O(1)
        ItposEst ← crearIT(cjtoPosEst)                                     //O(1)
        posMasCercana ← actual(ItposEst)                                   //O(1)
        siguiente(ItposEst)                                                //O(1)
        while hayMas(ItposEst) do                                          //O(1)
            if distancia(p,posMasCercana) ≥ distancia(p,actual(ItposEst)) then   //O(1)
                posMasCercana ← actual(ItposEst)                           //O(1)
            end
            siguiente(ItposEst)                                            //O(1)
        end
        res ← posMasCercana                                                //O(1)
    end
    res ← actual(crearIT(ingresoMasCercano(e.campus))                      //O(1)
end
```

Complejidad: $O(cantestudiantes)$ que esta acotado por el ancho y alto del campus ya que no puede existir 2 fichas en un mismo casillero

Comentarios: Me devuelvo la poss mas cercana entre los estudiantes si no hay devuelvo un ingreso total la funcion de mover lo va a filtrar

Algoritmo 31: estMasCercano

*i*hipMasCercano(**in** p: `pos`,**in** e: `estr`) → res: pos
begin
  if *cantHip(e)* ! =*0* **then** //**O(1)**
    cjtoPosHip ← claves(e.posOcupadasHippie) //**O(cant hippies)**
    ItposHip ← crearIT(cjtoPosHip) //**O(1)**
    posMasCercana ← actual(Itpoship) //**O(1)**
    siguiente(ItposHip) //**O(1)**
    while *hayMas(ItposHip)* **do** //**O(1)**
      if *distancia(p,posMasCercana)* ≥ *distancia(p,actual(ItposHip))* **then** //**O(1)**
        posMasCercana ← actual(Itpoship) //**O(1)**
      end
      siguiente(ItposHip) //**O(1)**
    end
    res ← posMasCercana //**O(1)**
  else
    res ← actual(crearIT(ingresoMasCercano(e.campus)) //**O(1)**
  end
end
Complejidad: $O(canthippies)$
Comentarios: Me devuelvo la poss mas cercana entre los hippies

Algoritmo 32: hipMasCercano

*i*muevoEstudiante(**in** n: `nombre`,**in** p: `pos`,**in** d: `dir`,**in/out** e: `estr`)
begin
  posVieja ← pos //**O(1)**
  posNueva ← proxPosicion(d,posVieja) //**O(1)**
  definirEstudiante(n,posNueva,e.HipYEst) //**O(**$|n_m|$**)**
  definir(posNueval,n,e.posOcupadasEstudiante) //**O(1)**
  borrar(posViejar,e.posOcupadasEstudiante) //**O(1)**
end
Complejidad: O($|n_m|$)
Comentarios: Me muevo a la pos que me da la direccion

Algoritmo 33: muevoEstudiante

*i*muevoHippie(**in** p: `pos`,**in** pD: `pos`,**in/out** e: `estr`)
begin
  nombrehip ← obtener(p,e.posOcupadasHippie) //**O(1)**
  if *cantEst(e)* ! = *0* **then** //**O(1)**
    if *EstaOcupado(proxPosicion(p,dameDireccion(p,posD),e))* **then** //**O(1)**
      definirHippie(nombreHip,damePoslibre(p,e),e.HipYEst) //**O(**$|n_m|$**)**
      definir(damePoslibre(p,e),nombreHip,e.posOcupadasHippie) //**O(1)**
      borrar(damePoslibre(p,e),nombreHip,e.posOcupadasHippie) //**O(1)**
    else
      definirHippie(nombreHip,proxPosicion(p,dameDireccion(p,posD),e.HipYEst) //**O(**$|n_m|$**)**
      definir(proxPosicion(p,dameDireccion(p,posD),nombreHip,e.posOcupadasHippie) //**O(1)**
      borrar(proxPosicion(p,dameDireccion(p,posD),nombreHip,e.posOcupadasHippie) //**O(1)**
    end
  end
end
Complejidad: O($|n_m|$)
Comentarios: Muevo un hippie al estudiante mas cercano.

Algoritmo 34: muevoHippie

*i*muevoAgente(**in** *p*: pos,**in** *pD*: pos,**in/out** *e*: estr)
begin
  nombreAg ← obtener(p,e.posOcupadasAg) //**O(1)**
  if *cantHip(e)* ! = *0* **then** //**O(1)**
    if *EstaOcupado(proxPosicion(p,dameDireccion(p,posD),e))* **then** //**O(1)**
      cambiarPosicion(nombreAg,damePoslibre(p,e),e.agentes) //**O(1) En caso promedio**
      definir(damePoslibre(p,e),nombreAg,e.posOcupadasAgente) //**O(1)**
      borrar(damePoslibre(p,e),nombreAg,e.posOcupadasAgente) //**O(1)**
    else
      cambiarPosicion(nombreAg,proxPosicion(p,dameDireccion(p,posD),e.agentes) //**O(1) En caso promedio**
      definir(proxPosicion(p,dameDireccion(p,posD),nombreAg,e.posOcupadasAgente) //**O(1)**
      borrar(proxPosicion(p,dameDireccion(p,posD),nombreAg,e.posOcupadasAgente) //**O(1)**
    end
  else
    posIngreso ← actual(crearIT(ingresosMasCercanos(e.campus))) //**O(1)**
    if *EstaOcupado(proxPosicion(p,dameDireccion(p,posD),e))* **then** //**O(1)**
      cambiarPosicion(nombreAg,damePoslibre(p,e),e.agentes) //**O(1) En caso promedio**
      definir(damePoslibre(p,e),nombreAg,e.posOcupadasAgente) //**O(1)**
      borrar(damePoslibre(p,e),nombreAg,e.posOcupadasAgente) //**O(1)**
    else
      cambiarPosicion(nombreAg,proxPosicion(p,dameDireccion(p,posIngreso),e.agentes) //**O(1) En caso promedio**
      definir(proxPosicion(p,dameDireccion(p,posIngreso),nombreAg,e.posOcupadasAgente) //**O(1)**
      borrar(proxPosicion(p,dameDireccion(p,posIngreso),nombreAg,e.posOcupadasAgente) //**O(1)**
    end
  end
end
Complejidad: En caso promedio, sabiendo que las placas estan distribuidas uniformemente, es O(1), si no es $O(n_a)$
Comentarios: muevo un agente al hippie mas cercano si esta bloqueada la poss me muevo a una cualquier si no me muevo a la correcta, si no hay hippies me muevo al ingreso mas cercano
con la misma metodologia que al moverme a un hippie

Algoritmo 35: moverAgente

---

*i*estaOcupado(**in** *p*: pos,**in** *e*: estr) → b: bool
begin
  res ← (def?(pos,e.posOcupadasAgente) ∨ def?(pos,e.posOcupadasHippie)) ∨
  def?(pos,e.posOcupadasEstudiante)) ∨ Ocupada?(pos, e.campus)) //**O(1)**
  return *res*
end
Complejidad: $O(1)$
Comentarios: me fijo si existe alguien en esa poss

Algoritmo 36: estaOcupado

---

*i*damePosLibre(**in** *p*: pos,**in** *e*: estr) → res: pos
begin
  cjtoVec ← vecinos(p,e.campus) //**O(1)**
  Itvec ← crearIT(cjtoVec) //**O(1)**
  while *hayMas(Itvec)* **do** //**O(1)**
    if *estaOcupado(actual(Itvec),e))* **then** //**O(1)**
      res ← actual(Itvec) //**O(1)**
    end
    siguiente(Itvec) //**O(1)**
  end
  return *res*
end
Complejidad: $O(1)$
Comentarios: devuelvo Una poss libre de mis vecinos

Algoritmo 37: damePosLibre

```
idameDireccion(in p: pos,in pD: estr) → res: dir
begin
    var dirFinal : dir                    //O(1) if |p.x - pD.x| ≥ |p.y - pD.y| then          //O(1)
        if p.x > pD.x then                                                                    //O(1)
            dirFinal ← Izquierda                                                              //O(1)
        else
            dirFinal ← Derecha                                                                //O(1)
        end
    else
        if p.y > pD.y then                                                                    //O(1)
            dirFinal ← Abajo                                                                  //O(1)
        else
            dirFinal ← Arriba                                                                 //O(1)
        end
    end
    res ← dirFinal                                                                            //O(1)
end
```

Complejidad: $O(1)$

Comentarios: devuelvo la direccion a la cual debo ir

Algoritmo 38: dameDirrecion