Reproducción en el laboratorio: justificació de les operacions

# La classe Organisme

## L’operació estirar\_organisme

Aquesta operació fa una estirada a l’organisme només si abans no ha estat retallat prèviament. Per fer-ho utilitza la operació “estirar\_recursiu”. En el programa principal es crida aquesta operació mitjançant la instrucció -1.

### Implementació

/\*

Pre: L’organisme ha de tenir una cèl·lula o més

Post: Totes les cèl·lules de l’organisme que no s’han dividit han passat a tenir

dues cèl·lules filles a l’àrbre de cèl·lules de l’organisme

\*/

void Organisme::estirar\_organisme()

{

if (not retallat and not cels.es\_buit())

{

estirar\_recursiu(cels, max\_id, tamany);

}

}

### Justificació

Només es fa una crida a una operació recursiva dins d’aquesta funció en el cas que l’organisme no hagi estat retallat abans. La crida a la funció estirar\_recursiu compleix com a condició que l’arbre no estigui buit ja que una de les condicions és l’arbre ‘cels’ no sigui buit, així doncs es compleix la precondició de la funció.

## L’operació estirar\_recursiu

L’operació estirar\_recursiu aplica una estirada a l’organisme fent que totes les cèl·lules que no s’havien dividit abans passin a tenir dues cèl·lules iguals a aquesta però amb un ID superior a la cèl·lula amb l’ID més gran de tot l’organisme. Aquestes dues cèl·lules noves es col·loquen com a filles a l’arbre de cèl·lules de l’organisme.

### Implementació

/\*

Pre: ‘a’ és un àrbre no buit.

Post: Totes les cèl·lules que no s'havien dividit abans han passat a

estar dividides.

\*/

void Organisme::estirar\_recursiu(Arbre<Celula> &a, int &max\_id, int &tam)

{

Arbre<Celula> a1, a2;

Celula c = a.arrel();

a.fills(a1, a2);

if (a1.es\_buit() and a2.es\_buit())

{

tam += 2;

Celula aux = c;

Arbre<Celula> buit;

++max\_id;

aux.id = max\_id;

a1.plantar(aux, buit, buit);

++max\_id;

aux.id = max\_id;

a2.plantar(aux, buit, buit);

}

else

{

if (not a1.es\_buit()) estirar\_recursiu(a1, max\_id, tam);

// HI1: a1 s’ha estirat, max\_id i tamany s’ha incrementat en tants elements

// com cèl·lules s’han afegit.

if (not a2.es\_buit()) estirar\_recursiu(a2, max\_id, tam);

// HI2: a1 s’ha estirat, max\_id i tamany s’ha incrementat en tants elements

// com cèl·lules s’han afegit.

}

a.plantar(c, a1, a2);

}

### Justificació

* *Cas senzill:* Si els dos subarbres de l’arbre ‘a’ són buits hem trobat una cèl·lula que no s’ha dividit i plantem dos cèl·lules idèntiques però amb un ID més gran a cada subarbres, fent que aquests deixin d’estar buits.
* *Cas recursiu:* Si un dels dos subarbres no és buit vol dir que la cèl·lula ja s’havia dividit i que hem de buscar cèl·lules que no s’han dividit als subarbres dret i esquerre.

Trobar cèl·lules no dividides s’obté per hipòtesi d’inducció amb les crides recursives. Abans de cridar cada funció recursiva comprovem que el subarbres ‘aX’ no sigui buit per tal de complir la precondició. Finalment plantem els dos subarbres que ara tenen les cèl·lules noves afegides a l’arrel que tenim.

* *Acabament:* A cada crida recursiva es va fent més petit l’arbre ‘a’.

# La classe ConjuntOrg

## L’operació reproduir

L’operació reproduir de la classe ConjuntOrg busca una parella per cada organisme amb la que no s’hagi aparellat abans i intenta fer una reproducció entre els dos organismes.

### Implementació

/\*

Pre: Cert

Post: Tots els organismes que poden s'han reproduit un cop com a màxim a més a més es retornen els fills nous de la ronda a la variable ‘fills’

\*/

bool ConjuntOrg::reproduir(Ranking &Rank, int &fills)

{

vector<bool> Escollit(tamany, false);

int num = tamany;

fills = 0;

int i = 0;

/\* INV1: 0 <= i <= num, 0 < tamany <= V.size() \*/

while (i < num and tamany != V.size())

{

if (not Escollit[i] and not V[i].es\_mort())

{

bool candidat = false;

int j = i + 1;

/\* INV2: i+1 <= j <= num, candidat = "s'ha trobat una parella V[j]

per a l'organisme V[i]" \*/

while (j < num and not candidat)

{

if (not Escollit[j] and not V[j].es\_mort() and

not Aparellat[i][j])

{

candidat = true;

Escollit[i] = Escollit[j] = true;

Aparellat[i][j] = Aparellat[j][i] = true;

}

else ++j;

}

if (candidat)

{

if(V[tamany].reproduir\_organisme(V[i], V[j]))

{

Rank.afegir\_fill(i, j, tamany);

++tamany;

++fills;

}

}

}

++i;

}

return hi\_cap;

}

### Justificació

Bucle 1:

* *Inicialitzacions:* Inicialment no hem comprovat cap element de V per tant posem ‘i’ a 0, satisfent així la primera part de la invariant. Per satisfer la segona cal posar ‘true’ a hi\_cap ja que el fet que haguem entrat a la funció ens diu que encara hi ha espai al subvector V[tamany...V.size()-1]. Així doncs d’aquesta manera podem satisfer la invariant.
* *Condició de sortida:* Es pot sortir del bucle per dues raons:
  + Si i arriba a ser num vol dir, per l’invariant que hem explorat tot el vector V[0...num] que és el que ens interessava perquè és on hi ha els organismes que volem que es reprodueixin.
  + En cas contrari com que l’invariant ens diu que i <= num es complirà que i < num i que per sortir del bucle tamany == V.size(). Si abans d’arribar al final del vector trobem que tamany == V.size() això voldrà dir que dos organismes han tingut un fill i que aquest ha omplert el vector d’organismes V de manera que ja no es poden seguir realitzant les reproduccions i, per tant, s’ha de sortir del bucle.