Reproducción en el laboratorio: justificació de les operacions

1. La classe `Organisme`

1.1. 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.

1.1.1. 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);
    }
}
```

1.1.2. 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.

1.2. 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.

1.2.1. 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);
}
```

1.2.2. 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'.

2. La classe `ConjuntOrg`

2.1. 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.

2.1.1. 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()
          -  num < V.size()
          -  s'ha reproduit tots els orgaismes que podien a V[0 ... i-1]
          -  fills = nombre d'organismes nous generats per les reproduccions
          -  Escollit[i] = "l'organisme ja s'ha reproduit en aquesta ronda"
*/
```

```
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]"
          -  No s'ha trobat cap organisme per emparellar entre V[i+1 ... j-1]
          -  Aparellat[i][j] = "l'organisme 'i' s'ha relacionat amb el 'j' i
             viceversa"
*/
            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;
}
```

2.1.2. Justificació

Bucle 1:

- *Inicialitzacions:* Inicialment no s'ha comprovat cap element de `V` així que el vector `Escollit` s'ha de inicialitzar tot a `false`, ja que no hi ha cap organisme que hagi estat escollit i sinó no es compliria l'invariant. Com que dins del bucle la variable `tamany` canvia i només es vol fer un nombre d'iteracions fix per tal que es reprodueixin només els organismes que hi ha en el moment d'entrada del bucle, s'assigna el valor inicial de `tamany` a `num`. No s'ha generat cap fill ja que no s'ha comprovat cap element, així doncs, es posa la variable `fills` a 0. Així mateix es posa la variable `i` a 0 ja que encara no s'ha comprovat cap element del vector.

- *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; així doncs, s'ha de sortir del bucle.

- *Cos del bucle:* A l'inici del bucle sabem, per l'invariant i per la condició d'entrada al bucle que `0 <= i < num` i que `tamany < V.size()`. Abans de incrementar la `i` s'ha de satisfer l'invariant canviant `i` per `i+1`. Primerament per comprovar que es compleixi la part de l'invariant que diu si s'han reproduït tots els organismes que poden, es comprova si l'organisme és viu i si no s'ha emparellat amb cap altre organisme. En el cas que es compleixin aquestes dues condicions llavors es busca un altre organisme per a l'organisme actual, el bucle intern s'ocupa de satisfer això, retornant una variable booleana `candidat` que ens diu si s'ha trobat una parella o no.

  En el cas d'haver trobat una parella es realitza una reproducció per tal de complir l'invariant i que entre `V[0 ... i]` s'hagin reproduït tots els organismes possibles. Si la reproducció ha estat possible (retorna `true`) s'ha d'incrementar la variable `fills` per tal de complir la part de l'invariant que demana quants fills s'han generat a la ronda de reproducció Finalment ja es pot incrementar i per comprovar el següent element del vector i així complir l'invariant ja que s'hauran reproduït tots els organismes possibles de `V[0 ... i-1]`.

- *Acabament:* A cada iteració la distància entre `i` i `num` es fa més petita.

Bucle 2:

- *Inicialitzacions:* Inicialment, com que no s'ha comprovat cap element del vector `V[i+1 ... num]`, es posa la variable `candidat` a `false`, ja que segur que no es pot trobar un candidat al vector `V[i+1 ... j-1]` amb `j >= i+1`. Per la condició d'entrada a aquest bucle se sap que tots els organismes fins a `i` ja s'han reproduït segur així que el candidat de reproducció ha d'estar a `V[i+1 ... num]`, així doncs s'inicialitza `j` a `i+1`.

- *Condició de sortida:* Es pot sortir del bucle per dues condicions
  - Si j arriba a ser `num`, vol dir, per l'invariant que s'ha explorat tot el subvector `V[i+1...num]` que és el que interessa per saber si hi ha un candidat a `V[i+1 ... num]`.
  - Si `candidat == true` això vol dir que s'ha trobat un candidat al subvector `V[i+1 ... j-1]` i que, per tant, no cal seguir buscant més candidats.

- *Cos del bucle:* Al principi sabem per la condició d'entrada al bucle i per l'invariant que no hi ha cap candidat al subvector `V[i+1 ... j-1]`. Abans d'incrementar `j` s'ha de satisfer l'invariant canviant `j` per `j+1`. Ara es comprova si l'organisme actual `V[j]`, que encara no ha estat comprovat, és un candidat per a l'organisme `V[i]`, també es comprova que aquest no estigui mort i que no hagi estat abans una parella de `V[i]`. Si es compleixen totes aquestes condicions llavors `V[j]` és un candidat i per tal de complir l'invariant s'ha de posar `candidat` a `true`. A part també hem de posar `Escollit[i]` i `Escollit[j]` a `true` per tal de complir l'invariant del primer bucle i hem de posar `Aparellat[i][j]` i `Aparellat[j][i]` a `true` per complir l'invariant del segon bucle. Ara ja es pot incrementar `j` assegurant així que es compleixi l'invariant.

- *Acabament:* A cada iteració la distància entre `j` i `num` es va fent més petita.