# 1 La operación condicion2()

Esta es una de las operaciones mas importantes del modulo. Ya que realiza la migración desde un nodo hacia al nodo central del árbol.

## 1.1 Implementación

```
pair<bool,int> Cjt_region::condicion2(Arbre<int> &a, const int &idreg, const int &idpla, const int &idesp, const int &h, const int &g) {
   //Pre: a = A, 1 <= idreg <= R, 1 <= idpla <= R, 1 <= idesp <= N,  0 <= h <= Region.numesp[].size(), 0 <= g < h
   //post: first: indica si contine en uno de sus arboles el idreg, second: indica el numero de individuos que vienen de la migracion de los hijos
   bool b = false;
   pair<bool,int> ret;
   ret.first = false;
   if(not a.es_buit()){
     int n = a.arrel();
     if(n == idreg) {
         vreg[idreg-1].decrementar_especie(h,idesp);
         ret.first = true;
         ret.second = h;
     }
     else{//caso recursivo
         Arbre<int> a1;
         Arbre<int> a2;
         a.fills(a1,a2);
         int c = 0;
         pair<bool,int> ret1 = condicion2(a1,idreg,idpla,idesp,h,g);
         //HP1: ret1.first= indica si contiene el idreg en a1.
         // ret1.second= indica el numero de individuos que vienen por a1.
         if(ret1.first) {
           c += ret1.second;
           ret.first = true;
         }else {
           pair<bool,int> ret2 = condicion2(a2,idreg,idpla,idesp,h,g);
           //HP2: ret2.first= indica si contiene el idreg en a2.
         // ret2.second= indica el numero de individuos que vienen por a2.
           if(ret2.first) {
             c += ret2.second;
             ret.first = true;
           }
         }
         if(n == idpla or c < g) {
           vreg[n-1].incrementar_especie(c,idesp);
         }
         else{
           vreg[n-1].incrementar_especie(c/2,idesp);
           ret.second = c-c/2;
```

```
        }
      }
    }
    return ret;
}
```

## 1.2 Justificación:

·***Caso directo:*** Si a no esta vacío y n es igual al idreg entonces ret.first = true y ret.second = al numero de individuos que queremos migrar. Por tanto, el código del caso directo nos lleva a la post condición.

·***Caso recursivo:*** El código del caso recursivo consiste en agregar el numero de individuos de la especie que migra según los resultado de las llamadas recursivas.
Primero se comprueba que
Si el árbol no es vacío y no hemos encontrado el idreg,
Obtenemos los hijos del árbol, entonces hacemos la llamada recursiva para el hijo izquierdo. Comprobamos si contiene el idreg en la estructura, si es así acumulamos el segundo valor que nos retorna a la variable que contiene el numero de individuos que migran a la región, si no esta contenido en la estructura del hijo izquierdo hacemos la llamada recursiva para el hijo derecho, Comprobamos si contiene el idreg en la estructura, si es así acumulamos el segundo valor que nos retorna a la variable que contiene el numero de individuos que migran a la región.
Usando HI1 y HI2 tenemos que para llegar hasta la post condición sólo hacen falta las operaciones que siguen a la llamada y son las que realizan: con el numero de individuos que han migrado a la región. Se quedan en la region c/2 y retornamos al nodo padre c-c/2.

·***Finalización:*** A cada llamada recursiva la mida del árbol disminuye.

## 2 Operación lucha():

Esta función genera la lucha entre las diferentes especies según una prioridad de especie y unas preferencias de alimentación según la especie.

## 2.1 Implementación:

```
void Cjt_especie::lucha(Region &reg) {
  for(int i = 0; i < C; ++i) {
    int ncarn = reg.consultar_num_indiv(pricar[i]);
    int mincarn = ves[pricar[i]-1].getMinAlimento();
    int npres = ves[pricar[i]-1].getNpres();
    int j = 0;
    int comido = 0;
    /*
     Inv: 0 <= j <= npres, ncar: indica el numero de carnivoros que quedan por
alimentarse, 0<=ncarn<=numero de carnivoros en la region
          comido: indica la cantidad de alimento ingerido por un carnivoro, 0 <=
comido < valor minimo a alimentarse
     */
    while (j < npres and ncarn != 0) {
      int nind = reg.consultar_num_indiv(ves[pricar[i]-1].getIessimo(j));
      if(nind != 0){
          int aliind = ves[ves[pricar[i]-1].getIessimo(j)-1].getValorAlimento();
          int minnec = mincarn/aliind;
          if((mincarn % aliind) != 0) ++minnec;

          if(comido != 0) {
            int nece = (mincarn-comido)/aliind;
            if(((mincarn-comido)%aliind) != 0) ++nece;
            if(nece <= nind) {
              reg.decrementar_especie(nece,ves[pricar[i]-1].getIessimo(j));
              --ncarn;
              comido = 0;
              nind -= nece;
            }
            else {
              comido += nind*aliind;
              reg.decrementar_especie(nind,ves[pricar[i]-1].getIessimo(j));
            }
          }

          if(ncarn != 0 and nind != 0 and comido == 0) {
            int maxcarn = nind/minnec;
            if(maxcarn >= 1) {
              if(maxcarn <= ncarn) {
                reg.decrementar_especie(maxcarn*minnec,ves[pricar[i]-
1].getIessimo(j));
                ncarn -= maxcarn;
```

```
            nind -= maxcarn*minnec;
          }
          else {
            reg.decrementar_especie(ncarn*minnec,ves[pricar[i]-
1].getIessimo(j));
            nind -= ncarn*minnec;
            ncarn = 0;
          }

        }
        if(nind > 0 and ncarn != 0) {
          comido = nind*aliind;
          reg.decrementar_especie(nind,ves[pricar[i]-1].getIessimo(j));
          nind = 0;
        }
      }
    }
    ++j;
  }
  if(ncarn != 0) {
    reg.decrementar_especie(ncarn,pricar[i]);
  }
 }
}
```

## 2.2 Justificación:

Justificación del bucle: (suponemos que es solo para el caso de una sola 'i')

·***Inicializaciones:*** consultamos el numero de carnívoros en la región, consultamos en la especie el valor mínimo a alimentarse y el numero de presas que puede atacar, como inicialmente no hemos recorrido las presas del carnívoro 'j = 0', como aun no hemos atacado a ninguna especie y no hemos comido nada 'comido=0'.

·***Condición de salida:***

- Si 'j' es igual 'npres' significa por el invariante que ya hemos recorrido todo el vector de presas, igualmente por el invariante 'ncarn' indica el numero de carnívoros que queda por alimentarse.
- Si 'npres' es igual a '0' significa que todos los carnívoros se han alimentado, entonces no hace falta recorrer mas presas, seguiremos cumpliendo el invariante ya que 'j <npres' y ncarn indicara el numero de carnívoros que faltan por alimentarse que será '0'.

·***Cuerpo del bucle:*** Inicialmente consultamos el numero de presas que

contiene la región, si es diferente de 0 entonces, consultamos el valor alimenticio de la presa, calculamos el numero mínimo de presas que tiene que alimentarse el carnívoro, una vez aquí:

Primero: Si comido es diferente de Zero eso significa que ya hemos atacado en manada y que un carnívoro a podido alimentarse pero no hasta su valor mínimo, entonces calculamos con el valor alimenticio de la especie el numero de presas necesario, si podemos llegar al valor mínimo a alimentarse del carnívoro con las presas que tiene, decrementamos el numero de presas de la región, decrementamos el numero de carnívoros a alimentarse 'ncar' y ponemos comido = 0 ya que por el invariante 0 <= comido < valor mínimo a alimentarse , si no hay presas suficientes nos comemos todas las presas, incrementamos comido con el valor que se alimentado el carnívoro ya que no contemplara el segundo caso y como sabemos que j < npres por la condición del bucle incrementamos 'j' .

Segundo: Puesto que la última instrucción de la iteración es ++i, justo antes de
hacerlo tenemos que satisfacer invariante para i+1, es decir,
Si llegamos aquí puede ser que:

- Se la primera vez que intentamos atacar con los carnívoros que tenemos en la región , entonces 'comido = 0' por la inicialización.
- O que ya han atacado en grupo y que un carnívoro no se alimentase lo mínimo y que entrase en el primer caso, se haiga quedad saciado y que queden carnívoros, entonces podemos mirar para atacar en grupo.
- Que hagamos atacado con una parte de grupo y que comido = 0.

Una vez aquí se calcula en numero máximo de individuos que se pueden alimentar con las presas que contiene la región se ataca con el grupo posible decrementando ncar ya que por el invariante contiene el numero de carnívoros a alimentarse y decrementando nind ya que contiene el numero de presas vivas por el invariante.
Comprobamos que el numero de presas es diferente de 0 y que aun quedan carnívoros por alimentarse, si es así sabemos que no se alimentara asta el mínimo necesario por lo cual se incrementa comido ya que por el invariante contiene el alimento ingerido sin llegar al minio necesario de un carnívoro, ponemos a 0 las presas de la región ya que nos alimentamos todo lo posible.

·***Finalización:*** A cada iteración  la diferencia entere j y npres se reduce, porque incrementamos j en cada una de ellas.

Justificación de las instrucciones posteriores al bucle:

·Como por el invariante podemos salir del bucle con j >= npres significa que hemos recorrido todas las preferencias de la especie carnívora, entonces es posible que queden carnívoros por alimentarse ‘ncar != 0’ ya que por el invariante ncar indica el numero de carnívoros que quedan por alimentarse, por lo cual estos carnívoros mueren de hambre así que decrementamos el numero de carnívoros en la región según ncarn.