Energía e Indústria

10 diciembre de 2014

A continuación mostraré todas las justificaciones de todas las operaciones recursivas necesarias para realizar un pedido y todos los procesos iterativos que intervengan en la aplicación de un ciclo de producción para un sector de un país.

1. La operación cicle de produccio

Esta operación actualiza los stocks de los sectores de todos los paises, basandose en la energía de la que disponen, las dependencias entre los sectores y cuanto pueden generar en un solo ciclo de producción.

1.1 Implementación

```
void Cjt_sectors::cicle_de_produccio(Cjt_paisos &cp){
  for (int i = 0; i<cp.total_paisos(); ++i){
    queue<int>copia(prioritat);
    int energia_modificar = 0;
    vector<int> v(ind.size(),0);
    while (not copia.empty()){
      int sect = copia.front();
      vector<pair <int,int> > dependencies = ind[sect-1].consultar_dependencies();
      int energia_disponible = ((cp.consulta_energia_pais(i)*ind[sect-1].consultar_tantxcent_energia())/100);
      int eup = ind[sect-1].consultar_EUP();
      int max_cant_prod = ind[sect-1].consultar_max_cant_prod();
      bool stop = false;
      while (not stop) {
        if (eup > energia_disponible){
          stop = true;
        }
        else if (max_cant_prod == v[sect-1]){
          stop = true;
        }
        else {
          int j = 0;
          while (j<dependencies.size() and not stop) {
            if (cp.consulta_stock_pais(i, dependencies[j].first) < dependencies[j].second){
              stop = true;
            }
            else ++j;
          }
        }
        if (not stop) {
          ++v[sect-1];
          energia_disponible -= eup;
          energia_modificar += eup;
          for (int j = 0; j<dependencies.size(); ++j) {
            cp.modificar_unitats_prod_pais(i+1, dependencies[j].first, -dependencies[j].second);
          }
        }
      }
      copia.pop();
    }
    for (int j = 0; j<v.size(); ++j) {
      cp.modificar_unitats_prod_pais(i+1, j+1, v[j]);
    }
    cp.modificar_energia_pais(i+1, -energia_modificar);
  }
}
```

1.2 Justificación

Procedo a justificar el primer for:

- *Invariante*: 0<=i<cp.total_paisos().
- *Inicialicaciones*: La variable i representará el pais en el cual estamos realizando el ciclo de producción, la inicializamos a 0 porque aun no hemos comprovado ningún pais y porque los paises estan organizados como un vector en la classe "Cjt_paisos", asi accederá a la primera posición satisfaciendo la condicion del invariante.
- *Condición de Salida*: Solo hay una forma de salir del bucle y será cuando i sea igual a la medida del vector de paises, en este caso cp.total_paisos(), que querrá decir, por el invariante, que ya hemos recorrido todos los paises.
- *Cuerpo del bucle*: Por el invariante sabemos que 0<=i<cp.total_paisos() por lo que tendremos que ir incrementando en 1 la variable i, pasando asi por todos los paises, realizando un ciclo de producción en cada uno de ellos y actualizando los stocks i energia corresponientes de cada uno de ellos (despues del ciclo de producción, ya que la producción que se cree no se puede usar en el mismo ciclo y la energia también se tiene que actualizar después de este).
- *Finalización*: A cada vuelta decrece la distáncia entre cp.total_paisos() i la variable i porque incrementamos esta ultima.

Justificación primer while:

- *Invariante:* copia es la copia la parte de prioritat que queda para tratar, siendo cada elemento de prioritat un sector del pais. Copia no esta inicialmente vacía.
- *Inicialicaciones*: copia es la copia de la cola prioritat para que no se destruya en el bucle. Inicialmente no se ha tratado ningun elemento de copia es decir copia contiene todos los elementos por el invariante.
- *Condición de Salida*: si copia llega a ser una cola vacía quiere decir por el invariante que ya hemos tratado todos los sectores del pais.
- *Cuerpo del bucle:* en la cola copia están ordenados los sectores en orden de prioridad por lo que iremos tratando el primer elemento de copia (siendo

este el más prioritario) y vaciandolo para dar paso a otro menos prioritario hasta que la cola quede vacía y se haya realizado completamente el ciclo de producción en todos los sectores del pais, por orden de prioridad.

- *Finalización*: A cada vuelta decrece el tamaño de copia.

Justificación segundo while:

- *Invariante*: stop será falso siempre que se pueda augmentar la producción.
- *Inicialicaciones*: inicialmente stop será falso porque por el invariante vemos que tendremos que comprovar si se puede augmentar la producción.
- *Condición de salida:* cuando stop llegue a ser true será que no se puede aumentar la producción por lo que se aturará el bucle.
- *Cuerpo del* bucle: Por el invariante vemos que tendremos que comprovar si se dan las propiedades necesarias para que aumente la producción. En el primer if comprovamos que la energia de la que dispone el sector no sea menor a la necesaria aumentar la producción, de ser asi se pone stop a true cumpliendo con el invariante. En el primer else if comprovamos que la máxima cantidad de producción por ciclo sea menor que la producción que llevamos, la condición del else if es que sean diferentes, esto se cumple ya que inicialmente la producción será 0 y la maxima cantidad de producción es un entero positivo, sabiendo esto se comprueva que cuando los valores sean iguales stop se pondrá a true, cumpliendo el invariante. En el primer else hay un bucle interno en el que se comprueva que el stock del sector del que depende el sector en el que nos encontramos no sea mayor a su dependencia, de ser esto cierto se pone stop a true cumpliendo con el invariante. En el ultimo if, si stop no es true por invariante será porque se puede aumentar la producción, asi pues aumentamos esta y actualizamos las dependencias y la energia que gastamos y que tenemos que modificar.
- *Finalización*: Cuando se cumplan o el primer if o else if o else no se podra aumentar la producción y por el invariante stop será true.

Justificación del while dentro del else:

- *Invariante*: 0<=j<dependencies.size(), stop será falseo siempre que las dependencias permitan aumentar la producción.
- *Inicializaciones*: Inicialmente no hemos comprovado ningun elemento del vector dependencies por lo que inicializaremos j a 0, la primer posición del vector, cumpliendo con la primera parte del invariante. Inicializaremos stop a false ya que inicialmente supondremos que cumple las condiciones necesarias para permitir aumentar la producción.
- *Condición de salida*: Se puede salir del bucle por dos razones:
  1. Si llegamos a j == dependencies.size() querrá decir que hemos recorrido todo el vector y stop sigue siendo false por lo que se cumplirá que las dependencias permiten aumentar la producción por invariante.
  2. En caso contrario si j < dependencies.size() i stop == true querrá decir que no hemos recorrido todo el vector, ya que no nos ha sido necesario dado que hemos llegado a un sector donde tiene mas dependencia que stock por lo que no se puede aumentar la producción.
- *Cuerpo del bucle*: En el if comprovamos que en el sector del que depende el sector que estamos tratando hay más stock que dependencia tienen, en caso contrario stop se pondra a true para cumplir con el invariante, si eso no se cumple aumentaremos la j para asi comprovar todos los sectores de los que depende.
- *Finalización*: A cada vuelta o bien decrece la distancia entre dependencies.size() y el índice j, porque incrementamos j, o bien ponemos el booleano stop a true y salimos del bucle.

Justificación del ultimo for:

- *Invariante*: 0 <= j < v.size()
- *Inicializaciones*: Inicialmente no hemos comprovado ningun elemento del vector por lo que inicializaremos a j a 0, primera posición del vector, cumpliendo con el invariante.
- *Condición de Salida*: Cuando lleguemos a j == v.size() querrá decir que

hemos recorrido todo el vector por lo que cumpliendo con el invariante saldremos del bucle.

- *Cuerpo del bucle*: j representará cada sector del pais y en el vector v habrá la producción que habrán aumentado durante el ciclo de producción de su pais, se recorrerá todo el vector con j marcando el sector en el que nos encontramos y actualizando el stock de estos después de haver realizado el ciclo de producción.
- *Finalización*: A cada vuelta decrece la distancia entre v.size() y el índice j, porque lo incrementamos a cada iteración.

2. La operación func2

Esta función actualiza las energias de los paises productores y consumidores a partir del pedido de estos ultimos. Los paises se encuentran distribuidos en forma de árbol y los paises consumidores más lejanos tienen preferencia en recibir la energia de los productores y los paises productores mas lejanos al nodo tienen más preferencia en vender su energía a los paises consumidores.

2.1 Implementación

```
void Cjt_paisos::func2 (Arbre<int>& a, int &ets, int &pedido){
  if (not a.es_buit()) {
    if (a.arrel() <= comanda.size()){
      Arbre<int> dre;
      Arbre<int> esq;
      Arbre<int> copia(a);
      copia.fills(esq, dre);
      if (not esq.es_buit() and dre.es_buit()){
        pedido = 0;
        func2(esq, ets, pedido);
        if (ets > 0) {
          if (comanda[a.arrel()-1] <= ets){
            mon[a.arrel()-1].modificar_energia(comanda[a.arrel()-1]);
            pedido += comanda[a.arrel()-1];
            ets -= comanda[a.arrel()-1];
          }
          else {
            mon[a.arrel()-1].modificar_energia(ets);
            pedido += ets;
            ets = 0;
          }
        }
      }
      else if (not esq.es_buit() and not dre.es_buit()) {
        int ets_esq;
        if (ets%2 != 0) {
          ets_esq = ets/2;
          ++ets_esq;
        }
        else ets_esq = ets/2;
        int ets_dre = ets/2;
        int pedido_esq = 0;
        int pedido_dre = 0;
        func2(esq, ets_esq, pedido_esq);
        func2(dre, ets_dre, pedido_dre);
        ets = ets_esq + ets_dre;
        pedido = pedido_esq + pedido_dre;
        if (ets > 0) {
          if (comanda[a.arrel()-1] <= ets) {
            mon[a.arrel()-1].modificar_energia(comanda[a.arrel()-1]);
            pedido += comanda[a.arrel()-1];
            ets -= comanda[a.arrel()-1];
          }
          else {
            mon[a.arrel()-1].modificar_energia(ets);
            pedido += ets;
            ets = 0;
          }
        }
      }
      else if (esq.es_buit() and dre.es_buit()) {
        if (comanda[a.arrel()-1] <= ets){
          mon[a.arrel()-1].modificar_energia(comanda[a.arrel()-1]);
          pedido += comanda[a.arrel()-1];
          ets -= comanda[a.arrel()-1];
        }
        else {
          mon[a.arrel()-1].modificar_energia(ets);
          pedido += ets;
          ets = 0;
        }
```

```
    else {
      Arbre<int>copia(a);
      Arbre<int>esq;
      Arbre<int>dre;
      copia.fills(esq,dre);
      if (not esq.es_buit() and dre.es_buit()) {
        ets += mon[a.arrel()-1].consultar_energia();
        pedido = 0;
        func2(esq, ets, pedido);
        if (pedido > 0) {
          if (mon[a.arrel()-1].consultar_energia() >= pedido) {
            mon[a.arrel()-1].modificar_energia(-pedido);
            pedido = 0;
          }
          else {
            pedido -= mon[a.arrel()-1].consultar_energia();
            mon[a.arrel()-1].modificar_energia(-mon[a.arrel()-1].consultar_energia());
          }
        }
      }
      else if (not esq.es_buit() and not dre.es_buit()) {
        int ets_esq = (ets+mon[a.arrel()-1].consultar_energia());
        if (ets_esq%2 != 0) {
          ets_esq /= 2;
          ++ets_esq;
        }
        else ets_esq /= 2;
        int ets_dre = (ets+mon[a.arrel()-1].consultar_energia())/2;
        int pedido_esq = 0;
        int pedido_dre = 0;
        func2(esq, ets_esq, pedido_esq);
        func2(dre, ets_dre, pedido_dre);
        pedido = pedido_esq + pedido_dre;
        if (pedido > 0) {
          int energ_esq = mon[a.arrel()-1].consultar_energia()/2;
          int energ_dre = mon[a.arrel()-1].consultar_energia()/2;
          if (mon[a.arrel()-1].consultar_energia()%2 != 0) ++energ_esq;
          if (energ_esq >= pedido_esq) {
            mon[a.arrel()-1].modificar_energia(-pedido_esq);
            pedido_esq = 0;
          }
          else {
            pedido_esq -= energ_esq;
            mon[a.arrel()-1].modificar_energia(-energ_esq);
          }
          if (energ_dre >= pedido_dre) {
            mon[a.arrel()-1].modificar_energia(-pedido_dre);
            pedido_dre = 0;
          }
          else {
            pedido_dre -= energ_dre;
            mon[a.arrel()-1].modificar_energia(-energ_dre);
          }
        }
        pedido = pedido_esq + pedido_dre;
      }
    }
  }
}
```

1.2 Justificación

Hipotesis de inducción

ets = Energía máxima que le puede llegar al pais consumidor donde estamos situados como suma de las energias de sus paises productores predecesores.
pedido = Energía que han consumido los paises consumidores sucesores y que le llega como energía que debe al pais productor donde estamos situados (si no lo puede asumir todo se lo devuelve a un pais productor predecesor, restandole la parte que él ha podido asumir).

Caso simple

Hay tres casos simples:

1. Si el parámetro *a* es un árbol vacío devolverá la misma ets y el mismo pedido que le correspondía a su predecesor.
2. Si el nodo del parametro *a* es un pais consumidor y sus dos hijos son vacíos, consultará en el vector *comandes* su pedido y de ser menor a la *ets* que le llega, actualizará su energía sumando a ella todo el pedido realizado (el cual se encuentra en el vector *comandes*) y le devolverá a su predecesor el *pedido* total y la *ets* restandole el pedido que haya hecho él. Si su pedido es mayor a la *ets* que le llega, se actualizará la energía del pais sumandole la *ets* y le devolverá al predecesor el *pedido*, que pasará a tener el valor de la *ets* y finalmente la *ets* pasará a ser 0.
3. Si el nodo del parametro *a* es de un pais productor y sus dos hijos son vacíos, devolverá la misma ets y el mismo pedido que correspondía a su predecesor.

Caso Recursivo

Hay 4 casos recursivos:

1. Si el nodo del parámetro *a* es de un pais consumidor y solo contiene el hijo de la izquierda, se volverá a llamar a la función *func2* con el parámetro ets i pedido que le han llegado por referencia i con el hijo izquierdo de *a*. Después de la ejecución de la función se evaluará si la ets es mayor que 0 (si no lo es por HI veremos que en el pais consumidor donde nos encontramos no se podrá realizar el pedido) en caso de ser 0 el

pedido i la *ets* tendran el mismo valor para el predecedor de *a*, sino se evaluará que *ets* sea mayor o menor al pedido realizado por este pais consumidor, si es mayor en el pais se sumará todo el pedido realizado a su energía, que también se restará de la *ets* que se pasará al predecesor y el *pedido* que se pasará al predecesor se sumará al pedido que había realizado el pais perteneciente al nodo de *a*, si es menor en el pais se sumará la *ets,* llegada del sucesor, a su energía, al *pedido* que se pasará al predecesor se le sumará la *ets* y finalmente la *ets* que se pasará al predecesor será 0.

2. Si el nodo del parámetro *a* es de un pais consumidor con los dos hijos no vacíos, se mirará si al *ets* que le llega es par o impar, si es par se dividirá por dos y se pasará la mitad de esta al hijo de la izquierda y la otra mitad al hijo de la derecha, si es impar se dividirá por 2 igual, cojiendo la parte entera de la división, se pasará al hijo de la izquierda este resultado sumando 1 i al de la derecha el resultado sin modificar, el pedido en ambos casos sera 0, cumpliendo con la HI, y se llamará a la función *func2* con el hijo de la izquierda y la ets de la izquierda y lo mismo con el de la derecha. Una vez finalizadas las dos llamadas a las funciones se sumaran las ets y los pedidos llegados de la llamada al hijo de la izquierda y al de la derecha, se evaluará si la ets resultante de la suma entre la ets llegada del hijo izquierdo y el de la derecha, es mayor que 0 (si no lo es por HI veremos que en el pais consumidor donde nos encontramos no se podrá realizar el pedido) en caso de ser 0 el *pedido y* la *ets* tendran el mismo valor para el predecedor de *a*, sino se evaluará que *ets* sea mayor o menor al pedido realizado por este pais consumidor, si es mayor en el pais se sumará todo el pedido realizado a su energía, que también se restará de la *ets* que se pasará al predecesor y el *pedido* que se pasará al predecesor se sumará al pedido llegado que había realizado el pais perteneciente al nodo de *a*, si es menor en el pais se sumará la *ets,* llegada del sucesor, a su energía, al *pedido* que se pasará al predecesor se le sumará la *ets* y finalmente la *ets* que se pasará al predecesor será 0.

3. Si el nodo del parámetro *a* corresponde al de un pais productor el cual solo tiene hijo izquierdo se sumará a la *ets* que le llega como parámetro toda su energía y se llamará a la función *func2* con la ets resultante, el *pedido* que será 0 y el hijo de la izquierda (cumpliendo con la HI en la ets y el pedido). Una vez se ha ejecutado la llamada a la función se evaluará si el *pedido* que ha llegado como parámetro es mayor que 0, de no ser asi se asumirá que no se ha usado energía de este pais productor para el pedido de los paises consumidores (por la HI), si *pedido* es mayor que 0 se mirará si este es mayor que la energía del pais productor donde estamos, de ser asi, se restará la energía del pais al *pedido* y la energía pasará a ser 0, si no es asi se restará a la energía del pais el pedido y este pasará a ser 0 para sus predecesores (cumpliendo la HI).

4. Si el nodo del parámetro *a* corresponde al de un pais productor el cual tiene ambos hijos izquierdo y derecho, se sumara a la ets que llega como parámetro la energía del pais y se dividirá entre 2, si esta era impar se cojera la parte entera del resultado de la división y se le sumará 1 en la ets que ira al hijo de la izquierda, se llama a la función *func2* dos veces, una para cada hijo, poniendo como parametros en cada una la parte de la ets que le corresponde y el pedido, que sera 0 (cumpliendo con la HI). Una vez finalizada la ejecución de las llamadas a estas dos funciónes, se sumará el pedido que llegue por la izquierda y el que llegue por la derecha, si esta suma da 0 se asumirá por HI que el pedido ya ha sido asumido totalmente por los paises productores sucesores o el pedido era 0, si esta suma da mayor a 0 se consultara la energía del pais que corresponde al nodo de *a* y se evalua si es par o impar, si es par la energía con la que se podrá tratar al pedido de la izquierda correspondrá a la mitad de su energía (igual que al de la derecha) si es impar se dividirá la energía del pedido entre 2 y se cojerá la parte entera en ambos casos, sumando uno para la energía con la que se podrá tratar al pedido de la izquierda. Esto se puede aplicar tanto a la energía con la que se puede tratar al pedido de la izquierda como la de la derecha ya que es igual en ambos casos, se mirará si el pedido es mayor que la energía con la que se puede tratar, de ser asi, se restará la energía al

pedido y la energía pasará a ser 0, si no es asi se restará a la energía del pais el pedido y este pasará a ser 0. Una vez realizado esto para el pedido de la izquierda y el de la derecha se sumaran estos dos y será el *pedido* que irá para el pais predecesor.

Acabamiento

A cada llamada recursiva que hagamos el árbol a será más pequeño, hasta que llegará a ser vacío.