JUSTIFICACIÓ PRÀCTICA

Funció iterativa (comerciar)

```
void Inventario::comerciar(Inventario &inv2, const Cjt_productos &productos)
{
    map<int, Cantidad>::iterator it1 = inv.begin();
    map<int, Cantidad>::iterator it2 = inv2.inv.begin();

    // Iteramos hasta que llegamos al final de un inventario
    while (it1 != inv.end() and it2 != inv2.inv.end()) {
        int id1 = it1->first;
        Cantidad cantidad1 = it1->second;
        int id2 = it2->first;
        Cantidad cantidad2 = it2->second;

        if (id1 == id2) // Si los dos inventarios contienen el producto con identificador id1=id2
        {
            // Cojo el minimo entre lo que le falta a inv y lo que le sobra a inv2
            int cantidad_comerciar = calculo_comercio(
                cantidad1.poseido, cantidad1.requerido,
                cantidad2.poseido, cantidad2.requerido
            );
            // Comercio entre los dos inventarios la cantidad indicada
            modificar_cantidad_poseida(id1, cantidad_comerciar, productos.consultar_producto(id1));
            inv2.modificar_cantidad_poseida(id2, -cantidad_comerciar, productos.consultar_producto(id2));

            // Repito el proceso pero al reves
            cantidad_comerciar = calculo_comercio(
                cantidad2.poseido, cantidad2.requerido,
                cantidad1.poseido, cantidad1.requerido
            );
            modificar_cantidad_poseida(id1, -cantidad_comerciar, productos.consultar_producto(id1));
            inv2.modificar_cantidad_poseida(id2, cantidad_comerciar, productos.consultar_producto(id2));

            // Avanzamos los dos iteradores
            it1++;
            it2++;
        }
        else if (id1 < id2) // Avanzamos el primer iterador ya que el producto con id1 no esta en inv2
        {
            it1++;
        }
        else // Avanzamos el segundo iterador ya que el producto con id2 no esta en inv
        {
            it2++;
        }

    }
}
```

Invariant

La invariant és que tots els elements “anteriors” als iteradors de cada inventari respectivament han estat modificats si és que podien fer-ho (si podien comerciar amb l’altre inventari).

Inici

A l'inici del programa inicialitzem 2 iteradors it1, it2 al principi de cada inventari (.begin()). D'aquesta forma no hi ha cap element anterior a ningun dels dos iteradors i com que encara no hem comercial de ninguna manera amb ningun element tenim que clarament es compleix la invariant.

```
map<int, Cantidad>::iterator it1 = inv.begin();
map<int, Cantidad>::iterator it2 = inv2.inv.begin();
```

Condició de finalització del bucle

La condició de sortida o finalització del bucle és que un dels dos iteradors hagi arribat al final del seu inventari (hagi arribat al inventari.end()).
Aquesta condició funciona ja que si un iterador ha arribat al final de l'inventari, segons la nostra invariant tots els elements de l'inventari ja han comercial si es que podien per tant aquest inventari no pot intercanviar més productes i en conseqüència l'altre inventari tampoc no pot intercanviar productes ja que depèn de l'altre per fer-ho

```
while (it1 != inv.end() and it2 != inv2.inv.end()) {
```

Cos del bucle

Primer de tot prenem l'identificador i la quantitat de producte dels elements corresponents a it1 i it2.

```
int id1 = it1->first;
Cantidad cantidad1 = it1->second;
int id2 = it2->first;
Cantidad cantidad2 = it2->second;
```

Seguidament hem de fer tres casos:

<u>Cas id1 = id2:</u>
En aquest cas com que els identificadors dels productes son iguals clarament volem que els dos inventaris s'intentin intercanviar productes.
Primer provem de que el primer inventari li passi productes al segon (agafem el mínim entre els productes que li sobren al primer inventari i els que li falten al segon) i modifiquem els inventaris.
Seguidament fem el procés invers ja que potser és l'altre inventari al que li sobren i el primer al que li'n falten.
Finalment avancem els dos iteradors ja que els dos elements actuals ja han intercanviat tots els productes que podien i per tant es segueix complint la invariant.

```
if (id1 == id2) // Si los dos inventarios contienen el producto con identificador id1=id2
{
    // Cojo el minimo entre lo que le falta a inv y lo que le sobra a inv2
    int cantidad_comerciar = calculo_comercio(
        cantidad1.poseido, cantidad1.requerido,
        cantidad2.poseido, cantidad2.requerido
    );
    // Comercio entre los dos inventarios la cantidad indicada
    modificar_cantidad_poseida(id1, cantidad_comerciar, productos.consultar_producto(id1));
    inv2.modificar_cantidad_poseida(id2, -cantidad_comerciar, productos.consultar_producto(id2));

    // Repito el proceso pero al reves
    cantidad_comerciar = calculo_comercio(
        cantidad2.poseido, cantidad2.requerido,
        cantidad1.poseido, cantidad1.requerido
    );
    modificar_cantidad_poseida(id1, -cantidad_comerciar, productos.consultar_producto(id1));
    inv2.modificar_cantidad_poseida(id2, cantidad_comerciar, productos.consultar_producto(id2));

    // Avanzamos los dos iteradores
    it1++;
    it2++;
}
```

Cas id1 < id2:
En aquest cas no podem comerciar amb els elements actuals ja que no corresponen al mateix producte.
Com que els elements dels inventaris estan ordenats per identificador de producte podem assegurar que el segon inventari no conté el producte amb identificador id1. Veiem-ho per reducció a l'absurd, suposem que el segon inventari si que contingues un producte amb identificador id1, llavors com que id2>id1 vol dir que el segon inventari ja ha passat aquest element però això no pot ser ja que llavors podríem haver comerciat amb el primer inventari però això no pot ser per la invariant per tant queda demostrat que el segon inventari no conté id1.
Al no contenir id1 llavors podem avançar it1 ja que l'element de id1 no pot comerciar amb el segon inventari i per tant se segueix complint la invariant.

Cas id1 < id2:
Aquest cas és simètric a l'anterior però intercanviant el paper dels inventaris.

Final

Un cop hem sortit del bucle tenim que almenys un iterador ja ha arribat al final de l'inventari i per tant com hem vist a un apartat anterior, l'altre inventari tampoc no pot intercanviar més productes d'aquesta manera ja s'han intercanviat tots els productes que podien fer-ho i es compleix la postcondició.

Finalment la funció simplement acaba.

Funció recursiva (calcular_viaje)

```
void Cuenca::calcular_viaje(const BinTree<string> &raiz, Barco &barco, vector<string> &ruta) const
{
    if (raiz.empty()) return;

    // Ciudad actual
    string ciudad_id = raiz.value();

    // Cojo los datos del barco
    int id_compra, num_compra, id_venta, num_venta;
    barco.consultar_barco(id_compra, num_compra, id_venta, num_venta);

    bool agregar = false;
    // Intento comprar y vender productos en la ciudad actual
    if (ciudades.contiene_producto(ciudad_id, id_compra))
    {
        // Calculo la cantidad maxima que puedo comprar
        int cantidad_comprar =
            max(0,
                min(
                    num_compra,
                    ciudades.consultar_cantidad_poseida(ciudad_id, id_compra) -
                    ciudades.consultar_cantidad_requerida(ciudad_id, id_compra)
                )
            );
        // Actualizo el numero de productos que me quedan por comprar
        num_compra -= cantidad_comprar;
        // Miro si he comerciado en esta ciudad
        if (cantidad_comprar > 0)
            agregar = true;
    }
    if (ciudades.contiene_producto(ciudad_id, id_venta))
    {
        // Calculo la cantidad maxima que puedo vender
        int cantidad_vender =
            max(0,
                min(
                    num_venta,
                    ciudades.consultar_cantidad_requerida(ciudad_id, id_venta) -
                    ciudades.consultar_cantidad_poseida(ciudad_id, id_venta)
                )
            );
        // Actualizo el numero de productos que me quedan por vender
        num_venta -= cantidad_vender;
        // Miro si he comerciado en esta ciudad
        if (cantidad_vender > 0)
            agregar = true;
    }
```

```
    // Modifico el barco con los nuevos valores una vez he comerciado
    barco.modificar_barco(id_compra, num_compra, id_venta, num_venta);

    Barco left_barco  = barco;
    vector<string> left_ruta;
    Barco right_barco = barco;
    vector<string> right_ruta;

    // Recursivamente calculo el viaje de la izquierda y de la derecha
    calcular_viaje(raiz.left(), left_barco, left_ruta);
    calcular_viaje(raiz.right(), right_barco, right_ruta);

    // Miro que ruta es mejor
    if (left_barco.restante() < right_barco.restante() or (left_barco.restante() == right_barco.restante() and left_ruta.size() <= right_ruta.size()))
    {
        barco = left_barco;
        ruta.swap(left_ruta);
    }
    else
    {
        barco = right_barco;
        ruta.swap(right_ruta);
    }

    // Si he comerciado en esta ciudad o no es la ultima en la que he comerciado entonces la agrego a la ruta
    if (not ruta.empty() or agregar)
    {
        ruta.push_back(ciudad_id);
    }
}
```

Invariant

La invariant és que en qualsevol node, al cridar la funció recursiva a left() i a right(), el vector de ruta passat per referència conté les ciutats per les quals passa el barco en la ruta òptima considerant els nodes des de l'arrel fins el node actual i el subarbre de left()/right(), i barco conté els número de productes que li falta per comprar i vendre en la ruta òptima considerant els nodes des de l'arrel fins el node actual i el subarbre de left()/right().

Cas base

El cas base és que si el node actual no existeix, és a dir, es compleix .empty() llavors no fem res ja que no podem comerciar amb cap ciutat ja que el subarbre està buit.

```
if (raiz.empty()) return;
```

Cos part 1

Primerament comerciarem de forma ficticia amb la ciutat actual per a que es compleixi la primera part de la invariant (**considerant els nodes des de l'arrel fins el node actual** i el subarbre de left()/right()), diem de forma ficticia ja que podria ser que la ruta óptima no passes pel node actual llavors simplement calculem quants productes es podrien intercanviar entre el barco i la ciutat actual si la ruta òptima passes per la ciutat actual. El comerci fictici el fem notar modificant el barco.
De forma similar a la funció “comerciar” l’intercanvi es fa agafant el mínim entre el nombre de productes que li falta per comprar/vendre al barco i el numero de productes que li sobren/falten a la ciutat actual,

```
// Ciudad actual
string ciudad_id = raiz.value();

// Cojo los datos del barco
int id_compra, num_compra, id_venta, num_venta;
barco.consultar_barco(id_compra, num_compra, id_venta, num_venta);

bool agregar = false;
// Intento comprar y vender productos en la ciudad actual
if (ciudades.contiene_producto(ciudad_id, id_compra))
{
    // Calculo la cantidad maxima que puedo comprar
    int cantidad_comprar =
        max(0,
            min(
                num_compra,
                ciudades.consultar_cantidad_poseida(ciudad_id, id_compra) -
                ciudades.consultar_cantidad_requerida(ciudad_id, id_compra)
            )
        );
    // Actualizo el numero de productos que me quedan por comprar
    num_compra -= cantidad_comprar;
    // Miro si he comerciado en esta ciudad
    if (cantidad_comprar > 0)
        agregar = true;
}
if (ciudades.contiene_producto(ciudad_id, id_venta))
{
    // Calculo la cantidad maxima que puedo vender
    int cantidad_vender =
        max(0,
            min(
                num_venta,
                ciudades.consultar_cantidad_requerida(ciudad_id, id_venta) -
                ciudades.consultar_cantidad_poseida(ciudad_id, id_venta)
            )
        );
    // Actualizo el numero de productos que me quedan por vender
    num_venta -= cantidad_vender;
    // Miro si he comerciado en esta ciudad
    if (cantidad_vender > 0)
        agregar = true;
}

// Modifico el barco con los nuevos valores una vez he comerciado
barco.modificar_barco(id_compra, num_compra, id_venta, num_venta);
```

Crida recursiva

Per a la crida recursiva el que fem es crear còpies del barco actual modificat pels comerços des de l'arrel fins el node actual i crear dos vectors més on guardarem la ruta òptima en els subarbres dels fills.
Passem el barco i el vector de ruta per referència per tal que al acabar la crida recursiva tinguem les dades òptimes del viatge en els subarbres.

```
    Barco left_barco  = barco;
    vector<string> left_ruta;
    Barco right_barco = barco;
    vector<string> right_ruta;

    // Recursivamente calculo el viaje de la izquierda y de la derecha
    calcular_viaje(raiz.left(), left_barco, left_ruta);
    calcular_viaje(raiz.right(), right_barco, right_ruta);
```

Cos part 2

Per a que es segueixi complint la invariant hem de triar quin viatge és el més òptim dels dos fills left() i right() per tant mirem primer quin dels dos ha fet que el barco pugui comprar i vendre una suma total de productes major, en cas d'empat agafa la ruta en la qual l'última ciutat que ha comerciat és més propera al node actual (està més aprop) i en cas d'empat s'agafa el viatge del fill esquerra.
Un cop sabem quin dels dos subarbres ha fet un viatge més òptim agafem els paràmetres (barco i vector ruta) corresponent i son els que compleixen la invariant.

Finalment cal mirar si hem comerciat en la ciutat actual (agregar és cert) o si hem comerciat ja en el subarbre (ruta no és buida) i si alguna de les dues coses es compleixen, afegim al vector ruta la ciutat actual. Aquest pas és important ja que ens ajuda a comprobar la part de quina ruta acaba en una ciutat més propera (condició mencionada en aquest apartat).

```
// Miro que ruta es mejor
if (left_barco.restante() < right_barco.restante() or ((left_barco.restante() == right_barco.restante()
and left_ruta.size() <= right_ruta.size())))
{
    barco = left_barco;
    ruta.swap(left_ruta);
}
else
{
    barco = right_barco;
    ruta.swap(right_ruta);
}

// Si he comerciado en esta ciudad o no es la ultima en la que he comerciado entonces la agrego a la ruta
if (not ruta.empty() or agregar)
{
    ruta.push_back(ciudad_id);
}
```

Final

Un cop s'acaba la funció, com que haviem cridat la funció des de l'arrel de la "cuenca" llavors per la invariant tenim que el vector ruta i el barco contenen les dades del viatge òptim tenint en compte tota la cuenca i ja he acabat.