El objetivo de este documento es la justificación de la función recursiva hacer_viaje:

```
BinTree<Barco::InfoNodo> Barco::travelled_tree_rec(BinTree<string> mapa_rio, map<string, Ciudad>&
lista_ciudades, int unidades_comprar_barco, int unidades_vender_barco, list<string>& ruta) {
    //Base case
    if(mapa_rio.empty() or (unidades_comprar_barco <= 0 and unidades_vender_barco <= 0)) {
        return BinTree<InfoNodo> (InfoNodo(), BinTree<InfoNodo>(), BinTree<InfoNodo>());
    }

    InfoNodo node = InfoNodo();

    string id_ciudad = mapa_rio.value();
    Ciudad city = lista_ciudades[id_ciudad];

    if(city.contiene_producto(producto_a_comprar)) {
        int cantidad_vender = city.exceso(producto_a_comprar);

        //La ciudad quiere vender este producto.
        if(cantidad_vender > 0) {
            int cantidad = min(cantidad_vender, unidades_comprar_barco);

            node.venta = cantidad;
            node.total_venta = cantidad;
            unidades_comprar_barco -= cantidad;
        }
    }

    if(city.contiene_producto(producto_a_vender)) {
        int cantidad_comprar = city.exceso(producto_a_vender);

        if(cantidad_comprar < 0) {
            int cantidad = min(abs(cantidad_comprar), unidades_vender_barco);

            node.compra = cantidad;
            node.total_compra = cantidad;
            unidades_vender_barco -= cantidad;
        }
    }

    node.total_trato = node.compra + node.venta;

    list<string> rutaleft, rutaright;

    auto tleft = travelled_tree_rec(mapa_rio.left(), lista_ciudades, unidades_comprar_barco,
unidades_vender_barco, rutaleft);
    auto tright = travelled_tree_rec(mapa_rio.right(), lista_ciudades, unidades_comprar_barco,
unidades_vender_barco, rutaright);

    int totalleft = 0, totalright = 0;

    if(not tleft.empty()) {
        totalleft = tleft.value().total_trato;
    }

    if(not tright.empty()) {
```

```
    totalright = tright.value().total_trato;
}

if(totalleft == 0 and totalright == 0) {
    node.total_compra = node.compra;
    node.total_venta = node.venta;

    if(node.total_compra + node.total_venta > 0) node.altura = 1;
    else node.altura = 0;
}
else if(totalleft > totalright) {

    node.altura = tleft.value().altura + 1;
    node.total_trato += totalleft;
    node.total_compra += tleft.value().total_compra;
    node.total_venta += tleft.value().total_venta;

    ruta = rutaleft;
    ruta.push_front(mapa_rio.left().value());

}
else if(totalleft < totalright) {

    node.total_trato += totalright;
    node.altura = tright.value().altura + 1;
    node.total_compra += tright.value().total_compra;
    node.total_venta += tright.value().total_venta;

    ruta = rutaright;
    ruta.push_front(mapa_rio.right().value());

}
else {
    if(tleft.value().altura > tright.value().altura) {

        node.total_trato += totalright;
        node.altura = tright.value().altura + 1;
        node.total_compra += tright.value().total_compra;
        node.total_venta += tright.value().total_venta;

        ruta = rutaright;
        ruta.push_front(mapa_rio.right().value());

    }
    else {
        node.total_trato += totalleft;
        node.altura = tleft.value().altura + 1;
        node.total_compra += tleft.value().total_compra;
        node.total_venta += tleft.value().total_venta;

        ruta = rutaleft;
        ruta.push_front(mapa_rio.left().value());

    }
}
```

```
    return BinTree<InfoNodo> (node, tleft, tright);
}
```

PRECONDICIÓN: La precondición de la función recursiva es TRUE.

POSTCONDICIÓN: Devuelve un árbol que tiene un Struct como nodo. Este árbol puede tener el mismo tamaño que el del mapa_rio o puede ser más pequeño.

CASO BASE: Hay dos casos base:

1. Caso 1: mapa_rio.empty()
   Este caso se da cuando la llamada anterior trataba un nodo que era una hoja, y se ha llamado la función para la rama izquierda o para la rama derecha, que ambos son árboles binarios vacíos; no son ciudades y no tendrán productos con quienes pueden comerciar con el barco.
   Entonces se debe hacer una llamada return de un árbol vacío.

2. Caso 2: unidades_comprar_barco <= 0 and unidades_vender_barco <= 0
   Este caso se da cuando en la llamada anterior el barco ha agotado todos los productos que tenía: ha conseguido comprar y vender todos los productos que necesitaba. Así pues, no es necesario seguir recorriendo el árbol, dado que ya no se harán más tratos.
   Entonces, igual que el caso anterior, se deberá retornar un árbol vacío.

¿PRECONDICIÓN AND CASO BASE => POSTCONDICIÓN?
La precondición es TRUE, entonces será un elemento neutro en la evaluación del AND. El caso base, que son dos, nos indica el final de la llamada recursiva. Así pues, hay dos casos:

1. Caso 1: mapa_rio.empty()
   Se ha llegado a una rama que es vacía, indicándonos que se habrá recorrido todo el árbol. Entonces se devuelve un árbol vacío. Por lo tanto, la función recursiva ha devuelto un árbol que contiene el mismo número de nodos que mapa_rio, tal que los valores de los nodos son Structs.
   Por lo tanto, se puede decir que la postcondición se ha cumplido.

2. Caso 2: unidades_comprar_barco <= 0 and unidades_vender_barco <= 0
   Previamente se ha llegado a un nodo tal que los productos del barco se han agotado. Entonces, en la llamada en que nos encontramos ya no es necesario evaluar el nodo, y se devuelve un árbol vacío.
   Así pues, la función recursiva ha devuelto un árbol que tiene menor número de nodos que mapa_rio, así cumpliéndose la postcondición.

CASO RECURSIVO: Si el árbol no es vacío y el barco aún tiene productos con los que puede comerciar, entonces primeramente se evalúa en pre orden el árbol:

1. Se analiza si la ciudad puede vender el producto que quiere comprar el barco. Esto significa que el exceso de la ciudad debe de ser positiva. Si se da este caso, entonces se debe evaluar la cantidad más pequeña: las unidades que quiere vender el barco o la cantidad que quiere comprar la ciudad.
   Es necesario restar de unidades_comprar_barco dicha cantidad.

2. Se analiza si la ciudad puede comprar el producto que quiere vender el barco. Esto significa que el exceso de la ciudad debe de ser negativa. Si se da el caso, entonces se dará lugar un trato con el barco, así evaluando la cantidad más pequeña entre el valor absoluto del exceso y unidades_vender_barco.
   Al final será necesario restar este valor de unidades_vender_barco.

En ambos casos, si hay un trato, además de restar ciertos valores de unidades_comprar_barco y unidades_vender_barco, será necesario inicializar/modificar las variables de los structs que serán los

valores de los nodos del árbol binario que devuelve la función recursiva.

Las variables más importantes son los de acumulación y la variable de altura, aunque estas se tratarán a posteriori.

Entonces, es necesario crear dos variables locales: rutaleft y rutaright, que son las rutas de la izquierda y de la derecha, que almacenan aquellas ciudades con quienes el barco ha hecho tratos.

Con esto hecho, entonces se hacen las dos llamadas recursivas para la rama izquierda (otro árbol) y para la rama derecha (otro árbol).

Como es necesario tener en cuenta la longitud de la ruta, que será la altura del árbol, es necesario ahora tratarlo en postorden, a diferencia de los casos anteriores, que trataban el árbol en pre orden.

Por lo tanto, ahora será necesario evaluar los nodos, que tendrán almacenados los tratos, igual que la altura, en addición a la lista de ciudades que contienen aquellas con quienes ha comerciado el barco.

Primero tratamos los tratos acumulados del nodo de la izquierda y de la derecha, primero comprobando que no son vacíos.

1. Caso 1: los nodos son vacíos.
   Esto significa que los tratos son 0, por lo tanto, nos encontramos en la última ciudad de la ruta: la cantidad acumulada de los tratos es la cantidad que compra y/o vende la ciudad al barco.
   Este nodo, entonces, tendrá altura 1.

2. Caso 2: los nodos no son vacíos y los tratos totales acumulados son diferentes.
   Se prende como ruta aquel nodo que tiene mayor trato. Si el nodo de la izquierda tiene un trato total acumulado mayor que el nodo de la derecha, entonces se suma 1 a la altura del nodo de la izquierda, y el trato total del nodo en el que nos encontramos se le suma aquello que se ha intercambiado con el barco.
   Además de tratar el struct del nodo, será necesario asignar la variable ruta pasada como referencia a la función rutaleft, dado que la ruta debe ir hacia la izquierda del árbol (o la derecha dependiendo de como se mire el árbol: si la desembocadura se encuentra arriba del árbol, será la izquierda. Si se mira del revés, se deberá ir hacia la derecha).
   Si el nodo de la derecha tiene un trato total acumulado mayor que el nodo de la izquierda, se hace el mismo proceso pero viceversa.

3. Caso 3: los tratos totales acumulados de los nodos son iguales.
   Será necesario evaluar qué ruta es la más corta: si el nodo de la derecha tiene una altura menor que el nodo de la derecha, entonces se realizan los mismos pasos que el caso anterior, teniendo en cuenta que la ruta debe pasar por el nodo de la derecha.
   En los otros dos casos (el nodo de la izquierda tiene menor o igual altura que el nodo de la derecha) se deberá escoger el nodo de la izquierda como ciudad que pertenece a la ruta por la cual pasa el barco.

Al final debemos devolver el árbol con el nuevo nodo que hemos modificado: aquello que hemos hecho en pre orden y aquello que hemos hecho en postorden. Las ramas izquierdad y derecha seran aquellas llamadas recursivas que hemos almacenado en variables de tipo BinTree<InfoNodo>.

DECRECIMIENTO: Como en cada llamada recursiva se llama la función con mapa_rio.left() o mapa_rio.right(), el árbol cada vez se hace más pequeño. Además, si en el nodo hay intercambio con el barco, las unidades totales de los productos con los cuales puede comerciar el barco también se reducirán, así aumentando la posibilidad de que la ruta no llegue hasta las hojas del árbol.

Es importante saber que durante la justificación de este problema se ha analizado el río teniendo en cuenta que la desembocadura era la raíz del árbol, que todas las ramas iban hacia abajo.