

Justificación de Métodos

Edgar Aguadé Nadal

Mayo 2024

```
void City::trade(City &visitor_city, const ProductSet &product_set);
```

Precondición Cierta.

Postcondición Las ciudades han comerciado entre sí todos aquellos productos posibles. O de un modo más formal:

Sean:

- $city_{local} :=$ Parámetro implícito
- $id(product) :=$ El identificador del producto "product"
- $exc(product) :=$ Exceso del producto "product"

$\forall product_{local} \in city_{local} \forall product_{visitor} \in city_{visitor}$ tales que:

$$id(product_{local}) = id(product_{visitor}) \wedge exc(product_{local}) * exc(product_{visitor}) < 0$$

Se cumple que, siendo $t := \min(|exc(product_{local})|, |exc(product_{visitor})|)$
 $exc(product_{local}) > 0 \implies own_{local} = own_{local} - t \wedge own_{visitor} = own_{visitor} + t$
 $exc(product_{local}) < 0 \implies own_{local} = own_{local} + t \wedge own_{visitor} = own_{visitor} - t$

O lo que viene a ser lo mismo, todos aquellos productos donde una ciudad tenga más de las que necesita, la otra menos de las que necesita y viceversa, han sido comerciados. La cantidad comerciada "t" es el mínimo de los excedentes, pues no puede ni haber superávits ni que una ciudad que tiene exceso quede con menos unidades de las que necesita.

Invariante

- $city_{local}.begin() \leq product_{local} \leq city_{local}.end()$
- $city_{visitor}.begin() \leq product_{visitor} \leq city_{visitor}.end()$
- $\forall product_i < product_{local} \in city_{local} \forall product_j < product_{visitor} \in city_{visitor}$
 $city_{local}, city_{visitor}$ han intercambiado dichos productos cumpliendo la Post-condición.

1 Justificación

1.1 Inicializaciones

Inicialmente no se ha comercializado ningún producto, por lo que inicializamos ambos iteradores a al inicio de sus respectivos inventarios: $product_{local} = city_{local}.begin() \wedge product_{visitor} = city_{visitor}.begin()$, cumpliendo así las dos partes del *Invariante*, por un lado que ambos iteradores se encuentran entre el principio y el final, y, por otro lado, que todos los productos anteriores a los iteradores han sido comercializados.

1.2 Codición de salida

Se puede salir del bucle por dos razones:

- $product_{local} = city_{local}.end()$

Lo cual indicaría, por el invariante, que hemos explorado todos los productos de $city_{local}$ y, en consecuencia, todos los de $city_{visitor}$ tales que $id(product_j) \leq id(product_{local})$.

Por lo tanto, $\forall product_j$ tal que $id(product_j) > id(product_{local}) \nexists product_i$ tal que $id(product_i) = id(product_j)$

De lo contrario significaría que $\exists product_j$ tal que $id(product_j) > product_{local} \wedge \exists product_i$ tal que $id(product_i) = id(product_j)$. Por el invariante obtenemos que $product_i \geq city_{local}.end()$ lo cual es absurdo pues no hay más productos después del final del inventario.

- $product_{visitor} = city_{visitor}.end()$

Completamente análogo a lo acabado de demostrar.

1.3 Cuerpo del bucle

Para analizar correctamente el cuerpo del bucle diferenciaremos claramente en tres casos:

- $id(product_{local}) < id(product_{visitor})$

Por el *Invariante* todos los productos anteriores a $product_{local}, product_{visitor}$ se han intercambiado si cumplan con $id(product_{local}) = id(product_{visitor})$. En estar ordenados de menor a mayor índice deducimos que $\nexists product_j \in city_{visitor}$ tal que $id(product_j) < id(product_{visitor}) \wedge id(product_{local}) = id(product_j)$ De lo contrario deduciríamos que $\exists product_j \in city_{visitor}$ tal que $id(product_j) < id(product_{visitor})$ que no ha comercializado, absurdo por el *Invariante*. De este modo, avanzamos/descartamos $product_{local}$ puesto que estamos seguros que ese producto no va a ser comercializado.

- $id(product_{local}) > id(product_{visitor})$

$id(product_{local}) > id(product_{visitor}) \iff id(product_{visitor}) < id(product_{local})$ y por lo tanto, análogo al caso anterior.

- $id(product_{local}) = id(product_{visitor})$

Para que se vuelva a cumplir el *Invariante*, las ciudades deben comerciar si se cumple $exc(product_{local}) * exc(product_{visitor}) < 0$. Con este objetivo precalculamos "t" la cantidad transaccionada. del modo que:

$$t := \min(|exc(product_{local})|, |exc(product_{visitor})|)$$

De este modo podemos ver que actuamos según dos casos:

$$\begin{aligned} exc(product_{local}) > 0 &\implies exc(product_{visitor}) < 0 \text{ por lo que } city_{local} \text{ vende y} \\ city_{visitor} \text{ compra: } own_{local} &= own_{local} - t \wedge own_{visitor} = own_{visitor} + t \end{aligned}$$

Por otro lado...

$$\begin{aligned} exc(product_{local}) < 0 &\implies exc(product_{visitor}) > 0 \text{ por lo que } city_{local} \text{ compra y} \\ city_{visitor} \text{ vende: } own_{local} &= own_{local} + t \wedge own_{visitor} = own_{visitor} - t \end{aligned}$$

A su vez, en cada paso actualizamos los pesos y volúmenes correspondientes de las ciudades.

Por último, avanzamos $product_{local}, product_{visitor}$ para cumplir con el *Invariante*.

1.4 Finalización

Como podemos apreciar, en cada iteración o avanzamos $product_{local}$ o avanzamos $product_{visitor}$ o ambos simultáneamente. Por lo que la distancia entre estos y $city_{local}.end(), city_{visitor}.end()$ respectivamente, disminuye en cada iteración, de lo que deducimos que eventualmente el bucle finalizará.

```
void River::findOptimalRoute(BinTree<string> structure, Travel
                             current_travel, Travel &best_travel) const
```

Precondición $travel_{current}$ contiene el estado del viaje realizado para llegar a la ciudad actual.

Postcondición $travel_{best}$ guarda el mejor viaje realizado hasta el momento y $travel_{current}$ contiene el estado del viaje realizado una vez pasada a la ciudad actual.

2 Justificación

2.1 Caso Base

2.1.1 $structure.empty()$

$structure.empty() \iff \neg structure.left(), structure.right() \iff$ No hay más nodos que explorar y por lo tanto se ha acabado el camino.

En este caso no se actualiza $travel_{best}$ ni $travel_{current}$ por lo que sigue cumpliendo la *Postcondición*.

2.1.2 $\neg structure.empty()$

$\neg structure.empty() \implies structure.value()$ es una ciudad de $travel_{current} \implies$ la longitud del viaje aumenta \wedge se debe intentar comerciar con dicha ciudad.

En haber cambiado el estado de $travel_{current} \implies$ hay que asegurar que $travel_{best}$ sigue siendo el mejor viaje por lo que comparamos ambos viajes y en caso que $travel_{current}$ sea mejor lo remplazamos. Así pues, se cumple la *Postcondición*.

2.2 Caso Inductivo

Hipótesis El árbol no es vacío. Es decir, $\neg \text{structure.empty}()$

Aislado el caso directo en que $\text{travel}_{\text{current}}.\text{objectiveAchieved}()$ es cierto (el barco ya ha vendido y comprado todo lo posible y por lo tanto no debe continuar la ruta), pues en este caso $\neg \text{structure.empty}() \implies \text{Postcondición}$. Veamos entonces el caso inductivo que es el interesante:

$\neg \text{structure.empty}() \implies \exists \text{structure.left}() \wedge \text{structure.right}()$ Por lo que podemos ver dos casos

- $\text{structure.left().empty}() \wedge \text{structure.right().empty}()$

En este caso no podemos continuar el camino, puesto que ya hemos llegado al final del árbol. Por otro lado, en ser $\neg \text{structure.empty}() \implies \text{Postcondición}$.

- $\neg \text{structure.left().empty}() \wedge \neg \text{structure.right().empty}()$

$\implies \exists \text{structure.left().value}(), \text{structure.right().value}()$ Por lo tanto, como se cumple la *Precondición* se cumplirá la *Postcondición*.

Decrecimiento

$\text{structure.left}(), \text{structure.right}() \subseteq \text{structure}$ por lo que las llamadas se hacen cada vez con un árbol más pequeño, asegurando así que eventualmente se llega a una hoja/final.