TestBruteForceAlgorithm

should_work_return_max_is_0_95()

- **Objeto de la prueba:** Verificar que el coste calculado esté entre 0.94 y 0.96.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType1()`.
- **Valores estudiados:** Estrategia de caja gris. Se verifica que el algoritmo BruteForce calcule el coste correctamente dentro de los límites establecidos.
- **Operativa:** Se proporcionan lista de productos y matriz de similitud como parámetros. Se comprueba que el coste esté en el rango esperado.

should_work_return_max_is_5_48()

- **Objeto de la prueba:** Verificar que el coste calculado esté entre 5.47 y 5.49.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType2()`.
- **Valores estudiados:** Estrategia de caja gris para validar la precisión del coste calculado.
- **Operativa:** Utiliza el algoritmo BruteForce con productos y matriz de similitud. Comprueba que el coste esté dentro del rango especificado.

should_work_shelf_has_one_product_return_max_is_0()

- **Objeto de la prueba:** Comprobar que con un único producto, el coste sea 0.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType3()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Se pasa un producto y matriz de similitud. El resultado del coste debe ser 0.

should_work_shelf_has_two_products_return_max_is_0_4()

- **Objeto de la prueba:** Verificar que con dos productos el coste sea 0.4.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType4()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Se proporciona lista de productos y matriz de similitud, comprobando que el coste sea 0.4.

should_work_return_max_is_4_20()

- **Objeto de la prueba:** Verificar que el coste calculado sea 4.2.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType5()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Calcula el coste con una configuración específica de matriz de similitud.

should_work_return_max_is_2_40()

- **Objeto de la prueba:** Verificar que el coste calculado sea 2.4.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType7()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Se comprueba que el coste sea 2.4, con una tolerancia de 0.001.

should_work_return_max_is_4_60()

- **Objeto de la prueba:** Verificar que el coste calculado sea 4.6.
- **Archivos de datos necesarios:** Datos introducidos manualmente con `createInputType8()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Verifica que el coste es 4.6, con una tolerancia de 0.001.

test_validateParameters_when_products_is_null()

- **Objeto de la prueba:** Comprobar que se lanza una excepción si `products` es nulo.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define `products` como nulo y comprueba el error.

test_validateParameters_when_products_is_empty()

- **Objeto de la prueba:** Verificar que una lista vacía de productos genera un error.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define `products` como lista vacía y verifica la excepción.

test_validateParameters_when_simmatrix_is_null()

- **Objeto de la prueba:** Comprobar que se lanza un error si `simMatrix` es nulo.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define `simMatrix` como nulo y espera el error correspondiente.

test_validateParameters_when_product_size_is_different_to_simmatrix_size()

- **Objeto de la prueba:** Verificar que un tamaño diferente entre productos y matriz genera un error.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define una matriz de tamaño incompatible y espera la excepción.

test_validateParameters_when_product_size_is_different_to_row_simmatrix_size()

- **Objeto de la prueba:** Comprobar que el tamaño de filas diferente al de productos lanza error.

- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define una matriz inconsistente y espera el error.

TestEdge

testGetProduct1()

- **Objeto de la prueba:** Verificar que `getProduct1()` devuelve el producto 1.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Verifica que `getProduct1()` retorna el producto correcto.

testGetProduct2()

- **Objeto de la prueba:** Verificar que `getProduct2()` devuelve el producto 2.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Comprueba que `getProduct2()` devuelve el producto 2.

testGetCost()

- **Objeto de la prueba:** Comprobar que `getCost()` devuelve el coste correctamente.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Verifica que el valor del coste es el esperado, con una tolerancia de 0.0001.

testCompareToWithEqualCosts()

- **Objeto de la prueba:** Verificar la comparación de costes iguales.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Compara dos `Edge` con el mismo coste y verifica que el resultado es 0.

testCompareToWithLesserCosts()

- **Objeto de la prueba:** Verificar que un coste menor es comparado correctamente.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Verifica que la comparación es menor a 0 cuando el primer coste es menor.

testCompareToWithGreaterCosts()

- **Objeto de la prueba:** Verificar que un coste mayor es comparado correctamente.
- **Archivos de datos necesarios:** Ninguno.

- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Verifica que la comparación es mayor a 0 cuando el primer coste es mayor.

testToString()

- **Objeto de la prueba:** Verificar que `toString()` devuelve la representación esperada.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Comprueba que `toString()` devuelva el formato correcto de la instancia `Edge`.

TestShelfOrganizer

test_validateParameters_when_products_is_null()

- **Objeto de la prueba:** Verificar que `products` nulo lanza una excepción.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define `products` como nulo y comprueba que se lanza la excepción.

test_validateParameters_when_products_is_empty()

- **Objeto de la prueba:** Comprobar que una lista vacía de productos genera un error.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define `products` como lista vacía y verifica que se lanza una excepción.

test_validateParameters_when_simmatrix_is_null()

- **Objeto de la prueba:** Comprobar que `simMatrix` nulo lanza una excepción.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define `simMatrix` como nulo y espera la excepción.

test_validateParameters_when_product_size_is_different_to_simmatrix_size()

- **Objeto de la prueba:** Verificar que se lanza un error si el tamaño de `products` y `simMatrix` no coinciden.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define una matriz con tamaño incorrecto y espera el error.

test_validateParameters_when_product_size_is_different_to_row_simmatrix_size()

- **Objeto de la prueba:** Verificar que las filas de `simMatrix` coincidan con `products`.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Define una matriz inconsistente en el tamaño de las filas y espera la excepción.

TestTwoAproxAlgorithm

should_work_return_max_is_between_0_475_and_0_95()

- **Objeto de la prueba:** Verificar que el coste calculado está entre 0.475 y 0.95.
- **Archivos de datos necesarios:** Datos con `createInputType1()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Utiliza `TwoAproxAlgorithm` y comprueba que el coste esté en el rango.

should_return_triangle_inequality_violation_1()

- **Objeto de la prueba:** Verificar la violación de la desigualdad triangular.
- **Archivos de datos necesarios:** Datos con `createInputType2()`.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Usa `isTriangleInequalityPropertyTrue` y comprueba que devuelva `false`.

should_work_shelf_has_one_product_return_max_is_0()

- **Objeto de la prueba:** Comprobar que el coste sea 0 con un único producto.
- **Archivos de datos necesarios:** Datos con `createInputType3()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Calcula el coste con un producto y espera 0 como resultado.

should_work_shelf_has_two_products_return_max_is_between_0_2_and_0_4()

- **Objeto de la prueba:** Verificar que el coste esté entre 0.2 y 0.4 con dos productos.
- **Archivos de datos necesarios:** Datos con `createInputType4()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Calcula el coste y verifica el rango especificado.

should_return_triangle_inequality_violation_2()

- **Objeto de la prueba:** Verificar otra violación de la desigualdad triangular.
- **Archivos de datos necesarios:** Datos con `createInputType5()`.
- **Valores estudiados:** Estrategia de caja negra.

- **Operativa:** Comprueba que `isTriangleInequalityPropertyTrue` retorna `false`.

should_work__return_max_is_between_1_2_and_2_5()

- **Objeto de la prueba:** Verificar que el coste está entre 1.3 y 2.5.
- **Archivos de datos necesarios:** Datos con `createInputType7()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Verifica que el coste calculado esté en el rango.

should_work__return_max_is_between_2_3_and_4_7()

- **Objeto de la prueba:** Verificar que el coste calculado esté entre 2.3 y 4.7.
- **Archivos de datos necesarios:** Datos con `createInputType8()`.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Comprueba que el coste esté en el rango.

TestUnionFind

testFind()

- **Objeto de la prueba:** Comprobar que los componentes iniciales no están unidos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Instancia `UnionFind` y comprueba que los elementos no están conectados.

testUnion()

- **Objeto de la prueba:** Verificar que `union()` conecta correctamente dos elementos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Conecta dos elementos y verifica la conexión con `find()`

TestProduct

testConstructor()

- **Objeto de la prueba:** Verificar que el constructor de `Product` inicialice correctamente los atributos `id`, `name` y `price`.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.

- **Operativa:** Se crea una instancia de `Product` y se comprueba que los valores se inicializan correctamente.

testSetPrice()

- **Objeto de la prueba:** Verificar que el método `setPrice` actualice correctamente el precio.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se modifica el precio de un producto y se comprueba que el nuevo precio es el esperado.

testToString()

- **Objeto de la prueba:** Verificar que el método `toString` devuelva una representación adecuada del producto.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se comprueba que `toString` devuelve el formato de cadena esperado.

TestShelf

testConstructor()

- **Objeto de la prueba:** Verificar que el constructor de `Shelf` inicialice correctamente el nombre y los productos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se crea una `Shelf` vacía y se verifica que el nombre y la lista de productos sean correctos.

testConstructorWithProducts()

- **Objeto de la prueba:** Verificar que el constructor de `Shelf` filtre duplicados en la lista de productos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Se crea una `Shelf` con productos duplicados y se verifica que solo se mantengan los únicos.

testAddProduct()

- **Objeto de la prueba:** Verificar que `addProduct` añada productos correctamente y evite duplicados.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.

- **Operativa:** Se añade un producto y se verifica que la operación solo sea exitosa la primera vez.

testRemoveProduct()

- **Objeto de la prueba:** Verificar que `removeProduct` elimine un producto existente y no elimine productos inexistentes.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se elimina un producto y se verifica que no pueda eliminarse nuevamente.

testGetProducts()

- **Objeto de la prueba:** Verificar que `getProducts` devuelva la lista correcta de productos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se añaden productos y se verifica que `getProducts` los contenga.

testEmptyShelf()

- **Objeto de la prueba:** Verificar que `Shelf` vacía no contenga productos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se crea una estantería vacía y se verifica que `getProducts` sea una lista vacía.

testRemoveNonExistentProduct()

- **Objeto de la prueba:** Verificar que `removeProduct` devuelva `false` al intentar eliminar un producto inexistente.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Intenta eliminar un producto inexistente y verifica el resultado.

testOrderOfProducts()

- **Objeto de la prueba:** Verificar que los productos se mantengan en el orden adecuado.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Añade productos y verifica que el orden sea el esperado.

testClearShelf()

- **Objeto de la prueba:** Verificar que `clearShelf` vacíe correctamente la lista de productos.
- **Archivos de datos necesarios:** Ninguno.

- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Vacía la estantería y verifica que `getProducts` sea una lista vacía.

testShelfSize()

- **Objeto de la prueba:** Verificar que `size` devuelva el tamaño correcto de la estantería.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Añade productos y verifica que `size` devuelva el tamaño esperado.

TestSimilarityMatrix

testAddProduct()

- **Objeto de la prueba:** Verificar que `addProduct` agregue correctamente productos a la matriz de similitud.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Añade productos y verifica que los valores por defecto en la matriz sean `0.0`.

testSetAndGetSimilarity()

- **Objeto de la prueba:** Verificar que `setSimilarity` establezca y `getSimilarity` obtenga la similitud entre productos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Establece la similitud entre productos y verifica que los valores en la matriz sean correctos.

testRemoveProduct()

- **Objeto de la prueba:** Verificar que `removeProduct` elimine productos correctamente de la matriz de similitud.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Elimina un producto y verifica que las filas y columnas correspondientes se eliminen.

testInvalidIndex()

- **Objeto de la prueba:** Verificar que `getSimilarity` arroje una `ProductNotFoundException` si el producto no existe en la matriz.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.

- **Operativa:** Intenta acceder a índices inválidos y verifica que se lance la excepción esperada.

TestUser

testConstructor()

- **Objeto de la prueba:** Verificar que el constructor de `User` inicialice correctamente `name`, `username` y `password`.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja blanca.
- **Operativa:** Se crea una instancia de `User` y se comprueba que los valores de sus atributos sean correctos.

testConstructorWithInvalidName()

- **Objeto de la prueba:** Verificar que el constructor arroje `IllegalArgumentException` si `name` es inválido.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Intenta crear un `User` con un nombre vacío y verifica que se lance una excepción.

testConstructorWithInvalidUsername()

- **Objeto de la prueba:** Verificar que el constructor arroje `IllegalArgumentException` si `username` es inválido.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Intenta crear un `User` con un nombre de usuario vacío y verifica la excepción.

testConstructorWithInvalidPassword()

- **Objeto de la prueba:** Verificar que el constructor arroje `IllegalArgumentException` si `password` es inválida.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Intenta crear un `User` con una contraseña vacía y verifica que se lance la excepción.

testAuthenticate()

- **Objeto de la prueba:** Verificar que `authenticate` funcione correctamente con credenciales válidas e inválidas.
- **Archivos de datos necesarios:** Ninguno.

- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Prueba varias combinaciones de usuario/contraseña y verifica el resultado.

testUpdateUsername()

- **Objeto de la prueba:** Verificar que `setUsername` actualice el nombre de usuario y que falle con nombres inválidos.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja negra.
- **Operativa:** Actualiza el nombre de usuario y verifica que falle al intentar usar uno vacío.

testUpdatePassword()

- **Objeto de la prueba:** Verificar que `setPassword` actualice la contraseña correctamente y falle con la actual incorrecta.
- **Archivos de datos necesarios:** Ninguno.
- **Valores estudiados:** Estrategia de caja gris.
- **Operativa:** Cambia la contraseña y verifica el resultado. Intenta actualizar con la contraseña actual incorrecta y comprueba la excepción.

.