# Test Description

**Test Name or ID**: BB\_ findTruckAndDiversion

**Test Type**: Black Box

**Description**: To ensure the function returns either 1 or 0, depending on whether an order can be added to a truck. Furthermore, to ensure the function correctly calculates the diversion route based on the provided distances, updates the truck's weight and volume, and sets the “diversion” Route struct in the OrderInfo structure.

**Setup:**

1. Initialize a Dispatch struct (called org)
2. By accessing the values inside org, define the CurrentWeight and CurrentVolume values and routes for each truck.
3. Create a 2D array of double values with a size of 3x2 (called dists) to hold the distance and colour codes of each truck.
4. Populate the array with data: the trucks’ distances and colour codes
5. Create an order by using the OrderInfo struct.
6. Assign a pointer to the order.
7. Create the expected diversion route by defining a route struct with various points.
8. Call the findTruckAndDiversion and assign it to an integer (called done)
9. Create assertions to validate the expected results:
   1. Check if “done” equals the expected value
   2. Check CurrentWeight and CurrentVolume of the chosen truck matches the expected value
   3. Using a loop, check that the diversion data of the order matches the expected diversion route

**Test Function**: BB\_g findTruckAndDiversion

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Check function finds truck, updates the truck’s weight and volume when all trucks’ current capacities are 0. | org.current.B.CurrentWeight = 0;  org.current.B.CurrentVolume = 0;  org.current.G.CurrentWeight = 0;  org.current.G.CurrentVolume = 0;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  OrderInfo order = {20, .5, {11, 11}, {-1, -1}};    findTruckAndDiversion(&org, dists, &order); | Done = 1  Weight = 20  Volume = 0.5  All assertions return TRUE |  |  |
| Check function finds truck, updates the truck’s weight and volume when one distance has many decimal places. | org.current.B.CurrentWeight = 0;  org.current.B.CurrentVolume = 0;  org.current.G.CurrentWeight = 0;  org.current.G.CurrentVolume = 0;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  dists[1][0] = 2.2360679774997898;  dists[1][1] = BLUE;  dists[2][0] = 14.0;  dists[2][1] = GREEN;  dists[0][0] = 1.0;  dists[0][1] = YELLOW;  *struct* OrderInfo order = {500, .25, {18, 8}, {-1, -1}}; | Done = 1  Weight = 500  Volume = 25  All assertions return TRUE |  |  |
| Check function finds truck, updates the truck’s weight and volume when an order has a max weight the truck can hold. | org.current.B.CurrentWeight = 0;  org.current.B.CurrentVolume = 0;  org.current.G.CurrentWeight = 0;  org.current.G.CurrentVolume = 0;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  OrderInfo order = {1000, 1, {8, 24}, {-1, -1}};    findTruckAndDiversion(&org, dists, &order); | Done = 1  Weight = 1000  Volume = 1.0  All assertions return TRUE |  |  |
| Check function finds truck, updates the truck’s weight and volume, and adds diversion route when the current weight and volume of two trucks are 0. | org.current.B.CurrentWeight = 20;  org.current.B.CurrentVolume = 0.5;  org.current.G.CurrentWeight = 0;  org.current.G.CurrentVolume = 0;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  dists[1][0] = 10.0;  dists[1][1] = BLUE;  dists[0][0] = 2.0;  dists[0][1] = GREEN;  dists[2][0] = 12.0;  dists[2][1] = YELLOW;  OrderInfo order =  {200, 1, {7, 24}, {-1, -1}};  findTruckAndDiversion(&org, dists, orderTemp); | Done = 1  Weight = 200  Volume = 1  diversion.numPoints  = 7  diversion.routeSymbol  = (char)DIVERSION  expectedDiversion =  {{  {6, 19},  {6, 20},  {6, 21},  {6, 22},  {6, 23},  {6, 24},  {7, 24}},7, DIVERSION};  All assertions return TRUE |  |  |
| Check function finds truck, updates the truck’s weight and volume, and adds diversion route using all valid values. | org.current.B.CurrentWeight = 20;  org.current.B.CurrentVolume = 0.5;  org.current.G.CurrentWeight = 200;  org.current.G.CurrentVolume = 1;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  dists[1][0] = 10.0;  dists[1][1] = BLUE;  dists[0][0] = 2.0;  dists[0][1] = GREEN;  dists[2][0] = 12.0;  dists[2][1] = YELLOW;  OrderInfo order =  {500, 1, {7, 24}, {-1, -1}};  findTruckAndDiversion(&org, dists, orderTemp); | Done = 1  Weight = 700  Volume = 2.0  diversion.numPoints  = 7  diversion.routeSymbol  = (char)DIVERSION  expectedDiversion =  {{  {6, 19},  {6, 20},  {6, 21},  {6, 22},  {6, 23},  {6, 24},  {7, 24}}, 7, DIVERSION};  All assertions return TRUE |  |  |
| Check function correctly finds the blue truck, updates the truck’s weight and volume, and adds diversion route. | org.current.B.CurrentWeight = 20;  org.current.B.CurrentVolume = 0.5;  org.current.G.CurrentWeight = 200;  org.current.G.CurrentVolume = 1;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  dists[0][0] = 4.0;  dists[0][1] = BLUE;  dists[1][0] = 4.0;  dists[1][1] = GREEN;  dists[2][0] = 6.0;  dists[2][1] = YELLOW;  OrderInfo order =  {500, 1, {13, 20}, {-1, -1}};  findTruckAndDiversion(&org, dists, orderTemp); | Done = 1  Weight = 520  Volume = 1.5  diversion.numPoints  = 6  diversion.routeSymbol  = (char)DIVERSION  expectedDiversion =  {{  {17, 21},  {16, 21},  {15, 21},  {14, 21},  {13, 21},  {13, 20}}, 6, DIVERSION};  All assertions return TRUE |  |  |
| Check function correctly finds the green truck, updates the truck’s weight and volume, and adds diversion route. | org.current.B.CurrentWeight = 520;  org.current.B.CurrentVolume = 1.5; org.current.G.CurrentWeight = 200;  org.current.G.CurrentVolume = 1;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  dists[0][0] = 4.0;  dists[0][1] = BLUE;  dists[1][0] = 4.0;  dists[1][1] = GREEN;  dists[2][0] = 6.0;  dists[2][1] = YELLOW;  OrderInfo order =  {500, 1, {13, 20}, {-1, -1}};  findTruckAndDiversion(&org, dists, orderTemp); | Done = 1  Weight = 700  Volume = 2.0  diversion.numPoints  = 6  diversion.routeSymbol  = (char)DIVERSION  expectedDiversion =  {{  {9, 21},  {10, 21},  {11, 21  {12, 21},  {13, 21},  {13, 20}}, 6, DIVERSION};  All assertions return TRUE |  |  |
| Check function correctly finds the yellow truck, updates the truck’s weight and volume, and adds diversion route. | org.current.B.CurrentWeight = 520;  org.current.B.CurrentVolume = 1.5;  org.current.G.CurrentWeight = 200;  org.current.G.CurrentVolume = 1;  org.current.Y.CurrentWeight = 0;  org.current.Y.CurrentVolume = 0;  dists[1][0] = 5.0;  dists[1][1] = BLUE;  dists[2][0] = 13.0;  dists[2][1] = GREEN;  dists[0][0] = 3.0;  dists[0][1] = YELLOW;  OrderInfo order =  {666, .25, {22, 22}, {-1, -1}};  findTruckAndDiversion(&org, dists, orderTemp); | Done = 1  Weight = 666  Volume = 0.25  diversion.numPoints  = 4  diversion.routeSymbol  = (char)DIVERSION  expectedDiversion =  {{  {19, 22},  {20, 22},  {21, 22},  {22, 22}}, 4, DIVERSION};  All assertions return TRUE |  |  |
| Check function works as expected (returns 0) when the fleet limit (432) is exceeded | org.current.B.CurrentWeight = 520;  org.current.B.CurrentVolume = 1.5;  org.current.G.CurrentWeight = 400;  org.current.G.CurrentVolume = 1;  org.current.Y.CurrentWeight = 666;  org.current.Y.CurrentVolume = .25;  dists[1][0] = 5.0;  dists[1][1] = BLUE;  dists[2][0] = 13.0;  dists[2][1] = GREEN;  dists[0][0] = 3.0;  dists[0][1] = YELLOW;  OrderInfo order =  {666, .25, {22, 22}, {-1, -1}};  findTruckAndDiversion(&org, dists, orderTemp); | Done = 0  Weight = 666  Volume = 0.25  (Assert not equal)  diversion.numPoints  = 4  (Assert not equal)  diversion.routeSymbol  = (char)DIVERSION  (Assert not equal)  expectedDiversion =  {{  {19, 22},  {20, 22},  {21, 22},  {22, 22}}, 4, DIVERSION};  All assertions return TRUE |  |  |

**Bugs Found**:

Description of each bug found above and how to reproduce it.