10.3 JSprit tasks

Task: Q01JSNW

Introduction

In this task you are given a program that contains a few syntactical mistakes. Your task is to spot and report these mistakes.

Task

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4
☐ Line 5	☐ Line 6	☐ Line 7	☐ Line 8
☐ Line 9	☐ Line 10	☐ Line 11	☐ Line 12
☐ Line 13	☐ Line 14	☐ Line 15	☐ Line 16
☐ Line 17	☐ Line 18	☐ Line 19	☐ Line 20
☐ Line 21			

```
1  public class Q01JSNW {
2
3    private int final static STUFF = 0;
4
5    public static void main(String[] args){
6      Location n0 = create Location(5, 1);
7      Location n1 = Location.newInstance(10, 9);
8      Location n2 = Location.createCustomerAt("12,1");
9
10     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance;
11
12     costMatrixBuilder.addTransportTime(n0,n1, 2 * EuclideanDistanceCalculator
13       .calculateDistance(n0.getCoordinate(),n1.getCoordinate()) + 2);
14     costMatrixBuilder.addTransportTime(n0, n2, 2 * EuclideanDistanceCalculator
15       .calculateDistance(n0.getCoordinate(),n2.getCoordinate() + 2);
16     costMatrixBuilder.addTransportTime(n1, n2, 4 * EuclideanDistanceCalculator
17       .calculateDistance(n1.getCoordinate(),n2.getCoordinate())+ 4);
18     costMatrixBuilder.addTransportTime(n2 n1, 4 * EuclideanDistanceCalculator
19       .calculateDistance(n2.getCoordinate(),n1.getCoordinate())+ 4);
20   }
21 }
```

Correct solution

☑Line 6, ☑Line 8, ☑Line 10, ☑Line 15, ☑Line 18

Evaluation 2020

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	2
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Evaluation 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q01JSAG

Introduction

In this task you are given a program that contains a few syntactical mistakes. Your task is to spot and report these mistakes.

Task

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4
☐ Line 5	☐ Line 6	☐ Line 7	☐ Line 8
☐ Line 9			

```
1 VehicleType type1 = VehicleTypeImpl.Builder
2     .newInstance(type1).addCapacityDimension(0, 130)
3     .build();
4 VehicleType type2 = VehicleTypeImpl.Builder
5     .newInstance("type2").addCapacityDimension(0, 140)
6     .instantiate();
7 VehicleType type3 = VehicleTypeImpl.Builder
8     .newInstance("type3").addCapacityDimension(150)
9     .build();
```

Correct solution

☑ Line 2, ☑ Line 6, ☑ Line 8

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q02JSAG (1/2)

Introduction

In this task you will find a program with a gap. Additionally, you are presented four code snippets that can be used to fill the gap. However, some of these snippets do not make sense (either for themselves or in the context of the completed program). It is your task to find the nonsensical snippets and report them.

Task

Below you see a VRPTW modelled with JSprit. In lines 92 – 95, the definition of a depot is required. Which of the proposed snippets given at the bottom of this page are semantically incorrect (in other words: which of the four snippets do not make complete sense)?

☐ Snippet 1	☐ Snippet 2	☐ Snippet 3	☐ Snippet 4

```
1  public class Q02JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7
8      Location n0 = Location.newInstance(1,1);
9
10     Service n1 = Service.Builder.newInstance("n1")
11       .setLocation(Location.newInstance(1, 8))
12       .addSizeDimension(STUFF, 30)
13       .build();
14
15     Service n2 = Service.Builder.newInstance("n2")
16       .setLocation(Location.newInstance(2, 11))
17       .addSizeDimension(STUFF, 30)
18       .build();
19
20     Location n3 = Location.newInstance(4, 6);
21
22     Service n4 = Service.Builder.newInstance("n4")
23       .setLocation(Location.newInstance(5, 12))
24       .addSizeDimension(STUFF, 30)
25       .build();
26
27     Service n5 = Service.Builder.newInstance("n5")
28       .setLocation(Location.newInstance(8, 11))
29       .addSizeDimension(STUFF, 30)
30       .build();
31
32     Service n6 = Service.Builder.newInstance("n6")
33       .setLocation(Location.newInstance(8, 7))
34       .addSizeDimension(STUFF, 30)
35       .build();
36
37     Location n7 = Location.newInstance(13, 12);
38
39     Location n8 = Location.newInstance(9, 5);
40
41     Location n9 = Location.newInstance(13, 1);
42
```

Task: Q02JSAG (2/2)

Task (continuation)

```
43      IncompleteCostMatrix.Builder costMatrix = IncompleteCostMatrix.Builder.newInstance();
44
45      Set<RelationKey> lcfgroup = new HashSet<>();
46      lcfgroup.add(RelationKey.newKey(n0, n1));
47      lcfgroup.add(RelationKey.newKey(n1, n2 ));
48      lcfgroup.add(RelationKey.newKey(n2, n4));
49      lcfgroup.add(RelationKey.newKey(n4, n5));
50      lcfgroup.add(RelationKey.newKey(n6, n5));
51      lcfgroup.add(RelationKey.newKey(n7, n4));
52      lcfgroup.add(RelationKey.newKey(n7, n9));
53      lcfgroup.add(RelationKey.newKey(n9, n0));
54      lcfgroup.add(RelationKey.newKey(n9, n8));
55      lcfgroup.add(RelationKey.newKey(n8, n6));
56      lcfgroup.add(RelationKey.newKey(n5, n7));
57
58      for(RelationKey key : lcfgroup)
59        costMatrix.addTransportTime(key.from, key.to, roadFunction(key.from, key.to, 2.0));
60
61      Set<RelationKey> hcfgroup = new HashSet<>();
62      hcfgroup.add(RelationKey.newKey(n5, n3));
63      hcfgroup.add(RelationKey.newKey(n3, n0));
64
65      for(RelationKey key : hcfgroup)
66        costMatrix.addTransportTime(key.from, key.to, roadFunction(key.from, key.to, 4.0));
67
68      costMatrix.completeTransportTimeMatrix();
69
70      IncompleteCostMatrix cm = costMatrix.build();
71      vrpBuilder.setRoutingCost(cm);
72
73      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
74        .addCapacityDimension(0, 180)
75        .build();
76
77      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
78        .setType(deliverType).setStartLocation(n0).build();
79      vrpBuilder.addVehicle(vehicle);
80      vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n4, n5, n6));
81
82      VehicleRoutingProblem vrp = vrpBuilder.build();
83    }
84
85    public static double roadFunction(Location end1, Location end2, Double cfactor) {
86      return EuclideanDistanceCalculator
87      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + cfactor;
88    }
89  }
```

Correct solution

☑ Snippet 1, ☑ Snippet 2, ☑ Snippet 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q03JSALL (1/7)

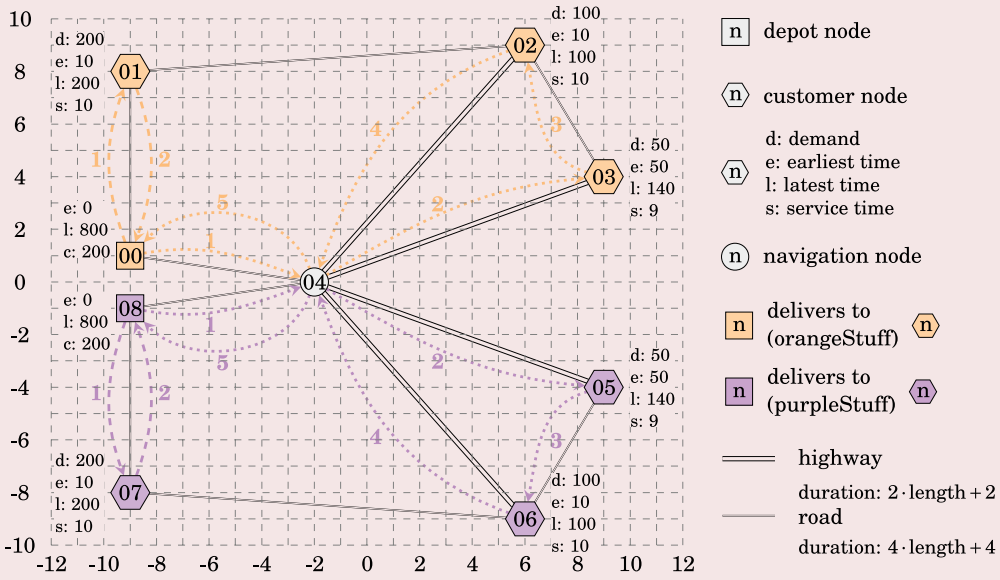
Introduction

In this task you see the illustration of a Network (comprised of customers, demands, roads / highways etc.). In addition, the illustration also shows optimised vehicle routes for a VRP based on the illustrated network. Your task is to determine which of the three models / programs corresponds to the illustrated network.

Task

Which of the three programs corresponds to the illustration?
Wrong answers may feature incorrect customer, depot, road / street, or vehicle definitions.

○Program 1	○Program 2	○Program 3



Task: Q03JSALL (2/7)

Task (continuation)

Program 1

```
1  public class Q03JSALLProgram1 {
2    public static final int ORANGE_STUFF = 0;
3    public static final int PURPLE_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-9, 1);
12
13     Delivery n1 = Delivery.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(-9, 8))
15       .addSizeDimension(ORANGE_STUFF, 200)
16       .setTimeWindow(TimeWindow.newInstance(10, 200))
17       .setServiceTime(10)
18       .build();
19
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(6, 9))
22       .addSizeDimension(ORANGE_STUFF, 100)
23       .setTimeWindow(TimeWindow.newInstance(10, 100))
24       .setServiceTime(10)
25       .build();
26
27     Delivery n3 = Delivery.Builder.newInstance("n3")
28       .setLocation(Location.newInstance(9, 4))
29       .addSizeDimension(ORANGE_STUFF, 50)
30       .setTimeWindow(TimeWindow.newInstance(50, 140))
31       .setServiceTime(9)
32       .build();
33
34     Location n4 = Location.newInstance(-2, 0);
35
36     Delivery n5 = Delivery.Builder.newInstance("n5")
37       .setLocation(Location.newInstance(9, -4))
38       .addSizeDimension(PURPLE_STUFF, 50)
39       .setTimeWindow(TimeWindow.newInstance(50, 140))
40       .setServiceTime(9)
41       .build();
42
43     Delivery n6 = Delivery.Builder.newInstance("n6")
44       .setLocation(Location.newInstance(6, -9))
45       .addSizeDimension(PURPLE_STUFF, 100)
46       .setTimeWindow(TimeWindow.newInstance(10, 100))
47       .setServiceTime(10)
48       .build();
49
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(-9, -8))
52       .addSizeDimension(PURPLE_STUFF, 200)
53       .setTimeWindow(TimeWindow.newInstance(10, 200))
54       .setServiceTime(10)
55       .build();
56
57     Location n8 = Location.newInstance(-9, -1);
58
59     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6, n7));
60
```

Task: Q03JSALL (3/7)

Task (continuation)

```
61      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62      // --- roads
63      Set<RelationKey> roads = new HashSet<>();
64      roads.add(RelationKey.newKey(n2, n4));
65      roads.add(RelationKey.newKey(n3, n4));
66      roads.add(RelationKey.newKey(n6, n4));
67      roads.add(RelationKey.newKey(n5, n4));
68
69      for(RelationKey key : roads)
70        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
71
72      // --- highways
73      Set<RelationKey> highways = new HashSet<>();
74      highways.add(RelationKey.newKey(n0, n1));
75      highways.add(RelationKey.newKey(n0, n4));
76      highways.add(RelationKey.newKey(n1, n2));
77      highways.add(RelationKey.newKey(n2, n3));
78      highways.add(RelationKey.newKey(n8, n7));
79      highways.add(RelationKey.newKey(n8, n4));
80      highways.add(RelationKey.newKey(n7, n6));
81      highways.add(RelationKey.newKey(n5, n6));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      // Vehicle type definition
90      VehicleType orangeStuffType = VehicleTypeImpl.Builder.newInstance("orangeStuffType")
91        .addCapacityDimension(ORANGE_STUFF, 200).build();
92      VehicleType purpleStuffType = VehicleTypeImpl.Builder.newInstance("purpleStuffType")
93        .addCapacityDimension(PURPLE_STUFF, 200).build();
94
95      // Vehicle instance definition
96      VehicleImpl orangeStuffInstance = VehicleImpl.Builder.newInstance("orangeStuffInstance")
97        .setType(orangeStuffType)
98        .setStartLocation(n0)
99        .build();
100     VehicleImpl purpleStuffInstance = VehicleImpl.Builder.newInstance("purpleStuffInstance")
101       .setType(purpleStuffType)
102       .setStartLocation(n8)
103       .build();
104
105     // Add vehicle instance to the problem
106     vrpBuilder.addVehicle(orangeStuffInstance);
107     vrpBuilder.addVehicle(purpleStuffInstance);
108
109     VehicleRoutingProblem vrp = vrpBuilder.build();
110   }
111
112   public static double highwayFunction(Location end1, Location end2) {
113     return 2 * EuclideanDistanceCalculator
114       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
115   }
116
117   public static double roadFunction(Location end1, Location end2) {
118     return 4 * EuclideanDistanceCalculator
119       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
120   }
121 }
```

Task: Q03JSALL (4/7)

Task (continuation)

Program 2

```
1  public class Q03JSALLProgram2 {
2    public static final int ORANGE_STUFF = 0;
3    public static final int PURPLE_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-9, 1);
12
13     Delivery n1 = Delivery.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(-9, 8))
15       .addSizeDimension(ORANGE_STUFF, 200)
16       .setTimeWindow(TimeWindow.newInstance(10, 200))
17       .setServiceTime(10)
18       .build();
19
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(6, 9))
22       .addSizeDimension(ORANGE_STUFF, 100)
23       .setTimeWindow(TimeWindow.newInstance(10, 100))
24       .setServiceTime(10)
25       .build();
26
27     Delivery n3 = Delivery.Builder.newInstance("n3")
28       .setLocation(Location.newInstance(9, 4))
29       .addSizeDimension(ORANGE_STUFF, 50)
30       .setTimeWindow(TimeWindow.newInstance(50, 140))
31       .setServiceTime(9)
32       .build();
33
34     Location n4 = Location.newInstance(-2, 0);
35
36     Delivery n5 = Delivery.Builder.newInstance("n5")
37       .setLocation(Location.newInstance(9, -4))
38       .addSizeDimension(PURPLE_STUFF, 50)
39       .setTimeWindow(TimeWindow.newInstance(50, 140))
40       .setServiceTime(9)
41       .build();
42
43     Delivery n6 = Delivery.Builder.newInstance("n6")
44       .setLocation(Location.newInstance(6, -9))
45       .addSizeDimension(PURPLE_STUFF, 100)
46       .setTimeWindow(TimeWindow.newInstance(10, 100))
47       .setServiceTime(10)
48       .build();
49
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(-9, -8))
52       .addSizeDimension(PURPLE_STUFF, 200)
53       .setTimeWindow(TimeWindow.newInstance(10, 200))
54       .setServiceTime(10)
55       .build();
56
57     Location n8 = Location.newInstance(-9, -1);
58
59     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6, n7));
60
```

Task: Q03JSALL (5/7)

Task (continuation)

```
61      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62      // --- roads
63      Set<RelationKey> roads = new HashSet<>();
64      roads.add(RelationKey.newKey(n0, n1));
65      roads.add(RelationKey.newKey(n0, n4));
66      roads.add(RelationKey.newKey(n1, n2));
67      roads.add(RelationKey.newKey(n2, n3));
68      roads.add(RelationKey.newKey(n8, n7));
69      roads.add(RelationKey.newKey(n8, n4));
70      roads.add(RelationKey.newKey(n7, n6));
71      roads.add(RelationKey.newKey(n5, n6));
72
73      for(RelationKey key : roads)
74        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
75
76      // --- highways
77      Set<RelationKey> highways = new HashSet<>();
78      highways.add(RelationKey.newKey(n2, n4));
79      highways.add(RelationKey.newKey(n3, n4));
80      highways.add(RelationKey.newKey(n6, n4));
81      highways.add(RelationKey.newKey(n5, n4));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      // Vehicle type definition
90      VehicleType orangeStuffType = VehicleTypeImpl.Builder.newInstance("orangeStuffType")
91      .addCapacityDimension(ORANGE_STUFF, 200).build();
92      VehicleType purpleStuffType = VehicleTypeImpl.Builder.newInstance("purpleStuffType")
93      .addCapacityDimension(PURPLE_STUFF, 200).build();
94
95      // Vehicle instance defintion
96      VehicleImpl orangeStuffInstance = VehicleImpl.Builder.newInstance("orangeStuffInstance")
97      .setType(orangeStuffType)
98      .setStartLocation(n8)
99      .build();
100     VehicleImpl purpleStuffInstance = VehicleImpl.Builder.newInstance("purpleStuffInstance")
101     .setType(purpleStuffType)
102     .setStartLocation(n0)
103     .build();
104
105     // Add vehicle instance to the problem
106     vrpBuilder.addVehicle(orangeStuffInstance);
107     vrpBuilder.addVehicle(purpleStuffInstance);
108
109     VehicleRoutingProblem vrp = vrpBuilder.build();
110   }
111
112   public static double highwayFunction(Location end1, Location end2) {
113     return 2 * EuclideanDistanceCalculator
114       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
115   }
116
117   public static double roadFunction(Location end1, Location end2) {
118     return 4 * EuclideanDistanceCalculator
119       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
120   }
121 }
```

Task: Q03JSALL (6/7)

Task (continuation)

Program 3

```
1  public class Q03JSALLProgram3 {
2    public static final int ORANGE_STUFF = 0;
3    public static final int PURPLE_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-9, 1);
12
13     Delivery n1 = Delivery.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(-9, 8))
15       .addSizeDimension(ORANGE_STUFF, 200)
16       .setTimeWindow(TimeWindow.newInstance(10, 200))
17       .setServiceTime(10)
18       .build();
19
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(6, 9))
22       .addSizeDimension(ORANGE_STUFF, 100)
23       .setTimeWindow(TimeWindow.newInstance(10, 100))
24       .setServiceTime(10)
25       .build();
26
27     Delivery n3 = Delivery.Builder.newInstance("n3")
28       .setLocation(Location.newInstance(9, 4))
29       .addSizeDimension(ORANGE_STUFF, 50)
30       .setTimeWindow(TimeWindow.newInstance(50, 140))
31       .setServiceTime(9)
32       .build();
33
34     Location n4 = Location.newInstance(-2, 0);
35
36     Delivery n5 = Delivery.Builder.newInstance("n5")
37       .setLocation(Location.newInstance(9, -4))
38       .addSizeDimension(PURPLE_STUFF, 50)
39       .setTimeWindow(TimeWindow.newInstance(50, 140))
40       .setServiceTime(9)
41       .build();
42
43     Delivery n6 = Delivery.Builder.newInstance("n6")
44       .setLocation(Location.newInstance(6, -9))
45       .addSizeDimension(PURPLE_STUFF, 100)
46       .setTimeWindow(TimeWindow.newInstance(10, 100))
47       .setServiceTime(10)
48       .build();
49
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(-9, -8))
52       .addSizeDimension(PURPLE_STUFF, 200)
53       .setTimeWindow(TimeWindow.newInstance(10, 200))
54       .setServiceTime(10)
55       .build();
56
57     Location n8 = Location.newInstance(-9, -1);
58
59     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6, n7));
60
```

Task: Q03JSALL (7/7)

Task (continuation)

```
61      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62      // --- roads
63      Set<RelationKey> roads = new HashSet<>();
64      roads.add(RelationKey.newKey(n0, n1));
65      roads.add(RelationKey.newKey(n0, n4));
66      roads.add(RelationKey.newKey(n1, n2));
67      roads.add(RelationKey.newKey(n2, n3));
68      roads.add(RelationKey.newKey(n8, n7));
69      roads.add(RelationKey.newKey(n8, n4));
70      roads.add(RelationKey.newKey(n7, n6));
71      roads.add(RelationKey.newKey(n5, n6));
72
73      for(RelationKey key : roads)
74        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
75
76      // --- highways
77      Set<RelationKey> highways = new HashSet<>();
78      highways.add(RelationKey.newKey(n2, n4));
79      highways.add(RelationKey.newKey(n3, n4));
80      highways.add(RelationKey.newKey(n6, n4));
81      highways.add(RelationKey.newKey(n5, n4));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      // Vehicle type definition
90      VehicleType orangeStuffType = VehicleTypeImpl.Builder.newInstance("orangeStuffType")
91        .addCapacityDimension(ORANGE_STUFF, 200).build();
92      VehicleType purpleStuffType = VehicleTypeImpl.Builder.newInstance("purpleStuffType")
93        .addCapacityDimension(PURPLE_STUFF, 200).build();
94
95      // Vehicle instance defintion
96      VehicleImpl orangeStuffInstance = VehicleImpl.Builder.newInstance("orangeStuffInstance")
97        .setType(orangeStuffType)
98        .setStartLocation(n0)
99        .build();
100     VehicleImpl purpleStuffInstance = VehicleImpl.Builder.newInstance("purpleStuffInstance")
101       .setType(purpleStuffType)
102       .setStartLocation(n8)
103       .build();
104
105     // Add vehicle instance to the problem
106     vrpBuilder.addVehicle(orangeStuffInstance);
107     vrpBuilder.addVehicle(purpleStuffInstance);
108
109     VehicleRoutingProblem vrp = vrpBuilder.build();
110   }
111
112   public static double highwayFunction(Location end1, Location end2) {
113     return 2 * EuclideanDistanceCalculator
114       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
115   }
116
117   public static double roadFunction(Location end1, Location end2) {
118     return 4 * EuclideanDistanceCalculator
119       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
120   }
121 }
```

Correct solution

◉ Program 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)

Task: Q04JSNW (1/4)

Introduction

In this task you are shown a program and four different graphical networks. One of these networks is exactly described (modelled) by the program. For the other three networks the program is not completely right. It is your task to find and report the exactly modelled network.

Task

Which of the above networks results from the given JSprit model?

○Network A	○Network B	○Network C	○Network D

```
1  public class Q04JSNW {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7
8      Location n0 = Location.newInstance(0, -6);
9
10     Service n1 = Service.Builder.newInstance("n1")
11       .setLocation(Location.newInstance(-9,4))
12       .addSizeDimension(STUFF, 15)
13       .setTimeWindow(TimeWindow.newInstance(15, 120))
14       .setServiceTime(5)
15       .build();
16
17     Service n2 = Service.Builder.newInstance("n2")
18       .setLocation(Location.newInstance(7,-9))
19       .addSizeDimension(STUFF, 20)
20       .setTimeWindow(TimeWindow.newInstance(10,130))
21       .setServiceTime(7)
22       .build();
23
24     Service n3 = Service.Builder.newInstance("n3")
25       .setLocation(Location.newInstance(8,5))
26       .addSizeDimension(STUFF, 50)
27       .setTimeWindow(TimeWindow.newInstance(20,90))
28       .setServiceTime(10)
29       .build();
30
31     Location n4 = Location.newInstance(2,0);
32
33     Service n5 = Service.Builder.newInstance("n5")
34       .setLocation(Location.newInstance(-2, -1))
35       .addSizeDimension(STUFF, 10)
36       .setTimeWindow(TimeWindow.newInstance(80, 150))
37       .setServiceTime(20)
38       .build();
39
40     Service n6 = Service.Builder.newInstance("n6")
41       .setLocation(Location.newInstance(-8,-6))
42       .addSizeDimension(STUFF,25)
43       .setTimeWindow(TimeWindow.newInstance(90,270))
44       .setServiceTime(5)
45       .build();
46
```

Task: Q04JSNW (2/4)

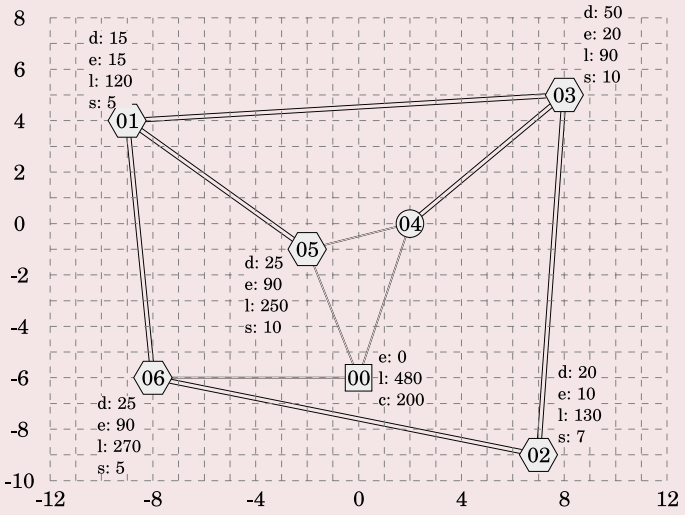
Task (continuation)

```
47      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
48
49      costMatrixBuilder.addTransportTime(n0, n5.getLocation(),
50        roadFunction(n0, n5.getLocation()));
51
52      costMatrixBuilder.addTransportTime(n0,n4,roadFunction(n0,n4));
53
54      costMatrixBuilder.addTransportTime(n5.getLocation(), n4,
55        roadFunction(n5.getLocation(),n4));
56
57      costMatrixBuilder.addTransportTime(n0, n6.getLocation(),
58        roadFunction(n0,n6.getLocation()));
59
60      costMatrixBuilder.addTransportTime(n1.getLocation(), n3.getLocation(),
61        highwayFunction(n1.getLocation(),n3.getLocation()));
62
63      costMatrixBuilder.addTransportTime(n3.getLocation(), n2.getLocation(),
64        highwayFunction(n3.getLocation(), n2.getLocation()));
65
66      costMatrixBuilder.addTransportTime(n2.getLocation(), n6.getLocation(),
67        highwayFunction(n2.getLocation(), n6.getLocation()));
68
69      costMatrixBuilder.addTransportTime(n6.getLocation(), n1.getLocation(),
70        highwayFunction(n6.getLocation(), n1.getLocation()));
71
72      costMatrixBuilder.addTransportTime(n4, n3.getLocation(),
73        highwayFunction(n4,n3.getLocation()));
74
75      costMatrixBuilder.addTransportTime(n5.getLocation(), n1.getLocation(),
76        highwayFunction(n5.getLocation(), n1.getLocation()));
77
78      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
79      vrpBuilder.setRoutingCost(cm);
80
81      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
82        .addCapacityDimension(0, 180)
83        .build();
84
85      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
86        .setType(deliverType).setStartLocation(n0).setLatestArrival(500).build();
87      vrpBuilder.addVehicle(vehicle);
88      vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6));
89
90      VehicleRoutingProblem vrp = vrpBuilder.build();
91    }
92
93    public static double highwayFunction(Location end1, Location end2) {
94      return 1.5 * EuclideanDistanceCalculator
95        .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
96    }
97
98    public static double roadFunction(Location end1, Location end2) {
99      return 3 * EuclideanDistanceCalculator
100       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) *  + 3;
101   }
102 }
```

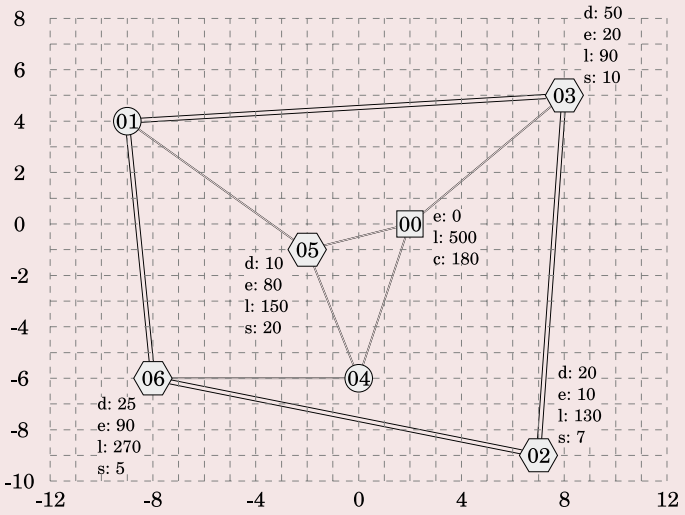
Task: Q04JSNW (3/4)

Task (continuation)

Network A



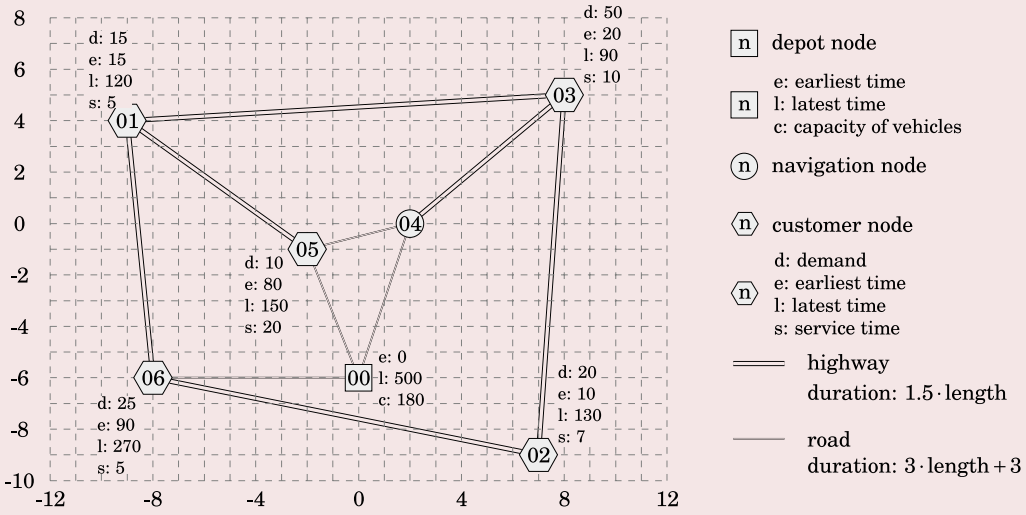
Network B



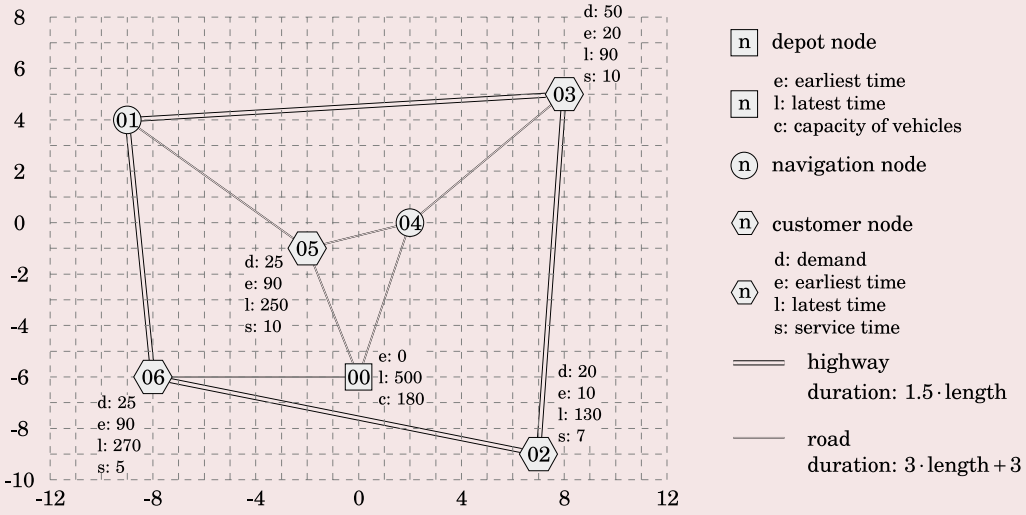
Task: Q04JSNW (4/4)

Task (continuation)

Network C



Network D



Correct solution

◉ Network C

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q04JSAG (1/4)

Introduction

In this task you are shown a program and four graphical networks on which four different tours are depicted. One of these networks shows a tour that is exactly described (modelled) by the program. For the other three tours the program is not completely right. It is your task to find and report the exactly modelled tour.

NOTE: If a tour step connects two nodes that do not share an edge, this means that the actual path from the start node to the target node of the respective step is not important. However, only nodes of the drawn tour are serviced!

Task

Which of the presented tours for a vehicle may result from the program given below?

○Tour 1	○Tour 2	○Tour 3	○Tour 4

```
1  public class Q04JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7
8      Location n0 = Location.newInstance(-10, -8);
9      Location n1 = Location.newInstance(-8,-4);
10     Service n2 = Service.Builder.newInstance("n2")
11       .setLocation(Location.newInstance(-11,2))
12       .addSizeDimension(STUFF, 20)
13       .setTimeWindow(TimeWindow.newInstance(10, 130))
14       .setServiceTime(7)
15       .build();
16     Location n3 = Location.newInstance(-5,4);
17     Service n4 = Service.Builder.newInstance("n4")
18       .setLocation(Location.newInstance(-8,7))
19       .addSizeDimension(STUFF, 50)
20       .setTimeWindow(TimeWindow.newInstance(20,90))
21       .setServiceTime(10)
22       .build();
23     Service n5 = Service.Builder.newInstance("n5")
24       .setLocation(Location.newInstance(2,0))
25       .addSizeDimension(STUFF, 25)
26       .setTimeWindow(TimeWindow.newInstance(90,250))
27       .setServiceTime(10)
28       .build();
29     Service n6 = Service.Builder.newInstance("n6")
30       .setLocation(Location.newInstance(2,7))
31       .addSizeDimension(STUFF, 25)
32       .setTimeWindow(TimeWindow.newInstance(90,250))
33       .setServiceTime(10)
34       .build();
35     Location n7 = Location.newInstance(6, 5);
36     Location n8 = Location.newInstance(11, 3);
37     Location n9 = Location.newInstance(2,-8);
38     Location n10 = Location.newInstance(8,-6);
39     Location n11 = Location.newInstance(6,-9);
40
```

Task: Q04JSAG (2/4)

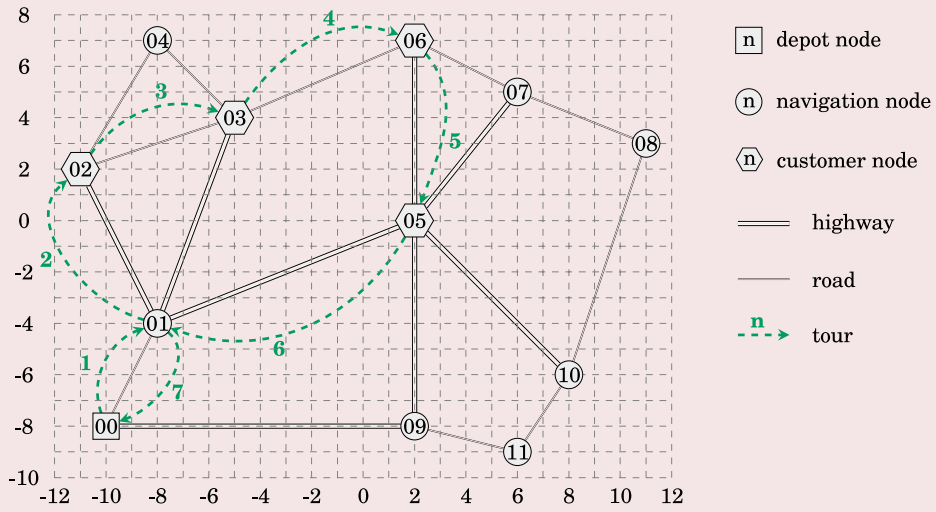
Task (continuation)

```
41      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
42      costMatrixBuilder.addTransportTime(n0, n1, roadFunction(n0, n1));
43      costMatrixBuilder.addTransportTime(n2.getLocation(), n4.getLocation(),
44        roadFunction(n2.getLocation(), n4.getLocation()));
45      costMatrixBuilder.addTransportTime(n2.getLocation(), n3,
46      roadFunction(n2.getLocation(), n3));
47      costMatrixBuilder.addTransportTime(n4.getLocation(), n3,
48        roadFunction(n4.getLocation(), n3));
49      costMatrixBuilder.addTransportTime(n3, n6.getLocation(),
50        roadFunction(n3, n6.getLocation()));
51      costMatrixBuilder.addTransportTime(n6.getLocation(), n7,
52        roadFunction(n6.getLocation(), n7));
53      costMatrixBuilder.addTransportTime(n7, n8, roadFunction(n7, n8));
54      costMatrixBuilder.addTransportTime(n8, n10, roadFunction(n8, n10));
55      costMatrixBuilder.addTransportTime(n10, n11, roadFunction(n10, n11));
56      costMatrixBuilder.addTransportTime(n11, n9, roadFunction(n11, n9));
57
58      costMatrixBuilder.addTransportTime(n1, n2.getLocation(),
59        highwayFunction(n1, n2.getLocation()));
60      costMatrixBuilder.addTransportTime(n1, n3, highwayFunction(n1, n3));
61      costMatrixBuilder.addTransportTime(n1, n5.getLocation(),
62        highwayFunction(n1, n5.getLocation()));
63      costMatrixBuilder.addTransportTime(n5.getLocation(), n6.getLocation(),
64        highwayFunction(n5.getLocation(), n6.getLocation()));
65      costMatrixBuilder.addTransportTime(n5.getLocation(), n7,
66        highwayFunction(n5.getLocation(), n7));
67      costMatrixBuilder.addTransportTime(n5.getLocation(), n10,
68        highwayFunction(n5.getLocation(), n10));
69      costMatrixBuilder.addTransportTime(n5.getLocation(), n9,
70        highwayFunction(n5.getLocation(), n9));
71      costMatrixBuilder.addTransportTime(n9, n0, highwayFunction(n9, n0));
72
73      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
74      vrpBuilder.setRoutingCost(cm);
75
76      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
77        .addCapacityDimension(0, 180)
78        .build();
79
80      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
81        .setType(deliverType).setStartLocation(n0).setLatestArrival(500).build();
82      vrpBuilder.addVehicle(vehicle);
83      vrpBuilder.addAllJobs(Arrays.asList(n2, n4, n5, n6));
84
85      VehicleRoutingProblem vrp = vrpBuilder.build();
86    }
87
88    public static double highwayFunction(Location end1, Location end2) {
89      return EuclideanDistanceCalculator
90        .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
91    }
92
93    public static double roadFunction(Location end1, Location end2) {
94      return 3 * EuclideanDistanceCalculator
95        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
96    }
97  }
```

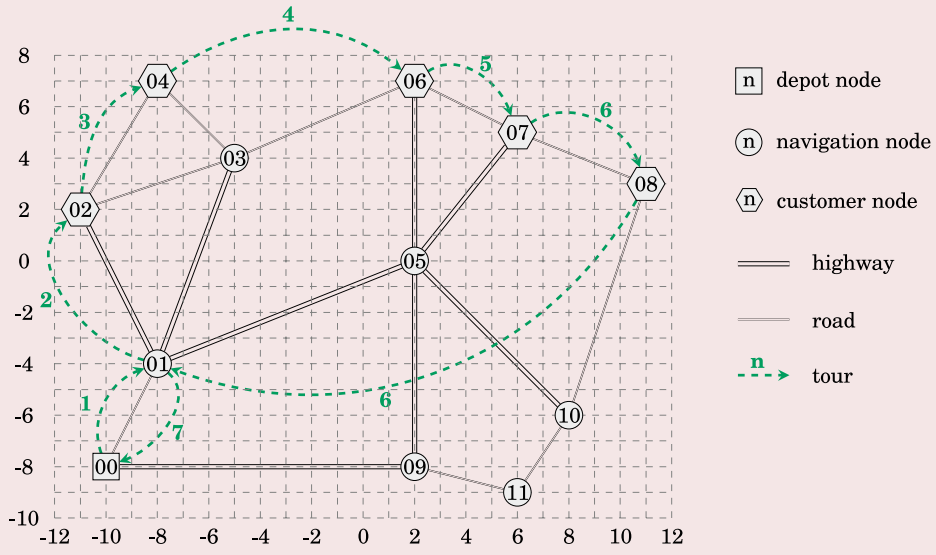
Task: Q04JSAG (3/4)

Task (continuation)

Tour 1



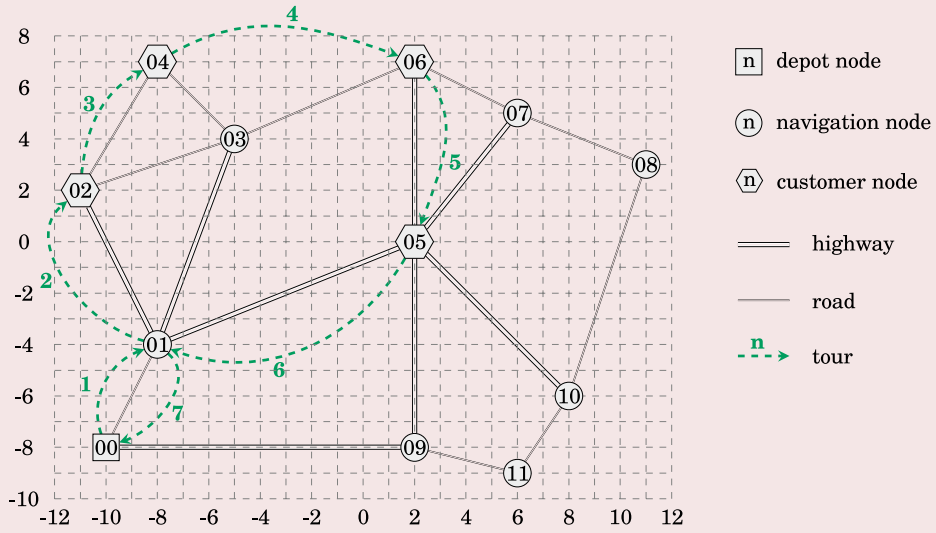
Tour 2



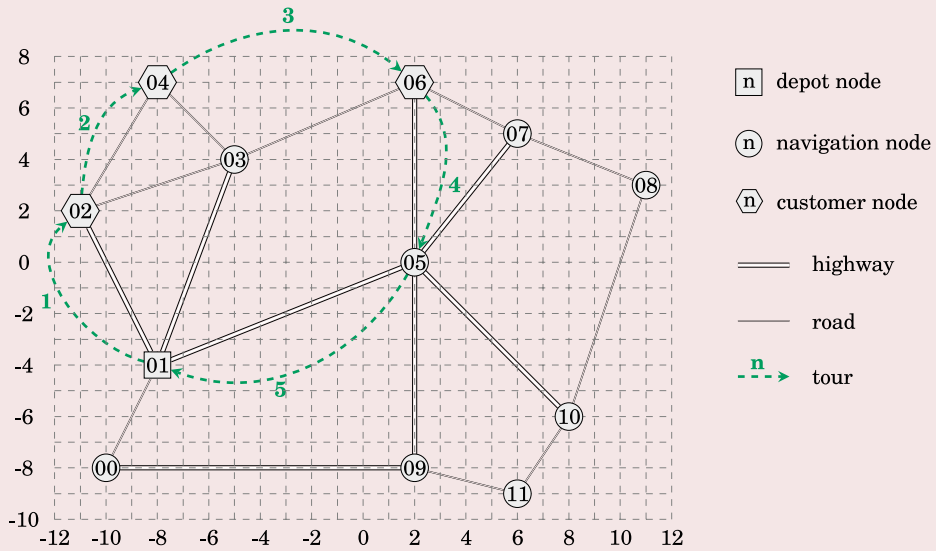
Task: Q04JSAG (4/4)

Task (continuation)

Tour 3



Tour 4



Correct solution

◉ Tour 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q05JSNW (1/3)

Introduction

In this task you will find a program and two networks. The first of the two networks is described by the program. If some of the lines of the program are changed, the program is a description of the second network. It is your task to find these lines and report them.

Task

The JSprit model below represents network 1 . What lines of the JSprit model would have to be changed if you wanted to model network 2 (multiple answers possible)?

☐ Line 01	☐ Line 02	☐ Line 03	☐ Line 04	☐ Line 05	☐ Line 06	☐ Line 07
☐ Line 08	☐ Line 09	☐ Line 10	☐ Line 11	☐ Line 12	☐ Line 13	☐ Line 14
☐ Line 15	☐ Line 16	☐ Line 17	☐ Line 18	☐ Line 19	☐ Line 20	☐ Line 21
☐ Line 22	☐ Line 23	☐ Line 24	☐ Line 25	☐ Line 26	☐ Line 27	☐ Line 28
☐ Line 29	☐ Line 30	☐ Line 31	☐ Line 32	☐ Line 33	☐ Line 34	☐ Line 35
☐ Line 36	☐ Line 37	☐ Line 38	☐ Line 39	☐ Line 40	☐ Line 41	☐ Line 42
☐ Line 43	☐ Line 44	☐ Line 45	☐ Line 46	☐ Line 47	☐ Line 48	☐ Line 49
☐ Line 50	☐ Line 51	☐ Line 52	☐ Line 53	☐ Line 54	☐ Line 55	☐ Line 56
☐ Line 57	☐ Line 58	☐ Line 59	☐ Line 60	☐ Line 61	☐ Line 62	☐ Line 63
☐ Line 64	☐ Line 65	☐ Line 66	☐ Line 67	☐ Line 68	☐ Line 69	☐ Line 70
☐ Line 71	☐ Line 72	☐ Line 73	☐ Line 74	☐ Line 75	☐ Line 76	☐ Line 77
☐ Line 78	☐ Line 79	☐ Line 80	☐ Line 81	☐ Line 82	☐ Line 83	☐ Line 84
☐ Line 85	☐ Line 86	☐ Line 87	☐ Line 88	☐ Line 89	☐ Line 90	☐ Line 91
☐ Line 92	☐ Line 93					

```
1  public class Q05JSNW {
2    // Products
3    public static int STUFF = 0;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-2, -2);
12
13     Location n1 = Location.newInstance(-8, -4);
14
15     Service n2 = Service.Builder.newInstance("n2")
16       .setLocation(Location.newInstance(-8,-1))
17       .addSizeDimension(STUFF, 20)
18       .setTimeWindow(TimeWindow.newInstance(10, 130))
19       .setServiceTime(7)
20       .build();
21
22     Service n3 = Service.Builder.newInstance("n3")
23       .setLocation(Location.newInstance(-5, 4))
24       .addSizeDimension(STUFF, 50)
25       .setTimeWindow(TimeWindow.newInstance(20, 90))
26       .setServiceTime(10)
27       .build();
28
29     Service n4 = Service.Builder.newInstance("n4")
30       .setLocation(Location.newInstance(-2, -6))
31       .addSizeDimension(STUFF, 25)
32       .setTimeWindow(TimeWindow.newInstance(90, 250))
33       .setServiceTime(10)
34       .build();
35
36     Location n5 = Location.newInstance(2, 0);
```

Task: Q05JSNW (2/3)

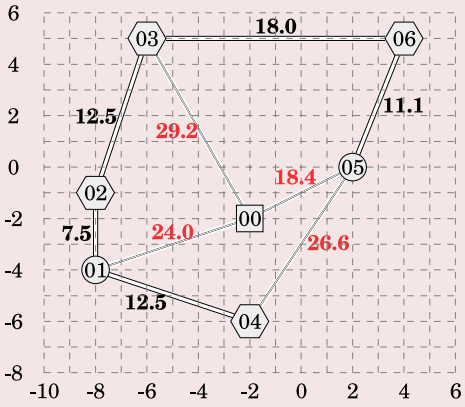
Task (continuation)

```
37
38      Service n6 = Service.Builder.newInstance("n4")
39        .setLocation(Location.newInstance(4, 5))
40        .addSizeDimension(STUFF, 25)
41        .setTimeWindow(TimeWindow.newInstance(90, 250))
42        .setServiceTime(10)
43        .build();
44
45      // --- Edges definition
46      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
47
48      // --- --- roads
49      costMatrixBuilder.addTransportTime(n0, n1, roadFunction(n0, n1));
50
51      costMatrixBuilder.addTransportTime(n0, n3.getLocation(),
52      roadFunction(n0,n3.getLocation()));
53
54      costMatrixBuilder.addTransportTime(n0,n5, roadFunction(n0,n5));
55
56      costMatrixBuilder.addTransportTime(n4.getLocation(), n5,
57      roadFunction(n4.getLocation(), n5));
58
59      // --- --- highways
60      costMatrixBuilder.addTransportTime(n1,n2.getLocation(),
61      highwayFunction(n1, n2.getLocation()));
62
63      costMatrixBuilder.addTransportTime(n2.getLocation(),n3.getLocation(),
64      highwayFunction(n2.getLocation(), n3.getLocation()));
65
66      costMatrixBuilder.addTransportTime(n3.getLocation(),n6.getLocation(),
67      highwayFunction(n3.getLocation(), n6.getLocation()));
68
69      costMatrixBuilder.addTransportTime(n5,n6.getLocation(),
70      highwayFunction(n5, n3.getLocation()));
71
72      costMatrixBuilder.addTransportTime(n1, n4.getLocation(),
73      highwayFunction(n1, n4.getLocation()));
74
75      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
76      vrpBuilder.setRoutingCost(cm);
77
78      vrpBuilder.addAllJobs(Arrays.asList(n2, n3, n4, n6));
79
80      // Vehicle type and vehicle and problem build omitted
81    }
82
83    public static double highwayFunction(Location end1, Location end2) {
84      return 1.5 * EuclideanDistanceCalculator
85        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
86    }
87
88    public static double roadFunction(Location end1, Location end2) {
89      return 3 * EuclideanDistanceCalculator
90        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
91    }
92
93  }
```

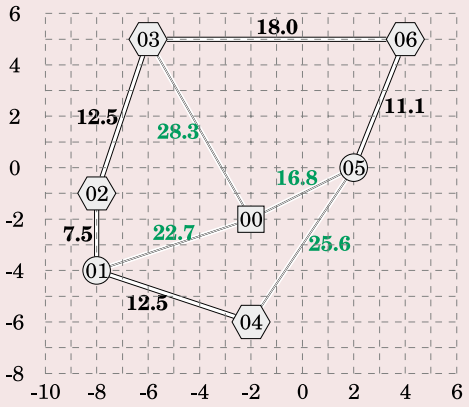
Task: Q05JSNW (3/3)

Task (continuation)

Network 1 (currently modelled)



Network 2 (target state)



Correct solution

☑Line 89, ☑Line 90

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q05JSAG (1/3)

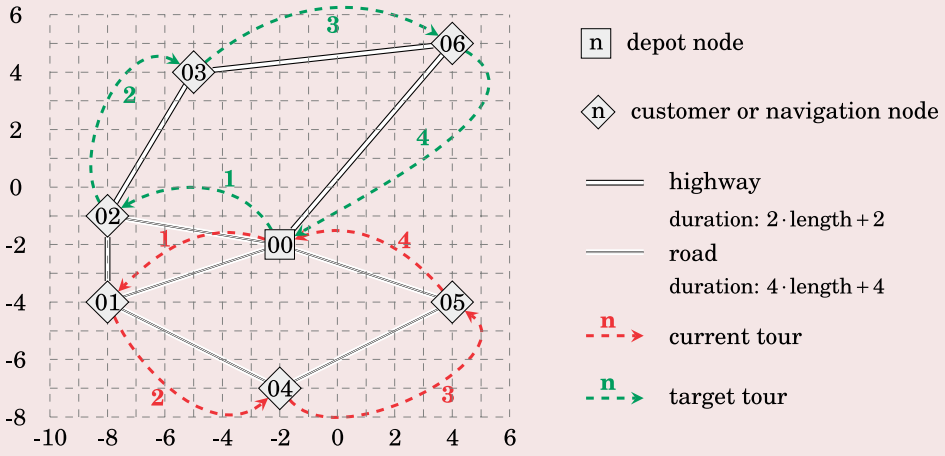
Introduction

In this task, you will find the graphical representation of a network together with a program. The network shows two tours. The program describes the network and a problem for which one of the tours is likely to be optimal. By changing some lines of the program, the program describes a problem for which the other tour is likely to be optimal. It is your task to find and report these lines.

Task

In the picture below, the red tour is a likely result from the problem modelled in the JSprit code given at the bottom. What lines of the JSprit model have to be changed so that the green tour r is likely to result from the model?

☐ Line 01	☐ Line 02	☐ Line 03	☐ Line 04	☐ Line 05	☐ Line 06	☐ Line 07
☐ Line 08	☐ Line 09	☐ Line 10	☐ Line 11	☐ Line 12	☐ Line 13	☐ Line 14
☐ Line 15	☐ Line 16	☐ Line 17	☐ Line 18	☐ Line 19	☐ Line 20	☐ Line 21
☐ Line 22	☐ Line 23	☐ Line 24	☐ Line 25	☐ Line 26	☐ Line 27	☐ Line 28
☐ Line 29	☐ Line 30	☐ Line 31	☐ Line 32	☐ Line 33	☐ Line 34	☐ Line 35
☐ Line 36	☐ Line 37	☐ Line 38	☐ Line 39	☐ Line 40	☐ Line 41	☐ Line 42
☐ Line 43	☐ Line 44	☐ Line 45	☐ Line 46	☐ Line 47	☐ Line 48	☐ Line 49
☐ Line 50	☐ Line 51	☐ Line 52	☐ Line 53	☐ Line 54	☐ Line 55	☐ Line 56
☐ Line 57	☐ Line 58	☐ Line 59	☐ Line 60	☐ Line 61	☐ Line 62	☐ Line 63
☐ Line 64	☐ Line 65	☐ Line 66	☐ Line 67	☐ Line 68	☐ Line 69	☐ Line 70
☐ Line 71	☐ Line 72	☐ Line 73	☐ Line 74	☐ Line 75	☐ Line 76	☐ Line 77
☐ Line 78	☐ Line 79	☐ Line 80				



Task: Q05JSAG (2/3)

Task (continuation)

```
1  public class Q05JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7      Location n0 = Location.newInstance(-2, -2);
8      Service n1 = Service.Builder.newInstance("n1")
9        .setLocation(Location.newInstance(-8,-4))
10       .addSizeDimension(STUFF, 20)
11       .setTimeWindow(TimeWindow.newInstance(0, 20000))
12       .setServiceTime(7).build();
13     Service n2 = Service.Builder.newInstance("n2")
14       .setLocation(Location.newInstance(-8,1))
15       .addSizeDimension(STUFF, 20)
16       .setTimeWindow(TimeWindow.newInstance(0, 20000))
17       .setServiceTime(7)
18       .build();
19     Service n3 = Service.Builder.newInstance("n3")
20       .setLocation(Location.newInstance(-5,4))
21       .addSizeDimension(STUFF, 50)
22       .setTimeWindow(TimeWindow.newInstance(0, 20000))
23       .setServiceTime(10)
24       .build();
25     Service n4 = Service.Builder.newInstance("n4")
26       .setLocation(Location.newInstance(-2,-7))
27       .addSizeDimension(STUFF, 25)
28       .setTimeWindow(TimeWindow.newInstance(0,20000))
29       .setServiceTime(10)
30       .build();
31     Service n5 = Service.Builder.newInstance("n5")
32       .setLocation(Location.newInstance(4,-4))
33       .addSizeDimension(STUFF, 25)
34       .setTimeWindow(TimeWindow.newInstance(0,20000))
35       .setServiceTime(10).build();
36     Service n6 = Service.Builder.newInstance("n6")
37       .setLocation(Location.newInstance(4,5))
38       .addSizeDimension(STUFF, 25)
39       .setTimeWindow(TimeWindow.newInstance(0,20000))
40       .setServiceTime(10)
41       .build();
42     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
43     costMatrixBuilder.addTransportTime(n0, n2.getLocation(),
44         roadFunction(n0, n2.getLocation()));
45     costMatrixBuilder.addTransportTime(n0, n1.getLocation(),
46         roadFunction(n0, n1.getLocation()));
47     costMatrixBuilder.addTransportTime(n1.getLocation(), n4.getLocation(),
48         roadFunction(n1.getLocation(), n4.getLocation()));
49     costMatrixBuilder.addTransportTime(n4.getLocation(), n5.getLocation(),
50         roadFunction(n4.getLocation(), n5.getLocation()));
51     costMatrixBuilder.addTransportTime(n5.getLocation(), n0,
52         roadFunction(n5.getLocation(), n0));
53     costMatrixBuilder.addTransportTime(n2.getLocation(), n3.getLocation(),
54         highwayFunction(n2.getLocation(), n3.getLocation()));
55     costMatrixBuilder.addTransportTime(n3.getLocation(), n6.getLocation(),
56         highwayFunction(n3.getLocation(), n6.getLocation()));
57     costMatrixBuilder.addTransportTime(n6.getLocation(), n0,
58         highwayFunction(n6.getLocation(), n0));
59     IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
60     vrpBuilder.setRoutingCost(cm);
61     VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
62       .addCapacityDimension(0, 180)
63       .build();
64     Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
65       .setType(deliverType).setStartLocation(n0).setLatestArrival(20000).build();
66     vrpBuilder.addVehicle(vehicle);
67     vrpBuilder.addAllJobs(Arrays.asList(n1, n4, n5));
68     VehicleRoutingProblem vrp = vrpBuilder.build();
69 }
70
71 }
```

Task: Q05JSAG (3/3)

Task (continuation)

```
71   public static double highwayFunction(Location end1, Location end2) {
72     return 2 * EuclideanDistanceCalculator
73       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
74   }
75
76   public static double roadFunction(Location end1, Location end2) {
77     return 4 * EuclideanDistanceCalculator
78       .calculateDistance(end1.getCoordinate(), end2.getCoordinate())  + 4;
79   }
80 }
```

Correct solution

☑ Line 67

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q06JSNW (1/7)

Introduction

In this task, you see a program that describes a network for a VRPTW. In addition, you see four more programs. Some of these programs describe the exact same network as the first program, even though they syntactically deviate from the first program. It is your task to find and report these equivalent programs.

Task

Look at the "Program to match" below. One or more of the four possible matches produce an equivalent network. Tick the respective boxes!

☐ Possible match 01	☐ Possible match 02	☐ Possible match 03	☐ Possible match 04

Program to match

```
1  public class Q06JSNW {
2
3    public static void main(String[] args) {
4      VehicleRoutingProblem.Builder vrpBuilder
5        = VehicleRoutingProblem.Builder.newInstance();
6
7      // Network
8      // --- Nodes
9      Location n0 = Location.newInstance(-2, -2);
10     Location n1 = Location.newInstance(-8, -4);
11     Location n2 = Location.newInstance(-8, -1);
12     Location n3 = Location.newInstance(-5, 4);
13     Location n4 = Location.newInstance(-2, -6);
14     Location n5 = Location.newInstance(2, 0);
15     Location n6 = Location.newInstance(4, 5);
16
17     // --- Edges
18     // --- --- roads
19     Set<RelationKey> roads = new HashSet<>();
20     roads.add(RelationKey.newKey(n0, n4));
21     roads.add(RelationKey.newKey(n0, n1));
22     roads.add(RelationKey.newKey(n0, n3));
23     roads.add(RelationKey.newKey(n4, n5));
24
25     // --- --- highways
26     Set<RelationKey> highways = new HashSet<>();
27     highways.add(RelationKey.newKey(n1, n2));
28     highways.add(RelationKey.newKey(n2, n3));
29     highways.add(RelationKey.newKey(n6, n5));
30
31     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
32
33     for(RelationKey key : roads)
34       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
35
36     for(RelationKey key : highways)
37       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
38
39     costMatrixBuilder.addTransportTime(n0,n5, specialFunction(n0, n5));
40     costMatrixBuilder.addTransportTime(n3, n6, specialFunction(n3,n6));
41
42     // adding to vrpBuilder etc. omitted
43   }
44
```

Task: Q06JSNW (2/7)

Task (continuation)

```
45   public static double highwayFunction(Location end1, Location end2) {
46     return 1.5 * EuclideanDistanceCalculator
47       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
48   }
49
50   public static double roadFunction(Location end1, Location end2) {
51     return 3 * EuclideanDistanceCalculator
52       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
53   }
54
55   public static double specialFunction(Location end1, Location end2) {
56     return 3 * EuclideanDistanceCalculator
57       .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
58   }
59   // RelationsKey class omitted for brevity
60 }
```

Task: Q06JSNW (3/7)

Task (continuation)

Matching option 1

```java
public class Q06JSNW_PossibleMatch1 {

  public static void main(String[] args) {
    VehicleRoutingProblem.Builder vrpBuilder
      = VehicleRoutingProblem.Builder.newInstance();

    // Network
    // --- Nodes
    Location n0 = Location.newInstance(-2, -2);
    Location n1 = Location.newInstance(-8, -4);
    Location n2 = Location.newInstance(-8, -1);
    Location n3 = Location.newInstance(-5, 4);
    Location n4 = Location.newInstance(-2, -6);
    Location n5 = Location.newInstance(2, 0);
    Location n6 = Location.newInstance(4, 5);

    IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();

    // --- Edges
    // --- --- highways
    Set<RelationKey> highways = new HashSet<>();
    highways.add(RelationKey.newKey(n1, n2));
    highways.add(RelationKey.newKey(n2, n3));
    highways.add(RelationKey.newKey(n6, n5));

    for(RelationKey key : highways)
      costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));

    costMatrixBuilder.addTransportTime(n0,n5, specialFunction(n0, n5));
    costMatrixBuilder.addTransportTime(n3, n6, specialFunction(n3, n6));

    // --- --- roads
    Set<RelationKey> roads = new HashSet<>();
    roads.add(RelationKey.newKey(n0, n4));
    roads.add(RelationKey.newKey(n0, n1));
    roads.add(RelationKey.newKey(n0, n3));
    roads.add(RelationKey.newKey(n4, n5));

    for(RelationKey key : roads)
       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));

    // adding to vrpBuilder etc. omitted
  }

  public static double highwayFunction(Location end1, Location end2) {
    return 1.5 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
  }

  public static double roadFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
  }

  public static double specialFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
  }
  // RelationsKey class omitted for brevity
}
```

Task: Q06JSNW (4/7)

Task (continuation)

Matching option 2

```
1  public class Q06JSNW_PossibleMatch2 {
2
3    public static void main(String[] args) {
4      VehicleRoutingProblem.Builder vrpBuilder
5        = VehicleRoutingProblem.Builder.newInstance();
6
7      // Network
8      // --- Nodes
9      Location n0 = Location.newInstance(-2, -2);
10     Location n1 = Location.newInstance(-8, -4);
11     Location n2 = Location.newInstance(-8, -1);
12     Location n3 = Location.newInstance(-5, 4);
13     Location n4 = Location.newInstance(-2, -6);
14     Location n5 = Location.newInstance(2, 0);
15     Location n6 = Location.newInstance(4, 5);
16
17     // --- Edges
18     // --- --- roads
19     Set<RelationKey> roads = new HashSet<>();
20     roads.add(RelationKey.newKey(n0,n4));
21     roads.add(RelationKey.newKey(n3, n6));
22     roads.add(RelationKey.newKey(n0, n3));
23     roads.add(RelationKey.newKey(n4, n5));
24
25     // --- --- highways
26     Set<RelationKey> highways = new HashSet<>();
27     highways.add(RelationKey.newKey(n1, n2));
28     highways.add(RelationKey.newKey(n2, n3));
29     highways.add(RelationKey.newKey(n6, n5));
30
31     // --- --- fastways
32     Set<RelationKey> fastways = new HashSet<>();
33     fastways.add(RelationKey.newKey(n0, n5));
34     fastways.add(RelationKey.newKey(n0, n1));
35
36     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
37
38     for(RelationKey key : roads)
39       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
40
41     for(RelationKey key : highways)
42       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
43
44     for(RelationKey key : fastways)
45       costMatrixBuilder.addTransportTime(key.from, key.to, specialFunction(key.from, key.to));
46
47     // adding to vrpBuilder etc. omitted
48   }
49
50   public static double highwayFunction(Location end1, Location end2) {
51     return 1.5 * EuclideanDistanceCalculator
52       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
53   }
54
55   public static double roadFunction(Location end1, Location end2) {
56     return 3 * EuclideanDistanceCalculator
57       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
58   }
59
60   public static double specialFunction(Location end1, Location end2) {
61     return 3 * EuclideanDistanceCalculator
62       .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
63   }
64
65   // RelationsKey class omitted for brevity
66
67 }
```

Task: Q06JSNW (5/7)

Task (continuation)

Matching option 3

labellabel
label

```
1  public class Q06JSNW_PossibleMatch3 {
2
3    public static void main(String[] args) {
4      VehicleRoutingProblem.Builder vrpBuilder
5        = VehicleRoutingProblem.Builder.newInstance();
6
7      // Network
8      // --- Nodes
9      Location n0 = Location.newInstance(-2, -2);
10     Location n1 = Location.newInstance(-8, -4);
11     Location n2 = Location.newInstance(-8, -1);
12     Location n3 = Location.newInstance(-5, 4);
13     Location n4 = Location.newInstance(-2, -6);
14     Location n5 = Location.newInstance(2, 0);
15     Location n6 = Location.newInstance(4, 5);
16
17     // --- Edges
18     // --- --- roads
19     Set<RelationKey> roads = new HashSet<>();
20     roads.add(RelationKey.newKey(n0, n4));
21     roads.add(RelationKey.newKey(n0, n1));
22     roads.add(RelationKey.newKey(n4, n5));
23
24     // --- --- highways
25     Set<RelationKey> highways = new HashSet<>();
26     highways.add(RelationKey.newKey(n1, n2));
27     highways.add(RelationKey.newKey(n3, n6));
28     highways.add(RelationKey.newKey(n0, n3));
29     highways.add(RelationKey.newKey(n6, n5));
30
31     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
32
33     for(RelationKey key : roads)
34       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
35
36     for(RelationKey key : highways)
37       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
38
39     costMatrixBuilder.addTransportTime(n0,n5, specialFunction(n0, n5));
40     costMatrixBuilder.addTransportTime(n2, n3, specialFunction(n2, n3));
41
42     // adding to vrpBuilder etc. omitted
43   }
44
45   public static double highwayFunction(Location end1, Location end2) {
46     return 1.5 * EuclideanDistanceCalculator
47       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
48   }
49
50   public static double roadFunction(Location end1, Location end2) {
51     return 3 * EuclideanDistanceCalculator
52      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
53   }
54
55   public static double specialFunction(Location end1, Location end2) {
56     return 3 * EuclideanDistanceCalculator
57       .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
58   }
59
60   // RelationsKey class omitted for brevity
61
62 }
```

Task: Q06JSNW (6/7)

Task (continuation)

Matching option 4

```
1  public class Q06JSNW_PossibleMatch4 {
2
3    public static void main(String[] args) {
4      VehicleRoutingProblem.Builder vrpBuilder
5        = VehicleRoutingProblem.Builder.newInstance();
6
7      // Network
8      // --- Nodes
9      Location n0 = Location.newInstance(-2, -2);
10     Location n1 = Location.newInstance(-8, -4);
11     Location n2 = Location.newInstance(-8, -1);
12     Location n3 = Location.newInstance(-5, 4);
13     Location n4 = Location.newInstance(-2, -6);
14     Location n5 = Location.newInstance(2, 0);
15     Location n6 = Location.newInstance(4, 5);
16
17     // --- Edges
18     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
19
20     costMatrixBuilder.addTransportTime(n0, n4, roadFunction(n0, n4));
21     costMatrixBuilder.addTransportTime(n0, n5, specialFunction(n0, n5));
22     costMatrixBuilder.addTransportTime(n0, n1, roadFunction(n0, n1));
23     costMatrixBuilder.addTransportTime(n0, n3, roadFunction(n0, n3));
24     costMatrixBuilder.addTransportTime(n4, n5, roadFunction(n4, n5));
25     costMatrixBuilder.addTransportTime(n1, n2, highwayFunction(n1, n2));
26     costMatrixBuilder.addTransportTime(n2, n3, highwayFunction(n2, n3));
27     costMatrixBuilder.addTransportTime(n3, n6, specialFunction(n3, n6));
28     costMatrixBuilder.addTransportTime(n6, n5, highwayFunction(n6, n5));
29
30     // adding to vrpBuilder etc. omitted
31   }
32
33   public static double highwayFunction(Location end1, Location end2) {
34     return 1.5 * EuclideanDistanceCalculator
35       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
36   }
37
38   public static double roadFunction(Location end1, Location end2) {
39     return 3 * EuclideanDistanceCalculator
40      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
41   }
42
43   public static double specialFunction(Location end1, Location end2) {
44     return 3 * EuclideanDistanceCalculator
45       .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
46   }
47
48   // RelationsKey class omitted for brevity
49
50 }
```

Task: Q06JSNW (7/7)

Task (continuation)

Correct solution

☑Possible match 2, ☑Possible match 4

Evaluation 2020

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	1 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Evaluation 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	1 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q07JSALL (1/4)

Introduction

In this task, you first see a complete program. After that, you are shown excerpts from this program and you are asked to associate the correct semantics (meaning) with the language elements shown in the excerpt. Thus, it is your task to associate the correct semantics to pre-selected language elements.

Task

```
1  public class Q07JSNW {
2
3    public static int STUFF = 0;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-2, -2);
12
13     Service n1 = Service.Builder.newInstance("n1")
14     .setLocation(Location.newInstance(3, -4))
15     .addSizeDimension(STUFF, 20)
16     .setTimeWindow(TimeWindow.newInstance(10, 300))
17     .setServiceTime(7)
18     .build();
19
20     Service n2 = Service.Builder.newInstance("n2")
21     .setLocation(Location.newInstance(0, -3))
22     .addSizeDimension(STUFF, 20)
23     .setTimeWindow(TimeWindow.newInstance(20, 250))
24     .setServiceTime(7)
25     .build();
26
27     Service n3 = Service.Builder.newInstance("n3")
28     .setLocation(Location.newInstance(-5, 4))
29     .addSizeDimension(STUFF, 50)
30     .setTimeWindow(TimeWindow.newInstance(0, 140))
31     .setServiceTime(10)
32     .build();
33
34     Service n4 = Service.Builder.newInstance("n4")
35     .setLocation(Location.newInstance(-2, -6))
36     .addSizeDimension(STUFF, 25)
37     .setTimeWindow(TimeWindow.newInstance(0, 120))
38     .setServiceTime(10)
39     .build();
40
41     Service n5 = Service.Builder.newInstance("n5")
42     .setLocation(Location.newInstance(2, 0))
43     .addSizeDimension(STUFF, 25)
44     .setTimeWindow(TimeWindow.newInstance(0, 140))
45     .setServiceTime(10)
46     .build();
47
48     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n4, n5));
49
50     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
51
52     // --- Edges
53     // --- --- roads
54     Set<RelationKey> roads = new HashSet<>();
55     roads.add(RelationKey.newKey(n0, n4));
56     roads.add(RelationKey.newKey(n0, n5));
57     roads.add(RelationKey.newKey(n0, n2));
58     roads.add(RelationKey.newKey(n0, n3));
59     roads.add(RelationKey.newKey(n1, n2));
60     roads.add(RelationKey.newKey(n1, n4));
61
```

Task: Q07JSALL (2/4)

Task (continuation)

```
62      for(RelationKey key : roads)
63        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
64
65      // --- --- highways
66      Set<RelationKey> highways = new HashSet<>();
67      highways.add(RelationKey.newKey(n3, n5));
68      highways.add(RelationKey.newKey(n2, n5));
69
70      for(RelationKey key : highways)
71        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
72
73      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
74      vrpBuilder.setRoutingCost(cm);
75
76      // Vehicle type and instance
77      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
78        .addCapacityDimension(0, 180)
79        .build();
80
81      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
82        .setType(deliverType).setStartLocation(n0).setLatestArrival(850).build();
83
84      vrpBuilder.addVehicle(vehicle);
85
86      VehicleRoutingProblem vrp = vrpBuilder.build();
87    }
88
89    public static double highwayFunction(Location end1, Location end2) {
90      return 1.2 * EuclideanDistanceCalculator
91        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1.3;
92    }
93
94    public static double roadFunction(Location end1, Location end2) {
95      return 2.5 * EuclideanDistanceCalculator
96        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3.2;
97    }
98  }
```

Task: Q07JSALL (3/4)

Task (continuation)

1st Element[a]

```
89 public static double highwayFunction(Location end1, Location end2) {
90       return 1.2 * EuclideanDistanceCalculator
91          .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1.3;
92 }
```

Which description of the sematics and usage of the the highwayFunction() method in the program above is the most appropriate?

- ○ It is used for single customers (Services). Together with an expression that refers to the respective customer it determines how long the time window of this customer is opened
- ○ It is used for single vehicles. Together with an expression it determines the costs that occur upon deployment of the respective vehicle.
- ○ It is used for single edges, that is the connection between two customers. Together with an expression it determines how long it takes a vehicle to travel the respective edge.
- ○ It is used for single depots. Together with an expression it determines the capacity of the vehicles starting from that depot.

2nd Element[b]

```
65 // --- --- highways
66 Set<RelationKey> highways = new HashSet<>();
67 highways.add(RelationKey.newKey(n3, n5));
68 highways.add(RelationKey.newKey(n2, n5));
69
70 for(RelationKey key : highways)
71    costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
```

Which explanation concerning the meaning of the depicted highways set in combination with the ensuing loop in the context of the complete program is most appropriate?

- ○ It allows to group vehicles that will then jointly travel the respective edge.
- ○ All edges (connections between two locations) must be associated with a group because only via a group it is possible to set the duration function of an edge.
- ○ All Service instances that are connected by edges of the same group are merged to one single Service so that the vehicle only needs to visit one of these customers.
- ○ They are an optional program construct that may facillitate the definition of edges. Especially in cases in which a large number of edges are assigned the same (duration) function.

[a] Element missing in actual survey.
[b] Element missing in actual survey.

Task: Q07JSALL (4/4)

Task (continuation)

Correct solution

1st Element

- It is used for single edges, that is the connection between two customers. Together with an expression it determines how long it takes a vehicle to travel the respective edge.

2nd Element

- They are an optional program construct that may facillitate the definition of edges. Especially in cases in which a large number of edges are assigned the same (duration) function.

Evaluation 2020, 2021

	Parameter	Value	Parameter	Value
1st Element	Answer Cell	Yes	NegPointsP	5 (default)
	CorAnswersString	Yes	WrongAnswAllowed	0 (default)
	Points	5	MissingAnswAllowed	0 (default)
	WrgAnswersString	No (default)	MaxPointsToGet	5 (default)
	Parameter	Value	Parameter	Value
2nd Element	Answer Cell	Yes	NegPointsP	5 (default)
	CorAnswersString	Yes	WrongAnswAllowed	0 (default)
	Points	5	MissingAnswAllowed	0 (default)
	WrgAnswersString	No (default)	MaxPointsToGet	5 (default)

Task: Q08JSALL (1/4)

Introduction

In this task, you first see a complete program. The program features some comments. These comments represent TODOs, i.e. future programming tasks. In addition, you'll find some questions that ask for the elements, that will be affected by these TODOs. Your task is to answer these questions correctly.

Task

```
1  public class Q08JSALL2ND {
2    public static int SOAP = 0;
3    public static int PAPER = 1;
4
5    public static void main(String[] args) {
6      Examples.createOutputFolder();
7
8      VehicleRoutingProblem.Builder vrpBuilder
9        = VehicleRoutingProblem.Builder.newInstance();
10
11     // Network
12     // --- Nodes
13     /* TODO: The customer with the shortest service time must be added to the
14     * respective depot */
15     Location n0 = Location.newInstance(2, 4);
16     Location n1 = Location.newInstance(-5, -2);
17     Location n2 = Location.newInstance(0, -3);
18     Location n3 = Location.newInstance(-5, 4);
19     Location n4 = Location.newInstance(-2, -2);
20     Location n5 = Location.newInstance(2, 0);
21     Location n6 = Location.newInstance(-2, -6);
22     Location n7 = Location.newInstance(3, -4);
23
24     Delivery dn1soap = Delivery.Builder.newInstance("dn1soap")
25       .setLocation(n1)
26       .addSizeDimension(SOAP, 10)
27       .setTimeWindow(TimeWindow.newInstance(20,220))
28       .setServiceTime(8)
29       .build();
30
31     Delivery dn1paper = Delivery.Builder.newInstance("dn1paper")
32       .setLocation(n1)
33       .addSizeDimension(PAPER, 20)
34       .setTimeWindow(TimeWindow.newInstance(20,220))
35       .setServiceTime(8)
36       .build();
37
38     Delivery dn2soap = Delivery.Builder.newInstance("dn2soap")
39       .setLocation(n2)
40       .addSizeDimension(SOAP, 20)
41       .setTimeWindow(TimeWindow.newInstance(20, 250))
42       .setServiceTime(2)
43       .build();
44
45     Delivery dn3soap = Delivery.Builder.newInstance("dn3soap")
46       .setLocation(n3)
47       .addSizeDimension(SOAP, 50)
48       .setTimeWindow(TimeWindow.newInstance(0, 140))
49       .setServiceTime(30)
50       .build();
51
52     Delivery dn4soap = Delivery.Builder.newInstance("dn4soap")
53       .setLocation(n4)
54       .addSizeDimension(SOAP, 60)
55       .setTimeWindow(TimeWindow.newInstance(0, 150))
56       .setServiceTime(10)
57       .build();
58
```

Task: Q08JSALL (2/4)

Task (continuation)

```
59      Delivery dn4paper = Delivery.Builder.newInstance("dn4paper")
60        .setLocation(n4)
61        .addSizeDimension(PAPER, 70)
62        .setTimeWindow(TimeWindow.newInstance(0, 150))
63        .setServiceTime(10)
64        .build();
65
66      Delivery dn5soap = Delivery.Builder.newInstance("dn5soap")
67        .setLocation(n5)
68        .addSizeDimension(SOAP, 30)
69        .setTimeWindow(TimeWindow.newInstance(0, 140))
70        .setServiceTime(10)
71        .build();
72
73      Delivery dn5paper = Delivery.Builder.newInstance("dn5paper")
74        .setLocation(n5)
75        .addSizeDimension(PAPER, 40)
76        .setTimeWindow(TimeWindow.newInstance(0, 140))
77        .setServiceTime(10)
78        .build();
79
80      Delivery dn6soap = Delivery.Builder.newInstance("dn6soap")
81        .setLocation(n6)
82        .addSizeDimension(SOAP, 25)
83        .setTimeWindow(TimeWindow.newInstance(0, 120))
84        .setServiceTime(10)
85        .build();
86
87      Delivery dn7soap = Delivery.Builder.newInstance("dn7soap")
88        .setLocation(n7)
89        .addSizeDimension(SOAP, 20)
90        .setTimeWindow(TimeWindow.newInstance(10, 300))
91        .setServiceTime(7)
92        .build();
93
94      Location n8 = Location.newInstance(-5, -6);
95
96      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
97
98      // --- Edges
99      // --- --- roads
100     Set<RelationKey> roads = new HashSet<>();
101     roads.add(RelationKey.newKey(n0, n6));
102     roads.add(RelationKey.newKey(n0, n5));
103     roads.add(RelationKey.newKey(n3, n4));
104     roads.add(RelationKey.newKey(n0, n3));
105     roads.add(RelationKey.newKey(n7, n2));
106     roads.add(RelationKey.newKey(n7, n6));
107     roads.add(RelationKey.newKey(n3, n0));
108     roads.add(RelationKey.newKey(n5, n0));
109     roads.add(RelationKey.newKey(n3, n1));
110     roads.add(RelationKey.newKey(n1, n8));
111     roads.add(RelationKey.newKey(n4, n2));
112     roads.add(RelationKey.newKey(n8, n6));
113
114     for(RelationKey key : roads)
115       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
116
117     // --- --- highways
118     /* TODO: two highways will later be modelled here:
119     * one going from the soap depot to the customer with the highest demands in both soap
120     * and paper and the other going from the customer with the highest soap and
121     * paper demands to the paper depot.    */
122     Set<RelationKey> highways = new HashSet<>();
123     highways.add(RelationKey.newKey(n3, n5));
124     highways.add(RelationKey.newKey(n2, n5));
125
126     for(RelationKey key : highways)
127       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
128
```

Task: Q08JSALL (3/4)

Task (continuation)

```
129     IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
130     vrpBuilder.setRoutingCost(cm);
131
132     // Vehicle type and instance
133
134     VehicleType soapVehicle = VehicleTypeImpl.Builder.newInstance("soapVehicle")
135       .addCapacityDimension(SOAP, 180).build();
136
137     VehicleType paperVehicle = VehicleTypeImpl.Builder.newInstance("paperVehicle")
138       .addCapacityDimension(PAPER, 180).build();
139
140     VehicleImpl soapVehicleInstance = VehicleImpl.Builder.newInstance("soapVehicleInstance")
141       .setType(soapVehicle)
142       .setStartLocation(n0)
143       .build();
144
145     VehicleImpl paperVehicleInstance = VehicleImpl.Builder.newInstance("paperVehicleInstance")
146       .setType(paperVehicle)
147       .setStartLocation(n8)
148       .build();
149
150     vrpBuilder.addAllJobs(Arrays.asList(dn1paper, dn1soap, dn2soap, dn3soap, dn4paper, dn4soap,
151                                         dn5paper, dn5soap, dn6soap, dn7soap));
152     vrpBuilder.addVehicle(soapVehicleInstance);
153     vrpBuilder.addVehicle(paperVehicleInstance);
154
155     VehicleRoutingProblem vrp = vrpBuilder.build();
156   }
157 }
```

1. Which customer must be added?

Read the comment in lines 13 and 14. Which customer must be added to the respective depot according to the comment?

Note: Some of the provided answers might not be customers.

○ n0	○ n1	○ n2
○ n3	○ n4	○ n5
○ n6	○ n7	○ n8

2. Which is the correct depot?

Read the comment in lines 13 and 14 . To which depot must the customer be added to?

Note: Some of the provided answers might not be depots.

○ n0	○ n1	○ n2
○ n3	○ n4	○ n5
○ n6	○ n7	○ n8

3. Which nodes are affected?

Read the comment that spans from lines 118 to line 121. What are the three nodes this comment refers to?

Note: Some of the provided answers might not be customers.

☐ n0	☐ n1	☐ n2
☐ n3	☐ n4	☐ n5
☐ n6	☐ n7	☐ n8

Task: Q08JSALL (4/4)

Task (continuation)

Correct solution

1.	◉ n2
2.	◉ n0
3.	☑ n0, ☑ n4, ☑ n8

Evaluation 2020, 2021

1.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	2

2.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	2

3.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	6

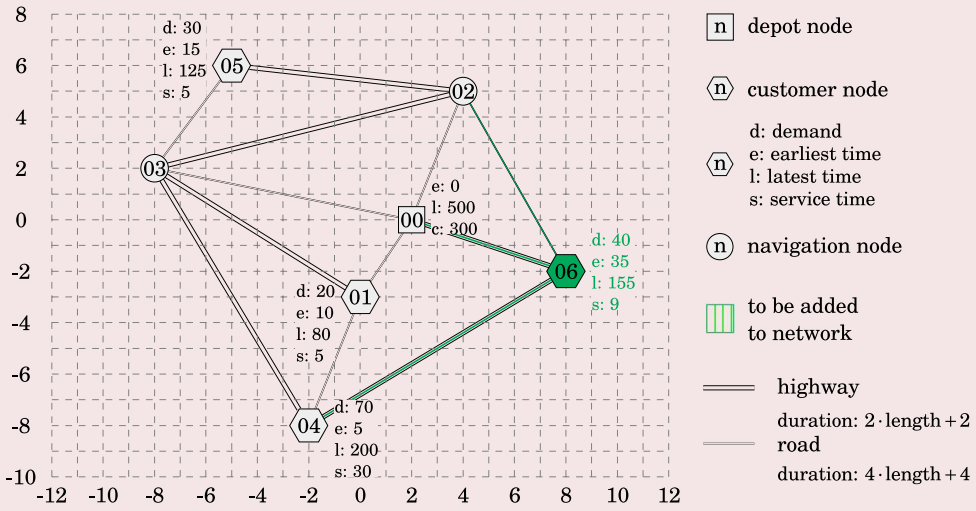
Task: Q09JSNW (1/4)

Introduction

In this task you'll find a graphical representation of a network. The network comprises a set of customers with demands, time windows and service times. Further below is the corresponding program. However, the program still shows some gaps. The gaps can also be seen in the illustration: they correspond to the elements drawn using green color. Your task is to fill in the gaps so that the program describes the complete network.

Task

Explanation: Below is the illustration of a network. The network comprises a set of customers with demands, time windows and service times.
Further below is the corresponding JSprit model. However, the model still has some gaps. The gaps can also be seen in the illustration: they correspond to the elements drawn using green color.
Fill in the gaps so that the JSprit model represents the complete network.




```
1  public class Q09JSNW {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(2, 0);
12
13     Service n1 = Service.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(0,-3))
15       .addSizeDimension(STUFF, 20)
16       .setTimeWindow(TimeWindow.newInstance(10, 80))
17       .setServiceTime(5)
18       .build();
19
20     Location n2 = Location.newInstance(4,5);
21
22     Location n3 = Location.newInstance(-8, 2);
23
```

Task: Q09JSNW (2/4)

Task (continuation)

```
24      Service n4 = Service.Builder.newInstance("n4")
25        .setLocation(Location.newInstance(-2, -8))
26        .addSizeDimension(STUFF, 70)
27        .setTimeWindow(TimeWindow.newInstance(5, 200))
28        .setServiceTime(30)
29        .build();
30
31      Service n5 = Service.Builder.newInstance("n5")
32        .setLocation(Location.newInstance(-5, 6))
33        .addSizeDimension(STUFF, 30)
34        .setTimeWindow(TimeWindow.newInstance(15, 125))
35        .setServiceTime(5)
36        .build();
37
38      //TASK 1.1: SOME TEXT TO BE ADDED HERE
39
40      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
41
42      // --- Edges
43      // --- --- roads
44      Set<RelationKey> roads = new HashSet<>();
45      roads.add(RelationKey.newKey(n0, n2));
46      roads.add(RelationKey.newKey(n0, n3));
47      roads.add(RelationKey.newKey(n0, n1));
48      roads.add(RelationKey.newKey(n1, n4));
49      roads.add(RelationKey.newKey(n3, n5));
50      // TASK 2: SOME TEXT TO BE ADDED HERE
51
52      for(RelationKey key : roads)
53        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
54
55      // --- --- highways
56      Set<RelationKey> highways = new HashSet<>();
57      highways.add(RelationKey.newKey(n2, n5));
58      highways.add(RelationKey.newKey(n2, n3));
59      highways.add(RelationKey.newKey(n1, n3));
60      highways.add(RelationKey.newKey(n3, n4));
61      // TASK 3: SOME TEXT TO BE ADDED HERE
62
63      for(RelationKey key : highways)
64        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
65
66      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
67      vrpBuilder.setRoutingCost(cm);
68
69      // Vehicle type and instance
70
71      VehicleType vehicles = VehicleTypeImpl.Builder.newInstance("vehicles")
72        .addCapacityDimension(STUFF, 300)
73        .build();
74
75      VehicleImpl vehiclesInstance = VehicleImpl.Builder.newInstance("soapVehicleInstance")
76        .setType(vehicles)
77        .setStartLocation(n0)
78        .setLatestArrival(300)
79        .build();
80
81      vrpBuilder.addAllJobs(Arrays.asList(n1, n4, n5 /* TASK 1.2: SOME TEXT TO BE ADDED HERE */ ));
82
83      vrpBuilder.addVehicle(vehiclesInstance);
84
85      VehicleRoutingProblem vrp = vrpBuilder.build();
86
87    }
88
```

Task: Q09JSNW (3/4)

Task (continuation)

```
89   public static double highwayFunction(Location end1, Location end2) {
90     return 2 * EuclideanDistanceCalculator
91       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
92   }
93
94   public static double roadFunction(Location end1, Location end2) {
95     return 4 * EuclideanDistanceCalculator
96       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
97   }
98
99   // static class RelationKey (omitted for brevity)
100 }
```

Task 1: In the gap program above, you find comments that read "TASK 1.1: TEXT TO BE ADDED HERE" and "TASK 1.2: TEXT TO BE ADDED HERE". Both comments mark places in the program where you need to insert additional text (code) in order for the program to comply with the network given in the illustration above. In the following two text areas, enter the code (one or more lines) that should replace these comments in order to complete the program.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Task 1.1: Enter answer here

Task 1.2: Enter answer here

Task 2: In the following text area, enter the code (one or more lines) that should replace the comment " TASK 2: SOME TEXT TO BE ADDED HERE " in the complete program above.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task 3: In the following text area, enter the code (one or more lines) that should replace the comment " TASK 3: SOME TEXT TO BE ADDED HERE " in the complete program above.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task: Q09JSNW (4/4)

Task (continuation)

Correct solution

Task 1.1:

```
38 Service n6 = Service.Builder.newInstance("n6")
39              .setLocation(Location.newInstance(8, -2))
40              .addSizeDimension(STUFF, 40)
41              .setTimeWindow(TimeWindow.newInstance(35, 155))
42              .setServiceTime(9)
43              .build();
```

Task 1.2:

```
81 vrpBuilder.addAllJobs(Arrays.asList(n1, n4, n5, n6 ));
```

Task 2:

```
50 roads.add(RelationKey.newKey(n2, n6));
```

Task 3:

```
61 highways.add(RelationKey.newKey(n0, n6));
62 highways.add(RelationKey.newKey(n4, n6));
```

Evaluation 2020, 2021

Task 1: 6 Points.
Task 2: 2 Points.
Task 3: 2 Points.

Task: Q09JSAG (1/5)

Introduction

In this task, you'll find an incomplete program together with a visual representation of the network represented by this program. Complete the program in a way so that the target network and agent behaviour visualised at the bottom of the page result. Note that the modelled behaviour, i.e. the tour, must be a likely outcome of the agent behaviour that you modelled in the program.

Note: In this task, there are some nodes for which demands, time windows, and service times are to be defined even though the vehicle must not service them! Navigation nodes (as well as those customer not supposed to be serviced) can be visited by the agent (vehicle) but they will not receive a delivery. Find a way to program this.

Task

```
1  public class Q09JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      /* TASK 1 BEGINNING */
10     // Network
11     // --- Nodes
12     Location l0 = Location.newInstance(2, 4);
13     Location l1 = Location.newInstance(-2, -8);
14     Location l2 = Location.newInstance(8, -6);
15     Location l3 = Location.newInstance(-10, -6);
16     Location l4 = Location.newInstance(-4, -2);
17     Location l5 = Location.newInstance(-5, 6);
18     Location l6 = Location.newInstance(7, 4);
19
20     Service n1 = Service.Builder.newInstance("n1")
21       .setLocation(l1)
22       .addSizeDimension(0, 30)
23       .setTimeWindow(TimeWindow.newInstance(10, 90))
24       .setServiceTime(5)
25       .build();
26
27     vrpBuilder.addAllJobs(Arrays.asList(n1));
28     /* TASK 1 END */
29
30     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
31
32     // --- roads
33     Set<RelationKey> roads = new HashSet<>();
34     roads.add(RelationKey.newKey(l0, l1));
35     roads.add(RelationKey.newKey(l0, l2));
36     roads.add(RelationKey.newKey(l0, l6));
37     roads.add(RelationKey.newKey(l0, l4));
38     roads.add(RelationKey.newKey(l4, l5));
39     roads.add(RelationKey.newKey(l1, l2));
40
41     for(RelationKey key : roads)
42       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
43
44     // --- highways
45     Set<RelationKey> highways = new HashSet<>();
46     highways.add(RelationKey.newKey(l2, l6));
47     highways.add(RelationKey.newKey(l6, l5));
48     highways.add(RelationKey.newKey(l3, l5));
49     highways.add(RelationKey.newKey(l1, l3));
50     highways.add(RelationKey.newKey(l3, l4));
51
52     for(RelationKey key : highways)
53       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
54
55     IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
56     vrpBuilder.setRoutingCost(cm);
57
```

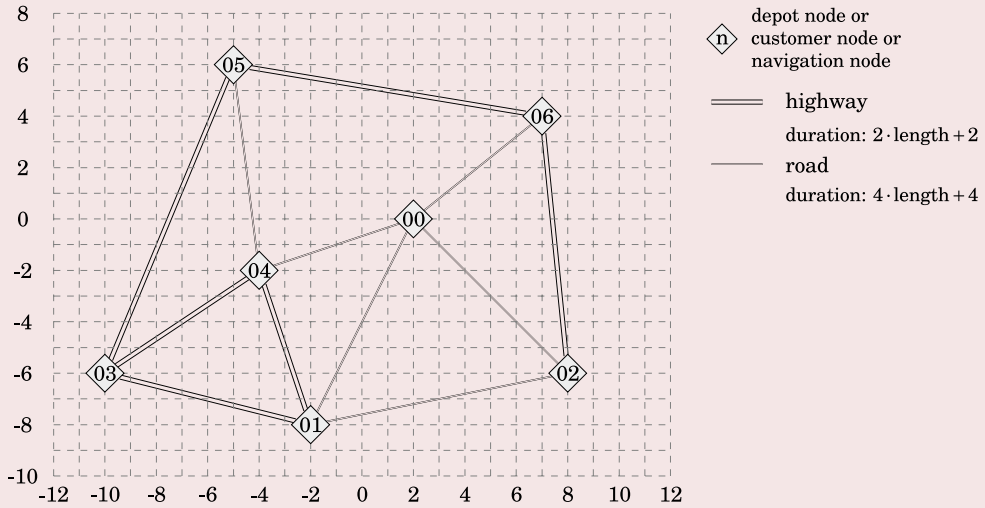
Task: Q09JSAG (2/5)

Task (continuation)

Task

```
58      /* TASK 2 BEGINNING */
59      // Vehicle type definition
60      VehicleType stuffVehicleType = VehicleTypeImpl.Builder.newInstance("stuffVehicleType")
61        .build();
62
63      // Vehicle instance defintion
64      VehicleImpl stuffVehicleInstance = VehicleImpl.Builder.newInstance("stuffVehicleInstance")
65        .setType(stuffVehicleType)
66        .setStartLocation(l0)
67        .build();
68      // Adding vehicle instance to the problem
69      vrpBuilder.addVehicle(stuffVehicleInstance);
70      /* TASK 2 END */
71
72      VehicleRoutingProblem vrp = vrpBuilder.build();
73    }
74    public static double highwayFunction(Location end1, Location end2) {
75      return 2 * EuclideanDistanceCalculator
76      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
77    }
78
79    public static double roadFunction(Location end1, Location end2) {
80      return 4 * EuclideanDistanceCalculator
81        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
82    }
83
84    //-- static class RelationKey omitted for brevity
85  }
```

Current state

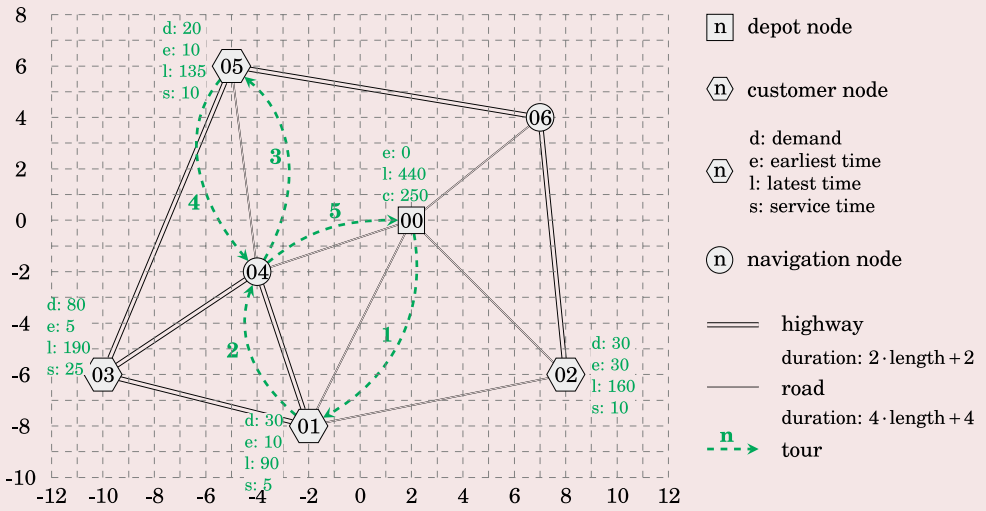


Task: Q09JSAG (3/5)

Task (continuation)

Task

Target state



Task 1: From the listing above, copy and paste the code betweenthe comment "TASK 1 : BEGINNING" and the comment "TASK 1 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task 2: From the listing above, copy and paste the code betweenthe comment "TASK 2 : BEGINNING" and the comment "TASK 2 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration. Hint: Take a good look! Something is missing!
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task: Q09JSAG (4/5)

Task (continuation)

Correct solution

Task 1:

```
//Network
      // --- Nodes
      Location l0 = Location.newInstance(2, 4);
      Location l1 = Location.newInstance(-2, -8);
      Location l2 = Location.newInstance(8, -6);
      Location l3 = Location.newInstance(-10, -6);
      Location l4 = Location.newInstance(-4, -2);
      Location l5 = Location.newInstance(-5, 6);
      Location l6 = Location.newInstance(7, 4);

      Service n1 = Service.Builder.newInstance("n1")
            .setLocation(l1)
            .addSizeDimension(0, 30)
            .setTimeWindow(TimeWindow.newInstance(10, 90))
            .setServiceTime(5)
            .build();

      Service n2 = Service.Builder.newInstance("n2")
                  .setLocation(l2)
                  .addSizeDimension(0, 30)
                  .setTimeWindow(TimeWindow.newInstance(30, 160))
                  .setServiceTime(10)
                  .build();

      Service n3 = Service.Builder.newInstance("n3")
                  .setLocation(l3)
                  .addSizeDimension(0, 80)
                  .setTimeWindow(TimeWindow.newInstance(5, 190))
                  .setServiceTime(25)
                  .build();

      Service n5 = Service.Builder.newInstance("n5")
                  .setLocation(l5)
                  .addSizeDimension(0, 20)
                  .setTimeWindow(TimeWindow.newInstance(10, 135))
                  .setServiceTime(10)
                  .build();

      vrpBuilder.addAllJobs(Arrays.asList(n1,n5));
```

Task 2:

```
    // Vehicle type definition
    VehicleType stuffVehicleType = VehicleTypeImpl.Builder.newInstance("stuffVehicleType")
          .addCapacityDimensions(STUFF, 50)
          .build();

    // Vehicle instance defintion
    VehicleImpl stuffVehicleInstance = VehicleImpl.Builder.newInstance("stuffVehicleInstance")
          .setType(stuffVehicleType)
          .setStartLocation(l0)
          .build();
    // Adding vehicle instance to the problem
    vrpBuilder.addVehicle(stuffVehicleInstance);
```

Evaluation 2020

Task 1: 10 Points.
Task 2: Not evaluated

Task: Q09JSAG (5/5)

Evaluation 2021

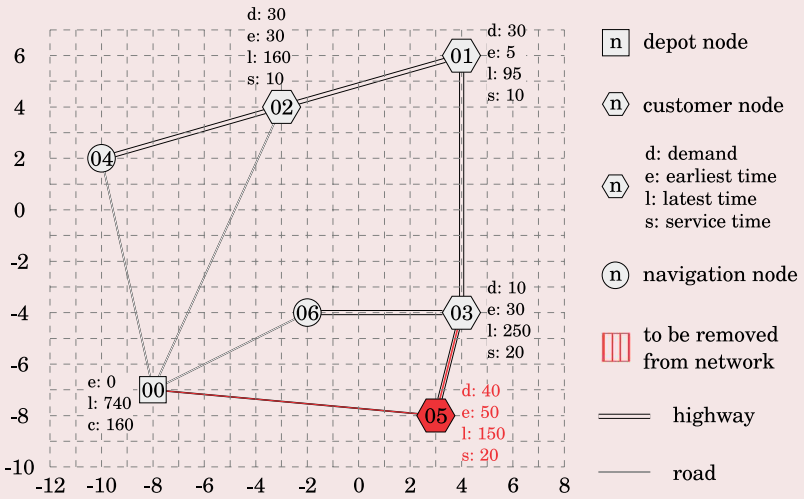
Task 1: 7 Points. Syntactical mistakes -1P; missing demand definitions (all) -4 P.; missing demand definitions (one correctly present) -2 P.; missing lines of code -1 P.; wrong value for demand or time -1 P. (max -2 P.); wrong or missing customer declaration -2 P.; wrong code (no own code additions) -8 P.; wrong customer label (specified in constructor) -1 P.; superflous code -1 P.; mixed-up customer (mistaken customer x for y) -1 P., wrong or incompatible asssigment (-2 P.); wrong location -1 P.; unecessary changes to pre-defined customer (introduction of mistakes) -1 P., missing call of build() method -1 P.
Task 2: 3 Points. Missing capacity dimension -2 P., Wrong product (only 'STUFF' and '0' are correct) -1P, Wrong value for either capacity dimension or latest time -1 P. (-2 P. max), missing latest time -2 P.

Task: Q10JSNW (1/3)

Introduction

In this task, you will find an illustration of a network comprised of highways, roads, navigation and customer nodes. In addition, you will find a program that corresponds to this illustration.In the illustration, some elements (e.g. nodes, demands, highways, etc.) are drawn in red color. These are the elements that are to be removed from the program.
Note: At the bottom of the page is a text area. Copy and paste the code that corresponds to the elements that need to be deleted into this text area. The order in which you paste the elements is not important.

Task




```
1  public class Q10JSNW {
2
3    public static int STUFF = 0;
4
5    public static void main(String[]] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      Location n0 = Location.newInstance(-8, -7);
10
11     Service n1 = Service.Builder.newInstance("n1")
12       .setLocation(Location.newInstance(4, 6))
13       .addSizeDimension(STUFF, 30)
14       .setTimeWindow(TimeWindow.newInstance(5, 95))
15       .setServiceTime(10)
16       .build();
17
18     Service n2 = Service.Builder.newInstance("n2")
19       .setLocation(Location.newInstance(-3, 4))
20       .addSizeDimension(STUFF, 30)
21       .setTimeWindow(TimeWindow.newInstance(5, 95))
22       .setServiceTime(10)
23       .build();
24
25     Service n3 = Service.Builder.newInstance("n3")
26       .setLocation(Location.newInstance(4, -4))
27       .addSizeDimension(STUFF, 10)
28       .setTimeWindow(TimeWindow.newInstance(30, 250))
29       .setServiceTime(20)
30       .build();
31
32     Location n4 = Location.newInstance(-10, 2);
33
```

Task: Q10JSNW (2/3)

Task (continuation)

```
60      Service n5 = Service.Builder.newInstance("n5")
61        .setLocation(Location.newInstance(3, -8))
62        .addSizeDimension(STUFF, 40)
63        .setTimeWindow(TimeWindow.newInstance(50, 150))
64        .setServiceTime(20)
65        .build();
66
67      Location n6 = Location.newInstance(-2, -4);
68
69      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
70
71      // --- roads
72      Set<RelationKey> roads = new HashSet<>();
73      roads.add(RelationKey.newKey(n0, n4));
74      roads.add(RelationKey.newKey(n0, n2));
75      roads.add(RelationKey.newKey(n0, n6));
76      roads.add(RelationKey.newKey(n0, n5));
77
78      for(RelationKey key : roads)
79        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
80
81      // --- highways
82      Set<RelationKey> highways = new HashSet<>();
83      highways.add(RelationKey.newKey(n2, n4));
84      highways.add(RelationKey.newKey(n1, n2));
85      highways.add(RelationKey.newKey(n1, n3));
86      highways.add(RelationKey.newKey(n3, n5));
87      highways.add(RelationKey.newKey(n3, n6));
88
89      for(RelationKey key : highways)
90        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
91
92      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
93      vrpBuilder.setRoutingCost(cm);
94
95      VehicleType stuffVehicleType = VehicleTypeImpl.Builder.newInstance("stuffVehicleType")
96        .addCapacityDimension(STUFF, 160).build();
97
98      VehicleImpl stuffVehicleInstance = VehicleImpl.Builder.newInstance("stuffVehicleInstance")
99        .setType(stuffVehicleType)
100       .setLatestArrival(740)
101       .setStartLocation(n0)
102       .build();
103     // Adding vehicle instance to the problem
104     vrpBuilder.addVehicle(stuffVehicleInstance);
105
106     VehicleRoutingProblem vrp = vrpBuilder.build();
107   }
108
109   public static double highwayFunction(Location end1, Location end2) {
110     return 1.5 * EuclideanDistanceCalculator
111       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
112   }
113
114   public static double roadFunction(Location end1, Location end2) {
115     return 3 * EuclideanDistanceCalculator
116       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
117   }
118
119 }
```

Task: Q10JSNW (3/3)

Task (continuation)

From above JSprit program, copy those lines that need to be deleted and paste them in the following text area (in an arbitrary order).
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Correct solution

```
60 Service n5 = Service.Builder.newInstance("n5")
61              .setLocation(Location.newInstance(3, -8))
62              .addSizeDimension(STUFF, 40)
63              .setTimeWindow(TimeWindow.newInstance(50, 150))
64              .setServiceTime(20)
65              .build();
76 roads.add(RelationKey.newKey(n0, n5));
```

Evaluation 2020, 2021

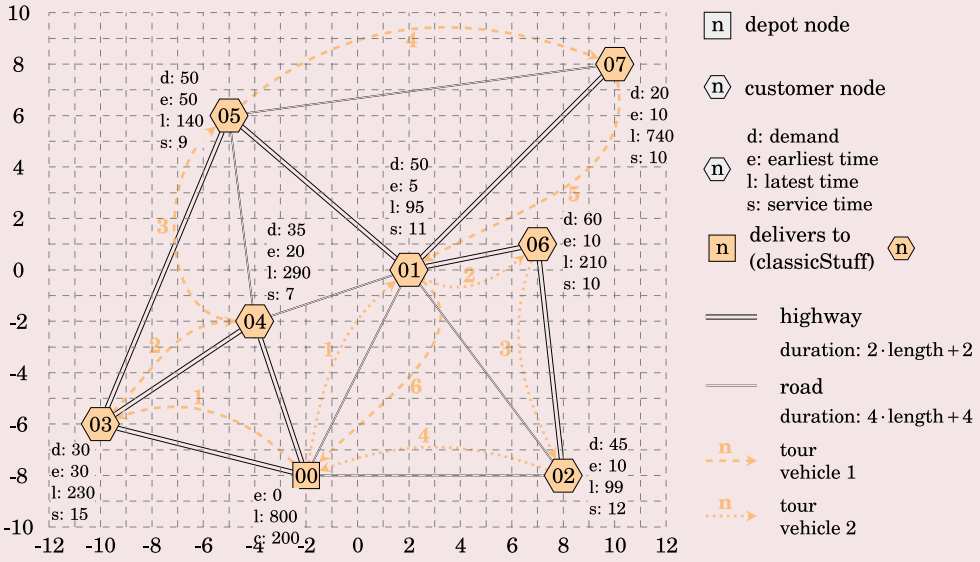
Scheme: 10 Points. Missing removal: -5 Points.

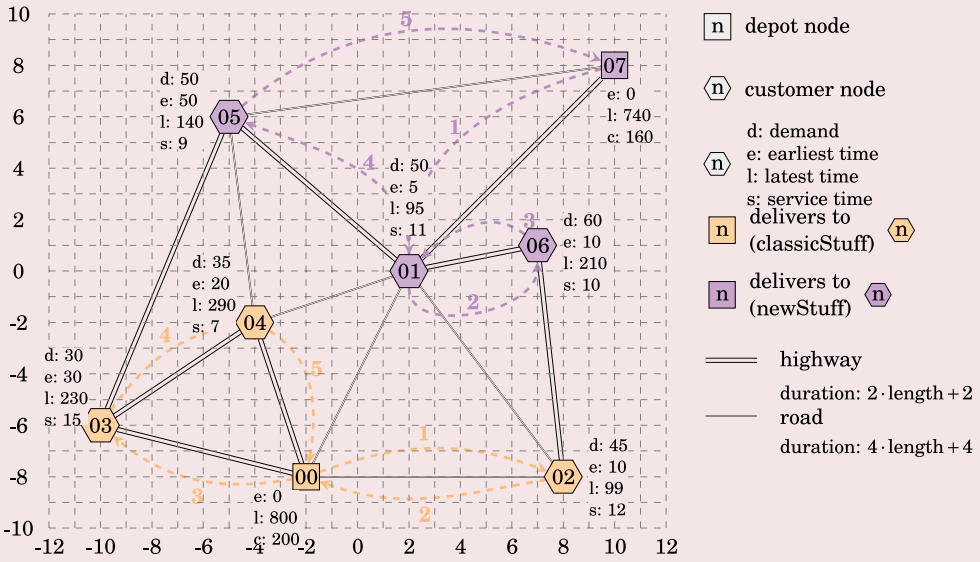
Task: Q11JSALL (1/5)

Introduction

In this task, you will find two graphical representations of networks that are comprised of highways, roads, navigation nodes and customer nodes.
In the first graphical representation you will find one depot together with **seven customer nodes.** The customer nodes are visited by two different tours.
After the two graphical network representations, you will find a program that corresponds **to the first graphical network representation** (i.e. it describes this representation).
The second graphical network representation displays the target state in which you are to transform the program: One of the customer nodes was transformed into a depot from which a new product is delivered to some of the customers. These customers only have a demand for the new product and do no longer require the old one (in other words, they are only supplied by one depot).
The program features comments that mark the beginning and the end of program sections that must be modified in order to transform the program into the target state. At the end of the page, there are corresponding text areas, in which you copy, paste and modify the original code in a suitable way.

Task





Task: Q11JSALL (2/5)

Task (continuation)

```
1  public class Q11JSALL {
2    public static final int CLASSIC_STUFF = 0;
3    public static final int NEW_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     /* TASK 1 BEGINNING */
12     Location n0 = Location.newInstance(-2, -8);
13
14     Delivery n1 = Delivery.Builder.newInstance("n1")
15       .setLocation(Location.newInstance(2,0))
16       .addSizeDimension(CLASSIC_STUFF, 50)
17       .setTimeWindow(TimeWindow.newInstance(5, 95))
18       .setServiceTime(11)
19       .build();
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(8,-8))
22       .addSizeDimension(CLASSIC_STUFF, 45)
23       .setTimeWindow(TimeWindow.newInstance(10, 99))
24       .setServiceTime(12)
25       .build();
26     Delivery n3 = Delivery.Builder.newInstance("n3")
27       .setLocation(Location.newInstance(-10,-6))
28       .addSizeDimension(CLASSIC_STUFF, 40)
29       .setTimeWindow(TimeWindow.newInstance(10, 99))
30       .setServiceTime(12)
31       .build();
32     Delivery n4 = Delivery.Builder.newInstance("n4")
33       .setLocation(Location.newInstance(-4, -2))
34       .addSizeDimension(CLASSIC_STUFF, 35)
35       .setTimeWindow(TimeWindow.newInstance(20, 290))
36       .setServiceTime(7)
37       .build();
38     Delivery n5 = Delivery.Builder.newInstance("n5")
39       .setLocation(Location.newInstance(-5, 6))
40       .addSizeDimension(CLASSIC_STUFF, 50)
41       .setTimeWindow(TimeWindow.newInstance(50, 140))
42       .setServiceTime(9)
43       .build();
44     Delivery n6 = Delivery.Builder.newInstance("n6")
45       .setLocation(Location.newInstance(7, 1))
46       .addSizeDimension(CLASSIC_STUFF, 60)
47       .setTimeWindow(TimeWindow.newInstance(10, 210))
48       .setServiceTime(10)
49       .build();
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(10, 8))
52       .addSizeDimension(CLASSIC_STUFF, 20)
53       .setTimeWindow(TimeWindow.newInstance(10, 740))
54       .setServiceTime(10)
55       .build();
56
57     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n4, n5, n6, n7));
58
59     /* TASK 1 END */
60
61     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62
63     // --- roads
64     Set<RelationKey> roads = new HashSet<>();
65     roads.add(RelationKey.newKey(n0, n2));
66     roads.add(RelationKey.newKey(n1, n2));
67     roads.add(RelationKey.newKey(n1, n4));
68     roads.add(RelationKey.newKey(n5, n7));
69
```

Task: Q11JSALL (3/5)

Task (continuation)

```
70      for(RelationKey key : roads)
71        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
72
73      // --- highways
74      Set<RelationKey> highways = new HashSet<>();
75      highways.add(RelationKey.newKey(n0, n3));
76      highways.add(RelationKey.newKey(n0, n4));
77      highways.add(RelationKey.newKey(n0, n3));
78      highways.add(RelationKey.newKey(n1, n5));
79      highways.add(RelationKey.newKey(n1, n6));
80      highways.add(RelationKey.newKey(n1, n7));
81      highways.add(RelationKey.newKey(n2, n6));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      /* TASK 2 BEGINNING */
90      // Vehicle type definition
91      VehicleType classicStuffType = VehicleTypeImpl.Builder.newInstance("classicStuffType")
92        .addCapacityDimension(CLASSIC_STUFF, 200).build();
93
94      // Vehicle instance defintion
95      VehicleImpl classicStuffInstance = VehicleImpl.Builder.newInstance("classicStuffInstance")
96        .setType(classicStuffType)
97        .setStartLocation(n0)
98        .build();
99
100     // Add vehicle instance to the problem
101     vrpBuilder.addVehicle(classicStuffInstance);
102
103     /* TASK 2 END */
104
105     VehicleRoutingProblem vrp = vrpBuilder.build();
106   }
107
108   public static double highwayFunction(Location end1, Location end2) {
109     return 1.5 * EuclideanDistanceCalculator
110       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
111   }
112
113   public static double roadFunction(Location end1, Location end2) {
114     return 3 * EuclideanDistanceCalculator
115       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
116   }
117 }
```

Task 1: From the listing above, copy and paste the code betweenthe comment "TASK 1 : BEGINNING" and the comment "TASK 1 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Task: Q11JSALL (4/5)

Task (continuation)

Task 2: From the listing above, copy and paste the code betweenthe comment "TASK 2 : BEGINNING" and the comment "TASK 2 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Correct solution

Task 1:

```
12 Location n0 = Location.newInstance(2, 4);
13
14 Delivery n1 = Delivery.Builder.newInstance("n1")
15    .setLocation(Location.newInstance(2,0))
16    .addSizeDimension(NEW_STUFF, 50)
17    .setTimeWindow(TimeWindow.newInstance(5, 95))
18    .setServiceTime(11)
19    .build();
20
21 Delivery n2 = Delivery.Builder.newInstance("n2")
22    .setLocation(Location.newInstance(8,-8))
23    .addSizeDimension(CLASSIC_STUFF, 45)
24    .setTimeWindow(TimeWindow.newInstance(10, 99))
25    .setServiceTime(12)
26    .build();
27
28 Delivery n3 = Delivery.Builder.newInstance("n3")
29    .setLocation(Location.newInstance(-10,-6))
30    .addSizeDimension(CLASSIC_STUFF, 40)
31    .setTimeWindow(TimeWindow.newInstance(10, 99))
32    .setServiceTime(12)
33    .build();
34
35 Delivery n4 = Delivery.Builder.newInstance("n4")
36    .setLocation(Location.newInstance(-4, -2))
37    .addSizeDimension(CLASSIC_STUFF, 35)
38    .setTimeWindow(TimeWindow.newInstance(20, 290))
39    .setServiceTime(7)
40    .build();
41
42 Delivery n5 = Delivery.Builder.newInstance("n5")
43    .setLocation(Location.newInstance(-5, 6))
44    .addSizeDimension(NEW_STUFF, 50)
45    .setTimeWindow(TimeWindow.newInstance(50, 140))
46    .setServiceTime(9)
47    .build();
48
49 Delivery n6 = Delivery.Builder.newInstance("n6")
50    .setLocation(Location.newInstance(7, 1))
51    .addSizeDimension(NEW_STUFF, 60)
52    .setTimeWindow(TimeWindow.newInstance(10, 210))
53    .setServiceTime(10)
54    .build();
55
56 Location n7 = Location.newInstance(10, 8);
57
58 vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n4, n5, n6));
```

Task: Q11JSALL (5/5)

Correct solution (continuation)

Task 2:

```
100 // Vehicle type definition
101 VehicleType newStuffType = VehicleTypeImpl.Builder.newInstance("newStuffType")
102   .addCapacityDimension(NEW_STUFF, 160).build();
103
104 // Vehicle instance defintion
105 VehicleImpl newStuffInstance = VehicleImpl.Builder.newInstance("newStuffInstance")
106   .setType(newStuffType)
107   .setStartLocation(n7)
108   .build();
109
110 // Add vehicle instance to the problem
111 vrpBuilder.addVehicle(newStuffInstance);
```

Evaluation 2020, 2021

Task 1: 5 Points.
Task 2: 5 Points.