11.3.1 A graph representing cities and roads

Next we describe a graph representing cities and roads between cities. This graph is represented by a class `RoadAndCityGraph`, which is a subclass of `Graph`:

```
class RoadAndCityGraph: Graph
   class Node::<
      cityName -> cn: ref String:
         cn := id
      display::<
          ...
   class Edge::<
      roadLength: var Integer
      display::
         ...
   addCity(nm: var String) -> n: ref Node:
      :::
   addRoad(from, to: ref Node, dist: var integer) -> e: ref Edge:
      :::
   inner(RoadAndCityGraph)
```

`RoadAndCityGraph` has the following attributes:

- A further binding of the virtual classes extended the descriptions of `Node` and `Edge` from `Graph`. A `Node` represents a city and an `Edge` represents a road.
- A method `addCity` and a method `addRoad`.

The further binding of `Node` adds the following attributes:

- A method `cityName` which returns the `id` of the node – the `id` is used to represent the name of the city.
- A further binding of `display`, we which extends the description of display from `Node`.
- Note that no further binding of `addEdge` is included here.

The further binding of `Edge` adds the following attributes:

- A integer variable `roadLength` holding the length of the road.
- A further binding of the `display` method.

Below we show the details of `addCity` and `addNode`:

```
   addCity(nm: var String) -> n: ref Node:
      n := addNode(nm)
   addRoad(from, to: ref Node, dist: var integer) -> e: ref Edge:
```

```
        e := from.addEdge(to)
        e.roadLength := dist
```

- `AddCity` has the name, `nm`, of the city as parameter and adds a `Node` with `nm` as argument. The new `Node` is returned as the value of `addNode`.
- `AddRoad` has the two cities being connected by the road and the distance between them as parameters: `from`, `to`, and `dist`.

We may now use class `RoadAndCityGraph` to define a region with some cities and roads. Herre we use Europe and Paris, London and Berlin:

```
Europe: obj RoadAndCityGraph
   setUp:
      n1, n2: ref Node
      e: ref Edge
      n1 := addCity("Paris")
      n2 := addCity("London")
      e := addRoad(n1,n2,291) -- 291 miles
      e := addRoad(n2,n1,291)
      n2 := addCity("Berlin")
      e := addRoad(n1,n2,1054) -- 1054 km
Europe.setUp
```

The example should be self explanatory.

Note, however, as a remainder of the importance of being aware of units of quantities, we have deliberately shown the distance from Paris to London in miles and the distance from Paris to Berlin in kilometers. When you use Google Maps (at least at the location of the authors) you get the distance from Paris to London in miles and the one from Paris to Berlin in kilometers. In section , it is show to be explicit about the units represented by numbers.

Exerises

1. *Shortest path.* Given two cities A and B, write a method that computes the shorts route from A to B.
2. *Traveling salesman*. For a given graph, write a method that computes the shortest path that visits/included all nodes in the graph.

(2) is called the travelling salesman since it is inspired by a salesman that needs to visit all cities in a given region and use the shortest route between them. It is due to the famous computer scientist E. Dijkstra.

These methods are not straight forward to write, and are examples of complex algorithms that may require the reader to study the topic of algorithms and data structures as mentioned in the introduction.

For small graphs, simple algorithms may be usable, but for large graph, it is necessary to find a algorithms that are efficient with respect to the time it takes to compute them.