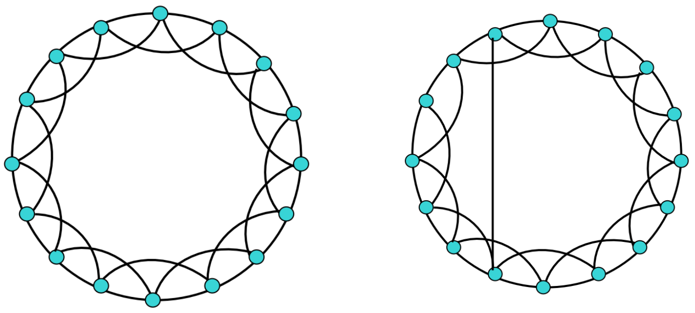
Assignment 2

Small World Graphs

A *graph* is a set of nodes and edges between the nodes. A *path* between two nodes is an sequence of connected edges in the graph. The *length* of a path is the number of edges in the path.

The **distance** between two nodes is the **length of the shortest path between the nodes**.

The **diameter** of a graph is the **maximum distance between any two nodes**.



The graph on the left above has 16 nodes arranged in a circle and each node is connected to it neighbours on the circle and to the nodes one place away. There is are many paths from any node back to itself.

The shortest path between any two nodes is 4 or less steps and the longest non repeating path has 15 steps. Thus the distance between any two nodes is 4 or less and the diameter of the graph is 4. If you consider two nodes that are on a diameter of the circle (hence the name diameter) then you can see that the shortest path between them is 4. Clearly any other pair of nodes must be closer than the pair on the diameter.

The right hand graph above is the same as the graph on the left except that an additional edge has been added. This additional edge can only reduce some of the path lengths, it cannot increase any path length and cannot decrease all paths.

As more edges are added ultimately the all shortest paths will be decreased and hence the diameter will be decreased.

A graph is **small(x)** if **the diameter is less that x times the number of nodes**.

$$small(g : graph, x : int) is(diameter(g) < x * nodes(g)$$

Write a Library for Small World Graphs

You must write an Ada library that defines a Graph Type, and contains functions for:

1. **distance** between tow nodes,
2. **diameter** of a graph and
3. **small(x)** a predicate over graphs, with parameter x of type fixed, that returns true if and only if the graph is small(x).

You need to submit:

Working Generic code this will be marked:

1. **30**% executable functions
2. **10**% Package structure + use of generics
3. **10**% Flow and Depends
4. **20**% Contracts (Pre,Post, ..) + proof
5. **10**% Testing (Proven code counts as tested)
6. **5**% Use of libraries (The pages [1] are, in my opinion, the best source fo what you need to know about using formal libraries)

Report covering:

1. **5**% brief documentation of your library
2. **5**% justification for the quality of your code
3. **5**% know weakness of you code

Suggested *.gpr set-up

You can edit the `*.gpr` file for your project and below is an example for the `MyLib` project that might help.

```
 project MyLib is
    for Source_Dirs use ("src/**", "src");
    for Main use ("main.adb");
    package Prove is
          for Switches use ("--report=all","--level=3");
    end Prove;
    package Compiler is
       for Default_Switches ("ada") use ("-fprofile-arcs",
                                         "-g" , "-ftest-coverage");
    end Compiler;
    package Linker is
         or Default_Switches ("ada") use ("-fprofile-arcs");
 end linker;
end MyLib;
```

The `"--level=3"` or `"level=2"` setting is needed to allow the tool to prove anything involving multiplication. The `"--report=all"` makes forces the tool to show what it has successfully proven.

Using `"-fprofile-arcs", "-g" , "-ftest-coverage"` should allow you to run tests and print code coverage.

Helpful link to Example Code
http://www.ada-auth.org/standards/12rm/html/RM-A-18-32.html

I found the this link after I set the question and thought it might help a little. It is not SPARK code but give some help as to how to use generic packages and give an algorithm for the shortest path.

References

[1] Formal SPARK libraries. `http://docs.adacore.com/spark2014-docs/html/ug/en/source/spark_libraries.html`.