Problem Set 7
Course **Security Engineering**
(Winter Term 2018)

Bauhaus-Universität Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Nathalie Dittrich
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/`

Due Date: 01 Feb 2019, 1:30 PM, via email to
`nathalie.jolanthe.dittrich(at)uni-weimar.de`.

Task 1 – Mini Project – Graph Algorithms (4 Credits)

Recall the graph implementation from Problem Set 2. Inform yourselves about the minimal-spanning-tree algorithm by Kruskal and the shortest-path algorithm by Dijkstra. Then implement the following package and write a sufficient test suite for your implementation with either **`AUnit`** or **`testgen`**.

```
with Ada.Unchecked_Deallocation;
with Graph;

generic
    type Vertex_Type is private;
    with function "="(Left: Vertex_Type; Right: Vertex_Type) return Boolean;
    with package Graph_Instance is new Graph(Vertex_Type, "=");
package Graph_Algorithms is
    use Graph_Instance;

    type Vertex_Array_Access is access all Vertex_Array;

    procedure Free is new Ada.Unchecked_Deallocation(
        Vertex_Array, Vertex_Array_Access);

    -- Implements Dijkstra's shortest-path algorithm in the given graph with
    -- edge weights. If a path exists, Path contains of the ordered sequence
    -- of vertices from From to To, excluding From and To.
    -- If no such path exists, Path will be empty.
    procedure Find_Shortest_Path(From: Vertex_Type;
                                 To: Vertex_Type;
                                 Path: out Vertex_Array_Access);
    -- Implements Kruskal's minimal-spanning-tree algorithm in the given graph
    -- with edge weights. If the graph is connected, Result will hold the
    -- minimal spanning tree; otherwise, Result will hold the minimal spanning
    -- forest.
    procedure Find_Min_Spanning_Tree(Result: out Vertex_Array_Access);
end Graph_Algorithms;
```

Task 2 – Mini Project – Parallel-Hofstadter Q Sequence (4 Credits)

Write a program that computes the Hofstadter Q sequence. The program takes two command-line arguments. The first parameter is the length of the Hofstadter Q sequence and the second a timeout that determines the maximum lifetime of the program. Your implementation should use **at least four tasks** to compute the Hofstadter Q sequence in parallel. Furthermore, pressing 'q' should stop the program immediately.

Example: The output of `#./<program_name> 4 2` is `1, 1, 2, 3` or a real subset of `1, 1, 2, 3` if the timeout is triggered before the sequence was finally computed.
Write a sufficient test suite for your implementation with either **`AUnit`** or **`testgen`**.