Index of content

1. Introduction

In the bonus system of the lecture Communication Networks 1, our group chooses the task to visualize Prim's algorithm.

Prim's algorithm finds the minimum spanning tree of a connected weighted undirected graph. This can for example be used in communication networks to determine the shortest path that packets or traffic will take between interconnected routers. Each graph consists of nodes and weighted undirected edges.

2. The algorithm – basic

The algorithm starts with a random start node that builds the trivial result graph R. In each iteration we search for the minimal weighted edge, which connects a new node with R. This node and the corresponding edge are added to R. This procedure will be repeated until all nodes are added into R.
The result graph R will be a minimum spanning tree.

1. A random start node will be chosen, as result graph R
2. As long as R not contains all nodes or all interconnected nodes
    a. Choose the edge with the minimum weight, which is connected to a node that is not in R
    b. Add the node and edge to R

3. Realization

For the implementation of a visual representation from Prim's algorithm, we chose Java as developing platform. The realization supports user defines graphs, for the visualization. User defines graphs have to be provided as XML file.

Graph Input file format:

```
<?xml version="1.0" encoding="UTF-8"?>
<graph>
    <nodes>
        <node>A</node>
        <node>B</node>
        <node>C</node>
        .
        .
    </nodes>
    <edges>
        <edge starts="A" ends="B" cost="7"/>
        <edge starts="A" ends="D" cost="6"/>
        <edge starts="B" ends="C" cost="8"/>
        <edge starts="B" ends="D" cost="4"/>
        <edge starts="B" ends="E" cost="3"/>
        <edge starts="C" ends="E" cost="5"/>
        .
        .
      </edges>
</graph>
```

The example above shows the structure of an XML input file, which represents a graph. All nodes have to be introduced in the <node> section. Edges that connect nodes, are defined after the node section in the <edge> section. Each edge needs a start and end node and the weight of the edge. Each input in the <edge> section has to be surrounded by quotation marks (e.g. "A"). In difference to the node declaration, the start and end nodes and the weight are declared as attributes of an edge tag.

The implementation is subdivided into two packages. The package algorithm includes the class Edge.java which is responsible for the edges of the graph. The class Prim.java implements the prim algorithm.

Edge.java

- Defines the getter and setters for start and end nodes of an edge and the cost of the edge (weight of the edge)
- Comparator to compare two edges by their costs
  1. Returns -1 if the given edge has higher costs
  2. Returns 1 if the given edge has lower costs

Prim.java

- Implementation of the prim algorithm
- Implementation of the GUI
- User interaction buttons

a.	Load Graph	->	load user defined graph as XML file from file system
b.	Random Start Node	->	selects a random node as start node from a loaded graph
c.	Update Layout	->	rebuild / rearrange the visual representation of the loaded graph
d.	Next	->	preforms one iteration step of the algorithm and highlights the result
e.	Previous	->	revers one iteration step

The package visualization includes the class GraphLayout.java which is responsible for the visual representation of a graph in the GUI.

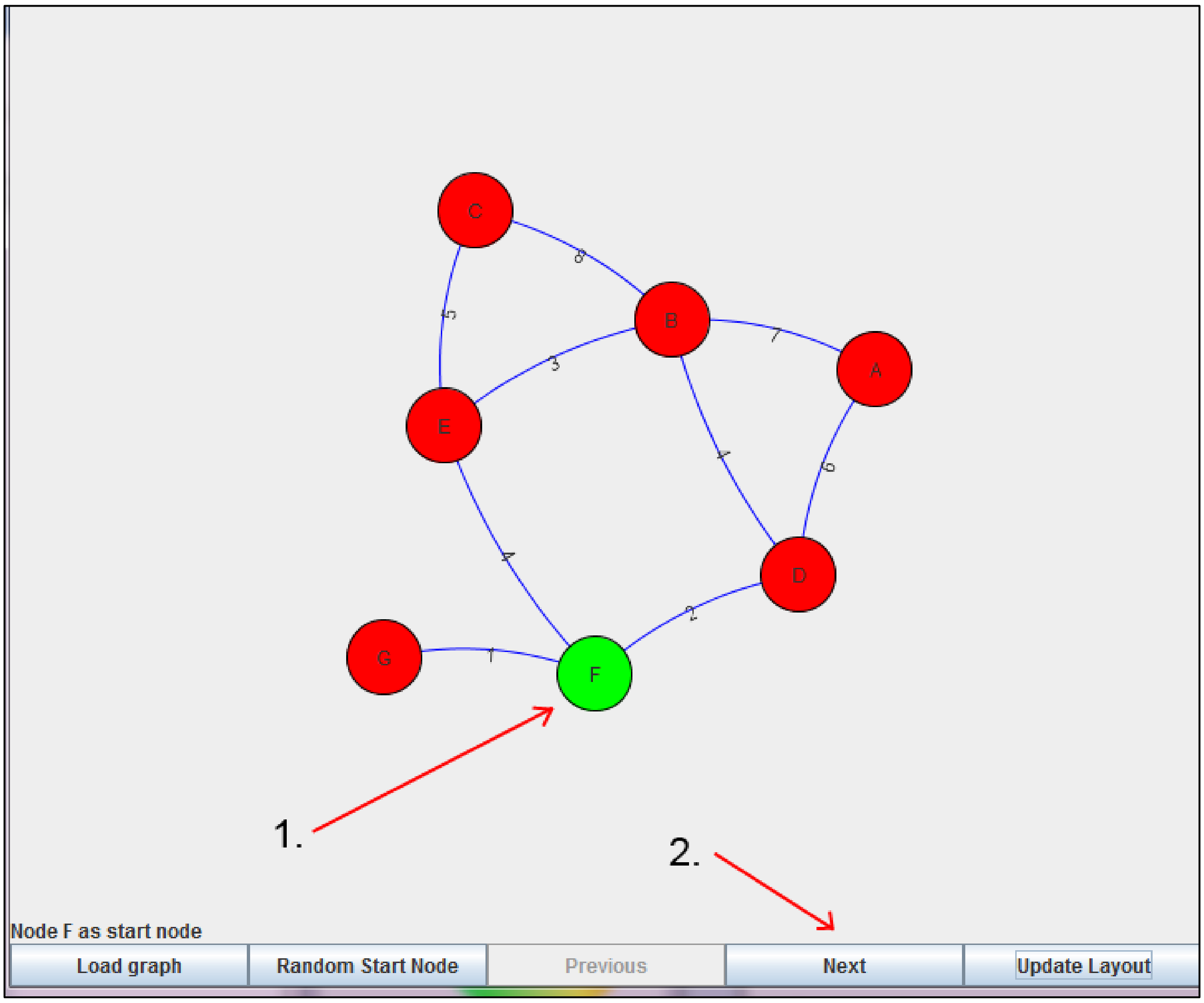
GraphLayout.java

- Visualization and rendering of the graph
  1. Nodes
     a. Start node and already processed nodes are green
     b. Not processed nodes are marked red
  2. Edges
     a. Unused edges are blue
     b. Used edges are green

For the visualization of a graph in JAVA, the Java Universal Network/Graph (in short JUNG) Framework (http://jung.sourceforge.net) provides existential features for modeling, analysis and visualization of data that can be represented as graph.
The JUNG framework offers variety of representations of entities and their relations, which allows an optimal representation of an undirected graph and a minimum spanning tree. In addition JUNG has the capabilities to visualize interactions which make an interactive exploration of an algorithm possible. The above described capabilities and the fact that the JUNG framework is an open source library made the decision easy to use this framework for the given task.

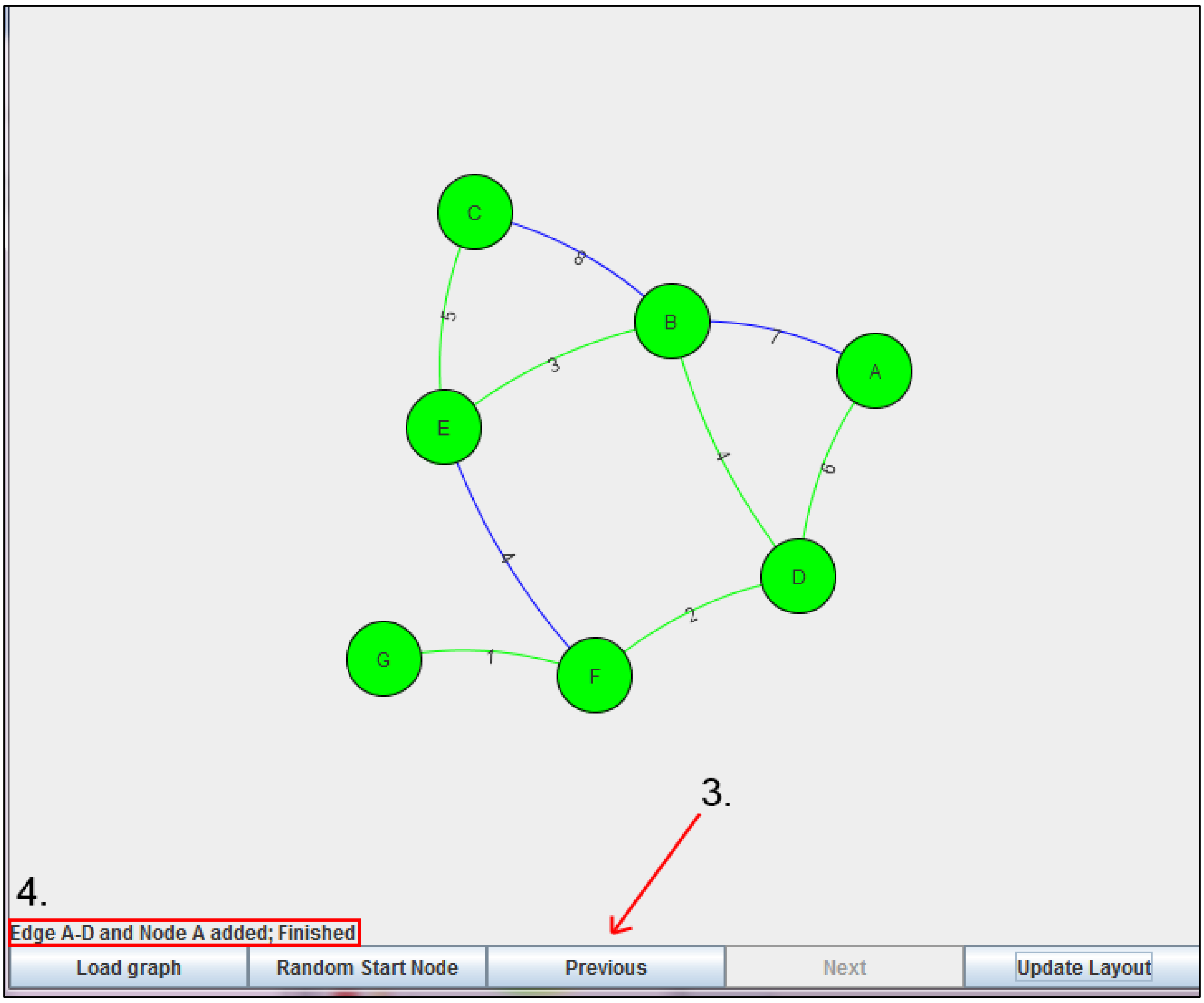
The result of the prim algorithm is a minimum spanning tree colored in green. In the lower left of the GUI, status messages are provided. Those will inform the user about the performed action in each iteration and if the algorithm has finished.

4. Example



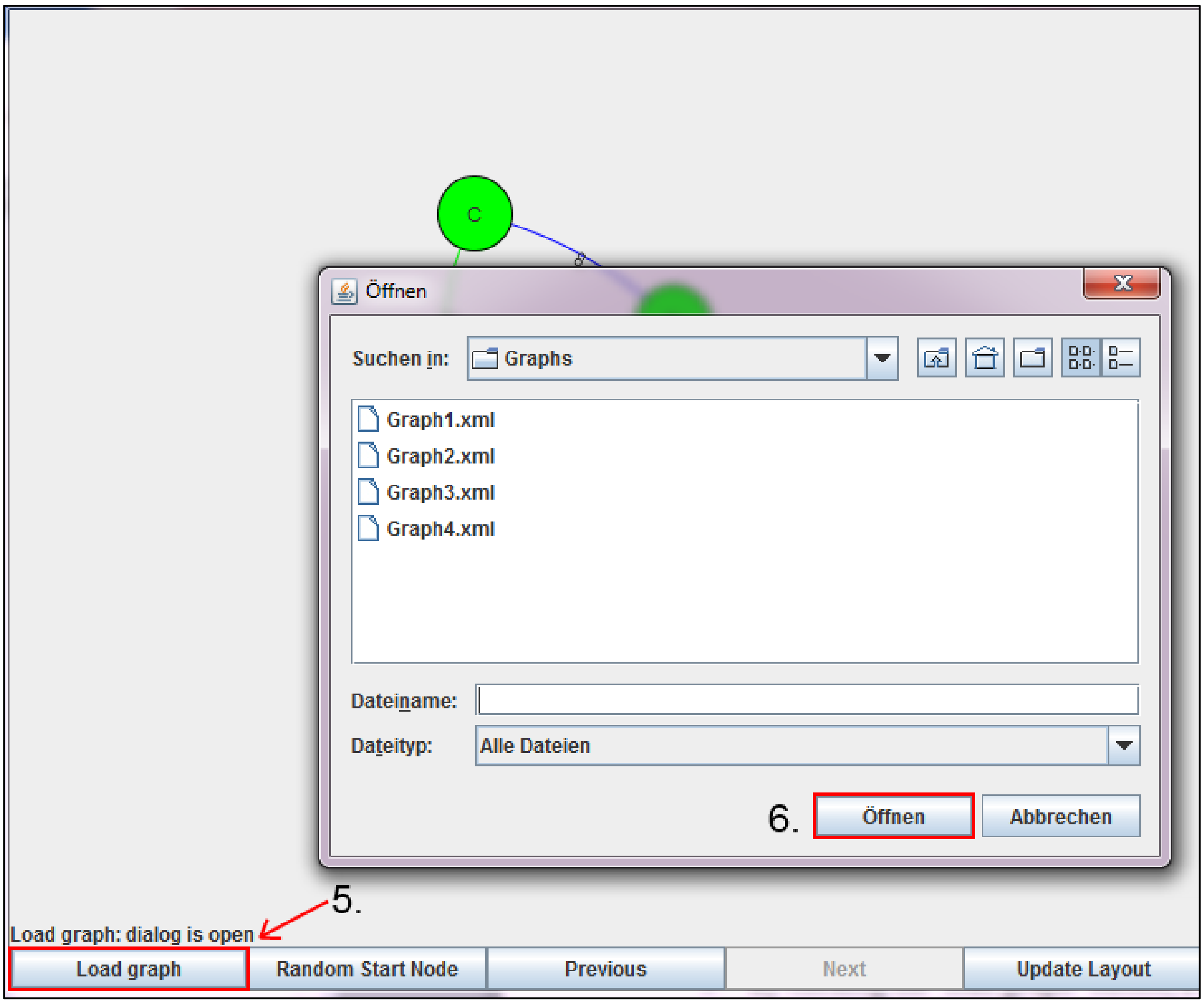
Node F as start node

1. Start node has been selected and algorithm is ready to start.
   - Node D is selected as start node
2. By clicking the "Next" button one iteration step is performed.
   - After each clicking on the "Next" the GUI and visual representation of the graph is updated.



Edge A-D and Node A added; Finished

3. By clicking the Previous button, one iteration step can be reversed.
   - This allows the user an interactive exploration of the algorithm
4. The lower left of the GUI offers an status message field
   - The status message provides the user information about preformed action in each iteration
   - And indicates when the algorithm is finished



5. By clicking the "load graph" button, a new window pops up
   - where user defined graphs can be imported / loaded

6. User defined graphs have to been marked and can be opened by clicking the "Öffnen/ Open" button
   - User defined graphs have to be provided as XML files

5. References

http://jung.sourceforge.net/ , last visited on May 25th, 2014
http://de.wikipedia.org/wiki/Algorithmus_von_Prim , last visited on May 25th, 2014
http://en.wikipedia.org/wiki/Prim%27s_algorithm , last visited on May 25th, 2014