ANNALS OF FACULTY ENGINEERING HUNEDOARA
– INTERNATIONAL JOURNAL OF ENGINEERING
Tome X (Year 2012) – FASCICULE 3 (ISSN 1584 – 2673)

1. Anca Elena IORDAN

DEVELOPMENT OF AN INTERACTIVE ENVIRONMENT USED FOR SIMULATION OF SHORTEST PATHS ALGORITHMS

1. UNIVERSITY "POLITEHNICA" OF TIMISOARA, FACULTY OF ENGINEERING HUNEDOARA, ROMANIA

ABSTRACT: *In this paper the author present the necessary stages in object orientated development of an interactive environment that is dedicated to the process of acquaintances assimilation in graphs theory domain, especially for simulation of the shortest paths algorithms. The modelling of the environment is achieved through specific UML diagrams representing the stages of analysis, design and implementation. This interactive environment is very useful for both students and professors, because computer programming domain, especially graphs theory domain is comprehended and assimilated with difficulty by students.*
KEYWORDS: *Graph, Dijkstra Algorithm, Bellman-Ford Algorithm, Roy-Floyd Algorithm, UML, Java.*

INTRODUCTION

Graphs theory represents the mathematic study of abstract relationship structure between the objects. Although graphs are themselves purely theoretical, their ability to model pair-wise relationships in systems of arbitrary complexity yields abundant direct correspondence with numerous important physical systems in the real world [1]. More then this, the graph structure permits a geometrical view of them in many ways. If they are taken together, these two properties suggest that graph theory teaching and also researching in this domain can obtain benefits using an interactive environment addressed graphs study.

Starting with this necessity, we designed an interactive soft instrument that can be used by students and also by graphs theory beginners, but strong enough to help the researchers in graphs theory domain.

In educational scope, the interface and the visual components will lead the students and the beginners in this domain to a better comprehending of the graphs theory basic concepts. The professors can use this soft as an instrument for graphical simulation of the graphs specify algorithms. The options to create, modify and view different types of graphs will help the students to comprehend the advanced specify concepts of this domain.

SHORTEST PATHS ALGORITHMS – *Dijkstra algorithm*

Dijkstra algorithm [2], conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms.

For a given source vertex in the graph, the algorithm finds the path with lowest cost between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra algorithm can be used to find the shortest route between one city and all other cities. Dijkstra original algorithm does not use a min-priority queue and runs in $O(|V|^2)$.

Bellman-Ford algorithm

The Bellman–Ford algorithm [3] computes single-source shortest paths in a weighted graph. For graphs with only non-negative edge weights, the faster Dijkstra algorithm also solves the problem. Thus, Bellman–Ford is used primarily for graphs with negative edge weights. The algorithm is named after its developers, Richard Bellman and Lester Ford, Jr.

If a graph contains a "negative cycle", i.e., a cycle whose edges sum to a negative value, then walks of arbitrarily low weight can be constructed, there may be no shortest path. Bellman-Ford can detect negative cycles and report their existence, but it cannot produce a correct answer if a negative cycle is not reachable from the source.

Bellman–Ford is in its basic structure very similar to Dijkstra algorithm, but instead of greedily selecting the minimum-weight node not yet processed to relax, it simply relaxes all the edges, and does this $|V| - 1$ times, where $|V|$ is the number of vertices in the graph. The repetitions allow minimum distances to accurately propagate throughout the graph, since, in the absence of negative cycles, the *shortest path can only visit each node at most once. Unlike the greedy approach, which depends on certain structural assumptions derived from positive weights, this straightforward approach extends to the general case. Bellman–Ford runs in O(|V|·|E|) time, where |V| and |E| are the number of vertices and edges respectively.*

Roy-Floyd algorithm

The Roy–Floyd algorithm [4] (also known as Floyd algorithm, Roy–Warshall algorithm, Floyd–Warshall algorithm, or the WFI algorithm) is a graph analysis algorithm for finding shortest paths in a weighted graph (with positive or negative edge weights) and also for finding transitive closure of a relation R. A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves. The algorithm is an example of dynamic programming. It was published in its currently recognized form by Robert Floyd in 1962. However, it is essentially the same as algorithms previously published by Bernard Roy in 1959 and also by Stephen Warshall in 1962 for finding the transitive closure of a graph.

The Roy–Floyd algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with only $O(|V|^3)$ comparisons in a graph. This is remarkable considering that there may be up to $\Omega(|V|^2)$ edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal. The complexity of the algorithm is $O(n^3)$.

STEPS FOR OF THE INTERACTIVE ENVIRONMENT DESIGN – ANALYSIS STEP

Elaboration of the interactive environment was made in concordance with a specification that shows all desired elements. These elements contain software requirements, making difference between obligatory and optional ones, and also contain the functionality and conditions that have to be accomplished in order to conform to standards. For object orientated development of the application, unified modeling language will be used. To achieve UML diagrams were used the ArgoUML [5] software.

The analysis of an informatics system consists in drawing the use case and activity diagrams [6]. The informatics system will be described in a clear and concise manner by representation of the use-cases. Each case describes the interaction between the user and the system. The diagram defines the system's domain, allowing visualization of the size and scope of the whole developing process. This diagram includes:

- *One actor - the user who is external entity with which the didactic game interacts.*
- *Five use cases that describe the functionality of the interactive game.*
- *Relationships between user and use cases (association relationships), and relationships between use cases (dependency and generalization relationships).*

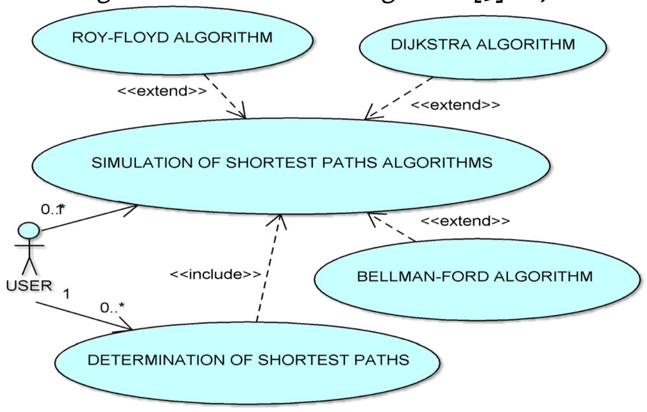

Figure 1 . Use Cases Diagram

DESIGN STEP

Conceptual modelling allows identifying the most important concepts for the interactive environment [7]. Inheritance was not used only as a generalization device, which is when derived classes are specializations of the base class.

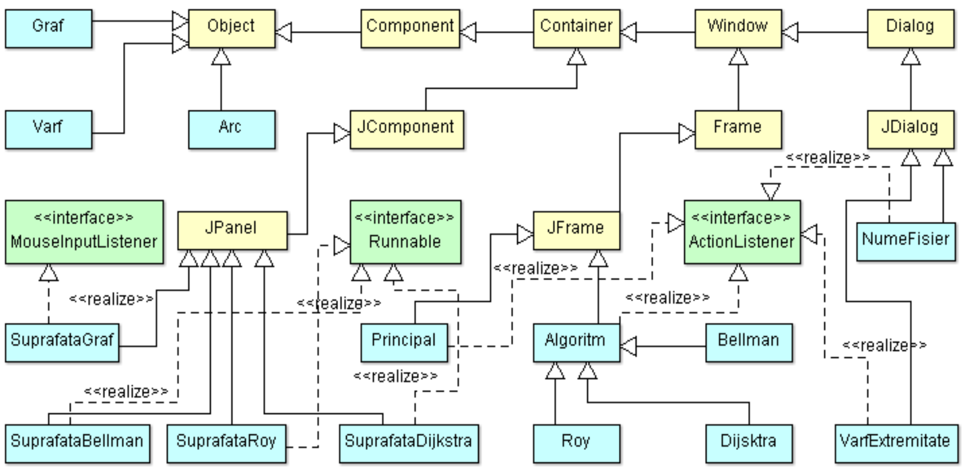

Figure 2. Class Diagram

In figure 2 are presented as inheritance relationships, as realization relationships. We can observe that the Principal and Algoritm classes inherit attributes and methods of the JFrame class, but implements the ActionListener interface.

NumeFisier class inherits attributes and methods of the JDialog class, but implements the interface ActionListener. SuprafataRoy class inherit attributes and methods of the JPanel class, but implements Runnable interface, and SuprafataGraf class inherits attributes and methods of the JPanel class and implements the MouseInputListener interface.

Between instances of the classes presented in figure 2 there are especially composition and aggregation relationships. In the composition relationship, unlike the aggregation relationship, the instance can not exist without the party objects. Analyzing figure 3 we can observe that an instance of SuprafataRoy type consists in two objects of Graf type, one of Graphics2D type and the other of Thread type.

Aggregation relationship is an association where it's specified who is integer and who is a part. For example, an object of Arc type or represent a part from an object of Graf type.

Sequence diagram [8] emphasizes the temporal aspect, being suitable for real-time specifications and complex scenarios. These diagrams determine the objects and classes involved in a scenario and sequence of messages sent between objects necessary to execute script functionality.

Diagram presented in figure 4 shows the interactions between objects that are aimed at determining optimal path using Dijkstra algorithm. We can observe that there are interactions between 17 objects of which the object of Pincipal type is already created, and objects of Dijkstra, JButton, SuprafataGraf, SuprafataDijkstra, Graphics2D, Graf, Varf, Arc and Thread type will instantiate during interactions.

At the beginning the execution control is taken by Principal object that create an instance to Dijkstra class, this object taking the control of the interactions. There will be instancing four JButton objects, a SuprafataGraf object and a SuprafataDijkstra object. Instancing the SuprafataGraf object has as effect creating a Graphics2D object and a Graf object. Instancing the SuprafataDijkstra object permits to create another Graphics2D object.

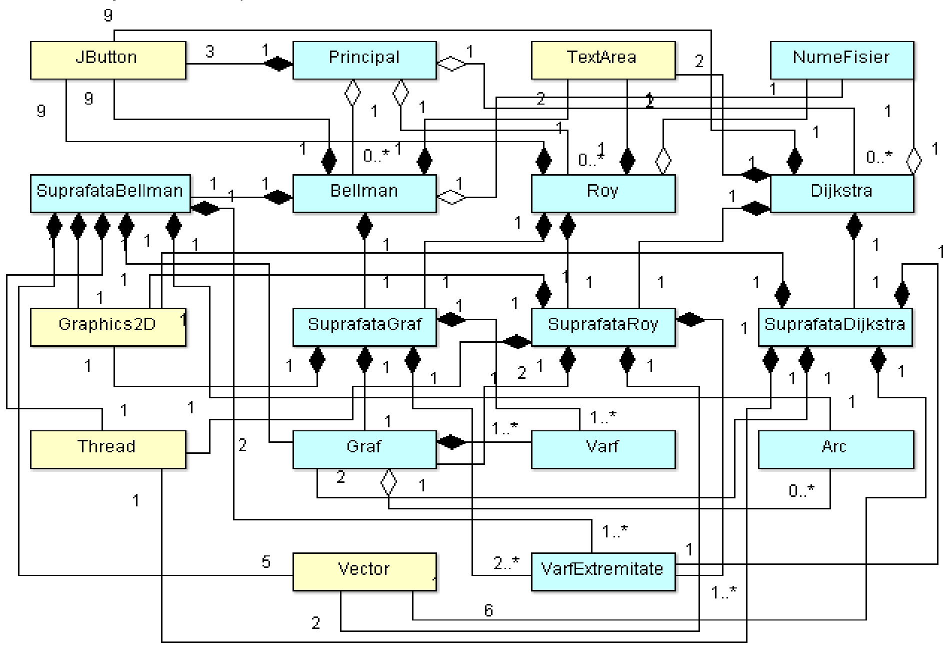

Figure 3. Class Diagram

After an event that interactions with nod of JButton object, there is transmitting the control of SuprafataGraf object that creates an instance of the class Varf. The control is transmitted to Graf object in order to add the object that was previously created. Control will be taken by SuprafataGraf object that will destroy Varf object, and by interaction with Graphics2D object the graph will be redrawn. There can be observed that Varf object life line is broken by X marking when appears the message stereotype mark <<destroy>>.

After an event that interactions with JButton object, there is transmitting the control of SuprafataGraf object that creates two instances of class Varf, then an instance of class Arc. The control is transmitted to Graf object in order to add the Arc object that was previously created. Control will be taken by SuprafataGraf object that will destroy Varf and Arc objects and by interaction with Graphics2D object the graph will be redrawn.

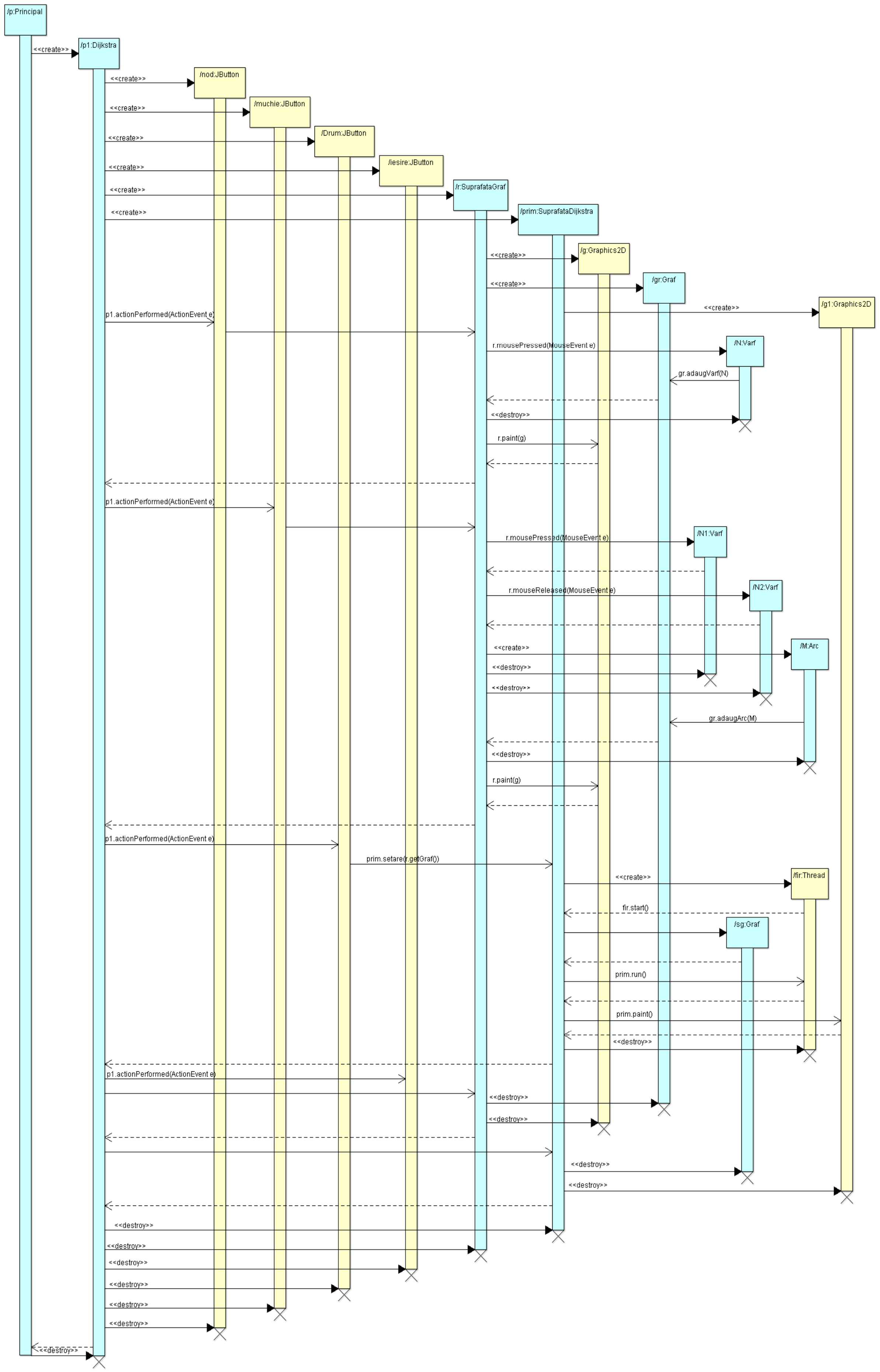

Figure 4. Sequence Diagram

Appealing run() method, the execution thread will pass in running state and by interaction between Graf object, used for memorising the optimal path, with Graphics2D object there will be represented step by step the optimal path. Finally there will be destroyed all the created objects and control will be taken by Principal object.

IMPLEMENTATION STEP

Component diagram [9] is similar to packages diagram, allowing visualization of how the system is divided and the dependencies between modules. Component diagram put emphasis on software physical elements and not on the logical elements like in case of package diagram.

The diagram in figure 5 describes the collection of components that together provide system functionality. Central component of the diagram is Principal.class, a component obtained by transforming by the Java compiler into executable code of the Principal.java component. As can be seen that component interacts directly with component Fereastra.class.

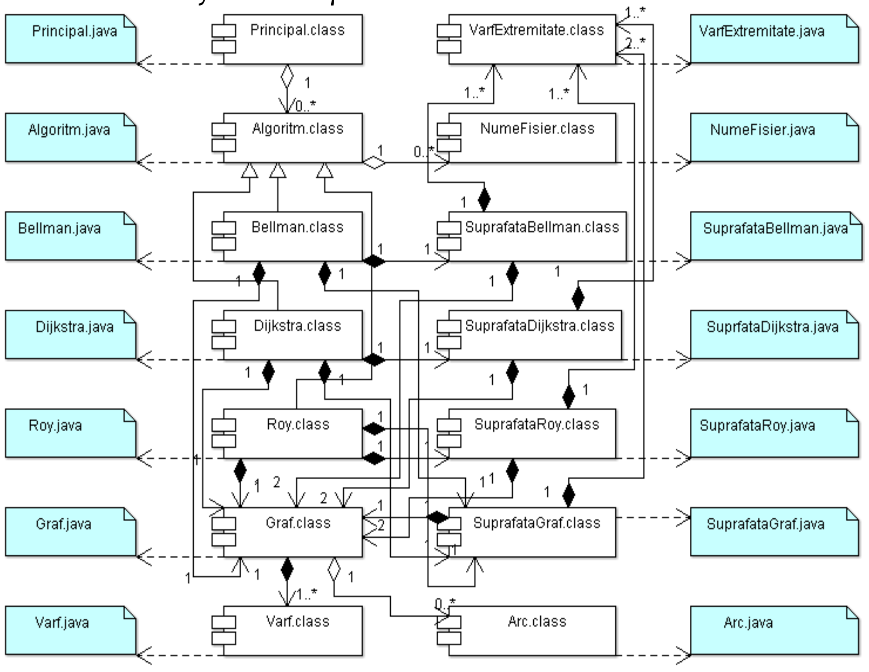

Figure 5. Components diagram

USER INTERFACE

The informatics system is accomplished using the Java programming language [10]. The application can easily convert in a Java applet.

Starting from specified requisites in uses cases diagram (figure 1) it was designed graphical user interface of the informatics system. The main page of the application contains buttons for selecting the following options: Dijkstra algorithm simulation (figure 6), Bellman-Ford algorithm simulation and Roy-Floyd algorithm simulation.

CONCLUSIONS

The main goal in realizing of this paper is designing and implementation of a dynamic interactive environment that has to lead the user to accomplish experience in comprehending and controlling acquaintances in graphs theory domain and to offer efficient and convenient access to the latest information.

The informatics application follows these actual problems and boards it in a new manner using modern educational technologies. Through the diagram representation all three

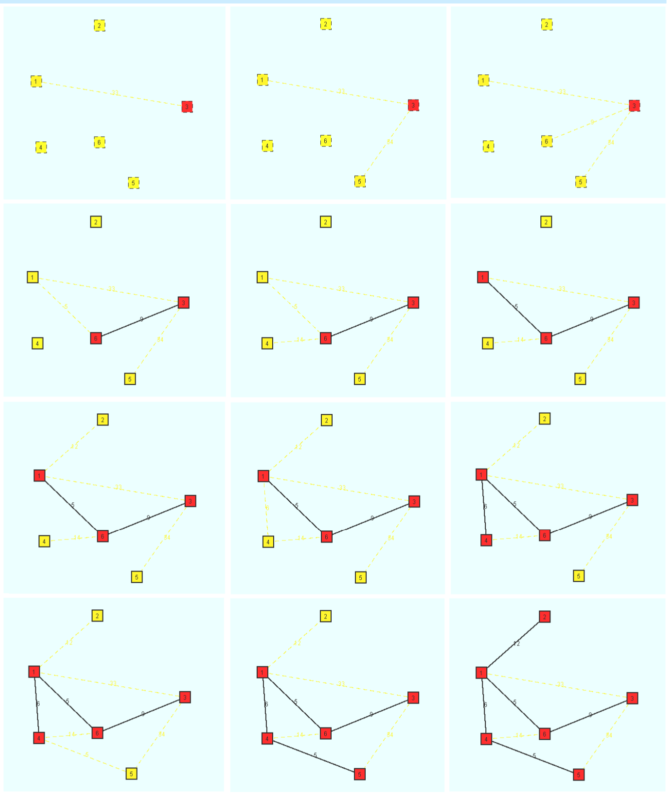

Figure 6. Simulation of Dijkstra algorithm

phases: analysis, design and implementation, the educational informatics system has been described in a clear and concise manner. The use of the UML modelling language for the creation of the diagrams is characterized by rigorous syntactic, rich semantic and visual modelling support.

REFERENCES

[1.] D. Bauer, H. Broersma, E. Schmeichel, *Toughness in graphs — a survey, Graphs and Combinatorics, vol.* **22**, *pp. 1–35, 2006*

[2.] M. Golumbic, *Algorithmic Graph Theory and Its Applications, Computer Science Interfaces Series, Springer, pp. 41-62, 2005*

[3.] J. Gross, J. Yellen, *Graph Theory and Its Applications, Taylor & Francis Group, 2005*

[4.] R. Diestel, *Graph Theory, Springer-Verlag, Heidelberg, 2010*

[5.] *http://argouml.tigris.org*

[6.] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide, Addison Wesley, 1999*

[7.] M. Fowler, K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison Wesley, Readings MA, USA, 2000*

[8.] J. Odell, *Advanced Object Oriented Analysis& Design using UML, Cambrige University Press, 1998*

[9.] S. Bennet, S. McRobb, R. Farmer, *Object Oriented Systems Analysis and Design, McGraw Hill, 1999*

[10.] B. Eckel, *Thinking in Java, Prentice Hall, 2003*