Aufgabenblatt 7
Dijkstra Algorithmus

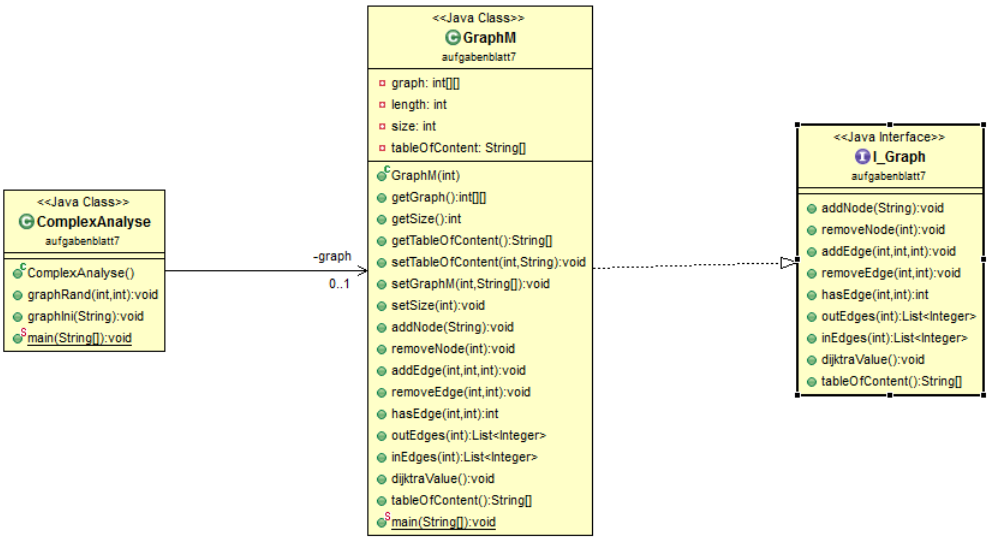
Cao, Thi Huyen(Lilli)-Kochinky,Dennis

November 8, 2016

Contents

1 UML

Ein Graph wird als Adjazenzmatrix implementiert.



2 API

Anbei sind Javadoc in Pdf Dateien von Interface *I_Graph* und Klasse GraphM.

aufgabenblatt7

Interface I_Graph

```
public interface I_Graph
```

This interface describes a graph

Author:

cao

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
void	**addEdge**(int i, int j, int value) This method adds a new edge from node i to node j
void	**addNode**(java.lang.String name) This method adds a new node in the graph.
void	**dijktraValue**() This method counts the shortest way after the dijktra algorithms.
int	**hasEdge**(int i, int j) This method checks out if a edge from node i to node j is available or not.
java.util.List<java.lang.Integer>	**inEdges**(int n) This method return a list of all nodes going in node n
java.util.List<java.lang.Integer>	**outEdges**(int n) This method returns a list of all nodes out from node n
void	**removeEdge**(int i, int j) This method removes the edge from node i to node j
void	**removeNode**(int n) This method removes the node with index n
java.lang.String[]	**tableOfContent**() This method return a list of Content of each nodes according to its index

Method Detail

addNode

```
void addNode(java.lang.String name)
```

This method adds a new node in the graph.

Parameters:

```
name: - name of node
```

removeNode

```
void removeNode(int n)
```

This method removes the node with index n

Parameters:

```
n: - index of node
```

addEdge

```
void addEdge(int i,
             int j,
             int value)
```

This method adds a new edge from node i to node j

Parameters:

```
i: - start node
```

```
j: - destination node
```

```
value: - distance from start node to destination node(weight of edge)
```

removeEdge

```
void removeEdge(int i,
                int j)
```

This method removes the edge from node i to node j

Parameters:

```
i: - start node
```

```
j: - destination node
```

hasEdge

```
int hasEdge(int i,
            int j)
```

This method checks out if a edge from node i to node j is available or not. If yes, return the value of the edge

Parameters:

`i: - start node`

`j: - destination node`

Returns:

`value of the edge`

outEdges

```
java.util.List<java.lang.Integer> outEdges(int n)
```

This method returns a list of all nodes out from node n

Parameters:

`n: - index of node`

Returns:

`a list of nodes`

inEdges

```
java.util.List<java.lang.Integer> inEdges(int n)
```

This method return a list of all nodes going in node n

Parameters:

`n: - index of node`

Returns:

`a list of nodes`

dijktraValue

```
void dijktraValue()
```

This method counts the shortest way after the dijktra algorithms.

tableOfContent

```
java.lang.String[] tableOfContent()
```

This method return a list of Content of each nodes according to its index

Returns:

an String array of index and content

aufgabenblatt7

Class GraphM

java.lang.Object
    aufgabenblatt7.GraphM

All Implemented Interfaces:

`aufgabenblatt7.I_Graph`

```
public class GraphM
extends java.lang.Object
implements aufgabenblatt7.I_Graph
```

This class describes a graph in a nxn matrix

Author:

`cao`

Constructor Summary

Constructors

Constructor and Description
`GraphM(int length)`

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
`void`	`addEdge(int i, int j, int value)` This method adds a new edge from node i to node j
`void`	`addNode(java.lang.String name)` This method adds a new node in the graph.
`void`	`dijktraValue()` This method counts the shortest way after the dijktra algorithms.
`int[][]`	`getGraph()`
`int`	`getSize()`
`java.lang.String[]`	`getTableOfContent()`
`int`	`hasEdge(int i, int j)`

	This method checks out if a edge from node i to node j is available or not.
java.util.List<java.lang.Integer>	**inEdges**(int n) This method return a list of all nodes going in node n
static void	**main**(java.lang.String[] args)
java.util.List<java.lang.Integer>	**outEdges**(int n) This method returns a list of all nodes out from node n
void	**removeEdge**(int i, int j) This method removes the edge from node i to node j
void	**removeNode**(int n) This method removes the node with index n
void	**setGraph**(int n)
void	**setGraphM**(int i, java.lang.String[] data)
void	**setSize**(int i)
void	**setTableOfContent**(int i, java.lang.String city)
java.lang.String[]	**tableOfContent**() This method return a list of Content of each nodes according to its index

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GraphM

```
public GraphM(int length)
```

Method Detail

getGraph

```
public int[][] getGraph()
```

getSize

```
public int getSize()
```

getTableOfContent

```
public java.lang.String[] getTableOfContent()
```

setTableOfContent

```
public void setTableOfContent(int i,
                              java.lang.String city)
```

setGraph

```
public void setGraph(int n)
```

setGraphM

```
public void setGraphM(int i,
                      java.lang.String[] data)
```

setSize

```
public void setSize(int i)
```

addNode

```
public void addNode(java.lang.String name)
```

Description copied from interface: `aufgabenblatt7.I_Graph`
This method adds a new node in the graph.

Specified by:
`addNode in interface aufgabenblatt7.I_Graph`

removeNode

```
public void removeNode(int n)
```

Description copied from interface: `aufgabenblatt7.I_Graph`
This method removes the node with index n

Specified by:

removeNode in interface aufgabenblatt7.I_Graph

addEdge

```
public void addEdge(int i,
                    int j,
                    int value)
```

Description copied from interface: aufgabenblatt7.I_Graph

This method adds a new edge from node i to node j

Specified by:

addEdge in interface aufgabenblatt7.I_Graph

removeEdge

```
public void removeEdge(int i,
                       int j)
```

Description copied from interface: aufgabenblatt7.I_Graph

This method removes the edge from node i to node j

Specified by:

removeEdge in interface aufgabenblatt7.I_Graph

hasEdge

```
public int hasEdge(int i,
                   int j)
```

Description copied from interface: aufgabenblatt7.I_Graph

This method checks out if a edge from node i to node j is available or not. If yes, return the value of the edge

Specified by:

hasEdge in interface aufgabenblatt7.I_Graph

Returns:

value of the edge

outEdges

```
public java.util.List<java.lang.Integer> outEdges(int n)
```

Description copied from interface: aufgabenblatt7.I_Graph

This method returns a list of all nodes out from node n

Specified by:

`outEdges in interface aufgabenblatt7.I_Graph`

Returns:

`a list of nodes`

inEdges

```
public java.util.List<java.lang.Integer> inEdges(int n)
```

Description copied from interface: `aufgabenblatt7.I_Graph`

This method return a list of all nodes going in node n

Specified by:

`inEdges in interface aufgabenblatt7.I_Graph`

Returns:

`a list of nodes`

dijktraValue

```
public void dijktraValue()
```

Description copied from interface: `aufgabenblatt7.I_Graph`

This method counts the shortest way after the dijktra algorithms.

Specified by:

`dijktraValue in interface aufgabenblatt7.I_Graph`

tableOfContent

```
public java.lang.String[] tableOfContent()
```

Description copied from interface: `aufgabenblatt7.I_Graph`

This method return a list of Content of each nodes according to its index

Specified by:

`tableOfContent in interface aufgabenblatt7.I_Graph`

Returns:

`an String array of index and content`

main

```
public static void main(java.lang.String[] args)
```

3 Aufwandsanalyse

Dijkstra Algorithms				
Cost Analyse				
Number of Nodes	Number of Edges	cost(ms)	Number of Edges	cost(ms)
10	45	1	25	1
100	580	4	300	2
200	1180	8	600	3
500	2985	16	1490	8
800	4770	36	2396	14
1000	6000	42	3000	18
2000	12000	70	6000	47
5000	30000	548	14990	228
8000	48000	1475	23997	616
10000	60000	2300	30000	900
100000	OutOfMemoryError			

$\Longrightarrow$ O(n)= n^2 mit n: Anzahl der Knoten

$\Longrightarrow$ Betrachtet man den Graph, ist auch leicht zu sehen, dass die Komplexität von Dijkstra Algorithmus sowohl von Anzahl der Knoten als auch von Anzahl der Kanten abhängig ist. Mit gleicher Anzahl von Knoten hat ein Graph mit mehreren Kanten grösseren Aufwand.Datenreihe 1 und 2 haben gleiche Anzahl von Knoten. Anzahl der Kanten von Datenreihe 2 = 1/2 Anzahl der Kanten von Datenreihe 1 (sieh Graph)

END