SWO 3x Übung zu Softwareentwicklung mit klassischen Sprachen u. Bibliotheken 3 **WS 2014/15, Übung 03**

Abgabetermin: Sa in der KW 43

☐ **Gr. 1,** DI Franz Gruber-Leitner **Name** _____ **Aufwand in h** _____

☐ **Gr. 2,** Dr. Erik Pitzer

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Repräsentation von *gewichteten* Graphen (7 + 9 Punkte)

Hinweis: Es werden nur Graphen mit einer fixen Anzahl von *n* Knoten betrachtet, d.h. die Anzahl der Knoten kann nicht wachsen oder schrumpfen und muss von Anfang an bekannt sein. Als "Knotennamen" können deshalb die ganzzahligen Werte von (z.B.) 1 bis *n* dienen.

a) Entwickeln Sie einen *abstrakten Datentyp* (in Form eines Moduls *DG_ADT_**M**.c*, wobei *DG* für *directed graph* und *M* für *matrix* steht) zur Verwaltung eines gewichteten Graphen mit *n* Knoten, wobei intern zur Repräsentation eine *Adjazenz**m**atrix* verwendet wird.

b) Entwickeln Sie einen *abstrakten Datentyp* (in Form eines Moduls *DG_ADT_**L**.c*, wobei *L* für *list* steht) zur Verwaltung eines gewichteten Graphen mit *n* Knoten, wobei zur Repräsentation intern eine *Adjazenz**l**iste* verwendet wird.

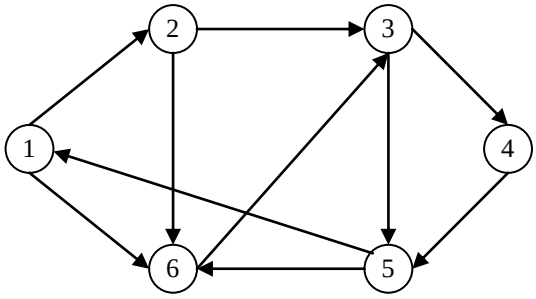
Die Mindestfunktionalität umfasst in beiden Fällen

1. das Erzeugen eines neuen Graphen mit dem Parameter *n*,
2. das Eintragen neuer Kanten zwischen Knoten mit Gewicht *w*,
3. das Löschen von Kanten zwischen Knoten,
4. das Ausgeben der internen Darstellung des Graphen in geeigneter Form und
5. das Freigeben eines Graphen.

Achten Sie darauf, dass beide Datenstrukturen "austauschbar" sind – indem beide Implementierungen die gleiche Schnittstelle *DG_ADT.h* erfüllen und durch binden („linken") die eine oder die andere Implementierung eingebunden werden kann.

Schreiben Sie *ein* Testprogramm *für beide* Datentypen, in dem Sie den Graphen aus der Abbildung rechts aufbauen und die interne Darstellung ausgeben. Sie können dabei beliebige positive Kantengewichte annehmen.

Testen Sie auch die Möglichkeit, eine Kante zu löschen.



So könnte die Schnittstelle in DG_ADT.h anfangs aussehen. Ein wichtiges Detail dabei ist, dass das „`struct graph`" nur deklariert, jedoch nicht definiert wird, was Sie dann in der konkreten Implementierung erst machen können.

```
struct Graph;
typedef struct Graph Graph;

Graph* createGraph(int n);
void freeGraph(Graph *g);
void insertEdge(Graph *g, int source, int target, double weight);
void removeEdge(Graph *g, int source, int target);
void printGraph(Graph *g);
```

2. Graphenalgorithmen (1 + 3 + 4 Punkte)

Erweitern Sie die Schnittstelle in DG_ADT.H sowie beide Implementierungen nach eigenem Ermessen um die Realisierung der nun folgenden weiteren Funktionalität **implementierungs-unabhängig** zu ermöglichen, also nur abhängig von der allgemeinen Schnittstelle DG_ADT.H. Halten Sie dazu diese Schnittstelle möglichst klein und allgemein.

Erstellen Sie zwei neue Dateien GRAPH_ALGS.H und GRAPH_ALGS.C in der Sie folgende Funktionen zur Verfügung stellen:

1. Erzeugung des invertierten Graphen (Alle Kanten werden umgedreht)
2. Testen auf Reflexivität, Symmetrie, Asymmetrie und Transitivität eines Graphen

Die Schnittstelle dieser Algorithmen könnte z.B. so aussehen:

```
#include "dg_adt.h"

Graph* invert(Graph *g);
bool isReflexive(Graph *g);
bool isSymmetric(Graph *g);
bool isAsymmetric(Graph *g);
bool isTransitive(Graph *g);
void printGraphProperties(Graph *g);
```

Hinweise:

1. Geben Sie für alle Ihre Lösungen immer eine Lösungsidee an.
2. Kommentieren und testen Sie Ihre Programme ausführlich.

1 Allgemein

Folgend ist die allgemeine Dokumentation dieser Übung angeführt.
Da diese Übung sich in drei verschiedene Teile aufteilt sollen diese unabhängig voneinander implementiert werden. Daher soll folgende Aufteilung erfolgen.

1.1 Projektstruktur

1. **Graph**
   Dieses Projekt enthält die Interfacespezifikation für das Handling eines Graphen, welche von allen Implementierungen verwendet werden soll. Ebenso soll hier die Testspezifikation enthalten werden, gegen die die Implementierungen getestet werden sollen.
2. **GraphMatrix**
   Dieses Projekt soll die Implementierung der Spezifikation mittels einer Adjazenzmatrix sowie alle implementierungsabhängigen Implementierungen für die Tests enthalten.
3. **GraphList**
   Dieses Projekt soll die Implementierung der Spezifikation mittels einer Adjazenzliste sowie alle implementierungsabhängigen Implementierungen für die Tests enthalten.
4. **GraphAlg**
   Dieses Projekt soll die Implementierung der Algorithmen, die auf einen Graphen angewendet werden können enthalten, welche aufgabengemäß so implementiert werden soll, dass diese implementierungsunabhängig auf die Graphen angewendet werden kann.
   Ebenfalls sollen hier die Testspezifikation enthalten sein, die gegen die Implementierungen getestet werden soll.
5. **GraphTest**
   Diese Projekt soll alle Test Implementierungen sowie die Shell Skripten enthalten, welche in der Lage sind beide Implementierungen mit denselben Testmethoden zu testen. Dies soll ebenfalls für die Tests der Graphenalgorithmen gelten.

1.2 Lösungsidee

Folgend ist die Dokumentation der Grundvoraussetzungen der beiden Implementierungen angeführt.
Da die Spezifikation bereits durch die Aufgabenstellung vorgegeben ist, soll entsprechend dieser Spezifikation auch der Source entwickelt werden, wobei die Funktionen nicht void sondern bool zurückliefern sollen (bereits implementierte common.h). Dies sollte kein Problem darstellen, da die angegebenen Schnittstellen als mögliche Lösung angeboten wurden.
Die boolschen Rückgabewerte werden dazu verwendet damit der Aufrufer prüfen kann ob eine Funktion erfolgreich durchgeführt wurde oder ob es einen Fehler gab ohne gleich das Programm abbrechen zu müssen.

Die Fehlermeldungen sollen über die Konsole ausgegeben werden. Ein Abbruch des Programms soll nur bei einer kritischen Situation erfolgen wie z.B.: wenn eine Alokierung des Speichers fehlschlägt ansonsten soll bool false zurückgeliefert werden.
Da zu bevorzugen ist, das die Implementierung sich nicht vollständig auf den Aufrufer verlässt, sollen die Funktionen defensiv implementiert werden. Damit ist gemeint das zumindest folgende Fehlerbehandlungen implementiert werden müssen.

1. Prüfung ob Graph Pointer beim Erstellen eines Pointers NULL ist.
2. Prüfung ob bereits eine Kante eingefügt wurde.
3. Prüfung ob eine Kante gültig ist. (Indexoverflow vermeiden)

Da die implementierungsspezifischen Test Implementierungen Zugriff auf die Definition des Graph Typ benötigen soll jede Implementierung ihrerseits eine Header Datei definieren in der lediglich dieser Typ definiert wird.
Hilfsfunktionen sollen nicht in die Headerdatei aufgenommen werden sondern sollen als statische Funktionen am Beginn des Source implementiert werden um eine vorherige Deklaration zu vermeiden und sie nach außen hin zu verstecken.
Die vorgegebene Spezifikation soll um alle Funktionen erweitert werden, die in beiden Implementierungen sowie in einem C Programm, welches diese Implementierungen nützt, verwendet werden können.

Es wird hier vereinbart dass das Aufbauen des Graphs von Aufrufer kontrolliert wird und dieser auch für den korrekten Aufbau eines gültigen Graphen verantwortlich ist, daher muss es in den Implementierungen auch keine Prüfung geben ob der abgebildete Graph auch einen gültigen Graph darstellt. Damit ist gemeint das nicht gewährleitstet werden muss dass alle Knoten des Graphen so miteinander verbunden sind, dass keine Konten innerhalb des implementierten Containers einen neuen Graphen bilden.

1.3 Tests

Folgend ist die Vorgehensweise der Tests dokumentiert.
Für die Tests soll die Headerdatei assert.h eingebunden werden um die Korrektheit der Funktionsweise über die Funktion assert prüfen zu können. Dies ist möglich, da die zu testenden Funktionen bool zurückliefern um anzuzeigen ob die Funktion vollständig durchgeführt wurde.

1.3.1 Test Abhängigkeiten

Folgend ist die Dokumentation für die implementierungsabhängigen Implementierungen der Tests angeführt.
Da beide Implementierungen dieselbe Spezifikation verwenden sollen die Tests so implementiert werden, dass diese in der Lage sind alle Implementierungen zu testen. Da die Tests nicht davon ausgehen können, das zu testende Funktionen bereits fehlerfrei funktionieren müssen folgende Funktionen für die Tests separat implementiert werden um nicht abhängig von der getesteten Implementierung zu sein:

1. CreateGraph
   Wird benötigt um zu testen ob die implementierungsabhängige Funktion createGraph auch mit bereits initialisierten Graphen umgehen kann.
2. InsertEdge
   Wird benötigt um zu testen ob die implementierungsabhängigen Funktion InsertEdge auch mit bereits eingefügten Kanten umgehen kann.

Dazu müssen die implementierungsabhängigen Projekte diese Funktionen zur Verfügung stellen, wobei auf eine defensive Implementierung verzichtet werden kann. Es muss aber gewährleistet werden dass diese Funktonen fehlerfrei funktionieren, da es für diese Funktionen naturgemäß keine Tests geben kann.
Damit kann erreicht werden, dass beide Implementierung mit ein und denselben Source getestet werden können und daher auch dieselben Ergebnisse liefern müssen. Die verschiedenen Implementierungen müssen lediglich die benötigten Funktionen bereitstellen.

1.3.2 Testskript

Folgend ist die Dokumentation für das zu implementierende Testskript angeführt.
Da die Tests ihrerseits auch EXIT_FAILURE beinhalten können sollen die Testfunktionen innerhalb einer main Funktion über switch (Integer) aufgerufen werden können, damit pro Ausführung immer nur eine Testfunktion ausgeführt wird. Diese ist erforderlich, da kein ExceptionHandling zur Verfügung steht.
Um die Tests automatisiert durchführen zu können, soll ein Schell Skript implementiert werden welches über einen while loop die verschiedenen Testfunktionen einzeln aufruft. Daher muss darauf geachtet werden, dass die Anzahl der Tests mit der Schleifenbedingung übereinstimmt.

Des Weiteren muss das Schell Skript folgende Argumente aufnehmen können um für alle Implementierungen verwendet werden zu können.

1. Argument 1: Source root
   Das Verzeichnis in dem sich die Implementierung befindet
2. Argument 2: make target
   Der Target in der implementierungsabhängigen Source Datei.

Ausführung eines Tests:

Folgend ist beschrieben wie die Tests ausgeführt werden können.

1. Entpacken des Archives auf einem Unix System
2. Sicherstellung das Benutzer genügend Rechte hat
3. Schell Skript ausführbar machen
   ```
   cd <ARCHIVE_ROOT>/GraphTest
   chmod +x *.sh
   ```
4. Ausführung des Testskripts
   a. `cd <ARCHIVE_ROOT>/GraphTest`
   b. `./<script_name> <folder_impl> <target_name>`

2 Graphspezifikation

Folgend ist die Spezifikation der Aufgabe Graphen angeführt.

2.1 Implementierung

Folgend ist der C Source bzw. die Headerdateien angeführt.

2.1.1 common.h

Diese Datei spezifiziert die gemeinsam verwendeten Typen.

```
/*
 * common.h
 *
 * This header file contains all common types which can be used in
 * multiple programs
 *
 *  Created on: Sep 20, 2014
 *      Author: cchet
 */

#ifndef COMMON_H_
#define COMMON_H_

/* allows to use bool in c */
typedef enum bool {
    false = 0, true = 1
} bool;

#endif /* COMMON_H_ */
```

2.1.2 dg_adt.h

Diese Headerdatei spezifiziert die Minimum Funktionalität, welche von den verschiedenen Implementierungen bereitgestellt werden müssen.

```
/*
 * dg_adt.h
 *
 *  This is the header file which defines the interface for handling graphs.
 *
 *  Created on: Oct 12, 2014
 *      Author: cchet
 */

#ifndef DG_ADT_H
#define DG_ADT_H

#include "common.h"

/* The type representing the graph */
typedef struct Graph Graph;
/* Enums which are used for edge validation */
typedef enum EdgeInfo {
    EDGE_EXISTS, EDGE_NEW, EDGE_INVALID
} EdgeInfo;

/* Visitor function for visiting the nodes */
typedef void (*Visitor)(int source, int target, double weight);

/**
 * Creates a graph with the given size
 *
 * Graph **graph: the reference to created graph
 * int size: the maximum size of the graph. The graph cannot grow larger then this size.
 * return true if the creation was successful, false otherwise
 */
bool createGraph(Graph **graph, int size);
```

```
/**
 * Removes all of the nodes of this graph.
 *
 * Graph **graph: the references to the graph to be deleted.
 * return bool: true if the deletion was successful, false otherwise.
 *              if the deletion was successful then the given references will be NULL
 */
bool deleteGraph(Graph **graph);

/**
 * Inserts a graph edge into the graph.
 *
 * Graph *graph: the where the node will be inserted
 * int source: the source node id which will reference the target
 * int target: the target node id which will be referenced by the source
 * double weight: the weight of the to insert node reference
 * return bool: true if the insert was successful, false otherwise
 */
bool insertEdge(Graph *graph, int source, int target, double weight);

/**
 * Removes a references between two graph nodes.
 *
 * Graph *graph: the graph where to delete the references
 * int source: the node id which references the target
 * int target: the node is which is referenced by the source
 * return bool: true if the deletion was successful, false otherwise
 */
bool removeEdge(Graph *graph, int source, int target);

/**
 * Prints the given graph.
 *
 * Graph *graph: the graph to be printed.
 * return bool: true if the graph could be printed
 */
bool printGraph(Graph *graph);

/**
 * Validates the edge for the given graph.
 *
 * Graph* graph: the graph to check for the valid edge
 * int source: the node id of the source
 * int target: the node id of the target
 * return EdgeInfo: the edge info enum which represents the function result.
 */
EdgeInfo validateEdge(Graph* graph, int source, int target);

/**
 * Visits all nodes in the graph if there is weighting present.
 * If a weighting is present the two nodes are connected to each other
 *
 * GRaph* graph: the graph to visit
 * Visitor visitor: the function pointer which provides the visitor logic
 */
void visitGraph(Graph* graph, Visitor visitor);

#endif /* DG_ADT_H_ */
```

2.1.3 dg_adt_tests_depend.h

Dieser Headerdatei spezifiziert die implementierungsabhängigen Funktionen, die für die Tests von den verschiedenen Implementierungen bereitgestellt werden müssen.

```
/*
 * dg_adt_tests_depend.h
 *
 *  This header file specifies the test dependencies which must
 *  be implemented by the tested implementation.
 *
 *  Created on: Oct 19, 2014
 *      Author: cchet
 */

#ifndef DG_ADT_TESTS_DEPEND_H
#define DG_ADT_TESTS_DEPEND_H

#include "dg_adt.h"

/**
 * Creates a test graph so that the tests are not depending on the
 * tested implementation. Don't implement it defensive but take care it
 * works as expected.
 *
 * int size: the size of the to create graph
 */
Graph* createTestGraph(int size);

/**
 * Inserts a edge in test graph so that the tests are not depending on the
 * tested implementation. Don't implement it defensive but take care it
 * works as expected.
 *
 * This functions does not need to check for valid edge, the test implementor is
 * responsible to take care of this.
 *
 * Graph* graph: the test graph to insert edge
 * int source, the source node id
 * int tagret: the target node id
 * double weight: the weight of the edge
 */
void insertTestEdge(Graph* graph, int source, int target, double weight);

#endif /* DG_ADT_TESTS_DEPEND_H_ */
```

2.1.4 dg_adt_tests.h

Diese Headerdatei spezifiziert die Test Funktionen für die Implementierungen.

```
/*
 * graph_tests.h
 *
 *  This header files specifies the tests functions to be implemented and
 *  used for each implementation of dg_adt.h
 *  Therefore that the functions have a self speaking name, no comments needed
 *
 *  Created on: Oct 17, 2014
 *      Author: cchet
 */

#ifndef GRAPH_TESTS_H_
#define GRAPH_TESTS_H_

#include "dg_adt.h"

/* ############### Invalid test functions ###############*/
/* createGraph */
void case1_not_null_graph();

void case2_invalid_size_1();

void case3_invalid_size_2();

/* insertEdge */
void case4_insertEdge_null_graph();

void case5_insertEdge_invalid_weight_1();

void case6_insertEdge_invalid_weight_2();

void case7_insertEdge_already_existing();

void case8_insertEdge_invalid_node_reference();

/* removeEdge */
void case9_removeEdge_does_not_exist();

void case10 removeEdge invalid node reference();

/* removeGraph */
void case11_removeGraph_null();

/* ############### Valid test functions ###############*/
/* printGraph */
void case12_printGraph_null();

void case13_printGraph_no_edges();

void case14_printGraph_reflexive();

void case15_printGraph_second_j();

/* createGraph */
void case16_createGraph();

/* insertEdge*/
void case17_insertEdge();

/* removeEdge */
void case18_removeEdge();

#endif /* GRAPH_TESTS_H_ */
```

2.1.5 dg_adt_ml_tests.c

Folgend ist der Source der Graph Tests angeführt, welches für beide Implementierungen verwendet werden kann.

Diese Datei befindet sich im Verzeichnis GraphTests.

```
/*
 * main.c
 *
 * This is the main entry point to the tests.
 * Depending on the given argument the backed test program will be invoked.
 *
 *  Created on: Oct 12, 2014
 *      Author: cchet
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "assert.h"
#include "dg_adt_tests_depend.h"
#include "dg_adt_tests.h"

const int SIZE = 5;

int main(int argc, char** argv) {
    int value;

    if (argc != 2) {
        printf("Test source needs exact one integer argument");
        exit(EXIT_FAILURE);
    }
    value = atoi(argv[1]);

    switch (value) {
    case 1:
        case1_not_null_graph();
        break;
    case 2:
        case2_invalid_size_1();
        break;
    case 3:
        case3_invalid_size_2();
        break;
    case 4:
        case4_insertEdge_null_graph();
        break;
    case 5:
        case5 insertEdge invalid weight 1();
        break;
    case 6:
        case6_insertEdge_invalid_weight_2();
        break;
    case 7:
        case7_insertEdge_already_existing();
        break;
    case 8:
        case8_insertEdge_invalid_node_reference();
        break;
    case 9:
        case9_removeEdge_does_not_exist();
        break;
    case 10:
        case10_removeEdge_invalid_node_reference();
        break;
    case 11:
        case11_removeGraph_null();
        break;
    case 12:
        case12_printGraph_null();
        break;
    case 13:
        case13_printGraph_no_edges();
        break;
```

```
    case 14:
        case14_printGraph_reflexive();
        break;
    case 15:
        case15 printGraph second j();
        break;
    case 16:
        case16_createGraph();
        break;
    case 17:
        case17_insertEdge();
        break;
    case 18:
        case18_removeEdge();
        break;
    default:
        printf("Unknown test case id !!!");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

/* ############### Invalid test functions ###############*/
/* createGraph */
void case1_not_null_graph() {
    Graph* graph;

    printf("createGraph: Graph already created\n");
    graph= createTestGraph(SIZE);
    assert(createGraph(&graph, SIZE) == false);
}

void case2 invalid size 1() {
    Graph* graph;
    graph = NULL;

    printf("createGraph: Size invalid (0)\n");
    assert(createGraph(&graph, 0) == false);
}

void case3_invalid_size_2() {
    Graph* graph;
    graph = NULL;

    printf("createGraph: Size invalid (-1)\n");
    assert(createGraph(&graph, -1) == false);
}

/* insertEdge */
void case4_insertEdge_null_graph() {
    printf("insertEdge: Insert edge on null graph\n");
    assert(insertEdge(NULL, 1, 1, 1) == false);
}

void case5_insertEdge_invalid_weight_1() {
    Graph* graph;

    printf("insertEdge: with invalid weight (0)\n");
    graph = createTestGraph(1);
    assert(insertEdge(graph, 1, 1, 0) == false);
}

void case6_insertEdge_invalid_weight_2() {
    Graph* graph;

    printf("insertEdge: with invalid weight (-1)\n");
    graph = createTestGraph(1);
    assert(insertEdge(graph, 1, 1, -1) == false);
}

void case7_insertEdge_already_existing() {
    Graph* graph;

    printf("insertEdge: Edge already existing\n");
    graph = createTestGraph(1);
```

```
    insertTestEdge(graph, 1, 1, 1);
    assert(insertEdge(graph, 1, 1, 2) == false);
}
void case8_insertEdge_invalid_node_reference() {
    Graph* graph;

    printf("insertEdge: Edge points to invalid nodes\n");
    graph = createTestGraph(1);
    assert(insertEdge(graph, 10, 10, 1) == false);
}

/* removeEdge */
void case9_removeEdge_does_not_exist() {
    Graph* graph;

    printf("removeEdge: Edge does not exists\n");
    graph = createTestGraph(1);
    assert(removeEdge(graph, 1, 1) == false);
}

void case10_removeEdge_invalid_node_reference() {
    Graph* graph;

    printf("removeEdge: Edge points to invalid nodes\n");
    graph = createTestGraph(1);
    assert(removeEdge(graph, 10, 10) == false);
}

/* removeGraph */
void case11_removeGraph_null() {
    Graph* graph;
    graph = NULL;

    printf("removeGraph: Null Graph\n");
    assert(deleteGraph(&graph) == false);
}

/* ############### Valid test functions ###############*/
/* printGraph */
void case12_printGraph_null() {
    Graph* graph;
    graph = NULL;

    printf("printGraph: Null Graph\n");
    assert(printGraph(graph) == false);
}

void case13_printGraph_no_edges() {
    Graph* graph;

    printf("printGraph: empty Graph\n");
    graph = createTestGraph(SIZE);
    assert(printGraph(graph) == true);
}

void case14_printGraph_reflexive() {
    int i;
    Graph* graph;

    printf("printGraph: reflexive Graph\n");
    graph = createTestGraph(SIZE);
    for (i = 1; i <= SIZE; ++i) {
        insertTestEdge(graph, i, i, 1);
    }

    assert(printGraph(graph) == true);
}

void case15_printGraph_second_j() {
    int i, j;
    Graph* graph;

    printf("printGraph: each second target set\n");
    graph = createTestGraph(SIZE);
    for (i = 1; i <= SIZE; ++i) {
        for (j = 1; j <= SIZE; ++j) {
```

```
            insertTestEdge(graph, i, j, 1);
            j++;
        }
    }

    assert(printGraph(graph) == true);
}
/* createGraph */
void case16_createGraph() {
    Graph* graph;

    printf("createGraph: create empty Graph\n");
    assert(createGraph(&graph, 5) == true);
    assert(printGraph(graph) == true);
}

/* insertEdge*/
void case17_insertEdge() {
    Graph* graph;
    int i, j;

    printf("createGraph: insert edge on each second target \n");
    graph = createTestGraph(SIZE);
    for (i = 1; i <= SIZE; ++i) {
        for (j = 1; j <= SIZE; ++j) {
            assert(insertEdge(graph, i, j, 1) == true);
            j++;
        }
    }
    printGraph(graph);
}

/* removeEdge */
void case18_removeEdge() {
    Graph* graph;
    int i;

    printf("createGraph: remove added edges \n");
    graph = createTestGraph(SIZE);
    for (i = 1; i <= SIZE; ++i) {
        assert(insertEdge(graph, i, i, 1) == true);
        i++;
    }
    printf("\nGraph with added edges\n");
    printGraph(graph);
    for (i = 1; i <= SIZE; ++i) {
        assert(removeEdge(graph, i, i) == true);
        i++;
    }
    printf("\nGraph with removed edges\n");
    printGraph(graph);
}
```

2.1.6 ml_tests.sh

Folgend ist das Schell Skript angeführt welche für beide Implementierungen zur Anwendung kommt.

Diese Datei befindet sich im Verzeichnis GraphTests.

```
#!/bin/sh
TEST_FOLDER=tests
i=1
TEST_CASES=18

echo "creating test folder: $TEST_FOLDER"

mkdir $TEST_FOLDER

echo "test folder created"
echo "\ncopying test resources to test folder"
echo "cp *.c ../$1/*.c ../$1/*.h ../$1/makefile ../Graph/*.h $TEST_FOLDER"

cp dg adt ml tests.c ../$1/*.c ../$1/*.h ../$1/makefile ../Graph/*.h $TEST_FOLDER
cd $TEST_FOLDER

echo "\nin directory: $(pwd)"
echo "build test source"
echo "make clean depend $2"

make clean depend $2

echo "\n\n################################################"
echo "performing tests"
echo "################################################"
while [ $i -le $TEST_CASES ]
do
    echo "--------------------------------"
    echo "Invoking test case with id $i: "
    echo "--------------------------------"
    ./$2 $i
    echo
    i=`expr $i + 1`
done

echo "\n\nremoving test resources"
cd ..
rm -rf $TEST_FOLDER
echo "test finished"
echo "in directory: $(pwd)"
```

3 Adjazentmatrix

Folgend ist die Dokumentation für die Implementierung mittels einer Adjazenzmatrix angeführt.

3.1 Lösungsidee

Da hierbei eine zweidimensionale Matrix mit dynamischer Länge verwendet werden muss soll das Erstellen und das Löschen dieser Matrix innerhalb von statischen Funktionen, wie vorgegeben, implementiert werden.

Die Funktionen, welche durch die Headerdatei dg_adt.h spezifiziert werden, sollen wie vereinbart implementiert werden.

3.2 Implementierung

Folgend ist der Source und die Headerdateien der Implementierung Adjazenzmatrix angeführt.

3.2.1 dg_adt_m.h

Folgend ist die Headerdatei angeführt, die den Graph Typ definiert.

```
/*
 * dg_adt_m.h
 *
 *  This header file defines the Graph type which is used in the implementation and tests,
 *  therefore defined here and not in the implementation itself.
 *
 *  Created on: Oct 17, 2014
 *      Author: cchet
 */

#ifndef DG_ADT_M_H_
#define DG_ADT_M_H_

#include "dg adt.h"

/* The definition of the graph used in this application */
struct Graph {
    double** matrix;
    int size;
};

#endif /* DG_ADT_M_H_ */
```

3.2.2 dg_adt_m.c

Folgend ist der Source der Implementierung der Spezifikation dg_adt.h angefürt.

```
/*
 * dg_adt_m.c
 *
 * This is the implementation of of a directed graph for handling a weighted graph
 * by usage of an Adjazenzmatrix.
 *
 *  Created on: Oct 12, 2014
 *      Author: cchet
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dg_adt_m.h"

/* ############## private visible functions ############## */
/**
 * Creates the matrix for the graph
 *
 * int n: the the x,y size of the matrix
```

```
 * return int**: the created matrix
 * failure: will exit with failure when n is not within the range 0 < n < MAX INT
 */
static double** createMatrix(int n) {
    double** array;
    int i, j;

    /* Check for valid n */
    if (n <= 0) {
        printf("n must in range 0 < n < MAX_INT !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    array = malloc(n * sizeof(double *));
    for (i = 0; i < n; i++) {
        array[i] = malloc(n * sizeof(double));
    } /* for */

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            array[i][j] = 0;
        } /* for */
    } /* for */

    return array;
}

/**
 * Deletes the matrix.
 *
 * int** matrix: the matrix to be deleted
 * int n: the size of the array
 */
static void deleteMatrix(double** matrix, int n) {
    int i;

    /* Check for null matrix */
    if (matrix != NULL) {
        for (i = 0; i < n; ++i) {
            if (matrix[i] != NULL) {
                free(matrix[i]);
                matrix[i] = NULL;
            } /* if */
        } /* for */
        free(matrix);
        matrix = NULL;
    } /* if */
}

/* ############## public visible functions ############## */
bool createGraph(Graph **graph, int size) {
    double** matrix;

    /* Graph references needs to be NULL */
    if ((*graph) != NULL) {
        printf("The given graph reference must be null !!!\n");
        return false;
    } /* if */

    /* Check for valid graph size */
    if (size <= 0) {
        printf("size must in range 0 < size < MAX_INT !!! size:%d\n", size);
        return false;
    } /* if */

    *graph = malloc(sizeof(Graph));
    /* Graph creation failed */
    if (!(*graph)) {
        printf("The graph could not be created !!!\n");
        exit(EXIT_FAILURE);
    } /* if */
    matrix = createMatrix(size);

    /* Set attributes on graph */
    (*graph)->size = size;
    (*graph)->matrix = matrix;
```

```
    return true;
}

bool deleteGraph(Graph **graph) {
    /* Check for invalid pointer */
    if ((graph != NULL) || ((*graph) != NULL)) {
        return false;
    }
    /* Delete if pointer is valid */
    else {
        deleteMatrix((*graph)->matrix, (*graph)->size);
        free((*graph));
        (*graph) = NULL;
        return true;
    } /* if */
}

bool insertEdge(Graph *graph, int source, int target, double weight) {
    EdgeInfo result;

    /* Check for null graph */
    if (graph == NULL) {
        printf("Cannot insert edge in null graph !!!\n");
        return false;
    } /* if */

    /* Check for valid weight */
    if (weight <= 0) {
        printf("Weight must be in range 0 < weight < MAX_DOUBLE !!!\n");
        return false;
    } /* if */

    result = validateEdge(graph, source, target);
    switch (result) {
    case EDGE_NEW:
        graph->matrix[(source - 1)][(target - 1)] = weight;
        return true;
    case EDGE_EXISTS:
        printf(
                "Edge already exists. Remove the edge and add it afterwards !!!\n");
        return false;
    case EDGE_INVALID:
        printf("Source and target do not represent a valid edge !!!\n");
        return false;
    default:
        printf("Invalid EdgeInfo enum value detected !!!\n");
        exit(EXIT_FAILURE);
    } /* switch */
}

bool removeEdge(Graph *graph, int source, int target) {
    EdgeInfo result;

    result = validateEdge(graph, source, target);
    switch (result) {
    case EDGE_NEW:
        printf("Cannot remove not existing Edge !!!\n");
        return false;
    case EDGE_EXISTS:
        graph->matrix[(source - 1)][(target - 1)] = 0;
        return true;
    case EDGE_INVALID:
        printf("Source and target do not represent a valid edge !!!\n");
        return false;
    default:
        printf("Invalid EdgeInfo enum value detected !!!\n");
        exit(EXIT_FAILURE);
    } /* switch */
}

bool printGraph(Graph *graph) {
    int i, j;
    double** matrix;
    double value;

    /* Check for printable graph */
```

```
    if (graph == NULL) {
        printf("Graph is null and therefore cannot be printed !!!\n");
        return false;
    } /* if */
    /* Print the graph */
    else {
        /* print header line */
        printf("%4s", "");
        for (i = 1; i <= graph->size; ++i) {
            printf("%5d%5s", i, "|");
        } /* for */
        printf("\n");
        for (i = 1; i <= graph->size; ++i) {
            printf("-----------");
        } /* for */
        printf("\n");
        matrix = graph->matrix;
        for (i = 0; i < graph->size; ++i) {
            printf("%2i |", (i + 1));
            for (j = 0; j < graph->size; ++j) {
                value = matrix[i][j];
                if (value != 0) {
                    printf("%3s%3.2f%3s", "", matrix[i][j], "");
                } else {
                    printf("%5s%5s", "-", "");
                } /* if */
            } /* for */
            printf("\n");
        } /* for */

        return true;
    } /* if */
}

EdgeInfo validateEdge(Graph* graph, int source, int target) {
    /* Check for valid source and target */
    if ((target <= 0) || (target > graph->size) || (source <= 0)
            || (source > graph->size)) {
        return EDGE_INVALID;
    }
    /* Check if edge already present */
    else if (graph->matrix[(source - 1)][(target - 1)] != 0) {
        return EDGE_EXISTS;
    }
    /* Else new edge */
    else {
        return EDGE_NEW;
    } /* if */
}

void visitGraph(Graph* graph, Visitor visitor) {
    int i, j;
    if (graph == NULL) {
        printf("Cannot visit NULL graph !!!\n");
        exit(EXIT_FAILURE);
    } else {
        for (i = 0; i < graph->size; ++i) {
            for (j = 0; j < graph->size; ++j) {
                if (graph->matrix[i][j] != 0) {
                    visitor((i + 1), (j + 1), graph->matrix[i][j]);
                } /* if */
            } /* for */
        } /* for */
    } /* if */
}
```

3.2.3 dg_adt_tests_m.c

Folgend ist die Implementierung der Test Abhängigkeiten angeführt.

```
/*
 * dg_adt_test_m.c
 *
 * This source file implements the test depended functions.
 *
 *  Created on: Oct 12, 2014
 *      Author: cchet
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dg adt tests depend.h"
#include "dg adt m.h"

/**
 * Creates the by graph backed matrix.
 *
 * int n: the size of the first and second dimension of the matrix.
 */
static double** createTestMatrix(int n) {
    double** array;
    int i, j;

    array = malloc(n * sizeof(double *));
    for (i = 0; i < n; i++) {
        array[i] = malloc(n * sizeof(double));
    } /* for */

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            array[i][j] = 0;
        } /* for */
    } /* for */

    return array;
}

Graph* createTestGraph(int size) {
    Graph* graph = malloc(sizeof(Graph));

    /* Set attributes on graph */
    graph->size = size;
    graph->matrix = createTestMatrix(size);
    return graph;
}

void insertTestEdge(Graph* graph, int source, int target, double weight) {
    graph->matrix[(source - 1)][(target - 1)] = weight;
}
```

3.2.4 makefile

Folgend ist die Makefile für den Build dieser Implementierung angeführt.

```
COMPILER = gcc
COMPILER_FLAGS = -ansi -g -Og -pedantic -Wall -Wextra
LINKER_FLAGS = -lm
PROG=matrix
C_FILES = $(wildcard *.c)
O_FILES = $(C_FILES:.c=.o)

%.o: %.c
    $(COMPILER) $(COMPILER FLAGS) -c $<

$(PROG): $(O_FILES)
    $(COMPILER) -o $@ $(O_FILES) $(LINKER_FLAGS)

clean:
    rm -f $(O_FILES) $(PROG)

depend:
    @cat < /dev/null > makedep
    @for i in $(C_FILES); do \
        ($(COMPILER) -MM $$i >> makedep); done
    @echo "/^# BEGIN DEPENDENCIES\$$/+1,\$$d" > edcmds
    @echo "r makedep" >> edcmds
    @echo "wq" >> edcmds
    @cp makefile makefile.bak
    @ed -v - makefile < edcmds
    @rm edcmds makedep
    @echo "# LINE REQUIRED FOR MAKEDEPEND" >> makefile

# BEGIN DEPENDENCIES
# LINE REQUIRED FOR MAKEDEPEND
```

3.3 Tests

Folgend sind die Tests für die Implementierung mittels einer Adjazenzmatrix angeführt.

Die Test werden wie folgt ausgeführt: `./ml_test.sh GraphMatrix matrix`

```
--------------------------------
Invoking test case with id 1:
--------------------------------
createGraph: Graph already created
The given graph reference must be null !!!

--------------------------------
Invoking test case with id 2:
--------------------------------
createGraph: Size invalid (0)
size must in range 0 < size < MAX_INT !!! size:0

--------------------------------
Invoking test case with id 3:
--------------------------------
createGraph: Size invalid (-1)
size must in range 0 < size < MAX_INT !!! size:-1

--------------------------------
Invoking test case with id 4:
--------------------------------
insertEdge: Insert edge on null graph
Cannot insert edge in null graph !!!

--------------------------------
Invoking test case with id 5:
--------------------------------
insertEdge: with invalid weight (0)
Weight must be in range 0 < weight < MAX_DOUBLE !!!

--------------------------------
Invoking test case with id 6:
--------------------------------
insertEdge: with invalid weight (-1)
Weight must be in range 0 < weight < MAX_DOUBLE !!!


--------------------------------
Invoking test case with id 7:
--------------------------------
insertEdge: Edge already existing
Edge already exists. Remove the edge and add it afterwards !!!

--------------------------------
Invoking test case with id 8:
--------------------------------
insertEdge: Edge points to invalid nodes
Source and target do not represent a valid edge !!!

--------------------------------
Invoking test case with id 9:
--------------------------------
removeEdge: Edge does not exists
Cannot remove not existing Edge !!!

--------------------------------
Invoking test case with id 10:
--------------------------------
removeEdge: Edge points to invalid nodes
Source and target do not represent a valid edge !!!

--------------------------------
Invoking test case with id 11:
--------------------------------
removeGraph: Null Graph

--------------------------------
Invoking test case with id 12:
--------------------------------
printGraph: Null Graph
Graph is null and therefore cannot be printed !!!
```

Sollten Funktionsargumente nicht der Spezifikation entsprechen so werden entsprechende Fehlermeldungen ausgeben um den Aufrufer mitzuteilen, warum die Funktion den Algorithmus nicht ausführen konnte.

```
------------------------------
Invoking test case with id 13:
------------------------------
printGraph: empty Graph
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -         -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

------------------------------
Invoking test case with id 14:
------------------------------
printGraph: reflexive Graph
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00       -         -         -         -
 2 |    -        1.00       -         -         -
 3 |    -         -        1.00       -         -
 4 |    -         -         -        1.00       -
 5 |    -         -         -         -        1.00

------------------------------
Invoking test case with id 15:
------------------------------
printGraph: each second target set
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00       -        1.00       -        1.00
 2 |   1.00       -        1.00       -        1.00
 3 |   1.00       -        1.00       -        1.00
 4 |   1.00       -        1.00       -        1.00
 5 |   1.00       -        1.00       -        1.00

------------------------------
Invoking test case with id 16:
------------------------------
createGraph: create empty Graph
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -         -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 17:
--------------------------------
createGraph: insert edge on each second target
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00       -        1.00       -        1.00
 2 |   1.00       -        1.00       -        1.00
 3 |   1.00       -        1.00       -        1.00
 4 |   1.00       -        1.00       -        1.00
 5 |   1.00       -        1.00       -        1.00

--------------------------------
Invoking test case with id 18:
--------------------------------
createGraph: remove added edges

Graph with added edges
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00       -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -        1.00       -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -        1.00

Graph with removed edges
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -         -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
```

Sollten die Argumente sowie Vorbedingungen für die Funktionen korrekt sein, so werden diese auch korrekt ausgeführt.

4 Adjazentliste

Folgend ist die Dokumentation der Implementierung mit einer Adjazenzliste angeführt.

4.1 Lösungsidee

Um das Handling mit den beiden Listen (Graphknoten, Kantenknoten) zu erleichtern soll eine doppelt verlinkte Liste mit Anker verwendet werden, da hier alle NULL Prüfungen wegfallen und es sehr einfach ist einen Konten in die Liste einzufügen.

Wie bei der Implementierung mittels Matrix sollen hier statische Hilfsfunktionen implementiert werden, die für einen Graphknoten sowie einen Kantenknoten

1. Einen Listknoten erstellen
2. Eine Liste löschen
3. Ein Listknoten löschen
4. Einen Listknoten finden
5. Einen Listenknoten in einer Liste einfügen

4.2 Implementierung

4.2.1 dg_adt_l.h

Folgend ist die Headerdatei angeführt, die den Graph Typ, sowie die enthaltenen Strukturen definiert.

```
/*
 * dg_adt_l.h
 *
 *  This is the header for the implementation which uses a list for a
 *  graph representation. It enhances the dg_adt.h specification for the
 *  list specific helper functions.
 * Also it contains the concrete struct definition of the graph and used list nodes.
 *
 *  Created on: Oct 13, 2014
 *      Author: Thomas Herzog
 */

#ifndef DG ADT L H
#define DG ADT L H

#include "dg_adt.h"

typedef struct GraphNode GraphNode; /* Represents a node in the graph list */
typedef struct WeightNode WeightNode; /* Represents a weight node in the weighting list */

/* Realizes a double-cyclic list with anchor because easier to handle */
struct GraphNode {
    int id;
    WeightNode* weightings;
    GraphNode* next;
    GraphNode* previous;
};

/* Realizes a double-cyclic list with anchor because easier to handle */
struct WeightNode {
    int id;
    double weight;
    WeightNode* next;
    WeightNode* previous;
};
```

```
/* Graph here holds the list which holds the nodes */
struct Graph {
    GraphNode* list;
    int size;
};

#endif /* DG_ADT_L_H_ */
```

4.2.2 dg_adt_l.c

Folgend ist der Source der Implementierung der Spezifikation dg_adt.h angefürt.

```
/*
 * dg_adt_l.c
 *
 * This is the implementation of of a directed graph for handling a weighted graph
 * by usage of an Adjazenzlist.
 *
 *  Created on: Oct 12, 2014
 *      Author: cchet
 */
#include <stdio.h>
#include <stdlib.h>
#include "dg_adt_l.h"
#include "common.h"

/* ####### local used functions ####### */

/**
 * Creates a weight node for the given id and weight.
 * This function does not care about the validity of the id and weight,
 * this is the responsibility of the caller.
 *
 * int id: the id of the referenced graph node
 * double weight: the weight for this edge
 * return WeightNode*: the create weight node with next and previous set to itself
 */
static WeightNode* createWeightNode(int id, double weight) {
    WeightNode* node = NULL;

    node = malloc(sizeof(WeightNode));
    /* Graph creation failed */
    if (!node) {
        printf("The weight node could not be created !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    node->id = id;
    node->weight = weight;
    node->next = node;
    node->previous = node;

    return node;
}

/**
 * Appends a weighting node to the list on the proper position,
 * so that the order of the references nodes is kept.
 *
 * WeightNode* list: the weighting list of the graph node
 * WeightNode* node: the node to insert into the list.
 */
static void appendWeightNode(WeightNode* list, WeightNode* node){
    WeightNode* temp = list->next;
    while((temp != list) && (temp->id < node->id)) {
        temp = temp->next;
    }
    node->next = temp;
    node->previous = temp->previous;
    temp->previous->next = node;
    temp->previous = node;
}

/**
```

```
 * Creates a graph node with the given id.
 * This function does not care about a valid node id, this is the responsibility of the
caller.
 *
 * int id: the id of the graph node
 * return GRaphNode*: the created graph node with next and previous set to itself
 */
static GraphNode* createGraphNode(int id) {
    GraphNode* node = NULL;
    WeightNode* weightings;

    node = malloc(sizeof(GraphNode));
    /* Graph creation failed */
    if (!node) {
        printf("The graph node could not be created !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    weightings = createWeightNode(-1, -1);

    node->id = id;
    node->weightings = weightings;
    node->next = node;
    node->previous = node;

    return node;
}

/**
 * Deletes the weight node list.
 *
 * WeightNode* list: the weight node representing the list. (anchor element)
 */
static void deleteWeightNodeList(WeightNode** list) {
    WeightNode* node;
    WeightNode* next;

    if (list != NULL) {
        next = (*list);
        node = (*list)->next;
        while (node != (*list)) {
            next = node->next;
            free(node);
            node = next;
        } /* while */
        free((*list));
        (*list) = NULL;
    } /* if */
}

/**
 * Finds the graph node in the graph hold list.
 *
 * GRaph* graph: the graph to search for the node
 * int id: the id of the node to be found
 * return GraphNode*: the found node, null otherwise
 */
static GraphNode* findGraphNode(Graph* graph, int id) {
    GraphNode* node = NULL;

    /* Check for null graph */
    if (graph != NULL) {
        node = graph->list->next;
        while ((node != graph->list) && (node->id != id)) {
            node = node->next;
        } /* while */
        /* GraphNode not found */
        if (node == graph->list) {
            node = NULL;
        } /* if */
    } /* if */

    return node;
}

/**
```

```
 * Finds the edge in the graph.
 *
 * Graph* graph: the graph to search for the edge
 * int source: the source node holding the weighting
 * int target: the target which is referenced by the weighting entry.
 * return WeightNode*: the found node null, otherwise
 */
static WeightNode* findEdge(Graph* graph, int source, int target) {
    GraphNode* gNode;
    WeightNode* wNode;

    if (graph != NULL) {
        gNode = graph->list->next;
        while (gNode != graph->list) {
            /* source found */
            if (gNode->id == source) {
                wNode = gNode->weightings->next;
                while ((wNode != gNode->weightings) && (wNode->id != target)) {
                    wNode = wNode->next;
                } /* while */
                /* target found */
                if (wNode != gNode->weightings) {
                    return wNode;
                } else {
                    return NULL;
                } /* if */
            } /* if */
            gNode = gNode->next;
        } /* while */
    } /* if */

    /* edge not found */
    return NULL;
}

/* ###################### public visible methods ######################*/
bool createGraph(Graph **graph, int size) {
    int i;
    GraphNode* nodeList;
    GraphNode* node;

    /* Graph references needs to be NULL */
    if ((*graph) != NULL) {
        printf("The given graph reference must be null !!!\n");
        return false;
    } /* if */

    /* Check for valid graph size */
    if (size <= 0) {
        printf("size must in range 0 < size < MAX_INT !!! size:%d !!!\n", size);
        return false;
    } /* if */

    /* Create graph */
    *graph = malloc(sizeof(Graph));
    nodeList = createGraphNode(-1);

    /* Add nodes to graph */
    for (i = 1; i <= size; ++i) {
        node = createGraphNode(i);
        node->next = nodeList;
        node->previous = nodeList->previous;
        nodeList->previous->next = node;
        nodeList->previous = node;
    } /* for */

    /* set graph attributes */
    (*graph)->list = nodeList;
    (*graph)->size = size;

    return true;
}

bool deleteGraph(Graph **graph) {
    GraphNode* node;
    GraphNode* next;
```

```
    if ((*graph) != NULL) {
        node = (*graph)->list->next;
        while (node != (*graph)->list) {
            next = node->next;
            deleteWeightNodeList(&(node->weightings));
            free(node);
            node = next;
        } /* while */
        (*graph)->list = NULL;
        free((*graph));
        (*graph) = NULL;
        return true;
    } /* if */

    return false;
}

bool insertEdge(Graph *graph, int source, int target, double weight) {
    WeightNode* weighting;
    GraphNode* node;
    EdgeInfo info;

    /* Check for null graph */
    if (graph == NULL) {
        printf("Cannot insert edge in NULL graph !!!\n");
        return false;
    } /* if */

    /* Check for valid weight */
    if (weight <= 0) {
        printf("Weight must be in range 0 < weight < MAX_DOUBLE !!!\n");
        return false;
    } /* if */

    /* handle result */
    info = validateEdge(graph, source, target);
    switch (info) {
    case EDGE_EXISTS:
        printf("Edge already exists. Remove it and afterwards add it !!!\n");
        return false;
        break;
    case EDGE_INVALID:
        printf(
                "Source and target must point to a valid graph node !!! source=%d, target=%d
!!! \n",
                source, target);
        return false;
        break;
    case EDGE_NEW:
        appendWeightNode(findGraphNode(graph, source)->weightings, createWeightNode(target,
weight));
        return true;
        break;
    default:
        printf("Unknown EdgeInfo instance detected !!!\n");
        exit(EXIT_FAILURE);
        break;
    } /* switch  */

    /* compiler wants it :) */
    return false;
}

bool removeEdge(Graph *graph, int source, int target) {
    WeightNode* weighting;
    EdgeInfo info;

    /* Check for null graph */
    if (graph == NULL) {
        printf("Cannot insert edge in NULL graph !!!\n");
        return false;
    } /* if */

    /* handle result */
    info = validateEdge(graph, source, target);
```

```
    switch (info) {
    case EDGE_EXISTS:
        weighting = findEdge(graph, source, target);
        weighting->previous->next = weighting->next;
        weighting->next->previous = weighting->previous;
        weighting->next = NULL;
        weighting->previous = NULL;
        free(weighting);
        return true;
        break;
    case EDGE_INVALID:
        printf(
                "Source and target must point to a valid graph node !!! source=%d, target=%d
!!!\n",
                source, target);
        return false;
        break;
    case EDGE_NEW:
        printf("Edge not found !!!\n");
        return false;
        break;
    default:
        printf("Unknown EdgeInfo instance detected !!!\n");
        exit(EXIT_FAILURE);
        break;
    } /* switch */

    /* compiler wants it :) */
    return false;
}

bool printGraph(Graph *graph) {
    int i, lastVisited;
    GraphNode* gNode;
    WeightNode* wNode;

    /* Check for printable graph */
    if (graph == NULL) {
        printf("Graph is null and therefore cannot be printed !!!\n");
        return false;
    } /* if */
    /* Print the graph */
    else {
        /* print header line */
        printf("%4s", "");
        for (i = 1; i <= graph->size; ++i) {
            printf("%5d%5s", i, "|");
        } /* for */
        printf("\n");
        for (i = 1; i <= graph->size; ++i) {
            printf("-----------");
        } /* for */
        printf("\n");
        gNode = graph->list->next;
        while (gNode != graph->list) {
            printf("%2i |", gNode->id);
            wNode = gNode->weightings->next;
            lastVisited = 1;
            while (wNode != gNode->weightings) {
                for (i = lastVisited; i < wNode->id; ++i) {
                    printf("%5s%5s", "-", "");
                } /* for */
                printf("%3s%3.2f%3s", "", wNode->weight, "");
                lastVisited = wNode->id + 1;
                wNode = wNode->next;
            } /* while */
            for (i = lastVisited; i <= graph->size; ++i) {
                printf("%5s%5s", "-", "");
            } /* for */
            gNode = gNode->next;
            printf("\n");
        } /* while */

        return true;
    } /* if */
}
```

```
EdgeInfo validateEdge(Graph* graph, int source, int target) {
    /* Check for valid source and target */
    if ((target <= 0) || (target > graph->size) || (source <= 0)
            || (source > graph->size)) {
        return EDGE_INVALID;
    } /* if */

    /* Check if edge already present */
    if (findEdge(graph, source, target) != NULL) {
        return EDGE_EXISTS;
    } /* if */

    return EDGE_NEW;
}

void visitGraph(Graph* graph, Visitor visitor) {
    WeightNode* wNode;
    GraphNode* gNode;

    if (graph != NULL) {
        gNode = graph->list->next;
        while (gNode != graph->list) {
            wNode = gNode->weightings->next;
            while (wNode != gNode->weightings) {
                if (wNode->weight != 0) {
                    visitor(gNode->id, wNode->id, wNode->weight);
                } /* if */
                wNode = wNode->next;
            } /* while */
            gNode = gNode->next;
        } /* while */
    } /* if */
}
```

4.2.3 dg_adt_tests_l.c

Folgend ist die Implementierung der Test Abhängigkeiten angeführt.

```
/*
 * dg_adt_test_m.c
 *
 * This source file implements the test depended functions.
 *
 *  Created on: Oct 12, 2014
 *      Author: cchet
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dg adt tests depend.h"
#include "dg adt m.h"

/**
 * Creates the by graph backed matrix.
 *
 * int n: the size of the first and second dimension of the matrix.
 */
static double** createTestMatrix(int n) {
    double** array;
    int i, j;

    array = malloc(n * sizeof(double *));
    for (i = 0; i < n; i++) {
        array[i] = malloc(n * sizeof(double));
    } /* for */

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            array[i][j] = 0;
        } /* for */
    } /* for */

    return array;
}

Graph* createTestGraph(int size) {
    Graph* graph = malloc(sizeof(Graph));

    /* Set attributes on graph */
    graph->size = size;
    graph->matrix = createTestMatrix(size);
    return graph;
}

void insertTestEdge(Graph* graph, int source, int target, double weight) {
    graph->matrix[(source - 1)][(target - 1)] = weight;
}
```

4.2.4 makefile

Folgend ist die Makefile für den Build dieser Implementierung angeführt.

```
COMPILER = gcc
COMPILER_FLAGS = -ansi -g -Og -pedantic -Wall -Wextra
LINKER_FLAGS = -lm
PROG=list
C_FILES = $(wildcard *.c)
O_FILES = $(C_FILES:.c=.o)

%.o: %.c
    $(COMPILER) $(COMPILER FLAGS) -c $<

$(PROG): $(O_FILES)
    $(COMPILER) -o $@ $(O_FILES) $(LINKER_FLAGS)

clean:
    rm -f $(O_FILES) $(PROG)

depend:
    @cat < /dev/null > makedep
    @for i in $(C_FILES); do \
        ($(COMPILER) -MM $$i >> makedep); done
    @echo "/^# BEGIN DEPENDENCIES\$$/+1,\$$d" > edcmds
    @echo "r makedep" >> edcmds
    @echo "wq" >> edcmds
    @cp makefile makefile.bak
    @ed -v - makefile < edcmds
    @rm edcmds makedep
    @echo "# LINE REQUIRED FOR MAKEDEPEND" >> makefile

# BEGIN DEPENDENCIES
# LINE REQUIRED FOR MAKEDEPEND
```

4.3 Tests

Folgend sind die Test der Implementierung Adjanzenzliste angeführt.

Die Test werden wie folgt ausgeführt: `./ml_test.sh GraphList list`

```
-------------------------------
Invoking test case with id 7:
-------------------------------
insertEdge: Edge already existing
Edge already exists. Remove it and afterwards add it !!!

-------------------------------
Invoking test case with id 8:
-------------------------------
insertEdge: Edge points to invalid nodes
Source and target must point to a valid graph node !!! source=10, target=10 !!!

-------------------------------
Invoking test case with id 9:
-------------------------------
removeEdge: Edge does not exists
Edge not found !!!

-------------------------------
Invoking test case with id 10:
-------------------------------
removeEdge: Edge points to invalid nodes
Source and target must point to a valid graph node !!! source=10, target=10 !!!

-------------------------------
Invoking test case with id 11:
-------------------------------
removeGraph: Null Graph

-------------------------------
Invoking test case with id 12:
-------------------------------
printGraph: Null Graph
Graph is null and therefore cannot be printed !!!

-------------------------------
Invoking test case with id 1:
-------------------------------
createGraph: Graph already created
The given graph reference must be null !!!

-------------------------------
Invoking test case with id 2:
-------------------------------
createGraph: Size invalid (0)
size must in range 0 < size < MAX_INT !!! size:0 !!!

-------------------------------
Invoking test case with id 3:
-------------------------------
createGraph: Size invalid (-1)
size must in range 0 < size < MAX_INT !!! size:-1 !!!

-------------------------------
Invoking test case with id 4:
-------------------------------
insertEdge: Insert edge on null graph
Cannot insert edge in NULL graph !!!

-------------------------------
Invoking test case with id 5:
-------------------------------
insertEdge: with invalid weight (0)
Weight must be in range 0 < weight < MAX_DOUBLE !!!

-------------------------------
Invoking test case with id 6:
-------------------------------
insertEdge: with invalid weight (-1)
Weight must be in range 0 < weight < MAX_DOUBLE !!!
```

Auch diese Implementierung führt alle erforderlichen Prüfungen durch und behandelt diese korrekt.

```
------------------------------
Invoking test case with id 17:
------------------------------
createGraph: insert edge on each second target
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |   1.00       -        1.00       -        1.00
 2 |   1.00       -        1.00       -        1.00
 3 |   1.00       -        1.00       -        1.00
 4 |   1.00       -        1.00       -        1.00
 5 |   1.00       -        1.00       -        1.00

------------------------------
Invoking test case with id 18:
------------------------------
createGraph: remove added edges

Graph with added edges
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |   1.00       -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -        1.00       -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -        1.00

Graph with removed edges
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |    -         -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -


------------------------------
Invoking test case with id 13:
------------------------------
printGraph: empty Graph
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |    -         -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

------------------------------
Invoking test case with id 14:
------------------------------
printGraph: reflexive Graph
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |   1.00       -         -         -         -
 2 |    -        1.00       -         -         -
 3 |    -         -        1.00       -         -
 4 |    -         -         -        1.00       -
 5 |    -         -         -         -        1.00

------------------------------
Invoking test case with id 15:
------------------------------
printGraph: each second target set
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |   1.00       -        1.00       -        1.00
 2 |   1.00       -        1.00       -        1.00
 3 |   1.00       -        1.00       -        1.00
 4 |   1.00       -        1.00       -        1.00
 5 |   1.00       -        1.00       -        1.00

------------------------------
Invoking test case with id 16:
------------------------------
createGraph: create empty Graph
        1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------------
 1 |    -         -         -         -         -
 2 |    -         -         -         -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
```

Auch diese Test zeigen, dass sich diese Implementierung wie die andere verhält und die Ausgabe der Matrix exakt gleich ist, obwohl diese über eine Liste implementiert wurde.

5 Graphalgorithmen

Folgend ist die Dokumentation für die Graph Algorithmen angeführt.

5.1 Lösungsidee

Nachdem es einfach ist auf einer Matrix die verlangten Operationen durchzuführen, soll der implementierte Source eine Matrix zur Abbildung des Graphen besitzen, welcher kopiert wird, damit auf diesen die Operationen durchgeführt werden können.
Hierbei soll das Kopieren des Graphen über die Initialisierung eines Kontexts erfolgen, welcher initialisiert und auch nach den Operationen zerstört werden muss.
Jegliche Funktion, die kontextabhängig ist muss sicherstellen, dass sie nicht außerhalb eines initialisierten Kontext aufgerufen werden kann. Des Weiteren darf kein Kontext initialisiert werden wenn bereits ein Kontext initialisiert wurde.

Für das Kopieren des Graphen soll die von den Implementierungen zur Verfügung gestellte Funktion visitAllGraph verwendet werden, welche über den gesamten Graphen iteriert. Dies soll über die Übergabe eines Funktionspointer erfolgen.

Für die verschiedenen Operationen muss jetzt nur mehr über die Matrix mit den Indexes gemäß der mathematischen Definition iteriert werden und die enthaltenen Knoten validiert werden ob Sie die gefragten Eigenschaften erfüllen. Die Implementierung muss wieder unabhängig von den Implementierungen des Graphen implementiert und auch gegen die Implementierungen getestet werden.

5.2 Implementierung

Folgend ist die Implementierung der Graphenalgorithmen und deren Testabhängigkeiten, sowie Tests angeführt.

5.2.1 dg_adt_alg.h

Folgend ist die Header Datei welche die Graphalgorithmen spezifiziert angeführt.

```
/*
 * graph_algs.h
 *
 *  This header specifies the functionality of the graph algorithm source
 *  which performs operations on a graph.
 *
 *  Created on: Oct 13, 2014
 *      Author: cchet
 */

#ifndef GRAPH_ALGS_H_
#define GRAPH_ALGS_H_

#include "dg_adt.h"

/* Context control */
/**
 * Initializes the context for the graph operations.
 * The whole graph will be copied and on the copied graph the
 * operations will be performed
 *
 * Graph* graph: the graph to be initialize context on
 * int n: the size of the graph. Can be greater but not smaller than the size
 * of the given graph, otherwise an error will be thrown.
 */
bool initContext(Graph* graph, int n);

/**
 * Destroys the context by releasing all hold resources.
 */
bool destroyContext();

/* Context depended functions */
/**
 * Answers the question if the graph is reflexive.
 * Depends on an initialized context
 */
bool isReflexive();

/**
 * Answers the question if the graph is symmetric.
 * Depends on an initialized context
 */
bool isSymmetric();

/**
 * Answers the question if the graph is asymmetric.
 * Depends on an initialized context
 */
bool isAsymmetric();

/**
 * Answers the question if the graph is transitive.
 * Depends on an initialized context
 */
bool isTransitive();

/* context free functions */
/**
 * Will invert the graph nodes and copies the result in an own graph.
 */
Graph* invert(Graph* graph, int n);

/**
```

```
 * Prints all of the graph nodes which are connected with an edge
 * which weight > 0
 */
bool printGraphProperties(Graph* graph);

#endif /* GRAPH_ALGS_H_ */
```

5.2.2 dg_adt_alg.c

Folgend ist der Source der Implementierung der Spezifikation dg_adt_alg.h angefürt.

```
/*
 * graph_algs.c
 *
 *  This source performs validations on a graph and is context depended.
 *  This this means a context must be initialized for some functions
 *  This to function properly.
 *  This The context will copy the graph into a matrix on which the
 *  the validations will be performed.
 *
 *  Created on: Oct 13, 2014
 *      Author: cchet
 */
#include <stdlib.h>
#include <stdio.h>
#include "dg_adt_alg.h"

/* ############## private global members ############## */
static double** matrix = NULL;
static int size = 0;
static bool contextActive = false;

/* ############## private sued functions ############## */
/**
 * Validates the context and handles it depending on the given flags.
 *
 * bool active: true if the context must be active
 * bool strict: true if the program shall exist
 * return bool: true if the context is active, false otherwise,
 * but depends on strict flag which will exit program with an failure
 */
static bool validateContext(bool active, bool strict) {
    bool valid = true;
    if (active) {
        if (!contextActive) {
            printf("Context not active !!!\n");
            valid = false;
        } /* if */
    } else {
        if (contextActive) {
            printf("Context already active !!!\n");
            valid = false;
        } /* if */
    } /* if */
    if ((!valid) && (strict)) {
        exit(EXIT_FAILURE);
    } /* if */
    return valid;
}

/**
 * Creates the matrix which holds the context depended graph.
 */
static void createAlgMatrix() {
    int i, j;

    validateContext(true, true);

    matrix = malloc(size * sizeof(double *));
    for (i = 0; i < size; i++) {
        matrix[i] = malloc(size * sizeof(double));
    } /* for */

    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            matrix[i][j] = 0;
```

```
        } /* for */
    } /* for */
}

/**
 * Deletes the matrix which held the context depended graph
 */
static void deleteAlgMatrix() {
    int i;

    validateContext(true, true);

    if (matrix != NULL) {
        for (i = 0; i < size; ++i) {
            if (matrix[i] != NULL) {
                free(matrix[i]);
                matrix[i] = NULL;
            } /* if */
        } /* for */
        free(matrix);
        matrix = NULL;
    } /* if */
}

/**
 * Copies the graph nodes into the backed matrix
 *
 * int source: the source node id
 * int target: the target node id
 * double: the weight of the edge which connects the nodes
 */
static void visit(int source, int target, double weight) {
    validateContext(true, true);
    if ((source > size) || (target > size) || (size < source)
            || (size < target)) {
        printf(
                "Visited graph not in conjunction with initialized context !!!\n");
        exit(EXIT_FAILURE);
    } /* if */
    matrix[(source - 1)][(target - 1)] = weight;
}

/**
 * Prints the graph nodes if the connecting edge has an weight > 0
 *
 * int source: the source node id
 * int target: the target node id
 * double: the weight of the edge which connects the nodes
 */
static void print(int source, int target, double weight) {
    if (weight != 0) {
        printf("source=%d -> target=%d (%3.2f)\n", source, target, weight);
    } /* if */
}

/* ############## public sued functions ############## */
bool initContext(Graph* graph, int n) {
    if (!validateContext(false, false)) {
        return false;
    } else if (n <= 0) {
        printf("Invalid size for graph !!!\n");
        return false;
    } else if (graph == NULL) {
        printf("Cannot initialize context on null graph !!!\n");
        return false;
    } /* if */
    size = n;
    contextActive = true;
    createAlgMatrix();
    visitGraph(graph, &visit);
    return true;
}

bool destroyContext() {
    if (!validateContext(true, false)) {
        return false;
```

```
    } /* if */
    deleteAlgMatrix();
    contextActive = false;
    size = 0;
    return true;
}

bool isReflexive() {
    int i, j;
    bool fail = false;

    validateContext(true, true);

    i = 0;
    while ((i < size) && (!fail)) {
        j = 0;
        while ((j < size) && (!fail)) {
            if (matrix[i][j] != 0) {
                fail = (matrix[i][i] == 0) || (matrix[j][j] == 0);
            } /* if */
            j++;
        } /* while */
        i++;
    } /* while */

    return !fail;
}

bool isSymmetric() {
    bool fail = false;
    int i, j;

    validateContext(true, true);

    i = 0;
    while ((i < size) && (!fail)) {
        j = 0;
        while ((j < size) && (!fail)) {
            if (matrix[i][j] != 0) {
                fail = matrix[j][i] == 0;
            } /* if */
            j++;
        } /* while */
        i++;
    } /* while */

    return !fail;
}

bool isAsymmetric() {
    validateContext(true, true);

    return !isSymmetric() && !isReflexive();
}

bool isTransitive() {
    bool fail = false;
    int i, j, k;

    validateContext(true, true);

    k = 0;
    while ((k < size) && (!fail)) {
        i = 0;
        while ((i < size) && (!fail)) {
            if (matrix[k][i] != 0) {
                j = 0;
                while ((j < size) && (!fail)) {
                    if ((matrix[i][j]) != 0) {
                        fail = (matrix[k][j] == 0);
                    } /* if */
                    j++;
                } /* while */
            } /* if */
            i++;
        } /* while */
```

```
        k++;
    } /* while */

    return !fail;

}

Graph* invert(Graph* graph, int n) {
    Graph* nGraph = NULL;
    int i, j;

    validateContext(false, true);

    /* prepare context */
    initContext(graph, n);

    /* create graph */
    createGraph(&nGraph, size);

    /* copy inverted */
    i = j = 0;
    while (i < n) {
        j = i + 1;
        while (j < n) {
            if (matrix[j][i] != 0) {
                insertEdge(nGraph, (i + 1), (j + 1), matrix[j][i]);
            } /* if */
            if (matrix[i][j] != 0) {
                insertEdge(nGraph, (j + 1), (i + 1), matrix[i][j]);
            } /* if */
            j++;
        } /* while */
        i++;
    } /* while */

    /* destroy context */
    destroyContext();

    return nGraph;
}

bool printGraphProperties(Graph* graph) {
    if (graph == NULL) {
        printf("Cannot print graph properties of null graph !!!\n");
        return false;
    } /* if */
    visitGraph(graph, &print);
    return true;
}
```

5.2.3 dg_adt_alg_tests.h

Folgend ist die Header Datei für die Tests angeführt.

```
/*
 * dg_adt_alg_tests.h
 *
 *  This header file specifies the tests for the dg_adt_alg.h specification
 *
 *  Created on: Oct 19, 2014
 *      Author: cchet
 */

#ifndef DG_ADT_ALG_TESTS_H_
#define DG_ADT_ALG_TESTS_H_

#include "dg adt.h"

/* ############## Invalid tests ############## */

/* Context control  */
void case1_initContext_null_graph();

void case2_initContext_invalid_size1();

void case3_initContext_invalid_size2();
```

```
void case4_initContext_existing_context();

void case5_initContext_graph_not_in_conjunction();

void case6 destroyContext non existing context();

/* isReflexive */
void case7_isReflexive_no_context();

/* isSymetric */
void case8_isSymetric_no_context();

/* isAsymetric */
void case9_isAsymetric_no_context();

/* isTransitiv */
void case10_isTransitiv_no_context();

/* invertGraph */
void case11_invertGraph_null_graph();

void case12_invertGraph_existing_context();

void case13_invertGraph_invalid_size1();

void case14_invertGraph_invalid_size2();

void case15_invertGraph_invalid_size3();

/* printGraphProperties */
void case16_printGraphProperties_null_graph();

/* ############## Valid tests ############## */
void case17_initContext_destroyContext();

void case18_isReflexive_true();

void case19_isSymetric_true();

void case20_isAsymetric_true();

void case21_isTransitive_true();

void case22_isReflexive_false();

void case23_isSymetric_false();

void case24_isAsymetric_false();

void case25_isTransitive_false();

void case26_invertGraph();

void case27_printGraphProperties();

#endif /* DG ADT ALG TESTS H  */
```

5.2.4 dg_adt_alg_tests.c

Folgend ist der Source der Tests für die Graphenalgorithmen angeführt.

Diese Datei befindet sich im Verzeichnis GraphTests.

```
/*
 * dg_adt_alg_tests.c
 *
 *  This source tests the source dg adt alg.c and does not depend on the concrete
 *  implementation of dg adt.h.
 *  Therefore that the function names are self speaking no further documentation will
 *  be provided.
 *
 *  Created on: Oct 19, 2014
 *      Author: cchet
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "dg adt alg tests.h"
#include "dg_adt_tests_depend.h"
#include "dg_adt_alg.h"
#include "dg_adt.h"

static const int SIZE = 5;

int main(int argc, char** argv) {
    int value;
    if (argc != 2) {
        printf("Exactly one argument must be given !!! argc=%d !!!\n", argc);
        exit(EXIT_FAILURE);
    }
    value = atoi(argv[1]);
    switch (value) {
    case 1:
        case1_initContext_null_graph();
        break;
    case 2:
        case2_initContext_invalid_size1();
        break;
    case 3:
        case3_initContext_invalid_size2();
        break;
    case 4:
        case4_initContext_existing_context();
        break;
    case 5:
        case5_initContext_graph_not_in_conjunction();
        break;
    case 6:
        case6_destroyContext_non_existing_context();
        break;
    case 7:
        case7_isReflexive_no_context();
        break;
    case 8:
        case8_isSymetric_no_context();
        break;
    case 9:
        case9_isAsymetric_no_context();
        break;
    case 10:
        case10_isTransitiv_no_context();
        break;
    case 11:
        case11_invertGraph_null_graph();
        break;
    case 12:
        case12_invertGraph_existing_context();
        break;
    case 13:
        case13_invertGraph_invalid_size1();
        break;
```

```
    case 14:
        case14_invertGraph_invalid_size2();
        break;
    case 15:
        case15 invertGraph invalid size3();
        break;
    case 16:
        case16_printGraphProperties_null_graph();
        break;
    case 17:
        case17_initContext_destroyContext();
        break;
    case 18:
        case18_isReflexive_true();
        break;
    case 19:
        case19_isSymetric_true();
        break;
    case 20:
        case20_isAsymetric_true();
        break;
    case 21:
        case21_isTransitive_true();
        break;
    case 22:
        case22_isReflexive_false();
        break;
    case 23:
        case23_isSymetric_false();
        break;
    case 24:
        case24_isAsymetric_false();
        break;
    case 25:
        case25_isTransitive_false();
        break;
    case 26:
        case26_invertGraph();
        break;
    case 27:
        case27_printGraphProperties();
        break;
    default:
        printf("Unknown test case !!! case=%d", value);
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}

/* Context control  */
void case1_initContext_null_graph() {
    printf("Case 1: initContext on NULL graph\n");
    assert(initContext(NULL, SIZE) == false);
}

void case2_initContext_invalid_size1() {
    Graph* graph;
    printf("Case 2: initContext invalid size(0)\n");
    graph = createTestGraph(SIZE);
    assert(initContext(graph, 0) == false);
}

void case3_initContext_invalid_size2() {
    Graph* graph;
    printf("Case 3: initContext invalid size(-1)\n");
    graph = createTestGraph(SIZE);
    assert(initContext(graph, -1) == false);
}

void case4_initContext_existing_context() {
    Graph* graph;
    printf("Case 4: initContext on existing context\n");
    graph = createTestGraph(SIZE);
    assert(initContext(graph, SIZE) == true);
    assert(initContext(graph, SIZE) == false);
}
```

```
void case5_initContext_graph_not_in_conjunction() {
    Graph* graph;
    printf("Case 5: graph not in conjunction with initialized context\n");
    graph = createTestGraph(SIZE);
    initContext(graph, SIZE - 1);
}

void case6_destroyContext_non_existing_context() {
    printf("Case 6: destroyContext on not existing context\n");
    assert(destroyContext() == false);
}

/* isReflexive */
void case7_isReflexive_no_context() {
    printf("Case 7: isReflexive on not existing context\n");
    assert(isReflexive() == EXIT_FAILURE);
}

/* isSymetric */
void case8_isSymetric_no_context() {
    printf("Case 8: isSymetric on not existing context\n");
    assert(isSymmetric() == EXIT_FAILURE);
}

/* isAsymetric */
void case9_isAsymetric_no_context() {
    printf("Case 9: isAsymetric on not existing context\n");
    assert(isAsymmetric() == EXIT_FAILURE);
}

/* isTransitiv */
void case10_isTransitiv_no_context() {
    printf("Case 10: isTransitiv on not existing context\n");
    assert(isTransitive() == EXIT FAILURE);
}

void case11_invertGraph_null_graph() {
    printf("Case 11: invertGraph on null graph\n");
    assert(invert(NULL, SIZE) == false);
}

void case12_invertGraph_existing_context() {
    Graph* graph;
    printf("Case 12: invertGraph on existing context\n");
    graph = createTestGraph(SIZE);
    initContext(graph, SIZE);
    assert(invert(graph, SIZE) == false);
}

void case13_invertGraph_invalid_size1() {
    Graph* graph;
    printf("Case 13: invertGraph invalid size(0)\n");
    graph = createTestGraph(SIZE);
    assert(invert(graph, 0) == false);
}

void case14_invertGraph_invalid_size2() {
    Graph* graph;
    printf("Case 14: invertGraph invalid size(-1)\n");
    graph = createTestGraph(SIZE);
    assert(invert(graph, -1) == false);
}

void case15_invertGraph_invalid_size3() {
    Graph* graph;
    printf("Case 14: invertGraph size not in conjunction with graph size\n");
    graph = createTestGraph(SIZE);
    invert(graph, SIZE - 1);
}

void case16_printGraphProperties_null_graph() {
    printf("Case 14: printGraphProperties on null graph\n");
    assert(printGraphProperties(NULL) == false);
}

/* ############## Valid tests ############## */
```

```
void case17_initContext_destroyContext() {
    Graph* graph;
    printf("Case 17: init- and destroyContext\n");
    graph = createTestGraph(SIZE);
    assert(initContext(graph, SIZE) == true);
    assert(destroyContext() == true);
    assert(initContext(graph, SIZE) == true);
    assert(destroyContext() == true);
}

void case18_isReflexive_true() {
    Graph* graph;
    printf("Case 18: isReflexive (true)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 1, 1);
    insertTestEdge(graph, 2, 2, 1);
    insertTestEdge(graph, 3, 3, 1);
    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 1, 3, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isReflexive() == true);
    assert(destroyContext() == true);
}

void case19_isSymetric_true() {
    Graph* graph;
    printf("Case 19: isSymetric (true)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 1, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 3, 2, 1);
    insertTestEdge(graph, 1, 3, 1);
    insertTestEdge(graph, 3, 1, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isSymmetric() == true);
    assert(destroyContext() == true);
}

void case20_isAsymetric_true() {
    Graph* graph;
    printf("Case 20: isAsymetric (true)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 3, 4, 1);
    insertTestEdge(graph, 1, 3, 1);
    insertTestEdge(graph, 2, 4, 1);
    insertTestEdge(graph, 1, 4, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isAsymmetric() == true);
    assert(destroyContext() == true);

}

void case21_isTransitive_true() {
    Graph* graph;
    printf("Case 21: isTransitive (true)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 1, 3, 1);
```

```
    insertTestEdge(graph, 3, 4, 1);
    insertTestEdge(graph, 1, 4, 1);
    insertTestEdge(graph, 2, 4, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isTransitive() == true);
    assert(destroyContext() == true);
}

void case22_isReflexive_false() {
    Graph* graph;
    printf("Case 22: isReflexive (false)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 1, 1);
    insertTestEdge(graph, 2, 2, 1);
    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 3, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isReflexive() == false);
    assert(destroyContext() == true);
}

void case23_isSymetric_false() {
    Graph* graph;
    printf("Case 23: isSymetric (false)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 1, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 3, 2, 1);
    insertTestEdge(graph, 1, 3, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isSymmetric() == false);
    assert(destroyContext() == true);
}

void case24_isAsymetric_false() {
    Graph* graph;
    printf("Case 24: isAsymetric (false)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 1, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 3, 2, 1);
    insertTestEdge(graph, 1, 3, 1);
    insertTestEdge(graph, 3, 1, 1);

    assert(printGraph(graph) == true);

    assert(initContext(graph, SIZE) == true);
    assert(isAsymmetric() == false);
    assert(destroyContext() == true);
}

void case25_isTransitive_false() {
    Graph* graph;
    printf("Case 25: isTransitive (false)\n");
    graph = createTestGraph(SIZE);

    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 3, 1);
    insertTestEdge(graph, 3, 4, 1);
    insertTestEdge(graph, 2, 4, 1);

    assert(printGraph(graph) == true);
```

```
    assert(initContext(graph, SIZE) == true);
    assert(isTransitive() == false);
    assert(destroyContext() == true);
}

void case26_invertGraph() {
    Graph* graph;
    Graph* new;
    printf("Case 26: invertGraph\n");

    graph = createTestGraph(SIZE);
    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 1, 2);
    insertTestEdge(graph, 2, 3, 3);
    insertTestEdge(graph, 3, 2, 4);
    insertTestEdge(graph, 1, 3, 5);
    printGraph(graph);

    new = invert(graph, SIZE);
    printGraph(new);
}

void case27_printGraphProperties() {
    Graph* graph;
    printf("Case 26: printGraphProperties\n");

    graph = createTestGraph(SIZE);
    insertTestEdge(graph, 1, 2, 1);
    insertTestEdge(graph, 2, 1, 2);
    insertTestEdge(graph, 2, 3, 3);
    insertTestEdge(graph, 3, 2, 4);
    insertTestEdge(graph, 1, 3, 5);
    printGraph(graph);
    printGraphProperties(graph);
}
```

5.2.5 makefile

Folgend ist die makefile für den Build der Applikation für die Graphenalgorithmen angeführt.

```
COMPILER = gcc
COMPILER_FLAGS = -ansi -g -Og -pedantic -Wall -Wextra
LINKER_FLAGS = -lm
PROG=alg
C_FILES = $(wildcard *.c)
O_FILES = $(C_FILES:.c=.o)

%.o: %.c
    $(COMPILER) $(COMPILER FLAGS) -c $<

$(PROG): $(O_FILES)
    $(COMPILER) -o $@ $(O_FILES) $(LINKER_FLAGS)

clean:
    rm -f $(O_FILES) $(PROG)

depend:
    @cat < /dev/null > makedep
    @for i in $(C FILES); do \
        ($(COMPILER) -MM $$i >> makedep); done
    @echo "/^# BEGIN DEPENDENCIES\$$/+1,\$$d" > edcmds
    @echo "r makedep" >> edcmds
    @echo "wq" >> edcmds
    @cp makefile makefile.bak
    @ed -v - makefile < edcmds
    @rm edcmds makedep
    @echo "# LINE REQUIRED FOR MAKEDEPEND" >> makefile

# BEGIN DEPENDENCIES
dg_adt_alg.o: dg_adt_alg.c dg_adt_alg.h dg_adt.h common.h
dg_adt_alg_tests.o: dg_adt_alg_tests.c dg_adt_alg_tests.h dg_adt.h \
 common.h dg adt tests depend.h dg adt alg.h
dg_adt_m.o: dg_adt_m.c dg_adt_m.h dg_adt.h common.h
# LINE REQUIRED FOR MAKEDEPEND
```

5.2.6 alg_tests.sh

Folgend ist das Schell Skript angeführt, welches die Graphalgorithmen Tests ausführt.

Diese Datei befindet sich im Verzeichnis GraphTests.

```
#!/bin/sh
TEST_FOLDER=tests
i=1
TEST_CASES=27

echo "creating test folder: $TEST_FOLDER"

mkdir $TEST_FOLDER

echo "test folder created"
echo "\ncopying test resources to test folder"
echo "cp dg_adt_alg_tests.c ../$1/*.c ../$1/*.h ../GraphAlg/makefile ../GraphAlg/*.h
../GraphAlg/*.c ../Graph/*.h   $TEST_FOLDER"

cp dg_adt_alg_tests.c ../$1/*.c ../$1/*.h ../GraphAlg/makefile ../GraphAlg/*.h ../GraphAlg/*.c
../Graph/*.h  $TEST_FOLDER
cd $TEST_FOLDER

echo "\nin directory: $(pwd)"
echo "build test source"
echo "make clean depend $2"

make clean depend $2

echo "\n\n################################################"
echo "performing tests"
echo "################################################"
while [ $i -le $TEST_CASES ]
do
    echo "-------------------------------"
    echo "Invoking test case with id $i: "
    echo "-------------------------------"
    ./$2 $i
    echo
    i=`expr $i + 1`
done

echo "\n\nremoving test resources"
cd ..
rm -rf $TEST_FOLDER
echo "test finished"
echo "in directory: $(pwd)"
```

5.3 Tests

Folgend sind dies Test für die Graphalgorithmen angeführt.

Die Testskripten befinden sich im Ordner GraphTest

`./alg_tests.sh GraphList alg`

List

```
------------------------------
Invoking test case with id 1:
------------------------------
Case 1: initContext on NULL graph
Cannot initialize context on null graph !!!

------------------------------
Invoking test case with id 2:
------------------------------
Case 2: initContext invalid size(0)
Invalid size for graph !!!

------------------------------
Invoking test case with id 3:
------------------------------
Case 3: initContext invalid size(-1)
Invalid size for graph !!!

------------------------------
Invoking test case with id 4:
------------------------------
Case 4: initContext on existing context
Context already active !!!

------------------------------
Invoking test case with id 5:
------------------------------
Case 5: graph not in conjunction with initialized context

------------------------------
Invoking test case with id 6:
------------------------------
Case 6: destroyContext on not existing context
Context not active !!!

------------------------------
Invoking test case with id 7:
------------------------------
Case 7: isReflexive on not existing context
Context not active !!!

------------------------------
Invoking test case with id 8:
------------------------------
Case 8: isSymetric on not existing context
Context not active !!!

------------------------------
Invoking test case with id 9:
------------------------------
Case 9: isAsymetric on not existing context
Context not active !!!

------------------------------
Invoking test case with id 10:
------------------------------
Case 10: isTransitiv on not existing context
Context not active !!!
```

`./alg_tests.sh GraphMatrix alg`

Matrix

```
------------------------------
Invoking test case with id 1:
------------------------------
Case 1: initContext on NULL graph
Cannot initialize context on null graph !!!

------------------------------
Invoking test case with id 2:
------------------------------
Case 2: initContext invalid size(0)
Invalid size for graph !!!

------------------------------
Invoking test case with id 3:
------------------------------
Case 3: initContext invalid size(-1)
Invalid size for graph !!!

------------------------------
Invoking test case with id 4:
------------------------------
Case 4: initContext on existing context
Context already active !!!

------------------------------
Invoking test case with id 5:
------------------------------
Case 5: graph not in conjunction with initialized context

------------------------------
Invoking test case with id 6:
------------------------------
Case 6: destroyContext on not existing context
Context not active !!!

------------------------------
Invoking test case with id 7:
------------------------------
Case 7: isReflexive on not existing context
Context not active !!!

------------------------------
Invoking test case with id 8:
------------------------------
Case 8: isSymetric on not existing context
Context not active !!!

------------------------------
Invoking test case with id 9:
------------------------------
Case 9: isAsymetric on not existing context
Context not active !!!

------------------------------
Invoking test case with id 10:
------------------------------
Case 10: isTransitiv on not existing context
Context not active !!!
```

```
--------------------------------
Invoking test case with id 11:
--------------------------------
Case 11: invertGraph on null graph
Cannot initialize context on null graph !!!
size must in range 0 < size < MAX_INT !!! size:0 !!!
Segmentation fault (core dumped)

--------------------------------
Invoking test case with id 12:
--------------------------------
Case 12: invertGraph on existing context
Context already active !!!

--------------------------------
Invoking test case with id 13:
--------------------------------
Case 13: invertGraph invalid size(0)
Invalid size for graph !!!
size must in range 0 < size < MAX_INT !!! size:0 !!!
Context not active !!!

--------------------------------
Invoking test case with id 14:
--------------------------------
Case 14: invertGraph invalid size(-1)
Invalid size for graph !!!
size must in range 0 < size < MAX_INT !!! size:0 !!!
Context not active !!!

--------------------------------
Invoking test case with id 15:
--------------------------------
Case 14: invertGraph size not in conjunction with graph size

--------------------------------
Invoking test case with id 16:
--------------------------------
Case 14: printGraphProperties on null graph
Cannot print graph properties of null graph !!!
```

```
--------------------------------
Invoking test case with id 11:
--------------------------------
Case 11: invertGraph on null graph
Cannot initialize context on null graph !!!
size must in range 0 < size < MAX_INT !!! size:0
Segmentation fault (core dumped)

--------------------------------
Invoking test case with id 12:
--------------------------------
Case 12: invertGraph on existing context
Context already active !!!

--------------------------------
Invoking test case with id 13:
--------------------------------
Case 13: invertGraph invalid size(0)
Invalid size for graph !!!
size must in range 0 < size < MAX_INT !!! size:0
Context not active !!!

--------------------------------
Invoking test case with id 14:
--------------------------------
Case 14: invertGraph invalid size(-1)
Invalid size for graph !!!
size must in range 0 < size < MAX_INT !!! size:0
Context not active !!!

--------------------------------
Invoking test case with id 15:
--------------------------------
Case 14: invertGraph size not in conjunction with graph size

--------------------------------
Invoking test case with id 16:
--------------------------------
Case 14: printGraphProperties on null graph
Cannot print graph properties of null graph !!!
```

Diese Test zeigen, dass sich die beiden Implementierungen bei der Fehlerbehandlung gleich verhalten, wobei einzelne Nachrichten sich unterscheiden können.

List

```
--------------------------------
Invoking test case with id 17:
--------------------------------
Case 17: init- and destroyContext

--------------------------------
Invoking test case with id 18:
--------------------------------
Case 18: isReflexive (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00      1.00      1.00       -         -
 2 |    -        1.00      1.00       -         -
 3 |    -         -        1.00       -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 19:
--------------------------------
Case 19: isSymetric (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00       -         -
 2 |   1.00       -        1.00       -         -
 3 |   1.00      1.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 20:
--------------------------------
Case 20: isAsymetric (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00      1.00       -
 2 |    -         -        1.00      1.00       -
 3 |    -         -         -        1.00       -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 21:
--------------------------------
Case 21: isTransitive (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00      1.00       -
 2 |    -         -        1.00      1.00       -
 3 |    -         -         -        1.00       -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
```

Matrix

```
--------------------------------
Invoking test case with id 17:
--------------------------------
Case 17: init- and destroyContext

--------------------------------
Invoking test case with id 18:
--------------------------------
Case 18: isReflexive (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00      1.00      1.00       -         -
 2 |    -        1.00      1.00       -         -
 3 |    -         -        1.00       -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 19:
--------------------------------
Case 19: isSymetric (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00       -         -
 2 |   1.00       -        1.00       -         -
 3 |   1.00      1.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 20:
--------------------------------
Case 20: isAsymetric (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00      1.00       -
 2 |    -         -        1.00      1.00       -
 3 |    -         -         -        1.00       -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 21:
--------------------------------
Case 21: isTransitive (true)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00      1.00       -
 2 |    -         -        1.00      1.00       -
 3 |    -         -         -        1.00       -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
```

```
--------------------------------
Invoking test case with id 22:
--------------------------------
Case 22: isReflexive (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00      1.00       -         -         -
 2 |    -        1.00      1.00       -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 23:
--------------------------------
Case 23: isSymetric (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00       -         -
 2 |   1.00       -        1.00       -         -
 3 |    -        1.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 24:
--------------------------------
Case 24: isAsymetric (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00       -         -
 2 |   1.00       -        1.00       -         -
 3 |   1.00      1.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 25:
--------------------------------
Case 25: isTransitive (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00       -         -         -
 2 |    -         -        1.00      1.00       -
 3 |    -         -         -        1.00       -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 26:
--------------------------------
Case 26: invertGraph
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      5.00       -         -
 2 |   2.00       -        3.00       -         -
 3 |    -        4.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        2.00       -         -         -
 2 |   1.00       -        4.00       -         -
 3 |   5.00      3.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 27:
--------------------------------
Case 26: printGraphProperties
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      5.00       -         -
 2 |   2.00       -        3.00       -         -
 3 |    -        4.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
source=1 -> target=2 (1.00)
source=1 -> target=3 (5.00)
source=2 -> target=1 (2.00)
source=2 -> target=3 (3.00)
source=3 -> target=2 (4.00)
```

```
--------------------------------
Invoking test case with id 22:
--------------------------------
Case 22: isReflexive (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |   1.00      1.00       -         -         -
 2 |    -        1.00      1.00       -         -
 3 |    -         -         -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 23:
--------------------------------
Case 23: isSymetric (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00       -         -
 2 |   1.00       -        1.00       -         -
 3 |    -        1.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 24:
--------------------------------
Case 24: isAsymetric (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      1.00       -         -
 2 |   1.00       -        1.00       -         -
 3 |   1.00      1.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 25:
--------------------------------
Case 25: isTransitive (false)
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00       -         -         -
 2 |    -         -        1.00      1.00       -
 3 |    -         -         -        1.00       -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 26:
--------------------------------
Case 26: invertGraph
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      5.00       -         -
 2 |   2.00       -        3.00       -         -
 3 |    -        4.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        2.00       -         -         -
 2 |   1.00       -        4.00       -         -
 3 |   5.00      3.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -

--------------------------------
Invoking test case with id 27:
--------------------------------
Case 26: printGraphProperties
       1    |    2    |    3    |    4    |    5    |
-----------------------------------------------------
 1 |    -        1.00      5.00       -         -
 2 |   2.00       -        3.00       -         -
 3 |    -        4.00       -         -         -
 4 |    -         -         -         -         -
 5 |    -         -         -         -         -
source=1 -> target=2 (1.00)
source=1 -> target=3 (5.00)
source=2 -> target=1 (2.00)
source=2 -> target=3 (3.00)
source=3 -> target=2 (4.00)
```

Diese Test zeigen dass sich beide Implementierungen gleich verhalten.