**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.

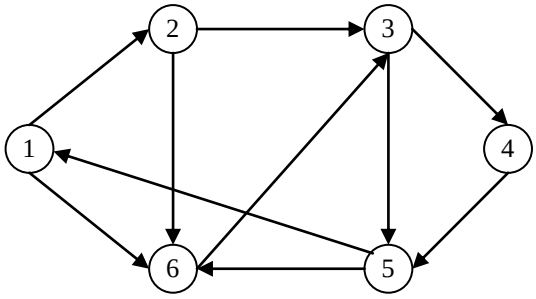
Die Mindestfunktionalität umfasst in beiden Fällen

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.