Abgabetermin: Sa in der KW 50

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

☐ **Gr. 2,** Dr. Erik Pitzer

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

Graphen, die Zweite (14 + 6 + 4 Punkte)

a) Die von Ihnen in der Übung 3 realisierte Implementierung einer abstrakten Datenstruktur (ADS) für die Darstellung von Graphen durch eine Adjazenzmatrix soll als Ausgangspunkt für eine objektorientierte Implementierung dienen, die gewichtete Graphen repräsentieren kann.

Gute Kandidaten für die zu implementierenden Klassen sind *Vertex* (für die Knoten) und *Graph* (für die Darstellung von gewichteten gerichteten Graphen in Form der Adjazenzmatrix).

Objekte der Klasse *Vertex* sind benannte Knoten, wobei deren Namen beliebige Zeichenketten sein können.

Objekte der Klasse *Graph* verwalten alle relevanten Informationen für einen Graphen (mit einer maximalen Anzahl *max* von Knoten). Die Klasse *Graph* sollte daher mindestens folgende Funktionalität (in Form von Methoden) aufweisen:

```
void addVertex(Vertex *v); // reports an error if more than max vertices
void addEdge(const Vertex *start, const Vertex *end, double weight);
  // reports an error if vertices not previously added or weight <= 0
```

Darüber hinaus, soll noch der Ausgabeoperator überladen werden um Graphen auszugeben, sowie Methoden um entweder nur die Kanten, oder sowohl alle Knoten als auch alle Kanten zu löschen.

Fügen Sie Ihren Klassen je nach Bedarf weitere notwendige Methoden und/oder Datenkomponenten hinzu und testen Sie Ihre Implementierung ausführlich.

b) So wie es auf Bäumen verschiedene Durchlaufstrategien gibt (*in*, *pre* oder *post order*) gibt es auch auf Graphen zwei Strategien, alle Knoten in einer definierten Reihenfolge zu besuchen, die Tiefen- und die Breiten-„Suche“. Machen Sie sich dazu in der Algorithmenliteratur schlau und implementieren Sie beide Strategien in Form von Methoden:

```
void printDepthFirst(const Vertex *start) const;
void printBreadthFirst(const Vertex *start) const;
```

Fügen Sie Ihren Klassen je nach Bedarf weitere notwendige Methoden und/oder Datenkomponenten hinzu und testen Sie Ihre Implementierung ausführlich.

c) Abschließen entwickeln Sie noch eine Methode um für einen beliebigen Graphen zu prüfen, ob er Zyklen enthält. Ob es also mindestens einen Konten gibt, von dem aus es möglich ist, über eine Folge von Kanten, wieder zum Ausganspunkt zurück zu kehren:

```
bool hasCycles() const;
```