# 5 Praktikum: Graphen

In dieser Aufgabe ist erneut Ihre Kompetenz als Systemarchitekt bei „Data Fuse“ gefragt. Die verschiedenen Standorte sollen mit neuen Routern verbunden werden und Sie sollen testen, ob die geplanten Router und deren Verbindungen ausreichen um alle Standorte zu verbinden. Um dies zu lösen, nutzen Sie Algorithmen der Graphentheorie.

In Abbildung 1 sehen Sie ein Beispiel, wie so ein Graph aufgebaut sein könnte, die Knoten stellen die Router an den Standorten dar, die Kanten die Verbindungen mit einem Gewicht als fiktive Entfernung. Hier lässt sich schnell erkennen, dass alle Standorte verbunden sind, mit der minimalen Entfernung wird es auch hier auf den ersten Blick schon schwieriger.

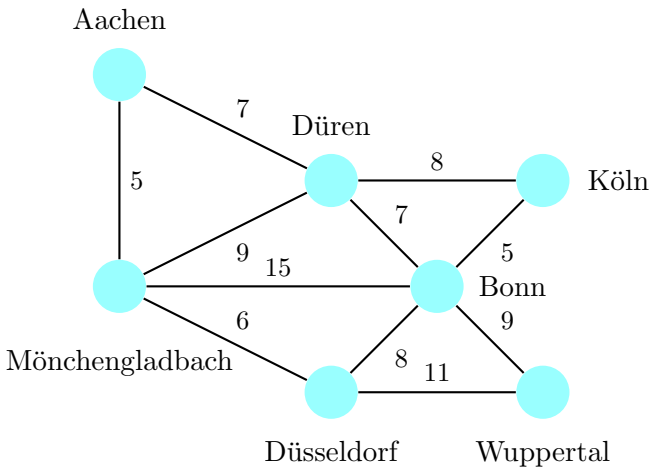


Abbildung 1: Beispiel Unternehmens Standorte mit Verbindungen

## 5.1 Aufgabenstellung

1. Implementieren Sie ein Programm, dass zur Verarbeitung von vorgegebenen Graphen verwendet werden soll. Legen Sie dazu eine Graphen- und Knotenklassean, die mindestens die Attribute und Methoden aus Abbildung 2. Folgende Algorithmen sollen Sie hier selber implementieren:

   - Ausgabe des Graphen
   - Rekursive Tiefensuche
   - Iterative Breitensuche
   - Prim
   - Kruskal

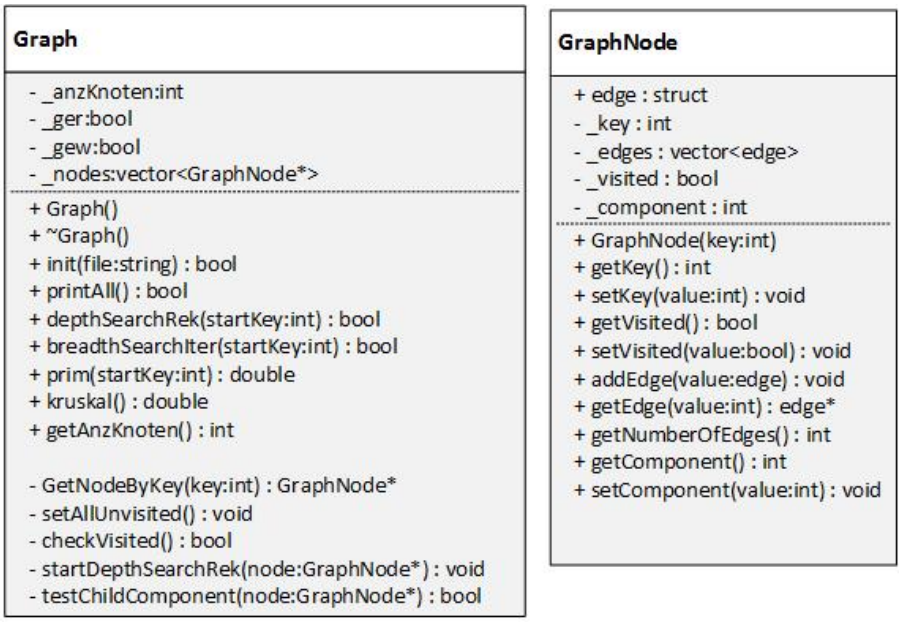


Abbildung 2: Graphen- und Knotenklasse

Es dürfen weitere Attribute und Methoden hinzugefügt werden und auch Übergabeparameter verändert werden. Begründen Sie diese Änderungen bei der Abgabe.

2. Implementieren Sie die Print-Methode als Adjazenzliste wie im Beispiel unten gezeigt.

```
0  -> 1 [7]  -> 3 [5]
1  -> 0 [7]  -> 2 [8]  -> 3 [9]  -> 4 [7]
2  -> 1 [8]  -> 4 [5]
3  -> 0 [5]  -> 1 [9]  -> 4 [15]  -> 5 [6]
4  -> 1 [7]  -> 2 [5]  -> 3 [15]  -> 5 [8]  -> 6 [9]
5  -> 3 [6]  -> 4 [8]  -> 6 [11]
6  -> 4 [9]  -> 5 [11]
```

3. Implementieren Sie wie schon in den letzten Aufgaben ein Menü zur Auswahl für den Bentzer wie folgt:

```
Working on Grphs. Please choose:
1) Graph einlesen
2) Tiefensuche
3) Breitensuche
4) Prim
5) Kruskal
6) Print Graph
0 zum beenden
?>
```

4. Überprüfen Sie mittels Tiefen- oder Breitensuche, ob alle Knoten verbunden sind, also alle Standorte miteinander Kommunizieren könnten.

   Implementieren Sie dazu die rekursive Tiefensuche und die iterative Breitensuche. Als Ergebnis soll true oder false zurück gegeben werde. Bei true konnten alle Knoten erreicht werden, bei false gibt es nach dem Durchlauf noch unbesuchte Knoten.

   **Hinweis:** Nutzen Sie das „besucht“-Flag der Knotenklasse.

5. Berechnen Sie mittels Prim und Kruskal die Kosten des minimalen Spannbaums. Geben Sie die Gesamtkosten der beiden Algorithmen aus und vergleichen Sie diese. Prüfen Sie durch zeichnen der Bäume, ob ihr Ergebnis stimmt (`https://graphonline.ru/en/`).

## 5.2 Lösungshinweise

In den nächsten Abschnitten folgen eine Hilfe zum Einlesen einer Textdatei, sowie ein paar allgemeine Hinweise zu den Algorithmen, die Ihnen die Programmierung erleichtern sollen. Beachten Sie, dass hier nur Hinweise gegeben werden. Gegebene Beispiele müssen erweitert oder überarbeitet und können nicht so übernommen werden.

### 5.2.1 Beispielgraphen

Als Beispielgraph stehen ihnen drei Graphen Dateien als „.txt “ zur Verfügung. Sie finden den Graph dort als Kantenliste aufgebaut. Die erste Zeile gibt die Knotenzahl $V$ an, alle folgenden Zeilen die Kanten in der Form: Startknoten $\rightarrow$ Endknoten $\rightarrow$ Gewicht, wobei die Knoten mit $0 \ldots (V-1)$ nummeriert sind. Die Graphen sind ungerichtet und gewichtet.

### 5.2.2 Einlesen einer Textdatei

Hier finden Sie Beispielcode zum einlesen einer Datei.

```
1  bool Graph::init(std::string path){
2    std::ifstream file;
3    file.open(filename, std::ios_base::in);
4
5    if (!file){
6      std::cout << "Cannot open file." << std::endl;
7      return false;
8    }
9    else{
10     std::string line;
11     std::getline(file, line);
12     std::istringstream iss(line);
```

```
13      iss >> _anzKnoten; //Erste Zeile Auslesen
14
15      //Alle Knoten anlegen
16      for(int i = 0; i < _anzKnoten; i++){
17        //TODO
18      }
19      //Alle Kanten anlegen
20      while (std::getline(file, line)){
21        //TODO
22      }
23      return true;
24    }
25    return false;
26  }
```

### 5.2.3 Graph- und Knoten-Klasse

Ein Knoten besteht wie bei den Listen und Bäumen aus einem Wert als „Key“ und bekommt nun noch zusätzlich Attribute wie z.B. „besucht“ und „component“. Wozu Sie diese verwenden können, wird aus dem Kontext klar..

Der Graph ist grundsätzlich aufgebaut wie ein Baum in Aufgabe 2. Zusätzlich werden aber mehr als 2 Nachbarn zugelassen und ein Knoten kann auch eine Verbindung wieder zum Elternknoten haben. Da es sich in diesen Aufgaben um ungerichtete Graphen handelt, wird jede Kante als Hin- und Rückkante angelegt. Jeder Knoten enthält daher eine Liste von Kanten. Eine Kante besteht dabei aus einem Zielknoten und einem Gewicht.

### 5.2.4 Hinweise Unittests

In dieser Aufgabe helfen Ihnen die Unittests wieder bei der Lösung. Für Tiefen- und Breitensuche wird erwartet, dass ihr Programm 0 liefert, wenn von dem Startknoten nicht alle Knoten erreicht werden konnten. Es soll 1 liefern wenn alle Knoten besucht wurden.

Bei Prim und Kruskal werden die Kosten des MST (keine Kanten) geprüft. Für Prim sind diese bei den ersten beiden Graphen mit beliebigen Startknoten immer gleich. Graph 3 ist nicht zusammenhängend und daher sollte Ihr Algorithmus abhängig vom Startknoten verschiedene Ergebnisse liefern. Sorgen Sie dafür, dass Ihr Programm trotzdem zum Ende kommt und die unvollständige Anzahl an Kanten im MST akzeptiert.

Für Kruskal gilt im Prinzip genau das gleiche, nur dass es hier keinen Startknoten gibt. Kruskal liefert auf den ersten beiden Graphen ein eindeutiges Ergebnis. Bei Graph 3, soll er ebenfalls zum Ende kommen und die Summe zurück geben, auch wenn damit nicht alle Knoten verbunden werden konnten.