Graphentheorie

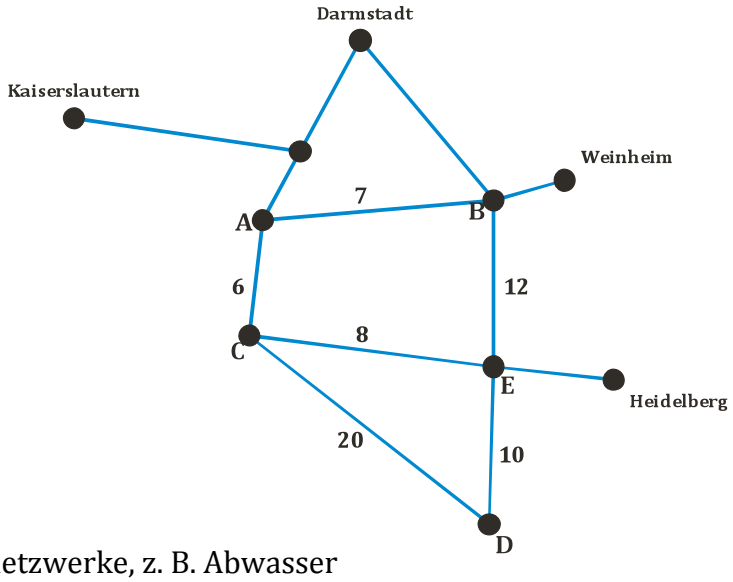
Graphen sind Modelle für Netzwerke.

Hier habe ich eine kleine Übersicht zusammengestellt, worum es bei dem Thema geht. Die Beispiele sind zunächst ganz einfach und klein gewählt. Man kann aber gut sehen, wie schnell die Aufgaben „sehr groß“ werden, wenn ein Netzwerk komplizierter und größer wird.

Beispiel-Anwendungen

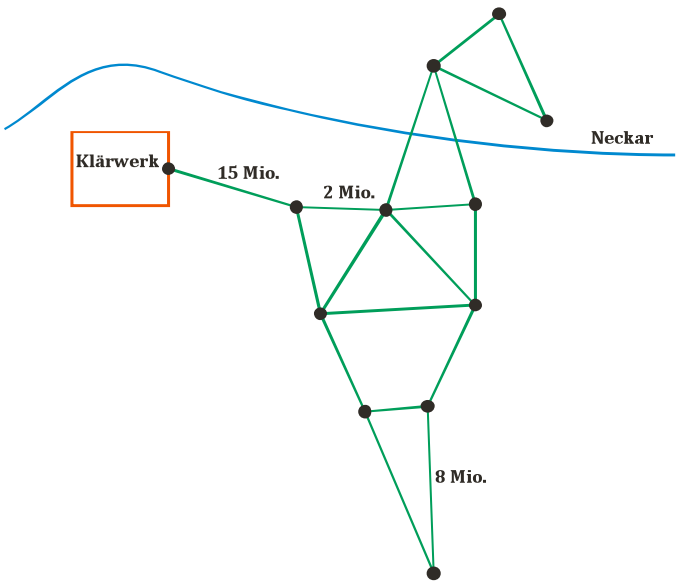
1.) Verkehrsnetzwerk, z. B. Autobahnen

Hier sieht man ein Autobahnnetz. Die Bewertungen der Kanten ist die Zeit in Minuten, die man für die Strecke benötigt. So kann ein Navigationssystem zum Beispiel, die kürzeste Strecke herausfinden und auch die Ankunftszeit bestimmen.



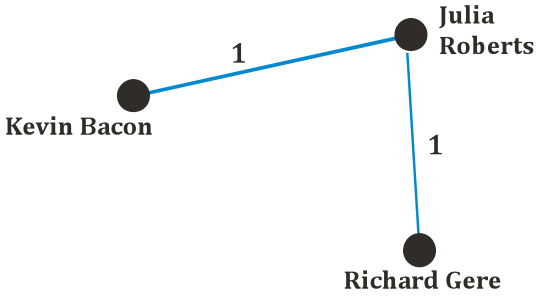
2.) Versorgungsnetzwerke, z. B. Abwasser

Das hier sind die Rohrleitungen der Stadt Heidelberg. Die Bewertung steht für die Kosten, die aufgebracht werden müssen, um das Rohrteilstück auszutauschen.



3.) Soziale Netzwerke

Hier ist dargestellt, wie viele Filme die genannten Schauspieler zusammen gedreht haben. Die Bewertung 1 steht für einen Film.



Darstellung von Graphen

a.) Als Bild
- Knoten
- Kanten
- Richtungen von Kanten
- Bewertungen von Kanten

b.) Als Liste
Knoten A – Knoten B – Bewertung – Richtung

c.) Als Adjazenzmatrix

	A	**B**	**C**	**D**	**E**
A		7	6		
B	7				12
C	6			20	8
D			20		10
E		12	8	10	

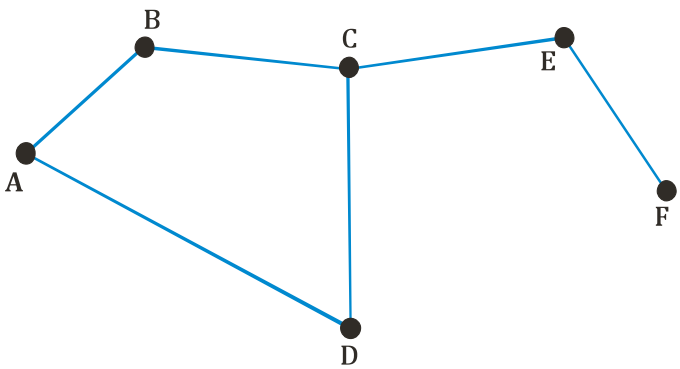
Bei hoher Anzahl an Knoten:
-> sehr groß
-> dünn besetzt (sparse)

Standardaufgaben und Algorithmen

1.) Zusammenhang

Frage: Handelt es sich um ein zusammenhängendes Netzwerk?

	A	**B**	**C**	**D**	**E**	**F**
A		1		1		
B	1		1			
C		1		1	1	
D	1		1			
E			1			1
F					1	



Die Frage lässt sich leicht beantworten, wenn man ein Bild des Netzwerks hat (siehe rechts). Alle Punkte sind hier miteinander verbunden.
Aber was ist, wenn man das Netzwerk nur in Form einer Tabelle (siehe links) oder Liste hat?

Mögliche Lösung

-> Färbealgorithmus

Start: Färbe einen beliebigen Knoten grün.
Regel: Suche einen grünen Knoten und
- Färbe ihn rot
- Färbe alle ungefärbten Nachbarn grün

Ziel: Wiederhole die Regel, bis es keine grünen Knoten mehr gibt.
- Alle rot -> Das Netzwerk ist zusammenhängend!
- Mindestens ein Knoten ungefärbt -> Es hängt nicht zusammen

Wenn man den Färbealgorithmus benutzt, muss man sich bei einer Adjazenzmatrix $n \cdot n$ Felder anschauen (einmal alle Spalten und einmal alle Zeilen).
Hat man jedoch eine Liste, muss man nur $2 \cdot n \cdot v = 2nv$ Listenpunkte anschauen. Faktor 2 ist da, weil man alles doppelt hat (Knoten A – Knoten B / Knoten B – Knoten A), n steht für die Anzahl der Knoten und v ist die größte Anzahl Kanten, die aufeinander treffen.

Beispiel v = 8, n Knoten

Knoten	**Adjazenzmatrix**	**Liste**
n	**n^2**	**$2 \cdot n \cdot v$**
10	100	160
50	2.500	800
200	40.000	3.200
3000	9.000.000	48.000

An der Tabelle kann man ablesen, dass der Algorithmus bei Vorliegen einer Adjazenzmatrix bei steigender Knotenzahl sehr schnell sehr viel länger dauert als bei einer Liste.

2.) Minimal Spanning Tree

Aufgabe: Die Stadt will die Abwasserrohre ausbessern lassen, hat aber nur genug Geld, um die wichtigsten Rohre auszubessern.
Ziel:
- Der neue Graph soll zusammenhängend sein
- Minimale Kosten

Mögliche Lösung

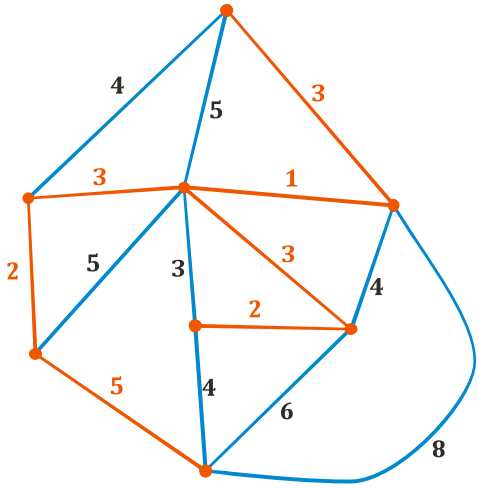
-> Algorithmus von Prim

Start: Färbe einen Knoten rot.

Regel: Suche die billigste Kante von irgendeinem roten zu irgendeinem ungefärbten Knoten

- Wähle die Kante
- Färbe den ungefärbten Knoten rot

Ziel: Wiederhole die Regel, bis alle Knoten rot sind.



3.) Travelling Salesman Problem (TSP)

- Graph mit n Knoten
- Alle Kanten (von jedem Punkt zu jedem anderen Punkt) mit Bewertung
- Gesucht: Die günstigste Route, zum Beispiel zwischen verschiedenen Städten (Günstigster Hamilton-Kreis)

Mögliche Lösungen

a.) Alles Ausprobieren

Bei n = 8
-> #Möglichkeiten: $7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5.040$

-> (n-1)!

! = Fakultät und bedeutet, dass man die Zahl mit allen niedrigeren ganzen Zahlen multipliziert.

Bei n = 11
-> 10! = 3.628.800 Möglichkeiten

Man sieht sehr gut, wie schnell die Anzahl der Möglichkeiten steigt, wenn n sich erhöht. Bei nur 3 Städten mehr auf der Route müssen 3,6 Mio. mehr Möglichkeiten geprüft werden.

b.) Dynamische Programmierung

Bei der dynamischen Programmierung wird ein umfangreiches Problem in viele kleine Teilprobleme zerlegt. Aus allen Teilergebnissen, die man sich merken muss, wird die Gesamtlösung ermittelt. Dafür braucht man nur noch die Größenordnung 2^n Rechenoperationen.

n	**n!**	**2^n**
5	120	32
10	3,6 Mio.	1024
20	10^{21}	1.000.000
50	10^{70}	10^{15}

-> Direkte, exakte Verfahren brauchen exponentiell viel Zeit (nicht nur polynomiell)!

Bsp: Man hat einen superschnellen PC erfunden, der 300 Städte in 1 ms absuchen kann.

Vollständiges Absuchen		Dynamisches Programmieren	
300 Städte	1 ms	300 Städte	1 ms
301 Städte	300 ms	301 Städte	2 ms
302 Städte	90 s	310 Städte	1 s
303 Städte	¾ h	322 Städte	1 h
304 Städte	7 d	327 Städte	1 d
305 Städte	6 y	336 Städte	1 y

-> Bringt einem nicht viel, da ja beim Vollständigen Absuchen bei nur einer weiteren Stadt die Zeit um das 300-fache steigt! Bei 5 Städten mehr braucht auch der superschnelle PC schon 6 Jahre !

-> Der Zeitaufwand bei der dynamischen Programmierung steigt zwar langsamer, aber ist trotzdem noch zu groß, wenn die Anzahl Städte größer wird.

-> Dieses Problem hat bis heute keine Lösung!
Die Frage aller Fragen: Gibt es einen polynomiellen Algorithmus oder gibt es keinen?