Graphen und der Algorithmus von Dijkstra

Def.: Ein *Graph* besteht aus *Ecken* und *Kanten* (engl. *vertices*, sing. *vertex*, and *edges*, sing. *edge*). Jede Kante verbindet zwei (nicht notwendig verschiedene) Ecken.

Ecken werden auch als *Knoten* bezeichnet (aber im Folgenden sind mit k, k1, k2, ... immer **K**anten und mit e, e1, e2, ... immer **E**cken gemeint).

Viele sehr verschiedene konkrete Probleme kann man auf solche abstrakte Graphen abbilden, und deshalb unterscheidet man (in der Informatik) viele verschiedene *Arten* von Graphen, die sich durch folgende Eigenschaften unterscheiden:

Gerichtet/ungerichtet: Bei einem *ungerichteten Graphen* (engl. undirected graph) verbindet eine Kante zwei Ecken e1, e2 ("die Kanten sind einfache Striche"). Bei einem *gerichteten Graphen* führt eine Kante von einer *Anfangsecke* e1 zu einer *Endecke* e2 ("die Kanten sind Pfeile").

Ohne/mit Schleifen: Eine Schleife (engl. a loop) verbindet eine Ecke e mit sich selbst (bzw. führt von einer Ecke e zur selben Ecke e). Ein schleifen-loser *Graph* enthält keine Schleifen.

Ohne/mit Mehrfachkanten: Wenn *mehrere* Kanten dieselben Ecken e1, e2 verbinden (bzw. von einer Anfangsecke e1 zu einer Endecke e2 führen), spricht man von einem *Graphen mit Mehrfachkanten*.

Einfache (oder: schlichte) **Graphen**: Ein *einfacher Graph* (engl. a simple graph) ist ein *ungerichteter* Graph *ohne Schleifen* und *ohne Mehrfachkanten*.

Kantengewichtete Graphen: Jeder *Kante* ist ein *Gewicht* zugeordnet. Das Gewicht ist meistens eine Zahl (eine reelle Zahl, eine ganze Zahl oder eine natürliche Zahl).

Ecken-gewichtete Graphen: Jeder *Ecke* ist ein *Gewicht* zugeordnet. Das Gewicht ist meistens eine Zahl (eine reelle Zahl, eine ganze Zahl oder eine natürliche Zahl).

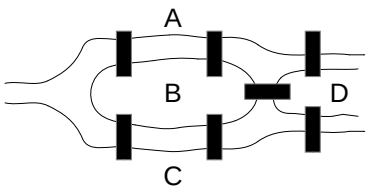
Zusammenhängende/unzusammenhängende Graphen: Ein *ungerichteter* Graph ist *zusammenhängend*, wenn es von jeder Ecke zu jeder anderen Ecke einen Weg (aus Kanten) gibt. Bei *gerichteten* Graphen unterscheidet man *schwach zusammenhängend* (auch Wege *gegen* die Pfeilrichtungen werden als Zusammenhang gezählt) und *stark zusammenhängend* (nur Wege in den Pfeilrichtungen gelten).

Bipartite Graphen: In einem bipartiten Graphen gibt es *zwei Arten* von Ecken und jede Kante verbindet eine Ecke der einen Art mit einer der anderen Art (aber nie zwei Ecken der gleichen Art).

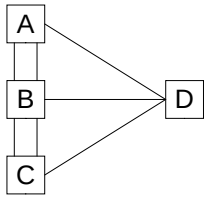
Diese Aufzählung von *Arten von Graphen* ist nicht vollständig.

Beispiel-01: Ein besonders berühmter Graph

Im 18. Jahrhundert wurde das Stadtgebiet von Königsberg (heute: Kaliningrad) durch Flussläufe in 4 Gebiete A, B, C, D geteilt, die durch 7 Brücken miteinander verbunden waren, etwa so wie in dem Stadtplan links angedeutet (die dicken schwarzen Striche sind die Brücken):



Stadtplan von Königsberg



Ein Graph BG

Der Mathematiker **Leonard Euler** (1707-1783) untersuchte folgende Frage: "Gibt es einen Rundweg, der genau einmal über jede der 7 Brücken führt?". Dazu konstruierte er aus dem Stadtplan den Brücken-Graphen BG, bei dem die *Ecken* den Stadtteilen entsprechen und die *Kanten* den Brücken. Die Anzahl der Kanten, die von einer Ecke ausgehen, bezeichnete er als *den Grad der Ecke* (im Graphen BG hat die Ecke B den Grad 5 und alle anderen Ecken haben den den Grad 3) und zeigte ganz allgemein: Einen Rundweg, der genau einmal über jede Kante führt, gibt es genau dann, wenn jede Ecke einen *geraden Grad* hat. Somit gab es in Königsberg keinen "eulerschen Rundweg", aber Euler hatte mit seinen Untersuchungen die Graphentheorie begründet.

Aufgabe-01: Wie viele Kanten *muss* ein einfacher, zusammenhängender Graph mit N Ecken *mindestens* haben?

Aufgabe-02: Wie viele Kanten *kann* ein einfacher Graph mit N Ecken *höchstens* haben?

Implementierungen von Graphen

Die Anzahl der Kanten eines einfachen, zusammenhängen Graphen mit 1000 Ecken kann irgendwo zwischen etwa 1 Tausend und 495 Tausend liegen. Einen Graphen mit 1 Tausend Kanten wird man in aller Regel anders implementieren als einen mit fast 500 mal so vielen Kanten. Deshalb gibt es keine "beste Implementierung" für alle Graphen, sondern verschiedene *Grundideen*, zwischen denen man von Fall zu Fall wählen kann (und die man manchmal noch variieren und anpassen muss).

Grundbegriffe:

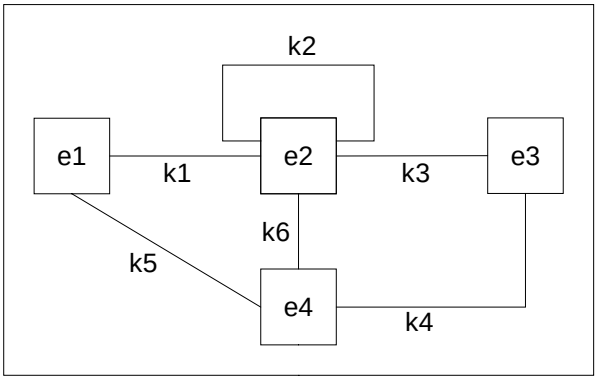
Ein Graph mit M Kanten und N Ecken heißt

dicht (engl. dense) wenn $M \approx (N^2-N)/2$ ("≈" bedeutet hier: "ungefähr gleich groß")
dünn (engl. sparse) wenn $M << N^2$ ("<<" bedeutet hier: "ist klein im Vergleich zu")

Zwei *Ecken* e1, e2 sind *benachbart* (lat./deutsch: adjazent, engl. adjacent), wenn sie durch eine Kante k = (e1, e2) *verbunden* sind.

Eine *Kante* k = (e1, e2) ist *inzident* mit den *Ecken* e1 und e2 (und *nicht-inzident* mit allen anderen Ecken)

Beispiel-02: Ein *ungerichteter* Graph UG1 mit **N** gleich 4 Ecken und **M** gleich 6 Kanten



Grundidee 1: **Nachbarschafts-Matrix** (Adjazenz-Matrix. engl. adjacency matrix)

	e1	**e2**	**e3**	**e4**
e1	f	t	f	t
e2	t	t	t	t
e3	f	t	f	t
e4	t	t	t	f

Eine NxN-Matrix von booleschen Werten (t und f).

In einem ungerichteten Graphen gilt: Wenn e_i mit e_j verbunden (d.h. adjazent) ist, dann ist auch e_j mit e_i verbunden. Deshalb ist eine Nachbarschafts-Matrix (Adjazenz-Matrix) A immer symmetrisch zu ihrer Hauptdiagonalen (an der Stelle $A[e_i, e_j]$ steht immer das Gleiche wie an der Stelle $A[e_j, e_i]$ und deshalb braucht man eigentlich nur "etwa eine Hälfte" der Matrix.

Grundidee 2: **Inzidenz-Matrix** (Ecken-Kanten-Matrix, engl. incidence matrix)

	k1	**k2**	**k3**	**k4**	**k5**	**k6**
e1	t	f	f	f	t	f
e2	t	t	t	f	f	t
e3	f	f	t	t	f	f
e4	f	f	f	t	t	t

Eine NxM-Matrix von booleschen Werten (t und f).

Grundidee 3: **Nachbarschafts-Listen** (Adjazenz-Listen, engl. adjacency lists)

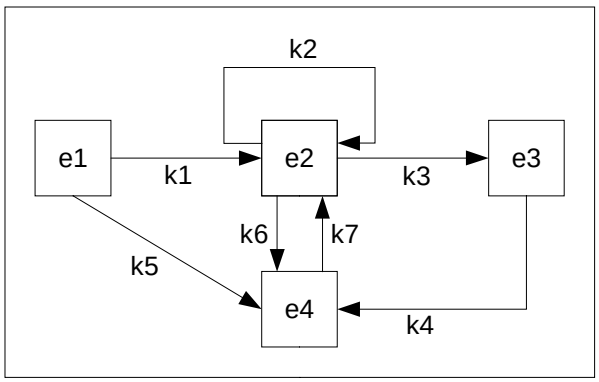
Ecken	**Liste von Nachbarn**
e1	[e2, e4]
e2	[e1, e2, e3, e4]
e3	[e2, e4]
e4	[e1, e2, e3]

Eine Reihung von `N` Listen. Bei einem ungerichteten Graphen mit `M` Kanten und `S` Schleifen enthalten diese Listen zusammen genau `2*M-S` Komponenten (für jede Schleife eine und für jede "normale" Kante zwei). Insgesamt besteht die Darstellung also aus `N + 2*M-S` Komponenten (`N` Referenzen auf Listen und in den Listen `2*M-S` Referenzen auf Ecken).

Aufgabe-03: Angenommen, ein einfacher Graph EG3 hat 100 Ecken und 200 Kanten.
Wie viele *Komponenten* ("Einträge in den Matrizen bzw. Reihungen und Listen") haben dann die drei Implementierungen (Nachbarschafts-Matrix, Inzidenz-Matrix und Nachbarschafts-Listen)?

Aufgabe-04: Angenommen, ein einfacher Graph EG4 hat 100 Ecken und 4000 Kanten.
Wie viele *Komponenten* ("Einträge in den Matrizen bzw. Reihungen und Listen") haben dann die drei Implementierungen (Nachbarschafts-Matrix, Inzidenz-Matrix und Nachbarschafts-Listen)?

Beispiel-02: Ein *gerichteter* Graph GG1 mit N gleich 4 Ecken und M gleich 7 Kanten:



Aufgabe-05: Geben Sie für GG1 die Nachbarschafts-Matrix an.

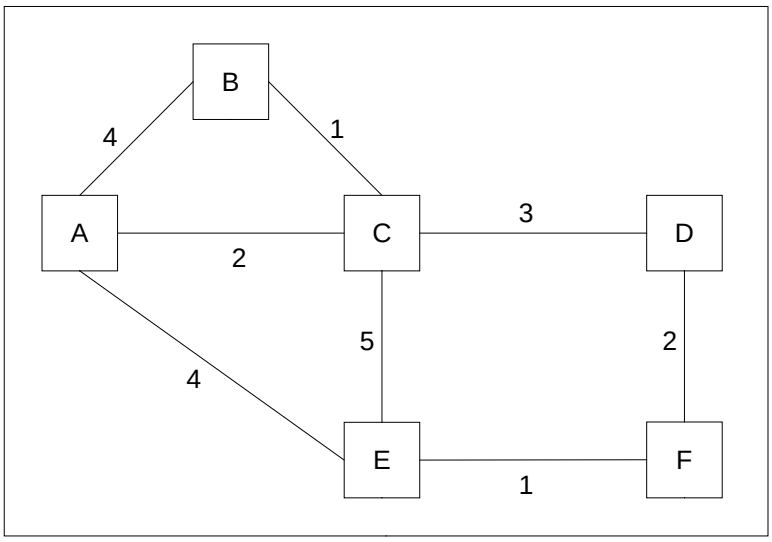
Aufgabe-06: Wie könnte eine Inzidenz-Matrix für GG1 aussehen? Seien Sie kreativ!

Aufgabe-07: Geben Sie für GG1 die Reihung der Nachbarschafts-Listen an.

Algorithmus von Dijkstra

Mit dem **Dijkstra Algorithmus** kann man
in einem (ungerichteten oder gerichteten) Graphen mit
nicht-negativen Kantengewichten und
ohne Mehrfachkanten
kürzeste Wege von *einer bestimmten* Ecke zu *jeder anderen* Ecke berechnen.

Beispiel-01: Berechnet werden sollen kürzeste Wege von der Ecke A zu den Ecken B, C, D, E, F.



Für die Berechnung der kürzesten Wege von Hand wurden verschiedene Formen von Tabellen entwickelt, unter anderem die hier verwendete Form. Anfangs sieht eine solche Tabelle etwa so aus:

besucht	**Ecken**								**Min**
	A	0							
	B	inf							
	C	inf							
	D	inf							
	E	inf							
	F	inf							

In der 2. Zeile bedeutet die **0**, dass ein Weg der Länge 0 von A nach A führt. Das **inf** (wie infinity) neben **B** bedeutet, dass zur Zeit der kürzeste uns bekannte Weg von A nach B unendlich lang ist, d.h. dass wir noch keinen Weg von A nach B gefunden haben. Für die anderen Ecken C bis F entsprechend.

Wir "besuchen" die Ecken in einer bestimmten Reihenfolge (jede Ecke höchstens einmal) und tragen in der Spalte **besucht** ein, als wievielte wir die betreffende Ecke besucht haben. Ecken mit einer Zahl in der **besucht**-Spalte sind also bereits *besucht*, solche ohne Zahl sind noch *unbesucht*. Anfangs sind alle Ecken unbesucht.

Schritt 1a: Wir suchen unter den *unbesuchten* Ecken die, zu der wir den kürzesten Weg von A aus kennen, markieren sie als besucht, und schreiben sie (und die Länge des Weges zu ihr) als Überschrift über die nächste freie Spalte.

Da **0** (viel) kleiner ist als **inf**, besuchen wir als erstes die Ecke A.
Nach dem **Schritt 1a** sieht die Tabelle so aus:

besucht	**Ecken**		**A 0**						**Min**
1	A	0							
	B	inf							
	C	inf							
	D	inf							
	E	inf							
	F	inf							

Schritt 1b: Wenn von der Ecke A (die wir gerade besuchen) ein Weg zu einer anderen noch *unbesuchten* Ecke X führt, tragen wir in die A-Spalte und X-Zeile die Entfernung zwischen der Ecke A und der Ecke X ein:

besucht	**Ecken**		**A 0**						**Min**
1	A	0							
	B	inf	4						
	C	inf	2						
	D	inf							
	E	inf	4						
	F	inf							

Damit ist der Schritt 1 fertig.

Schritt 2a: Wir suchen unter den *unbesuchten* Ecken die, zu der wir den kürzesten Weg von A aus kennen, markieren sie als besucht, und schreiben sie (und die Länge des Weges zu ihr) als Überschrift über die nächste frei Spalte.

In diesem Beispiel ist das die Ecke C (denn 2 ist kleiner als 4 und kleiner als **inf**, und die Ecke A mit der Entfernung 0 kommt nicht mehr in Frage, weil sie schon *besucht* ist) und die Länge 2.

Schritt 2b: Wenn von der Ecke C (die wir gerade besuchen) ein Weg zu einer anderen noch *unbesuchten* Ecke **X** führt, tragen wir die Entfernung (A-nach-C + C-nach-**X**) in die C-Spalte und **X**-Zeile ein, also die Länge eines Weges von A über C nach **X**.
Beispiel mit **X** gleich B: (A-nach-C) ist gleich 2, (C-nach-B) ist gleich 1, 2 + 1 ist gleich 3.
Also tragen wir 3 in die C-Spalte und B-Zeile ein.

besucht	**Ecken**		**A 0**	**C 2**					**Min**
1	A	0							
	B	inf	4	3					
2	C	inf	2						
	D	inf		5					
	E	inf	4	7#					
	F	inf							

Die 3 in der C-Spalte und B-Zeile bedeutet: Von A nach B gibt es einen Weg der Länge 3, der über C führt. Die 5 bedeutet: Von A nach D gibt es einen Weg der Länge 5, der über C führt.

Es gibt auch einen Weg der Länge 7, der von A über C nach E führt. Da wir aber schon einen kürzeren Weg von A nach E kennen (den Weg der Länge 4, der nur über A läuft), ist die 7 mit einem Nummernzeichen # als *unwirksam* markiert.

Wichtig: Auch die unwirksamen (mit # gekennzeichneten) Zahlen gehören zu einer vollständigen Lösung (sie repräsentieren "verworfene" oder "abgelehnte" Wege).

Nach einigen weiteren Schritten sieht die Tabelle schließlich so aus:

besucht	**Ecken**		**A 0**	**C 2**	**B 3**	**E 4**	**D 5**	**F 5**	**Min**
1	A	0							0
3	B	inf	4	3					3
2	C	inf	2						2
5	D	inf		5					5
4	E	inf	4	7#					4
6	F	inf				5	7#		5

Die Ecke B haben wir als dritte Ecke besucht (nach A und C). Zur Zeit unseres B-Besuchs gab es keinen Weg mehr, der von B zu einer unbesuchten Ecke führte. Deshalb ist die B-Spalte leer geblieben.

Die 7 in der D-Spalte und F-Zeile bedeutet: Von A nach F gibt es einen Weg der Länge 7, der über D führt. Diese 7 ist aber unwirksam, weil (in derselben Zeile) in der E-Spalte schon die kleinere Zahl 5 steht, d.h. weil wir schon einen Weg der Länge 5 von A (über E) nach F kennen.

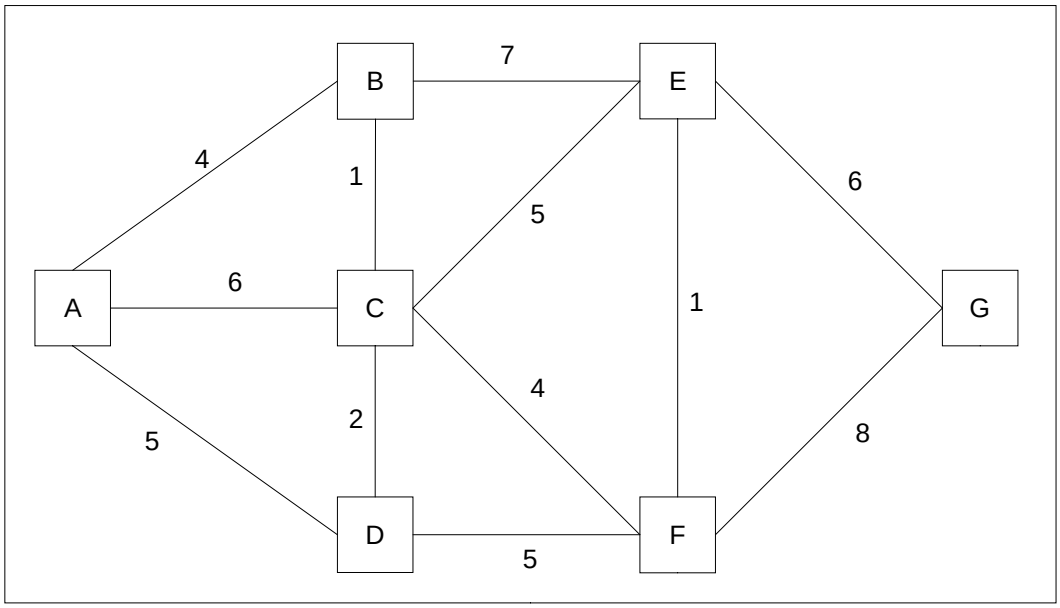
Wenn es am Anfang eines Schrittes zu *mehreren* unbesuchten Ecken kürzeste Wege von A aus gibt, können wir eine beliebige davon *wählen*. Im Beispiel hatten wir beim Schritt 5 die Wahl zwischen den Ecken D und F (zu beiden führt von A aus ein Weg der Länge 5) und haben uns für D entschieden. Hätten wir uns für F entschieden, sähe unsere Tabelle so aus:

besucht	**Ecken**		**A 0**	**C 2**	**B 3**	**E 4**	**F 5**	**D 5**	**Min**
1	A	0							0
3	B	inf	4	3					3
2	C	inf	2						2
6	D	inf		5			7#		5
4	E	inf	4	7#					4
5	F	inf				5			5

Ganz am Schluss können wir als Endergebnis die Längen der kürzesten Wege von A zu den einzelnen Ecken in die Spalte **Min** eintragen. Man sieht: Die **Min**-Spalte ist in beiden Lösungen gleich (egal, ob wir uns im Schritt 5 für D oder für F entscheiden).

Ende von **Beispiel-01**.

Aufgabe-08: Betrachten Sie den folgenden einfachen Graphen:

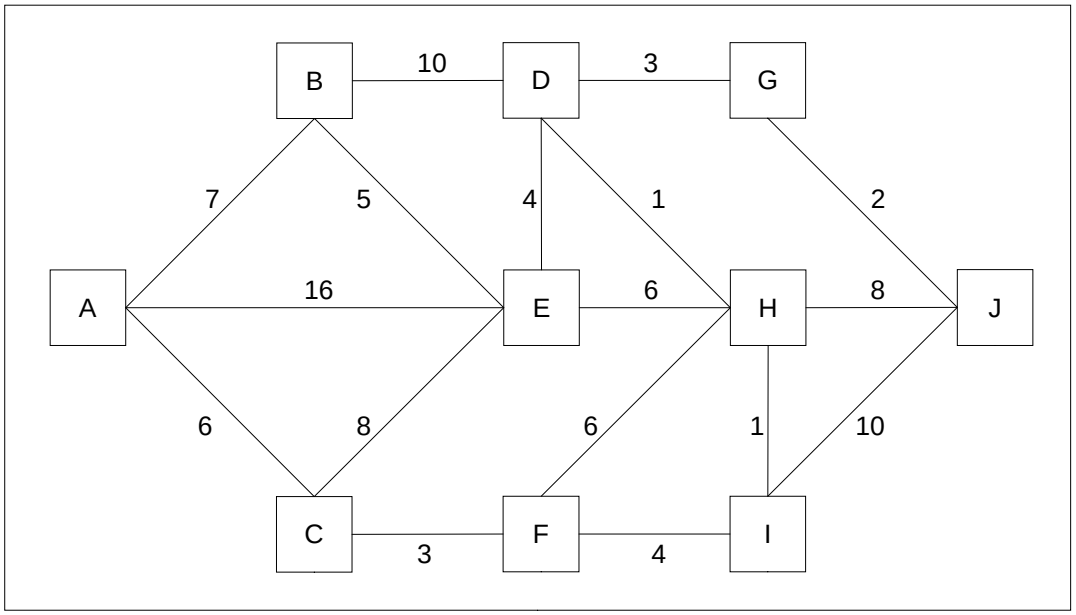


Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie oben im **Beispiel-01** beschrieben.

besucht	**Ecken**									**Min**
	A	0								
	B	inf								
	C	inf								
	D	inf								
	E	inf								
	F	inf								
	G	inf								

Geben Sie dann an, über welche Ecken ein kürzester Weg von A nach G führt.

Aufgabe-09: Betrachten Sie den folgenden Graphen:

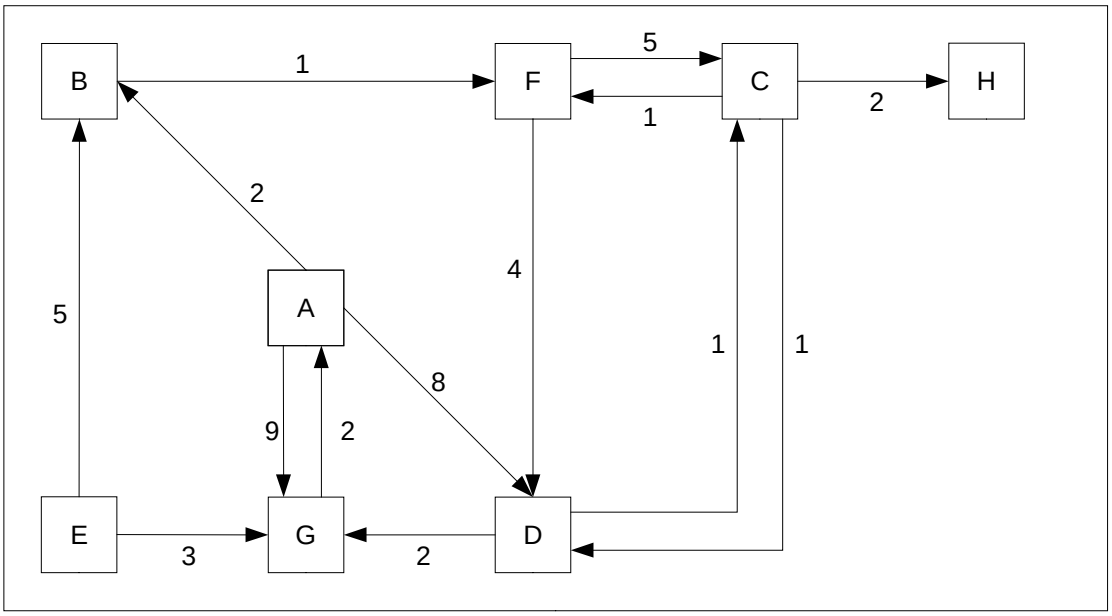


Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie oben im **Beispiel-01** beschrieben. **Achtung**: In der folgenden Tabelle ist *eine Spalte weniger vorgegeben* als in den vorigen Tabellen (aber Sie haben ja auch schon genug Erfahrung, um diese Spalte auszufüllen).

besucht	**Ecken**												**Min**
	A												
	B												
	C												
	D												
	E												
	F												
	G												
	H												
	I												
	J												

Geben Sie dann an, über welche Ecken ein kürzester Weg von A nach J führt.

Aufgabe-10: Ein gerichteter Graph

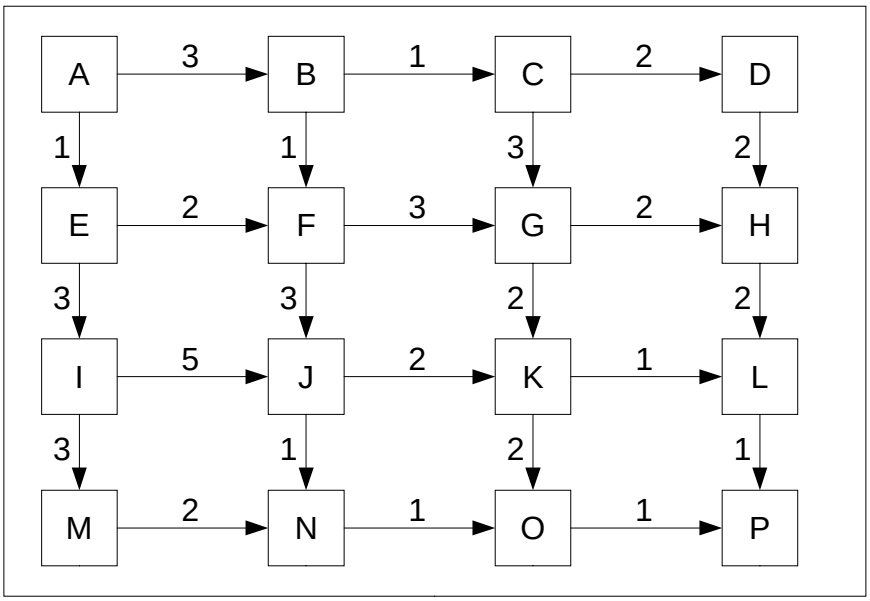


Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie oben im **Beispiel-01** beschrieben.

besucht	**Ecken**										**Min**
	A										
	B										
	C										
	D										
	E										
	F										
	G										
	H										

Geben Sie dann an, über welche Ecken ein kürzester Weg von A nach H führt.

Aufgabe-11: Ein gerichteter Graph:



Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie oben im **Beispiel-01** beschrieben.

be.	**Eck**																		**Min**
	A																		
	B																		
	C																		
	D																		
	E																		
	F																		
	G																		
	H																		
	I																		
	J																		
	K																		
	L																		
	M																		
	N																		
	O																		
	P																		

Geben Sie dann an, über welche Ecken ein kürzester Weg von A nach P führt.

Lösung-01: Ein zusammenhängender, einfacher Graph mit N Ecken hat mindestens **N-1** Kanten.

Lösung-02: Ein einfacher Graph mit N Ecken hat höchstens (N * (N - 1)) / 2 gleich $\mathbf{N^2/2 - N/2}$ Kanten.
Erläuterung: Von jedem der N Ecken gehen N-1 Kanten zu den anderen Ecken aus. Da wir bei dieser Zählung jede Kante *zweimal* zählen, müssen wir das Produkt N * (N - 1) noch durch 2 teilen.

Warnung: Die Formel $\mathbf{N^2/2 - N/2}$ (die in der **Lösung-02** vorkommt) sollte man möglichst selten mit der Formel $\mathbf{N^2/2 + N/2}$ (für die "Grundform des Gauß-Tricks") verwechseln.

Lösung-03: Der einfache Graph EG3 hat 100 Ecken und 200 Kanten

Nachbarschafts-Matrix:	5.050 Komponenten
Inzidenz-Matrix:	20.000 Komponenten
Nachbarschafts-Listen	500 Komponenten

Lösung-04: Der einfache Graph EG4 hat 100 Ecken und 4.000 Kanten

Nachbarschafts-Matrix:	5.050 Komponenten
Inzidenz-Matrix:	400.000 Komponenten
Nachbarschafts-Listen	8.100 Komponenten

Lösung-05: Nachbarschafts-Matrix für GG1

	e1	**e2**	**e3**	**e4**
e1	f	t	f	t
e2	f	t	t	t
e3	f	f	f	t
e4	f	t	f	f

Lösung-06: Wie könnte eine Inzidenz-Matrix für GG1 aussehen?

Entweder eine Matrix mit Komponenten, die 4 Werte haben können: a (wie Anfang), e (wie Ende), b (wie beides, für Schleifen) und leer (für nix):

	k1	**k2**	**k3**	**k4**	**k5**	**k6**	**k7**
e1	a				a		
e2	e	b	a			a	e
e3			e	a			
e4				e	e	e	a

Oder zwei boolean-Matrizen, eine für Kanten-Anfänge und eine für Kanten-Enden (alle nicht mit t gekennzeichneten Komponenten enthalten f):

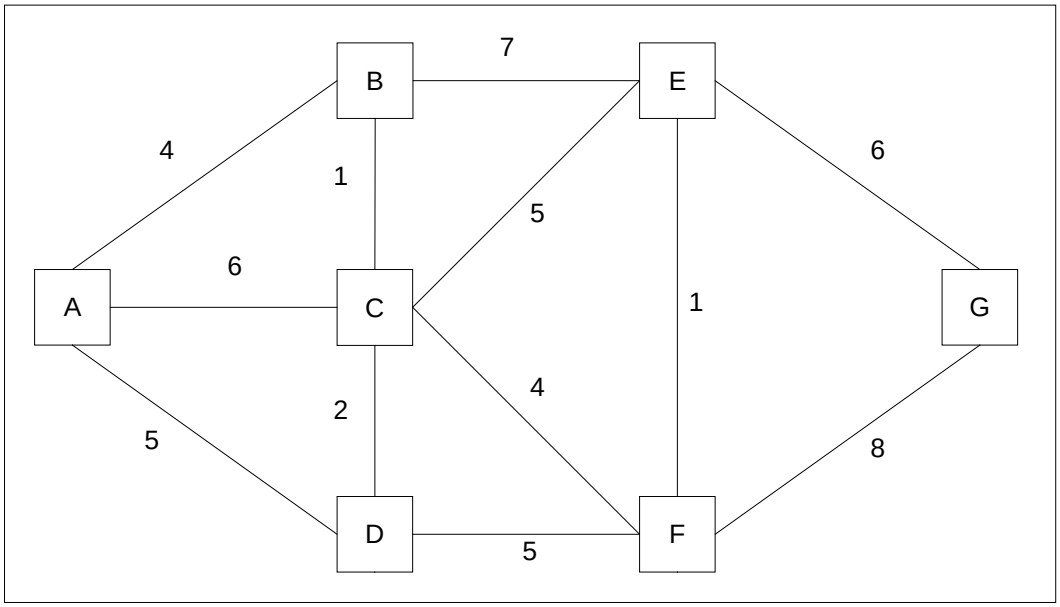
ANF	**k1**	**k2**	**k3**	**k4**	**k5**	**k6**	**k7**
e1	t				t		
e2		t	t			t	
e3				t			
e4							t

END	**k1**	**k2**	**k3**	**k4**	**k5**	**k6**	**k7**
e1							
e2	t	t					t
e3			t				
e4				t	t	t	

Lösung-07: Nachbarschafts-Listen für GG1

Ecken	**Liste von Nachbarn**
e1	[e2, e4]
e2	[e2, e3, e4]
e3	[e4]
e4	[e2]

Lösung-08: Für Aufgabe-08 gibt es *zwei* Lösungen, da man nach der Ecke B wahlweise C oder D besuchen kann.



Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie im Beispiel-01 beschrieben.

Lösung-08-1: Nach A und B wird C besucht

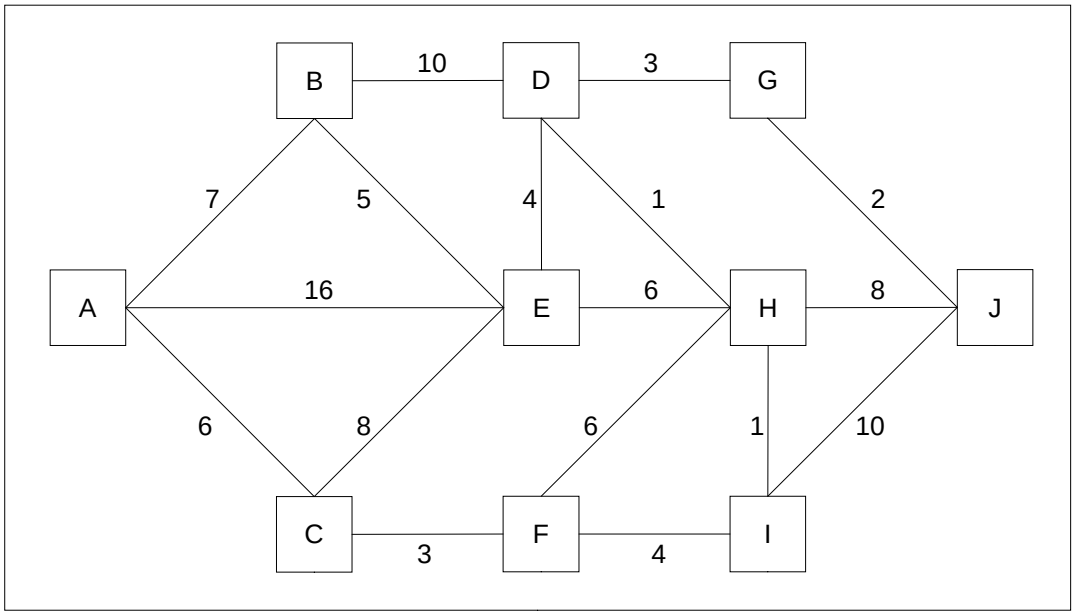
besucht	**Ecken**		**A 0**	**B 4**	**C 5**	**D 5**	**F 9**	**E 10**	**G 16**	**Min**
1	A	0								0
2	B	inf	4							4
3	C	inf	6	5						5
4	D	inf	5		7#					5
6	E	inf		11	10		10#			10
5	F	inf			9	10#				9
7	G	inf					17	16		16

Lösung-08-2: Nach A und B wird D besucht

besucht	**Ecken**		**A 0**	**B 4**	**D 5**	**C 5**	**F 9**	**E 10**	**G 16**	**Min**
1	A	0								0
2	B	inf	4							4
4	C	inf	6	5	7#					5
3	D	inf	5							5
6	E	inf		11		10	10#			10
5	F	inf			10	9				9
7	G	inf					17	16		16

Kürzester Weg von A nach G: `A 4 B 1 C 5 E 6 G`, Länge 4+1+5+6 = 16

Lösung-09: Betrachten Sie den folgenden Graphen:

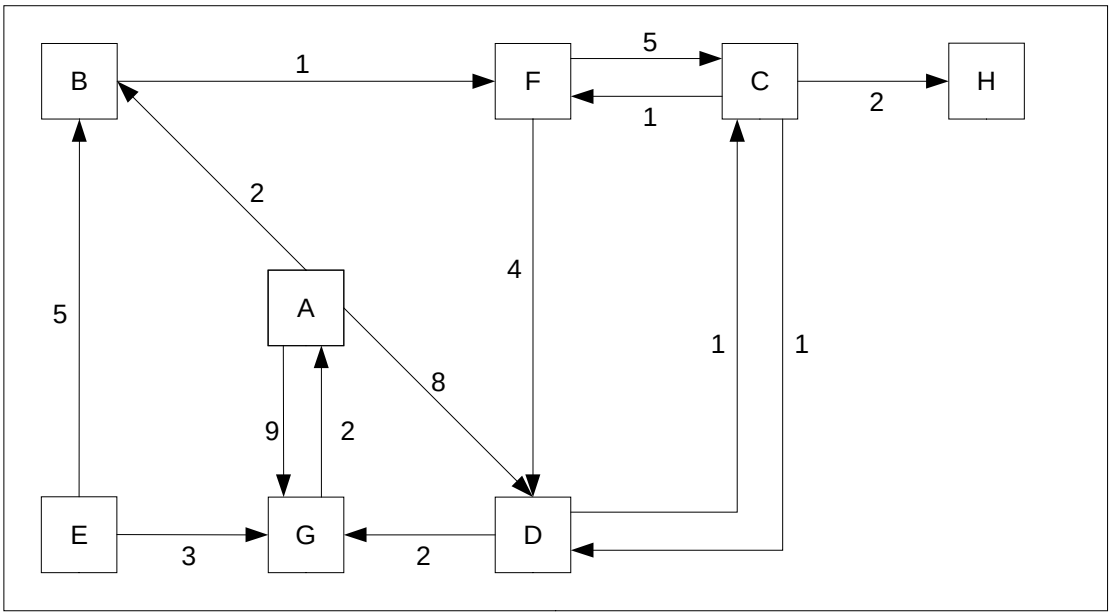


Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie im Beispiel-01 beschrieben.

besucht	**Ecken**		**A 0**	**C 6**	**B 7**	**F 9**	**E 12**	**I 13**	**H 14**	**D 15**	**G 18**	**J 20**	**Min**
1	A	0											0
3	B	inf	7										7
2	C	inf	6										6
8	D	inf			17		16		15				15
5	E	inf	16	14	12								12
4	F	inf		9									9
9	G	inf								18			18
7	H	inf				15	18#	14					14
6	I	inf				13							13
10	J	inf						23	22		20		20

Kürzest. Weg von A nach J: `A 6 C 3 F 4 I 1 H 1 D 3 G 2 J`, Länge: 6+3+4+1+1+3+2 = 20

Lösung-10: Ein gerichteter Graph

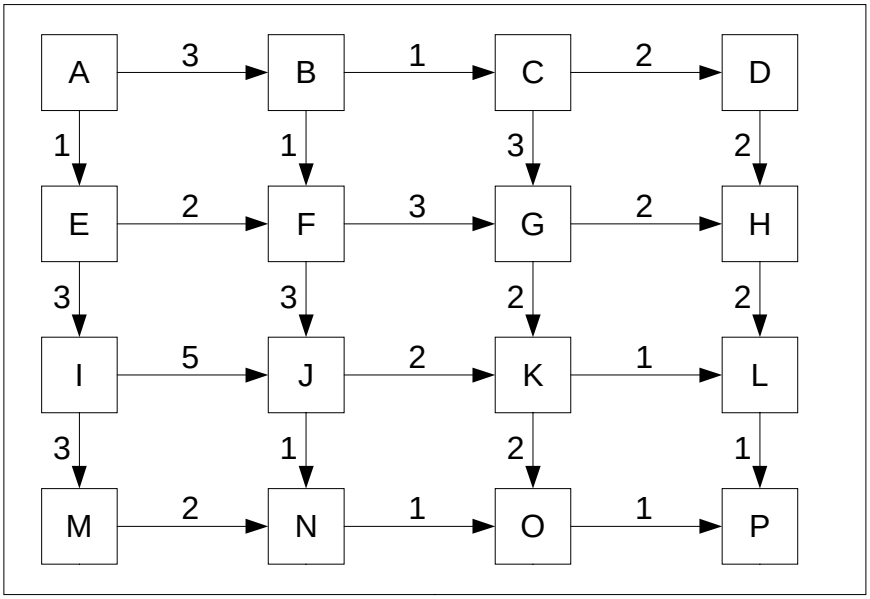


Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie im Beispiel-01 beschrieben.

besucht	**Ecken**		**A 0**	**B 2**	**F 3**	**D 7**	**C 8**	**G 9**	**H 10**		**Min**
1	A	0									0
2	B	inf	2								2
5	C	inf			8	8#					8
4	D	inf	8		7						7
8	E	inf									inf
3	F	inf		3							3
6	G	inf	9			9#					9
7	H	inf					10				10

Kürzester Weg von A nach H: `A 2 B 1 F 5 C 2 H`, Länge 2+1+5+2=10

Lösung-11: Ein gerichteter Graph:



Wenden Sie den Algorithmus von Dijkstra an, um die kürzesten Weg von der Ecke A zu allen anderen Ecken zu berechnen. Füllen Sie dabei die folgende Tabelle aus wie im Beispiel-01 beschrieben.

be.	**Eck**		**A 0**	**E 1**	**B 3**	**F 3**	**C 4**	**I 4**	**D 6**	**G 6**	**J 6**	**M 7**	**N 7**	**H 8**	**K 8**	**O 8**	**L 9**	**P 9**	**Min**
1	A	0																	0
3	B	inf	3																3
5	C	inf			4														4
7	D	inf					6												6
2	E	inf	1																1
4	F	inf		3	4#														3
8	G	inf				6	7#												6
12	H	inf							8	8#									8
6	I	inf		4															4
9	J	inf				6		9#											6
13	K	inf								8	8#								8
15	L	inf												10	9				9
10	M	inf						7											7
11	N	inf									7	9#							7
14	O	inf											8		10#				8
16	P	inf														9	10#		9

Geben Sie dann an, über welche Ecken ein kürzester Weg von A nach P führt.

`A 1 E 2 F 3 J 1 N 1 O 1 P`, Länge: 1+2+3+1+1+1=9