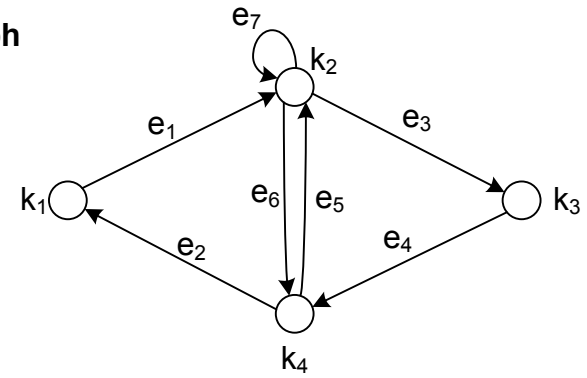
zu speichernder Beispiel-Graph



knotenorientiert Darstellung (Adjazenz-Darstellung)

sequenzielle Adjazenzliste

	NeighPlus				last
k_1	2	?	?	?	1
k_2	2	3	4	?	3
k_3	4	?	?	?	1
k_4	1	2	?	?	2

`NeighPlus[i][j]` beinhaltet Knotenbezeichner des j-ten Nachfolgers von Knoten i sofern j <= last[i]. Sonst steht dort ein beliebiger Wert.

+ Zugriff effizient, da wahlfrei (per Index)
- Speicherplatzverschwendung im Normalfall (siehe die ‚?')
- Statisch (Knoten können nicht hinzugefügt/gelöscht werden)

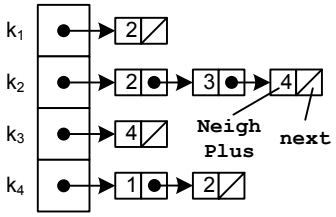
verdichtete sequenzielle Adjazenzliste

	Neigh Plus	last
1	2	1
2	2	4
3	3	5
4	4	7
5	4	
6	1	
7	2	

Hier wird für **`NeighPlus`** ein eindimensionales Feld verwendet. In **`last[i]`** wird immer der letzte Nachfolger von Knoten i gespeichert.

+ laufzeiteffizient, da wahlfreier Zugriff (per Index)
+ speicherplatzeffizient, da nur Speicher für vorhandene Nachfolger gebraucht wird
- Statisch

verkettete Adjazenzliste



In **`first[i]`** wird befindet sich der Kopfzeiger für die Nachfolgerliste von Knoten i
+ speicherplatzeffizient, da nur Speicher für vorhandene Nachfolger gebraucht wird
+ leicht zu einer dynamischen Realisierung erweiterbar (**`first`** muss dann auch z.B. verkettete Liste sein)
- laufzeitineffizient, bei Zugriff Durchlauf der entsprechenden Liste notwendig

Adjazenzmatrix

	k_1	k_2	k_3	k_4
k_1	0	1	0	0
k_2	0	1	1	1
k_3	0	0	0	1
k_4	1	1	0	0

`NeighPlus`

`NeighPlus[i][j] == 1` gdw. Kante von k_i nach k_j existiert, sonst 0.
+ Kann statt 1 auch Kantengewichte speichern.
+ laufzeiteffizient
- statisch
- speicherplatzineffiziernt

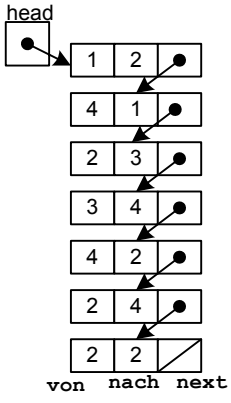
kantenorientiert Darstellung (Inzidenz-Darstellung)

sequenzielle Inzidenzliste

	Kanten	
e_1	1	2
e_2	4	1
e_3	2	3
e_4	3	4
e_5	4	2
e_6	2	4
e_7	2	2

Kante **`k`** verläuft von Knoten **`Kante[k][0]`** nach **`Kante[k][1]`**
+ Zugriff effizient, da wahlfrei (per Index)
- Statisch (Knoten können nicht hinzugefügt/gelöscht werden)
- Isolierte Knoten ohne Schlinge können nicht gespeichert werden

verkettete Inzidenzliste



Wie sequenzielle Liste, nur über verkettete Liste
+ Dynamisch, da Realisierung über dynamische Datenstruktur
- Zugriff ineffizient (Listendurchlauf)
- Isolierte Knoten ohne Schlinge können nicht gespeichert werden

Inzidenzmatrix

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
k_1	1	-1	0	0	0	0	0
k_2	-1	0	1	0	-1	1	2
k_3	0	0	-1	1	0	0	0
k_4	0	1	0	-1	1	-1	0

`Kanten`

`Kante[i][j] == 1` falls Kante j von Knoten i ausläuft
`Kante[i][j] == -1` falls Kante j von Knoten i einläuft, sonst 0 (Spezialfall Schlinge == 2 (Beispiel))
+ laufzeiteffizient
+ erlaubt Speicherung von Kantengewichten (z.B. -13.42)
- speicherplatzineffizient
- „sehr" statisch (weder Knoten noch Knoten können gelöscht/eingefügt werden)