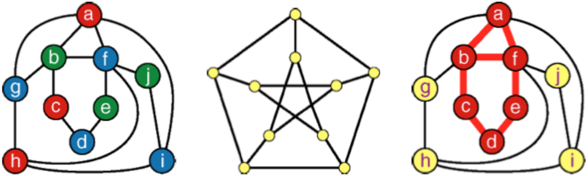
Algorithmische Graphentheorie

Vorlesung 3: Einführung in die Graphentheorie - Teil 3

Babeş-Bolyai Universität, Department für Informatik, Cluj-Napoca
csacarea@cs.ubbcluj.ro



Zusammenhang

Definition

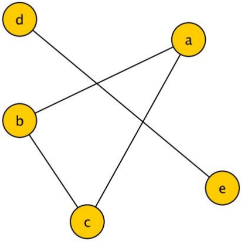
Es sei $G = (V, E)$ ein Graph.

1. *Zwei Knoten $v, w \in V$ heißen verbindbar gdw. ein Weg von v nach w existiert.*
2. *G heißt zusammenhängend (connected) gdw. je zwei Knoten von G verbindbar sind.*
3. *Eine Zusammenhangskomponente (connected component) von G ist*
   - *ein durch eine Knotenmenge $U \subseteq V$ induzierter Untergraph $G(U)$, der zusammenhängend und*
   - *bezüglich der Knotenmenge maximal ist, d.h. $G(W)$ ist nicht zusammenhängend für alle $U \subset W$.*

ZUSAMMENHANG (2)

Beispiel Ein nicht zusammenhängender Graph mit Zusammenhangskomponenten induziert durch

- $\{a, b, c\}$ und
- $\{d, e\}$



Zusammenhang (3)

Satz

Jeder zusammenhängende Graph mit n Knoten hat mindestens $n-1$ Kanten.

Beweis:

Mittels vollständiger Induktion über die Anzahl der Knoten, also über n.
Induktionsanfang $n = 1$: Ein Graph mit genau einem Knoten ist zusammenhängend und hat keine Kanten.
Induktionsschritt $n \to n+1$: Wie nehmen an, dass jeder zusammenhängende Graph mit $n' \leq n$ Knoten, mindestens $n' - 1$ Kanten hat.
Induktionsbehauptung: Jeder zusammenhängende Graph mit $n+1$ Knoten hat mindestens n Kanten.

Zusammenhang (4)

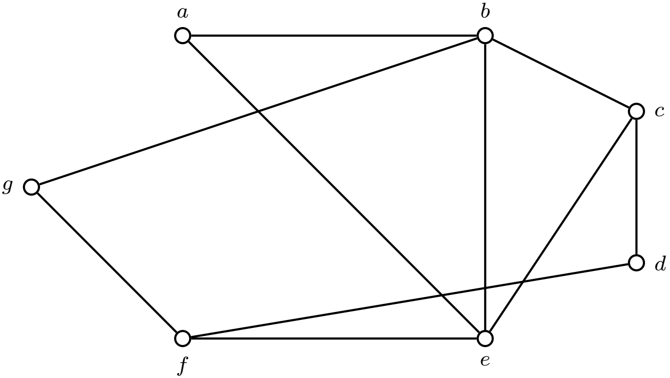
Es sei $G = (V, E)$ ein Graph mit $n + 1$ Knoten.

	Wähle beliebigen Knoten $v \in V$.	
	$k := \deg(v)$.	Definition
$\Rightarrow$	$k \geq 1$	weil G z.h.
	Es sei G' der Graph der entsteht, wenn wir aus G den Knoten v und alle mit v inzidenten Kanten entfernen.	Definition
	G' besteht aus höchstens $l \leq k$ ZHKs, $ZHK_1, \ldots, ZHK_l$.	wegen $\deg(v) = k$
	Jede ZHK_i enthält höchstens n Knoten.	weil G' insgesamt nur n Knoten hat
$\Rightarrow$	Wir können für jede ZHK_i die Induktionsvoraussetzung anwenden.	
	Es sei n_i die Anzahl der Knoten in ZHK_i.	Definition
$\Rightarrow$	Jede ZHK_i hat mindestens $n_i - 1$ Kanten.	Induktionsvoraussetzung

ZUSAMMENHANG (5)

		Weiterhin gilt $n_1 + n_2 + \cdots + n_l = n$.	Alle Knoten von G ausgenommen v sind in den ZHKs von G'.
$\|E\|$	$\geq$	$(n_1 - 1) + \cdots + (n_l - 1) + k$	Kanten in ZHKs von G' plus die mit v inzidenten Kanten
	$=$	$n_1 + \cdots + n_l - l + k$	
	$\geq$	$n - k + k$	s.o., und weil $l \leq k$
	$=$	n	q.e.d.

Beispiel



Zusammenhang mit SAGE

```
sage: g = Graph({"a":["b","e"], "b":["a","g","e","c"], \
...     "c":["b","e","d"], "d":["c","f"], "e":["f","a","b","c"], \
...     "f":["g","d","e"], "g":["b","f"]})
sage: g.is_connected()
True
sage: g.shortest_path("g", "d")
['g', 'f', 'd']
```

Bäume

Definition

Es sei $G = (V, E)$ ein Graph. G heißt Wald (forest) gdw. G keinen Kreis enthält. G heißt Baum (tree) gdw. G ein Wald ist und zusammenhängend ist.

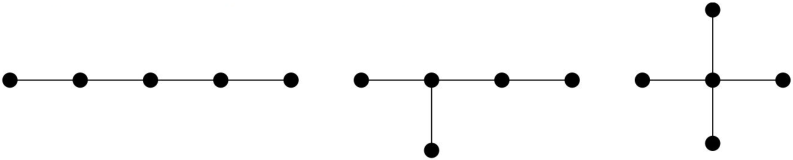


Abbildung 1: Die (nichtisomorphen) Bäume mit 5 Knoten

Satz

Ein Baum mit n Knoten besitzt genau $n-1$ Kanten.

Beweis:

Vollständige Induktion nach n.

- I.V.: Für Bäume mit 1 oder 2 Knoten ist der Satz wahr.
- I.S. $n \to n+1$: Sei T_{n+1} ein Baum mit $n+1$ Knoten und e eine Kante. Der Graph $T_{n+1} - e$ besteht aus zwei Komponenten G und H, die wiederum Bäume sind.
- Die Anzahl der Knoten von G und H sei g, bzw. h.
- Es gilt $g + h = n + 1$. Außerdem hat G genau $g - 1$ Kanten und H genau $h - 1$ Kanten.
- Es gilt $g + h - 2 = n - 1$. Die Kanten von G und H sind auch Kanten von T_{n+1}. Mit der Kante e hat T_{n+1} genau n Kanten.

Charakterisierung von Bäume

Theorem

Für einen Graphen $G = (V, E)$ mit $|V| = n$ sind die folgenden Aussagen äquivalent:

1. *G ist ein Baum,*
2. *Je zwei Knoten von G sind durch genau einen Weg verbunden,*
3. *G ist zusammenhängend, aber für jede Kante $e \in E$ ist $G' = (V, E \setminus \{e\})$ nicht zusammenhängend,*
4. *G ist zusammenhängend und hat genau $n - 1$ Kanten*
5. *G ist kreisfrei und hat genau $n - 1$ Kanten,*
6. *G ist kreisfrei, aber für je zwei nicht adjazente Knoten v, w von G enthält $G'' = (V, E \cup \{\{v, w\}\})$ genau einen Kreis.*

Gerichtete Graphen

Für viele Anwendungen ist es sinnvoll, die Kanten mit einer Richtung zu versehen.

Definition

Ein gerichteter Graph (directed graph, digraph) ist ein Paar $G = (V, A)$ bestehend aus den Mengen

- *V, der Menge von Knoten und A, der Menge der gerichteten Kanten (arcs), die aus geordneten Paaren $v, w)$ mit $v, w \in V, v \neq w$ besteht.*

Für eine gerichtete Kante $a = (v, w)$ heißt v der Angangsknoten (initial vertex) und w der Endknoten (terminal vertex) von a.

Gerichtete Graphen (2)

Bemerkung

Man kann ungerichtete Graphen als gerichtete Graphen betrachten, bei denen die Relation A symmetrisch ist.

Definition

Es sei $G = (V, A)$ ein gerichteter Graph.

- $indeg(v) = \mid \{(x, v) \mid (x, v) \in A\} \mid$ *heit der Eingangsgrad von* $v \in V$
- $outdeg(v) = \mid \{(v, y) \mid (v, y) \in A\} \mid$ *heit Ausgangsgrad von* $v \in V$
- *Ein gerichteter Kantenzug ist eine Folge* $(v_0, v_1, \ldots, v_n)$ *von Knoten mit* $e_i := (v_{i-1}, v_i) \in A$ *für* $i = 1, \ldots, n$ *gerichtete Graphen übertragen.*

Gerichtete Graphen (3)

Definition

- *Der einem gerichteten Graph $G = (V, A)$ zugeordnete ungerichtete Graph $G' = (V, A')$ ist definiert durch $\{v, w\} \in A'$ gdw. $(v, w) \in A$ oder $(w, v) \in A$.*
- *G heißt zusammenhängend gdw. der zugeordnete ungerichtete Graph G' zusammenhängend ist.*
- *G heißt stark zusammenhängend gdw. es fur je zwei Knoten $v, w \in V$ einen gerichteten Weg von v nach w gibt.*

Lemma

Für einen gerichteten Graphen $G = (V, A)$ gilt:

$$\sum_{v \in V} indeg(v) = \sum_{v \in V} outdeg(v).$$

DAGs

Definition

Ein gerichteter Graph $G = (V, A)$ heißt DAG (directed acyclic graph) gdw. G keinen einfachen gerichteten Kreis der Länge ≥ 2 enthält.

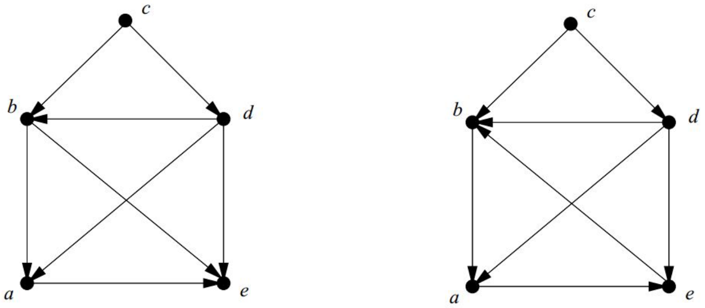


Abbildung 2: Der linke Graph ist ein DAG, der rechte nicht.

Graphinvarianten

1. Anzahl der Knoten
2. Anzahl der Kanten
3. Minimalgrad
4. Maximalgrad
5. Gradfolge
6. längster Weg
7. Zusammenhang
8. Anzahl der Komponenten

Graphinvarianten (2)

Gradfolgen

Die Anzahl der Knoten eines gegebenen Grades ist eine Grapheninvariante. Die Gradfolge eines Graphen ist definiert als

$$(d_1, \ldots, d_n) := (deg(v_1), \ldots, deg(v_n)).$$

Es gilt $\delta(G) = d_1 \leq d_2 \leq \cdots \leq d_n = \Delta(G)$.

Korollar

Für einen Baum T_n mit $n \geq 2$ gilt $d_1 = d_2 = 1$.

Beweis

Hausaufgabe. Wende hierfür das Handschlaglemma

Repräsentation von Graphen in Computern

Definition

Gegeben sei ein Graph $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}, n \geq 1$. Dann kann E in Form einer $n \times n$-Matrix repräsentiert werden. Es sei

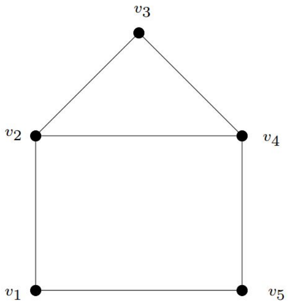
$$a_{ij} = \begin{cases} 1, & \text{falls } \{v_i, v_j\} \in E, \\ 0, & \text{sonst.} \end{cases}$$

Die Matrix $A(G)$ heißt die Adjazenzmatrix (adjacency matrix) von G.

$A(G)$ ist symmetrisch und $a_{ii} = 0, 1 \leq i \leq n$.

Analog kann die Adjazenzmatrix für die Darstellung gerichteter Graphen verwendet werden. Sie ist dann in der Regel nicht symmetrisch.

Adjazenzmatrix

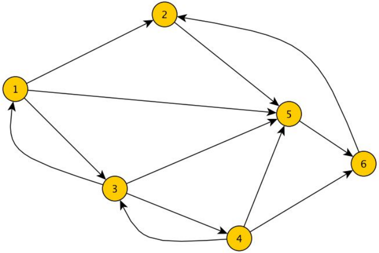
$G =$ 

$$\mathbf{A}_G = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Adjazenzmatrix (2)

- Es kann in Zeit $O(1)$ überprüft werden, ob zwei Knoten v_i und v_j adjazent sind.
- $\deg(v_i)$ ist gleich der Zeilensumme der i-ten Zeile (bzw. der Spaltensumme der i-Spalte). Aufwand:$O(\mid V \mid)$.
- Ermittlung der Nachbarn zu einem Knoten v_i: Suche in der i-ten Zeile/Spalte.
- notwendiger Speicherplatz:$O(|V|^2)$.
- Platzverbrauch ineffizient für bestimmte Graphklassen, z.B. Bäume, planare Graphen.

Adjazenzmatrix für gerichtete Graphen



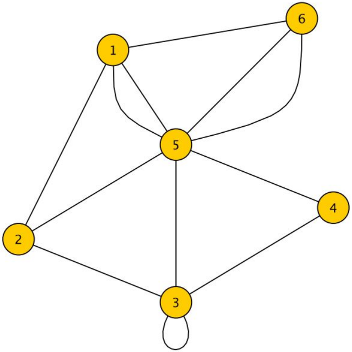
$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Beispiel: Adjazenzmatrix für nichtschlichte Graphen

- Für nicht schlichte Graphen hibt a_{ij} die Anzahl der Kanten zwischen v_i und v_j an.
- Wenn Schlingen vorliegen, sind die Diagonalelemente der entsprechenden Knoten ungleich 0. Das Element a_{ii} gibt dann die Anzahl der Schlingen am Knoten v_i an.
- Bei der Gradermittlung müssen die Diagonalelemente doppelt gezählt werden:

$$deg(v_i) = 2 \cdot a_{ii} + \sum_{k=1, k \neq i}^{n} a_{ik}.$$

Adjazenzmatrix für nichtschlichte Graphen

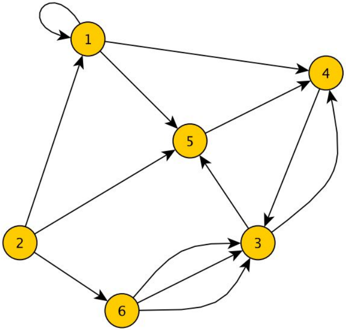


$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 2 & 0 \end{pmatrix}$$

ADJAZENZMATRIX: GERICHTET UND NICHT SCHLICHT

- Prinzipiell können naturlich auch gerichtete Graphen nicht schlicht sein,
- d.h. an Knoten existieren Schlingen oder
- zwischen zwei Knoten a und b gibt es mehrere Kanten mit der gleichen Richtung (von$a$ nach b).

BEISPIEL: GERICHTET UND NICHT SCHLICHT



$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \end{pmatrix}$$

Adjazenzliste

Adjazenzlisten ermöglichen die Darstellung eines Graphen in einem Programm, indem für jeden Knoten eine Liste mit all seinen Nachfolgern aufgebaut wird. Man erzeugt also für jedes $i \in \{1, \ldots, n\}$ eine Liste A_i, die alle $j \in \{1, \ldots, n\}$ und $(v_i, v_j) \in E$ enthält.
Die Adjazenzlisten kann man mit Hilfe von Arrays, Vektoren oder verketteten Listen implementieren.

Beispiel: Adjazenzliste (2)

Definition

Gegeben sei ein Graph $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}, n \geq 1$. Dann kann E in Form einer Liste von n-Listen A_i repräsentiert werden. Für $1 \leq i \leq n$ seien $v_{i_1}, v_{i_2}, \ldots, v_{n_i}$ die mit $v_i \in V$ adjazenten Knoten. Die Liste

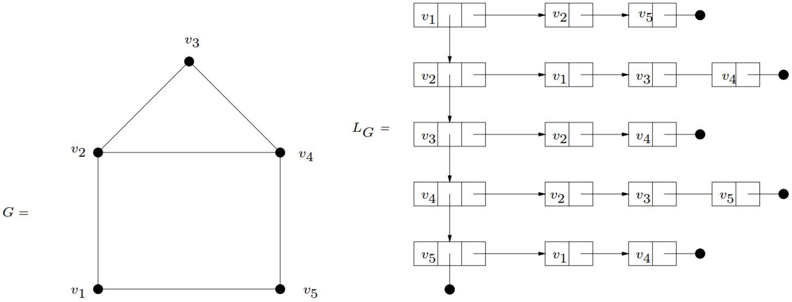
$$A_i = (v_{i_1}, v_{i_2}, \ldots, v_{n_i})$$

heißt die Adjazenzliste von $v_i \in V$.

Die Liste $L_G = (A_1, \ldots, A_n)$ ist die Adjazenzlistendarstellung von G.

Für einen gerichteten Graphen $G = (V, A)$ enthält die Adjazenzliste A_i die Knoten $w \in V$, für die $(v_i, w) \in A$ gilt.

Beispiel: Adjazenzliste



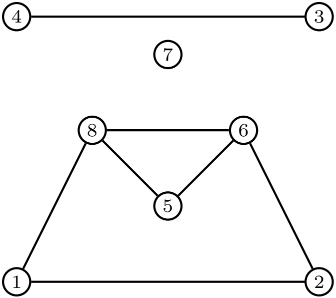
Adjazenzliste (3)

- Um zu überprüfen, ob zwei Knoten v_i und v_j adjazent sind, muss die Adjazenzliste von v_i durchsucht werden.
- Dies ist nicht in $O(1)$ möglich, der genaue Aufwand hängt von der Implementierung der Adjazenzliste ab.
- Der Knotengrad entspricht der Länge der Adjazenzliste.
- Die Nachbarn zu einem Knoten liegen direkt in der Adjazenzliste vor.
- notwendiger Speicherplatz: $O(|V| + |E|)$.

ADJAZENZLISTE (4)

- $L_i = []$ gdw. v_i ist ein isolierter Knoten
- Die Reihenfolge der Elemente in der Anjazenzliste spielt keine Rolle!
- Die Anzahl der Elemente einer Liste gibt uns die Länge der Liste
- Aus der Adjazenzliste können wir den Graphen rekonstruieren.

Beispiel



$L_1 = [2, 8]$

$L_2 = [1, 6]$

$L_3 = [4]$

$L_4 = [3]$

$L_5 = [6, 8]$

$L_6 = [2, 5, 8]$

$L_7 = [\,]$

$L_8 = [1, 5, 6]$

Abbildung 3: Ein Graph und seine Adjazenzlisten.

Beispiel: Kneser Graphen

Kneser Graphen mit Parameter (n, k), oder (n, k)-Kneser Graphen sind Graphen deren Knotenmengen sind alle k-elementige Teilmengen von $\{1, 2, \ldots, n\}$. Zwei Knoten sind adjazent, falls die entsprechenden Teilmengen disjunkt sind.

Man zeichne den $(5, 2)$-Kneser Graphen, man berechne seine Ordnung und die Adjazenzlisten.

$(5,2)$-Kneser Graph

Seine Knoten sind alle zweielementige Teilmengen von $\{1,2,3,4,5\}$:

$\{1,2\}$, $\{1,3\}$, $\{1,4\}$, $\{1,5\}$, $\{2,3\}$, $\{2,4\}$, $\{2,5\}$, $\{3,4\}$, $\{3,5\}$, $\{4,5\}$

Jeder Knoten ist eine Kombination aus zwei Elemente einer 5-elementiger Menge, d.h. der Graph hat Ordnung

$$\binom{5}{2} = \frac{5 \times 4}{2!} = 10.$$

$(5,2)$-Kneser Graph
Kanten

$$(\{1,3\},\ \{2,4\}),\ (\{2,4\},\ \{1,5\}),\ (\{2,4\},\ \{3,5\}),\ (\{1,3\},\ \{4,5\}),\ (\{1,3\},\ \{2,5\})$$

$$(\{3,5\},\ \{1,4\}),\ (\{3,5\},\ \{1,2\}),\ (\{1,4\},\ \{2,3\}),\ (\{1,4\},\ \{2,5\}),\ (\{4,5\},\ \{2,3\})$$

$$(\{4,5\},\ \{1,2\}),\ (\{1,5\},\ \{2,3\}),\ (\{1,5\},\ \{3,4\}),\ (\{3,4\},\ \{1,2\}),\ (\{3,4\},\ \{2,5\})$$

$(5,2)$-Kneser Graph

Adjazenzlisten

$$
\begin{aligned}
L_{\{1,2\}} &= [\{3,4\},\ \{3,5\},\ \{4,5\}], & L_{\{1,3\}} &= [\{2,4\},\ \{2,5\},\ \{4,5\}],\\
L_{\{1,4\}} &= [\{2,3\},\ \{3,5\},\ \{2,5\}], & L_{\{1,5\}} &= [\{2,4\},\ \{3,4\},\ \{2,3\}],\\
L_{\{2,3\}} &= [\{1,5\},\ \{1,4\},\ \{4,5\}], & L_{\{2,4\}} &= [\{1,3\},\ \{1,5\},\ \{3,5\}],\\
L_{\{2,5\}} &= [\{1,3\},\ \{3,4\},\ \{1,4\}], & L_{\{3,4\}} &= [\{1,2\},\ \{1,5\},\ \{2,5\}],\\
L_{\{3,5\}} &= [\{2,4\},\ \{1,2\},\ \{1,4\}], & L_{\{4,5\}} &= [\{1,3\},\ \{1,2\},\ \{2,3\}].
\end{aligned}
$$

(5, 2)-KNESER GRAPH

KNESER GRAPH MIT Sage

```
sage: K = graphs.KneserGraph(5, 2); K
Kneser graph with parameters 5,2: Graph on 10 vertices
sage: for v in K.vertices():
...        print(v, K.neighbors(v))
...
({4, 5}, [{1, 3}, {1, 2}, {2, 3}])
({1, 3}, [{2, 4}, {2, 5}, {4, 5}])
({2, 5}, [{1, 3}, {3, 4}, {1, 4}])
({2, 3}, [{1, 5}, {1, 4}, {4, 5}])
({3, 4}, [{1, 2}, {1, 5}, {2, 5}])
({3, 5}, [{2, 4}, {1, 2}, {1, 4}])
({1, 4}, [{2, 3}, {3, 5}, {2, 5}])
({1, 5}, [{2, 4}, {3, 4}, {2, 3}])
({1, 2}, [{3, 4}, {3, 5}, {4, 5}])
({2, 4}, [{1, 3}, {1, 5}, {3, 5}])
```

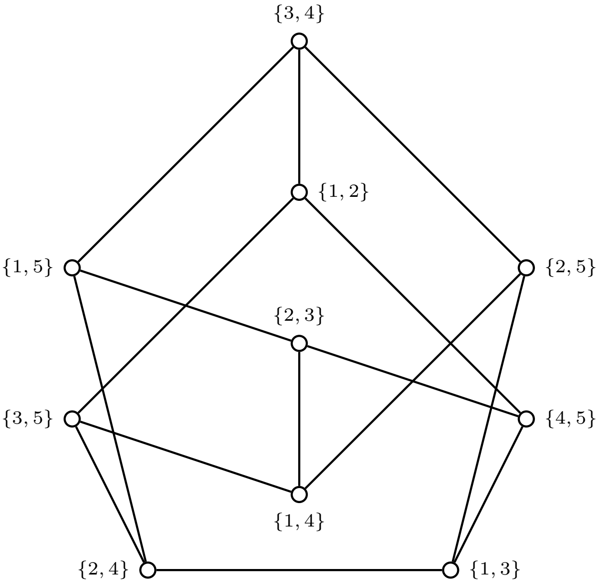
(n,k)-Kneser Graph

Der (n,k)-Kneser Graph hat

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!}$$

Knoten.

$(5,2)$-Kneser Graph



Kantenlisten

Analog zu den Adjazenzlisten, können wir Listen benutzen, um Kanten zu speichern. Sei G ein Graph. Die Kantenliste L wird folgendermaßen aufgebaut:

- Sei uv eine Kante in G. Dann ist das geordnete Paar (u, v) ein Element der Kantenliste L.
- Seien

  $$v_0v_1,\ v_2v_3,\ \ldots,\ v_kv_{k+1}$$

  alle Kanten von G mit k gerade. Die Kantenliste von G ist gegeben durch

$$L = [v_0v_1,\ v_2v_3,\ \ldots,\ v_kv_{k+1}].$$

DER `graph6` FORMAT

- `graph6`
- `sparse6`
- Benutzen Bitvektoren und ASCII, um Graphen zu repräsentieren.
- ASCII mit Dezimalkode von 63 bis 126.

Bitvektoren

- Bitvektor = Liste von Bits
- Die Länge eines Bitvektors ist die Anzahl der Bits
- Das höchstwertige Bit (most significant bit) ist das Bit mit dem größten Wert
- mindestwertige Bit (least significant bit) oder Parität Bit (parity bit).

Bitvektoren

- Die Reihenfolge der Bit Verarbeitung heißt Byte-Reihenfolge (endianness).
- Big endian: von links nach rechts
- Little endian: von rechts nach links

Big endian vs. little endian

	0	1	2	3	4	5	6
position	0	1	2	3	4	5	6
bit value	1	0	0	0	1	0	1
position value	2^0	2^1	2^2	2^3	2^4	2^5	2^6

Tabelle 1: Big-endian Ordnung der ASCII Code von `E`.

position	0	1	2	3	4	5	6
bit value	1	0	0	0	1	0	1
position value	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Tabelle 2: Little-endian Ordnung der ASCII Code von `E`.

Bitvektoren

- Sei $v = b_{n-1}b_{n-2}\cdots b_0$ ein Bitvektor gelesen in big-endian.
- Polynomiale Darstellung:

$$p(x) = \sum_{i=0}^{n-1} x^i b_i = x^{n-1}b_{n-1} + x^{n-2}b_{n-2} + \cdots + xb_1 + b_0.$$

- $p(2)$ gibt uns den numerischen Wert von v.

graph6

- Die Länge des Bitvektors ist ein vielfaches von k.
- Was wenn die Länge k kein vielfaches von 6 ist?
  - Berechne $r = k \mod 6$ und fülle rechts v mit $6 - r$ Nullen.
- Ab jetzt $k \mod 6 = 0$: Spalte v in $k/6$ Bitvektoren v_i der Länge 6.

$$v_i = b_{6i-5}b_{6i-4}b_{6i-3}b_{6i-2}b_{6i-1}b_{6i}, 0 \leq i \leq k/6.$$

graph6

- Jedes v_i ist in big-endian gelesen und sein numerischer Wert N_i wird berechnet.
- $N_i' = N_i + 63$ und speichere N_i' auf einem Byte, d.h. jedes N_i' wird repräsentiert als ein Bitvektor der Länge 8.
- Die Anzahl der Bytes, die wir benötigen, um v zu speichern ist $\lceil k/6 \rceil$.

graph6

- Sei B_i die Byte Repräsentation von N_i', so dass

$$R(v) = B_1 B_2 \cdots B_{\lceil k/6 \rceil}$$

  die Repräsentation von v als Folge von $\lceil k/6 \rceil$ Bytes darstellt.
- Wie stellt man eine natürliche Zahl $0 \leq n \leq 2^{36} - 1$ mit der obigen Formel dar? Wir bezeichnen diese Darstellung mit $N(n)$.

graph6

- Sei B_i die Byte Repräsentation von N_i', so dass

$$R(v) = B_1 B_2 \cdots B_{\lceil k/6 \rceil}$$

  die Repräsentation von v als Folge von $\lceil k/6 \rceil$ Bytes darstellt.
- Wie stellt man eine natürliche Zahl $0 \leq n \leq 2^{36} - 1$ mit der obigen Formel dar? Wir bezeichnen diese Darstellung mit $N(n)$.
-

$$N(n) = \begin{cases} n + 63, & \text{falls } 0 \leq n \leq 62, \\ 126\, R(v), & \text{falls } 63 \leq n \leq 258047, \\ 126\,126\, R(v), & \text{falls } 258048 \leq n \leq 2^{36} - 1. \end{cases}$$

graph6

Bemerkung

- $n + 63$ benötigt ein Byte Speicherplatz
- $126\,R(v)$ benötigt 4 Bytes Speicherplatz
- $126\,126\,R(v)$ benötigt 8 Bytes Speicherplatz

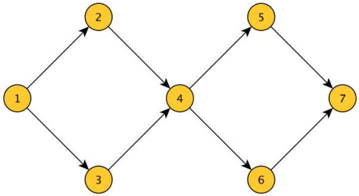
graph6

- graph6 wird benutzt, um einfache, nicht gerichtete Graphen mit maximal $2^{36}-1$ Knoten darzustellen.
- Sei nun G solch ein Graph mit $|V| = n$ und $0 \le n \le 2^{36}-1$.
  - $n = 0$: Die graph6 Repräsentation von G ist "?"
  - $n > 0$. Sei $M = [a_{ij}]$ die Adjazenzmatrix von G. Betrachte das obere Dreieck von M ohne der Hauptdiagonale und schreibe es als Bitvektor:

$$v = \underbrace{a_{0,1}}_{c_1}\ \underbrace{a_{0,2}a_{1,2}}_{c_2}\ \underbrace{a_{0,3}a_{1,3}a_{2,3}}_{c_3} \cdots \underbrace{a_{0,i}a_{1,i}\cdots a_{i-1,i}}_{c_i} \cdots \underbrace{a_{0,n}a_{1,n}\cdots a_{n-1,n}}_{c_n}$$

  - c_i sind die Werte $a_{0,i}a_{1,i}\cdots a_{i-1,i}$ in der Spalte i von M.
- Die graph6 Repräsentation von G ist $N(n)R(v)$.
- $N(n)$ kodiert die Ordnung von G und $R(v)$ die Kanten von G.

Anwendung: Anzahl der Wege zwischen zwei Knoten



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Anwendung: Anzahl der Wege zwischen zwei Knoten (2)

Wir bilden die Potenzen der Adjazenzmatrix $\mathbf{A}$:

$$\mathbf{A}^2 = \begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{A}^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Anwendung: Anzahl der Wege zwischen zwei Knoten (3)

$$\mathbf{A}^4 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{A}^k = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ für } k \geq 5$$

- Das Element $a_{i,j}$ der Matrizen $\mathbf{A}^k$ gibt hier die Anzahl der (einfachen) Wege der Länge k von i nach j an.

Anzahl der Kantenzüge zwischen zwei Knoten

Theorem

Es sei $G = (V, E)$ ein Graph mit der Adjazenzmatrix $A = (a_{ij})$. Dann gibt das Element $(a_{ij}^{(r)}$ der Matrix A^r die Anzahl der Kantenzüge der Länge r von v_i nach v_j an.

Beweis

Induktion über r.
Für $r = 1$ gilt $A^r = A$. Die Adjazenzmatrix gibt genau die Kantenzüge der Länge 1 an.
$r \to r + 1$: Jeder Kantenzug der Länge $r + 1$ zwischen zwei Knoten v_i und v_j besteht aus einem Kantenzug der Länge r zwischen v_i und einem Knoten v_k sowie der Kante $\{v_k, v_j\}$.

Anzahl der Kantenzüge zwischen zwei Knoten (2)

Beweis

- Nach Induktionsvoraussetzung gibt A^r die Anzahl der Kantenzüge der Länge r zwischen zwei Knoten an.
- Es gilt

$$a_{ij}^{(r+1)} = \sum_{k=1}^{|V|} a_{ik}^{(r)} \cdot a_{kj}.$$

- Da $a_{kj} = 1$ gdw. zwischen v_i und v_j eine Kante ist, beschreibt diese Formel die Anzahl der Möglichkeiten, einen Kantenzug der Länge $r+1$ zwischen v_i und v_j aus einem Kantenzug der Länge r zwischen v_i und einem Knoten v_k sowie der Kante $\{v_k, v_j\}$ zu bilden.

Anzahl der Kantenzüge zwischen zwei Knoten (3)

Korollar

Es sei $G = (V, E)$ ein Graph mit Adjazenzmatrix A. Dann gibt das Element b_{ij} der Matrix

$$B = A + A^2 + \cdots + A^p$$

die Anzahl der Kantenzüge mit einer Länge $\leq p$ von v_i nach v_j an.

Anzahl der Kantenzüge zwischen zwei Knoten (4)

Korollar

Es sei $G = (V, E)$ ein Graph mit der Adjazenzmatrix A und es sei

$$B = A + A^2 + \cdots + A^{|V|-1}.$$

Dann gilt: G ist genau dann zusammenhängend, wenn $b_{ij} > 0$ für alle $i \neq j$ gilt.

Bemerkungen

- Weil in DAGs jeder Kantenzug ein gerichteter einfacher Weg ist, liefert A^r dort sogar die Anzahl der einfachen Wege der Länge r.
- Auch können wir mit diesem Ansatz prinzipiell testen, ob ein gerichteter Graph kreisfrei ist (für $p = |V|$ müssen die b_{ii} alle ungleich 0 sein).
- Sowohl für die Kreisfreiheit als auch für den Zusammenhang sind diese Berechnungsansätze aber ineffizient.
- Im nächsten Kapitel werden wir effizientere Algorithmen für diese Probleme kennenlernen.

PFADMATRIX

Definition

Die Pfadmatrix für einen Graphen $G = (V, E)$ mit $G = \{v_1, \ldots, v_n\}$ ist eine $n \times n$ Matrix $M = (m_{ij})_{i,j \in \{1,\ldots,n\}}$ mit $m_{ij} = 1$, falls es einen Pfad (einfachen Weg) von v_i zu v_j gibt und $m_{ij} = 0$ sonst. Wenn $m_{ii} = 1$ ist, bedeutet das, dass es einen Kreis für den Knoten i gibt.

Ein einfacher Algorithmus, der die Pfadmatrix aufbaut ist der Floyd-Warshall-Algorithmus. Diesen Algorithmus kann man auch verwenden, um den kürzesten Pfad zwischen zwei gegebenen Knoten zu bestimmen.

Floyd-Warshall Algorithmus

```
ALGORITHM_FLOYD_WARSHALL(G)
 Initialize matrix M with adjacency matrix A
 For (k←1, n; step 1) Execute
  For (i←1, n; step 1, i≠k) Execute
   For (j←1, n; step 1, i≠k) Execute
    If(M[i][j]=0 AND M[i][k]=1 AND M[k][j]=1) Then
      M[i][j] ← 1
    End_If
   End_For
  End_For
 End_For
 return M
END_ALGORITHM_FLOYD_WARSHALL(G)
```

Inzidenzmatrix

Definition

Sei G ein gerichteter Graph (Digraph) mit Kantenmenge $= \{e_1, \ldots, e_m\}$ *und Knotenmenge* $V = \{v_1, \ldots, v_n\}$*. Die* *Inzidenzmatrix* *von G ist die* $n \times m$ *Matrix* $B = (b_{ij})$ *gegeben durch*

$$b_{ij} = \begin{cases} -1, & \text{if } v_i \text{ is the tail of } e_j, \\ 1, & \text{if } v_i \text{ is the head of } e_j, \\ 2, & \text{if } e_j \text{ is a self-loop at } v_i, \\ 0, & \text{otherwise.} \end{cases}$$

Gradmatrix

Definition

Die Gradmatrix ist eine $(n \times n)$-Diagonalmatrix, deren Diagonalelemente die Grade der Knoten von G sind.

Laplacematrix

Die Laplacematrix $\mathcal{L}$ von G ist definiert durch

$$\mathcal{L} = D - A$$

Für einen ungerichteten ungewichteten schlichten Graphen $\mathcal{L} = (\ell_{ij})$ ist gegeben durch

$$\ell_{ij} = \begin{cases} -1, & \text{if } i \neq j \text{ and } v_i v_j \in E, \\ d_i, & \text{if } i = j, \\ 0, & \text{otherwise,} \end{cases}$$

wobei $d_i = \deg(v_i)$ ist der Grad des Knoten v_i.

Gradmatrix (2)

Es gilt...

$$BB^T = A + D.$$

- Die Elemente von BB^T sind Skalarprodukte der Zeilenvektoren der Inzidenzmatrix.
- Die Komponenten dieser Vektoren sind nur

GRADMATRIX (2)

Es gilt...

$$BB^T = A + D.$$

- Die Elemente von BB^T sind Skalarprodukte der Zeilenvektoren der Inzidenzmatrix.
- Die Komponenten dieser Vektoren sind nur Nullen und Einsen
- Das Skalarprodukt ist gleich

Gradmatrix (2)

Es gilt...

$$BB^T = A + D.$$

- Die Elemente von BB^T sind Skalarprodukte der Zeilenvektoren der Inzidenzmatrix.
- Die Komponenten dieser Vektoren sind nur Nullen und Einsen
- Das Skalarprodukt ist gleich der Anzahl der übereinstimmenden Einsen in den beiden Vektoren.
- Zwei Zeilen i und j mit $i \neq j$ enthalten genau dann beide an der Stelle k eine Eins, wenn

Gradmatrix (2)

Es gilt...

$$BB^T = A + D.$$

- Die Elemente von BB^T sind Skalarprodukte der Zeilenvektoren der Inzidenzmatrix.
- Die Komponenten dieser Vektoren sind nur Nullen und Einsen
- Das Skalarprodukt ist gleich der Anzahl der übereinstimmenden Einsen in den beiden Vektoren.
- Zwei Zeilen i und j mit $i \neq j$ enthalten genau dann beide an der Stelle k eine Eins, wenn die entsprechenden Knoten i und j mit der Kante k verbunden werden.

Gradmatrix (3)

- Das Element auf Platz (i, j) von BB^T ist gleich

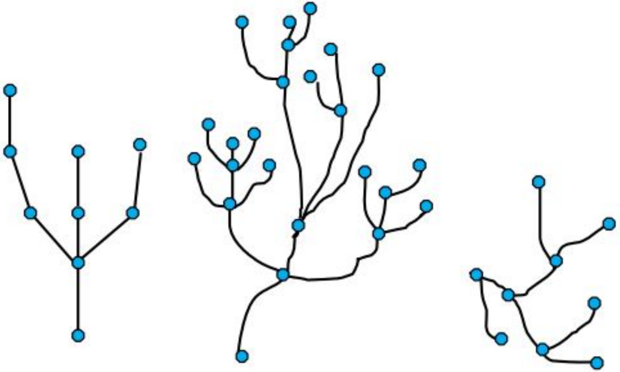
Gradmatrix (3)

- Das Element auf Platz (i, j) von BB^T ist gleich der Anzahl der Kanten zwischen i und j.
- Die Diagonalelemente von BB^T sind Skalarprodukte der Zeilenvektoren von B mit sich selbst. Diese liefern genau die Anzahl

GRADMATRIX (3)

- Das Element auf Platz (i, j) von BB^T ist gleich der Anzahl der Kanten zwischen i und j.
- Die Diagonalelemente von BB^T sind Skalarprodukte der Zeilenvektoren von B mit sich selbst. Diese liefern genau die Anzahl der Einsen der Zeilen von B - die Grade der Knoten des Graphen.

Bäume und Wälder



Bäume und Wälder (2)

Sei $(G = (V, E)$ ein ungerichteter Graph. Die folgenden Aussagen sind äquivalent:

1. G ist ein Baum
2. G ist minimal zusammenhängend (wenn man eine beliebige Kante $e \in E$ entfernt, ist der resultierende Graph nicht mehr zusammenhängend)
3. G ist maximal kreislos (wenn man eine beliebige Kante hinzufügt, ist der entstehende Graph nicht mehr kreislos)
4. zwischen je zwei Knoten enthält G genau einen Weg.

Minimale Spannbäume

Definition

Ein aufspannender Untergraph des Graphen $G = (V, E)$ ist ein Graph $H = (V, F)$ mit der Eigenschaft $F \subseteq E$ (H hat alle Knoten aber nicht alle Kanten von G).

Definition

Sei $G = (V, E)$ ein Graph. Eine Funktion $c\colon E \to \mathbb{R}_+$, die jede Kante $e \in E$ auf eine positive reelle Zahl abbildet, heißt Gewichtsfunktion des Graphen G.

Minimale Spannbäume

Definition

Sei $H = (V, F)$ ein aufspannender Untergraph von G. Mit dem Gewicht des Untegraphen H bezeichnen wir die Summe der Gewichte seiner Kanten

$$c(H) = \sum_{e \in F} c(e).$$

Minimale Spannbäume (2)

Definition

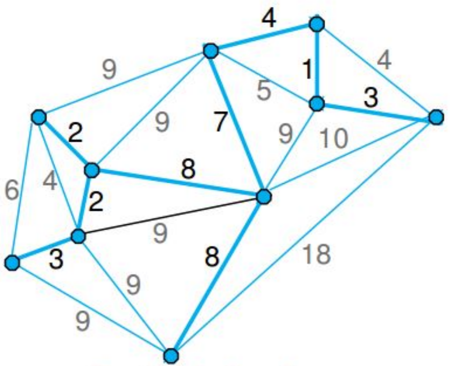
Ein Untergraph $H = (V, F)$ des Graphen $G = (V, E)$ der alle Knoten von G beinhaltet und auch ein Baum ist, heißt Spannbaum.

Definition

Einen Spannbaum, der von allen Spannbäumen des Graphen G minimales Gewicht hat, nennt man minimalen Spannbaum.

PROBLEM

Man finde einen minimalen Spannbaum eines gewichteten Graphen G.



Ein minimaler Spannbaum

Algorithmus von Kruskal

Sei der Graph $G = (V, E)$ mit n Knoten und die Gewichtsfunktion $c\colon E \to \mathbb{R}$ gegeben.

- Man beginnt mit dem Untergraphen $H = (V, \emptyset)$. Dieser Graph hat n Komponenten.
- Wir bauen daraus ein Baum mit n Knoten und $n-1$ Kanten.
- Füge Kante mit minimalem Gewicht zu $\to$ Untergraph mit $n-1$ Komponenten.
- Füge immer wieder Kante mit minimalem Gewicht zu, die keinen Zyklus erzeugt $\to$ Endpunkte in verschiedenen Komponenten $\Rightarrow$ Anzahl der Komponenten vermindert sich um 1.
- Der Algorithmus endet wenn man $n-1$ Kanten ausgewählt hat.

ALGORITHMUS VON KRUSKAL (2)

```
ALGORITHM_KRUSKAL(Graph G)
  Sort(e1, e2, …, em)
  For (i←1, n; step 1) Execute
    K[i] ← i                          // die Komponenten bezeichnen
  End_For
  weight ← 0
  selEdges ← 0
  While (selEdges<n-1) Do
     (u, v) ← nextEdge(E)
     While (K[u] ≠ K[v]) Do
        (u, v) ← nextEdge(E)
     End_While
     H.add((u, v))
     weight ← weight+c(u, v)
     selEdges ← selEdges+1
     max ← max(K[u], K[v])
     min ← min(K[u], K[v])
     For (i←1, n; step 1) Execute      // zwei Komponenten vereinigen
       If( K[i]=max) Then K[i] ← min End_If
     End_For
  End_While
  return H, weight
END_ALGORITHM_KRUSKAL(Graph G)
```