Übungen

- Übungsanmeldung
  - Verlängert bis Fr. 09.11., 12:00 Uhr
- Übungskorrektur
  - Ab dieser Woche alle vier Aufgaben
  - Bitte Aufgaben 1+2 und 3+4 auf **getrennte Zettel**

Zusammenfassung

- Weg, Pfad, Kreis
- schwacher/induzierter Teilgraph
- Zusammenhangskomponenten (ZHK)
- Bäume und Wälder
  - Baum: kreisfrei, zusammenhängend, n-1 Kanten

Eigenschaften von Bäumen

Satz: Sei G=(V,E). Die folgenden Aussagen sind äquivalent

1. **G ist ein Baum.**
2. **$\forall$ u,v $\in$ V: $\exists$! u-v-Pfad in G**
3. **G ist zusammenhängend, G=(V,E\{e}) ist nicht zusammenhängend für alle e $\in$ E.**
4. **G ist zusammenhängend und hat genau n-1 Kanten.**
5. **G ist kreisfrei und hat genau n-1 Kanten.**
6. **G ist kreisfrei und für alle nicht inzidenten u,v $\in$ V: G‘=(V, E $\cup$ {u,v}) enthält Kreis.**

Spannbäume

Def: Sei G=(V,E) ein zusammenhängender Graph.
$T=(V, E_T)$, $E_T \subseteq E$ ist Spannbaum von G $\Leftrightarrow$ T ist Baum.

Algorithmus SPANNBAUM

Eingabe: zusammenhängender G=(V,E)

- while G=(V,E) enthält Kreis K
  - Sei e beliebige Kreiskante. E:=E \ {e}.

Ausgabe: Spannbaum T=(V,E) von G

Korrektheit des Algorithmus

Satz: Algorithmus SPANNBAUM berechnet bei Eingabe eines zusammenhängenden G=(V,E) einen Spannbaum T mit m-n+1 Schleifendurchläufen.

Korrektheit: T ist zusammenhängend.

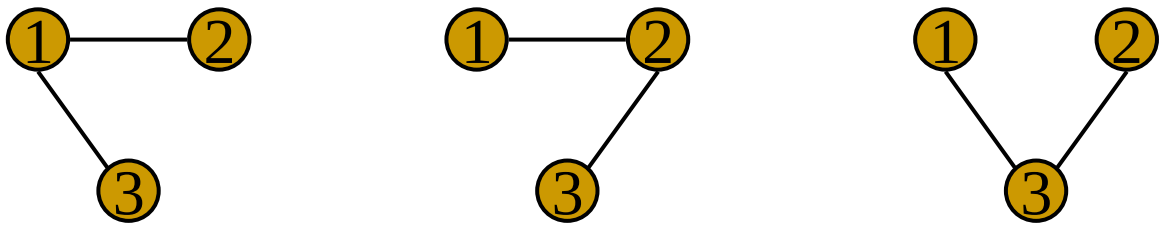
- Schleifeninvariante: G bleibt zusammenhängend.
- Sei u,v Knoten mit u-v-Pfad, der eine Kreiskante $e=\{v_i,v_{i+1 \bmod k}\}$ aus dem Kreis $K=(v_0,\ldots,v_{k-1})$ verwendet. Dann kann e durch den Pfad $(v_i, v_{i-1}, \ldots,v_0, v_{k-1},\ldots, v_{i+1})$ ersetzt werden.
- Schleifenoperation erhält Schleifeninvariante.
- Nach Terminierung ist T kreisfrei. Algorithmus muss terminieren, da die Kantenzahl sukzessive verringert wird.

Laufzeit:

- Spannbaum ist Baum und hat daher n-1 Kanten.
- Jeder Schleifendurchlauf verringert die Kantenanzahl um Eins:
  $\Rightarrow$ #Schleifendurchläufe = m-(n-1).

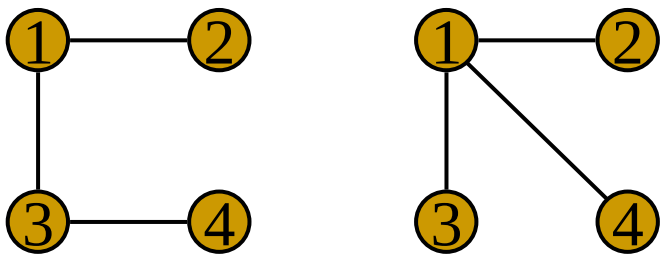
Spannbaum nicht eindeutig

Betrachten Spannbäume des K_3:



- Anzahl markierter Spannbäume: 3
  - Unterscheidbarkeit nur durch Knotenmarkierung
- Alle 3 markierten Spannbäume sind isomorph.

Nicht-isomorphe Spannbäume des K_4:



Anzahl Bäume

Knoten #Bäume	2	3	4	5	6	7	8
markiert	1	3	16	125	1296	16807	2621441
nicht-isomorph	1	1	2	3	6	11	23

Satz von Cayley

Satz (Cayley):
Für $n \geq 2$ gibt es genau n^{n-2} markierte Bäume.

- Sei V=[n].
- Sei T_n die Menge markierter Bäume mit n Knoten.

- Idee: Definieren Bijektion PK: $T_n \rightarrow [n]^{n-2}$.
- Daher gilt $|T_n| = n^{n-2}$.

Prüferkode PK

Algorithmus Kodierung

Eingabe: $T=(G,V) \in T_n$

1. $i \leftarrow 1$
2. while $|V|>2$ do
   1. Sei $v_i \in T$ Blatt mit kleinster Markierung
   2. $t_i \leftarrow$ Nachbar von v_i in T; $i \leftarrow i+1$;
   3. $V \leftarrow V \setminus \{v_i\}$, $E \leftarrow E \setminus \{v_i,t_i\}$

Ausgabe: Prüferkode $(t_1,\ldots,t_{n-2})$

Terminierung:

- In jedem Schritt wird ein Knoten entfernt.
- Algorithmus terminiert nach $|V|-2$ Schleifendurchläufen.

Korrektheit:

z.z.: (1) In jedem Baum mit $n>2$ Knoten existiert stets ein Blatt.
(2) Durch Entfernen eines Blattes bleibt T ein Baum.

Korrektheit

ad (1): Sei T ein Baum. Suchen Blatt algorithmisch

- Markiere beliebigen Knoten v.
- while (deg(v)>1)
  - Markiere unmarkierten Nachbarknoten u von v. v $\leftarrow$ u;

- Algorithmus terminiert, da kein Knoten zweimal besucht werden kann.
- Bei Terminierung ist v ein Knoten mit deg(v)=1, d.h. ein Blatt.

ad (2): Angenommen Blatt w werde entfernt.

- Sei u,v$\neq$w beliebig mit u-v-Pfad p=(u=v_0,…,v_k=v).
- w kann nicht in p enthalten sein, da innere Knoten v_1,…,v_{k-1} mindestens Grad 2 besitzen

Grad der Knoten aus Prüferkode

Lemma: Sei T=(V,E) ein Baum. Jedes v $\in$ V kommt im Prüferkode von T genau deg(v)-1 mal vor.

- Blätter kommen im Prüferkode nicht vor.
- Sei v innerer Knoten in T.
  - Am Ende des Algorithmus Kodierung ist v entweder entfernt, oder v ist ein Blatt.

  Fall 1: v entfernt
  - D.h. deg(v)-1 Nachbarn von v wurden entfernt.
  - Pro Entfernen eines Nachbarn von v taucht v einmal im Prüferkode auf.
  - Danach war v ein Blatt und wurde entfernt.

  Fall 2: v ist Blatt
  - Wie zuvor nur ohne den dritten Schritt.

Invertieren von PK

Dekodierung des PK: $[n]^{n-2} \Rightarrow T$

Algorithmus Dekodierung

Eingabe: $t_1,\ldots,t_{n-2}$

1. $V \leftarrow \{1, 2,\ldots,n\}$
2. for i $\leftarrow$1 to n
   1. deg(i) $\leftarrow$ 1
   2. for j $\leftarrow$ 1 to n-2
      1. if ($i=t_j$) then deg(i) $\leftarrow$ deg(i)+1
3. for i $\leftarrow$ 1 to n-2
   1. Sei v_i kleinster Knoten mit $deg(v_i)=1$.
   2. $E \leftarrow E \cup \{v_i,t_i\}$
   3. $deg(v_i) \leftarrow deg(v_i)-1$; $deg(t_i) \leftarrow deg(t_i)-1$
4. Seien u, v Knoten mit deg(u)=deg(v)=1. $E \leftarrow E \cup \{u,v\}$.

Ausgabe: T=(V,E)

- Terminierung nach $\mathcal{O}(n^2)$ Schleifendurchläufen.$\cup$

Korrektheit

z.z.: Algorithmus Dekodierung fügt im i-ten Schritt diejenige Kante $\{v_i,t_i\}$ in T ein, die im i-ten Schritt von Algorithmus Kodierung entfernt wird.

- Schritt 1: Falls Prüferkodelänge n-2, dann gilt |V|=n.
- Schritt 2: Rekonstruktion aller Knotengrade aus T.
- Schritt 3: Analog zur Kodierung:
  - Bestimmung des kleinsten Blattes v_i
  - Hinzufügen anstatt Entfernen der Kante $\{v_i, t_i\}$
  - Anpassung der (Rest-)Grade von $\{v_i, t_i\}$

Speicherung von Graphen

Def: Sei G=(V,E) mit V=[n]. Die (n×n)-Matrix A=($a_{u,v}$) mit

$$a_{u,v} = \begin{cases} 1 & \text{falls } \{u,v\} \in E \\ 0 & \text{sonst} \end{cases}$$

heisst Adjazenzmatrix von G.

Def: Sei G=(V,E) mit V=[n].

Die Adjazenzlistendarstellung von G ist ein

- **Array von n verketteten Listen.**
- **Die i-te Liste beinhaltet die Nachbarn von i**
  - **vorzugsweise in aufsteigend sortierter Reihenfolge**

Vergleich der Darstellungen

	Adjazenz-Matrix	Adjazenz-Liste
Speicherbedarf	$\Theta(n^2)$	$\Theta(n+m)$, d.h. $o(n^2)$ für $m=o(n^2)$
$\{u,v\} \in E$?	$\mathcal{O}(1)$	$\mathcal{O}(\min\{\deg(u),\deg(v)\})$
Bestimme $\Gamma(v)$	$\mathcal{O}(n)$	$\Theta(\deg(v))$

Warteschlange

Datenstruktur Queue Q:

- Generieren einer Q:
  - Q $\leftarrow$ new Queue
- Einfügen:
  - Q.Enqueue(u) fügt u in Q ein.
- Prüfung, ob Q leer ist.
  - Q.Isempty()$\neq$TRUE $\Leftrightarrow$ Q enthält mindestens ein Element
- Entfernen: Falls Q.Isempty()$\neq$TRUE:
  - Q.Dequeue() liefert dasjenige Element, das am längesten in der Warteschlange Q ist

Queue ist FIFO-Datenstruktur (First in, first out).
Annahme: Laufzeit für Enqueue, Dequeue und Isempty ist $\mathcal{O}(1)$.

Breitensuche (Breadth First Search)

Algorithmus BFS

Eingabe: G=(V,E) in Adjazenzlistendarstellung mit Startknoten s $\in$ V

1. Für alle v $\in$ V
   1. if (v=s) then d[v] $\leftarrow$ 0 else d[v] $\leftarrow \infty$
   2. pred(v) $\leftarrow$ nil;
2. Q $\leftarrow$ new Queue;
3. Q.Enqueue(s);
4. while (Q.Isempty $\neq$ TRUE)
   1. v $\leftarrow$ Q.Dequeue(Q).
   2. Für alle u $\in \Gamma$(v)
      1. if (d[u]=∞) then
         1. d[u] $\leftarrow$ d[v]+1; pred[u] $\leftarrow$ v;
         2. Q.Enqueue(u);

Ausgabe: Arrays d[v], pred[v] für v $\in$ V.

Zusammenfassung

- Spannbaum
  - Algorithmisch: entferne Kreiskanten
  - Anzahl markierter Spannbäume, Prüferkode
- Darstellung von Graphen
  - Adjazenzliste
  - Adjazenzmatrix
- Datenstruktur Warteschlange
- BFS