Anzahl von Kanten in Wäldern

Satz Anzahl Kanten im Wald

Sei $G = (V, E)$ ein Wald mit k Bäumen. Dann besitzt G genau $|V| - k$ Kanten.

Beweis:

- Seien $G[V_1], \ldots, G[V_k]$ die Bäume in G.
- Jeder Baum $G[V_i]$, $i \in [k]$ besitzt $|V_i| - 1$ Kanten, d.h.

$$m = \sum_{i=1}^{k} (|V_i| - 1) = \left(\sum_{i=1}^{k} |V_i|\right) - k = |V| - k.$$

Spannbäume

Definition Spannbaum

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph. Ein schwacher Teilgraph $T = (V, E_T)$ von G heißt *Spannbaum* gdw T ein Baum ist.

- Jeder zusammenhängende Graph G besitzt einen Spannbaum.
- Spannbäume von G sind im Allgemeinen nicht eindeutig.
- Konstruktion eines Spannbaumes durch Entfernen von Kreisen.

Berechnung von Spannbäumen

Algorithmus SPANNBAUM

EINGABE: $G = (V, E)$ zusammenhängend

1. Solange $G = (V, E)$ einen Kreis K enthält.
   1. Wähle eine beliebige Kreiskante e aus K. Setze $E \leftarrow E \setminus \{e\}$.

AUSGABE: Spannbaum $T = (V, E)$ von G

Satz Korrektheit von SPANNBAUM

Sei $G = (V, E)$ zusammenhängend. Dann berechnet SPANNBAUM einen Spannbaum T von G in $|E| - |V| + 1$ Schleifendurchläufen.

Korrektheit und Laufzeit von SPANNBAUM

Beweis:

- **Korrektheit:** Wir zeigen, dass G stets zusammenhängend bleibt.
- Sei $K = (v_0, \ldots, v_{k-1})$ ein Kreis in G.
- Wir entfernen eine Kreiskante $e = \{v_i, v_{i+1 \bmod k}\}$, $i \in \mathbb{Z}_k$.
- Seien u, v Knoten mit einem u-v Pfad, der e verwendet.
- Ersetze e durch den Restkreis $(v_i, v_{i-1}, \ldots, v_0, \ldots, v_{i+1})$.
- Bei Terminierung ist G zusammenhängend und kreisfrei.
- SPANNBAUM muss terminieren, da die Kantenzahl sukzessive verringert wird.

- **Laufzeit:** T ist ein Spannbaum und besitzt daher $|V| - 1$ Kanten.
- Jeder Schleifendurchlauf verringert die Kantenanzahl um Eins.
- D.h. die Anzahl Schleifendurchläufe ist $|E| - (|V| - 1)$.

Markierte und isomorphe Spannbäume

Bsp:

- Der vollständige Graph K_2 ist selbst ein Spannbaum.
- Der K_3 besteht aus einem Kreis der Länge 3.
- Durch Entfernen einer Kanten entsteht ein Spannbaum.
- Alle 3 Spannbäume sind isomorph, sie unterscheiden sich lediglich durch die Knotenmarkierungen.
- Der K_4 besitzt 2 nicht-isomorphe Spannbäume, den C_4 und einen Sterngraphen mit einem Knoten vom Grad 3.

# Bäume \ $\lvert V\rvert$	2	3	4	5	6	7	8
markiert	1	3	16	125	1296	16807	262144
nicht-isomorph	1	1	2	3	6	11	23

Satz von Cayley

Satz von Cayley

Für $n \geq 2$ gibt es genau n^{n-2} markierte Bäume.

Beweisidee:

- Sei $V = [n]$.
- Sei T_n die Menge der markierten Bäume mit n Knoten.
- Wir definieren eine bijektive Kodierung $f\colon T_n \to [n]^{n-2}$.
- Diese Kodierung bezeichnen wir als den *Prüfercode* eines Baums.
- Anwenden von f entspricht dem Kodieren im Prüfercode.
- Anwenden von f^{-1} entspricht dem Dekodieren im Prüfercode.
- Da f bijektiv ist, folgt $|T_n| = n^{n-2}$.

Berechnung von f

Algorithmus KODIERUNG

EINGABE: Baum $T = ([n], E) \in T_n$

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

AUSGABE: Prüfercode $(t_1, \ldots, t_{n-2})$

Laufzeit:

- In jedem Schleifendurchlauf wird ein Knoten entfernt.
- D.h. KODIERUNG terminiert nach $|V| - 2$ Durchläufen.

Korrektheit von KODIERUNG

Korrektheit:

- Zeigen: In jedem Baum T mit $n > 2$ Knoten existiert ein Blatt.
- T besitzt genau $|E| = n - 1$ Kanten.
- Annahme: Alle Knoten besitzen Grad mindestens 2.
- Mit Handschlaglemma gilt

$$2 \cdot |E| = 2(n-1) = \sum_{v \in V} \deg(v) \geq 2n.$$

- Widerspruch, d.h. es muss einen Knoten mit Grad 1 geben.

- Zeigen nun: Durch Entfernen eines Blattes w bleibt T ein Baum.
- Seien $u, v \neq w$ Knoten mit u-v Pfad $p = (v_0 = u, \ldots, v_k = v)$.
- w kann nicht in p enthalten sein, da alle inneren Knoten $v_1, \ldots, v_{k-1}$ des Pfades mindestens Grad 2 besitzen.

Prüfercode und Grad der Knoten

Lemma Knoten im Prüfercode

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

Beweis:

- Blätter von T kommen im Prüfercode nicht vor.
- Sei v ein innerer Knoten von T, d.h. $\deg(v) \geq 2$.

Fall 1: v ist bei Terminierung von KODIERUNG entfernt.

- Dann war v im Laufe des Algorithmus ein Blatt.
- Dazu wurden zunächst $\deg(v) - 1$ Nachbarn von v entfernt.
- Pro Entfernen eines Nachbarn taucht v einmal im Prüferkode auf.

Fall 2: v ist bei Terminierung nicht entfernt.

- Analog zu Fall 1: $\deg(v) - 1$ Nachbarn wurden bereits entfernt.

Dekodieren des Prüfercodes

Algorithmus DEKODIERUNG

EINGABE: $t_1, \ldots, t_{n-2} \in [n]$

1. $V \leftarrow [n]$
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 kleinster Knoten mit $\deg(v) = 1$.
   2. $E \leftarrow E \cup \{v, t_i\}$
   3. $\deg(v) \leftarrow \deg(v) - 1$; $\deg(t_i) \leftarrow \deg(t_i) - 1$
4. Seien u, v die verbliebenen Knoten mit Grad 1. $E \leftarrow E \cup \{u, v\}$.

AUSGABE: Baum $T = (V, E)$

Korrektheit der DEKODIERUNG

Laufzeit:

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

Korrektheit:

- Schritt 1: Rekonstruktion der Knotenanzahl $|V| = n$.
- Schritt 2: Rekonstruktion aller Knotengrade aus T. Korrektheit folgt aus zuvor gezeigtem Lemma.
- Schritt 3: Erfolgt analog zur Kodierung:
  - Bestimmung des Blattes v mit kleinster Markierung.
  - Hinzufügen anstatt Entfernen der Kante $\{v, t_i\}$.
  - Anpassen der verbleibenden Nachbarzahl von v, t_i.
- D.h. DEKODIERUNG fügt in der i-ten Iteration von Schritt 3 diejenige Kante $\{v, t_i\}$ in T ein, die im i-ten Schritt von KODIERUNG entfernt wurde.

Speicherung von Graphen

Definition Adjazenzmatrix und Adjazenzliste

Sei $G = ([n], E)$ ein ungerichteter Graph.

1. Die *Adjazenzmatrix* $A = (a_{u,v}) \in \{0,1\}^{n \times n}$ von G ist definiert als
$$a_{u,v} = \begin{cases} 1 & \text{falls } \{u,v\} \in E \\ 0 & \text{sonst} \end{cases}.$$
2. Die *Adjazenzliste* $A[1 \ldots n]$ von G ist ein Array von n verketteten Listen. Die i-te verkettete Liste $A[i]$ beinhaltet alle Nachbarn von i in aufsteigend sortierter Reihenfolge.

Vergleich der Darstellungen: Sei $\min_{u,v} := \min\{\deg(u), \deg(v)\}$.

	Speicherbedarf	$\{u,v\} \in E$?	Bestimme $\Gamma(v)$
Adjazenzmatrix	$\Theta(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Adjazenzliste	$\Theta(n+m)$	$\mathcal{O}(\min_{u,v})$	$\Theta(\deg(v))$

Datenstruktur Warteschlange

Datenstruktur Warteschlange

Sei U eine Menge. Eine *Warteschlange bzw. Queue* Q ist eine Datenstruktur auf U mit den folgenden Operationen:

1. Generieren einer Queue: $Q \leftarrow$ new Queue.
2. Einfügen von $u \in U$ in Q: Q.Enqueue(u).
3. Prüfung auf Leerheit von Q: Q.Isempty().
4. Entfernen des *zuerst* eingefügten Elements in Q: Q.Dequeue().

Bemerkungen:

- Queue ist eine FIFO-Datenstruktur, d.h. first in, first out.
- Wir nehmen an, dass alle Operationen Laufzeit $\mathcal{O}(1)$ benötigen.

Datenstruktur Stack

Datenstruktur Stack

Sei U eine Menge. Ein *Stapel bzw. Stack* S ist eine Datenstruktur auf U mit den folgenden Operationen:

1. Generieren eines Stacks S: $S \leftarrow$ new Stack.
2. Einfügen von u in S: S.Push(u).
3. Test auf Leerheit von S: S.Isempty()
4. Entfernen des *zuletzt* eingefügten Elements in S: S.Pop()

Bemerkungen:

- Stack ist eine LIFO-Datenstruktur, d.h. last in, first out.
- Wir nehmen an, dass alle Operationen Laufzeit $\mathcal{O}(1)$ benötigen.