Wiederholung

- Knotenfärbung von Graphen
  - Chromatische Zahl χ(G)
  - Beweis: Jeder planare Graph ist 5-färbbar
  - Vierfarbensatz: Jeder planare Graph ist 4-färbbar.
  - Kantenfärbung: χ'(G) = Δ(G) oder Δ(G)+1
- Matchings M
  - Heiratsproblem
  - Bipartite Graphen G=(A $\uplus$ B, E)
    - Satz von Hall: M=|A| $\Leftrightarrow$ |X| $\leq$ |Γ(X)|

Heiratssatz

Satz(Hall): Sei G=(A $\uplus$ B, E) bipartit.

G enthält Matching M der Größe |M|=|A|

$\Leftrightarrow$ $|\cup_{x \in X} \Gamma(x)| = |\Gamma(X)| \geq |X|$ für alle X $\subseteq$ A.

„$\Rightarrow$“: Sei M Matching, |M|=|A|.

- Betrachte G‘=(A $\uplus$ B,M).
- Jedes X $\subseteq$ A hat in G‘ genau |X| Nachbarn

  $\Rightarrow$ Jedes X $\subseteq$ A hat in G mindestens |X| Nachbarn.

$|\Gamma(X)| \geq |X| \Rightarrow$ Matching, $|M| = |A|$

Ann.: G=(A $\uplus$ B, E) hat max. Matching M, |M|<|A|
$\Rightarrow \exists$ nicht überdecktes $a_1 \in A$ mit Nachbarn b_1.Existenz von b_1 wegen $|\Gamma(\{a_1\})| \geq 1$.

Algorithmus Augmentierender-Pfad
Eingabe: G=(A $\uplus$ B, E), M, a_1, b_1

1. k $\leftarrow$ 1
2. while (b_k wird von M überdeckt)
   1. $a_{k+1} \leftarrow$ Nachbar von b_k im Matching M
   2. $b_{k+1} \leftarrow$ beliebiges v $\in \Gamma(\{a_1,\ldots,a_{k+1}\}) \setminus \{b_1, \ldots, b_k\}$
   3. k $\leftarrow$ k+1

Ausgabe: augmentierender Pfad $p_a = (a_1, b_1, \ldots, a_k, b_k)$

- Korrektheit: b_{k+1} existiert wegen $|\Gamma(\{a_1,\ldots,a_{k+1}\}) \setminus \{b_1,\ldots,b_k\}| \geq (k+1)-k = 1$
- $\{a_i, b_i\} \notin M$ für i=1,…,k: k Kanten nicht in M
- $\{b_i, a_{i+1}\} \in M$ für i=1,…,k-1: k-1 Kanten in M
- a_1, b_k nicht überdeckt.
  - Nimm $\{a_i, b_i\}$ in Matching auf und $\{b_i, a_{i+1}\}$ aus Matching raus.
  - M wird um Eins größer (Widerspruch zur Maximalität von M)

Konstruktion eines maximalen Matchings

Algorithmus MaxMatching

Eingabe: $G=(A \uplus B, E)$ mit $|\Gamma(X)| \geq |X|$ für alle $X \subseteq A$

- $M \leftarrow \emptyset$
- while (es gibt nichtüberdeckten $a_i \in A$)
  - $b_1 \leftarrow$ Nachbar von a_i
  - $p_a=(a_1,b_1,\ldots,a_k,b_k) \leftarrow$ Augmentierender-Pfad(G,M,a_1,b_1)
  - for i $\leftarrow$ 1 to k
    - $M=M \cup \{a_i,b_i\}$; if(i<k) $M=M \setminus \{b_i,a_{i+1}\}$

Ausgabe: Matching M mit $|M|=|A|$

Korrektheit:

- M wird in jeder Iteration um ein Element vergrößert.
- Nach |A| Iterationen gilt $|M|=|A|$.

k-reguläre bipartite Graphen

Satz: Sei G=(A ⊎ B, E) ein k-regulärer bipartiter Graph. Dann gilt:

(1) G besitzt ein perfektes Matching.

(2) χ'(G) = k.

zu (1): Ann: $\exists$ X $\subseteq$ A: |Γ(X)| < |X|

- Betrachte Multimenge M=$\cup_{x \in X}$ Γ(x) mit |M|=k*|X|
- Verallgemeinertes Schubfachprinzip:

  $\exists$ b $\in$ Γ(X) mit deg(b) $\geq$ $\lceil$|M|/|Γ(X)$\rceil$ > k*|X|/|X| = k
  (Widerspruch: G ist k-regulär)

$\chi'(G) = k$

zu (2):

- $|E| = \sum_{a \in A} \deg(a) = |A| * k = \sum_{b \in B} k \Rightarrow |A| = |B|$

Induktion über k

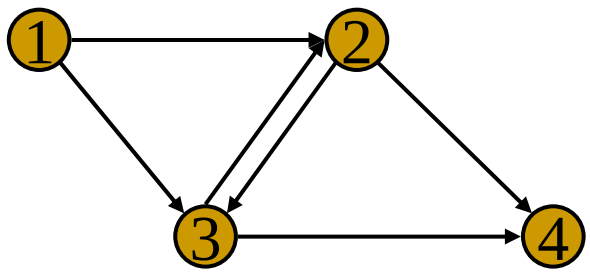
- IA: k=1: G besitzt perfektes Matching E:
  - Jedes $a \in A$ ist mit einem $b \in B$ verbunden.
  - M=E und alle Kanten können mit 1 gefärbt werden.
- IS: k-1 $\rightarrow$ k
  - G besitzt ein perfektes Matching M.
  - G'=(A $\uplus$ B, E $\setminus$ M) ist (k-1)-regulär:
    - G' besitzt (k-1)-Kantenfärbung nach IV.
  - Färbe alle Kanten des Matchings M mit der Farbe k.

Gerichtete Graphen D

Def: Gerichteter Graph oder Digraph D=(V,E) mit

- **V Menge der Knoten, |V|=n**
- **E $\subseteq$ V$\times$V Menge der gerichteten Kanten, |E|=m**

Bsp.: D=([4], (1,2),(1,3),(2,3),(2,4),(3,2),(3,4))



Analog zu ungerichteten Graphen:

- gerichteter Weg (1,2,3,2,4)
- gerichteter Pfad (1,2,3,4) der Länge 3
- Kürzester 1-4-Pfad (1,3,4) der Länge (mittels BFS)
- gerichteter Kreis (2,3) der Länge 2
- $\Gamma(3) = \{2,4\}$

DAG und topologische Sortierung

Def: Ein Digraph D=(V,E) heisst DAG, falls D kreisfreis ist.

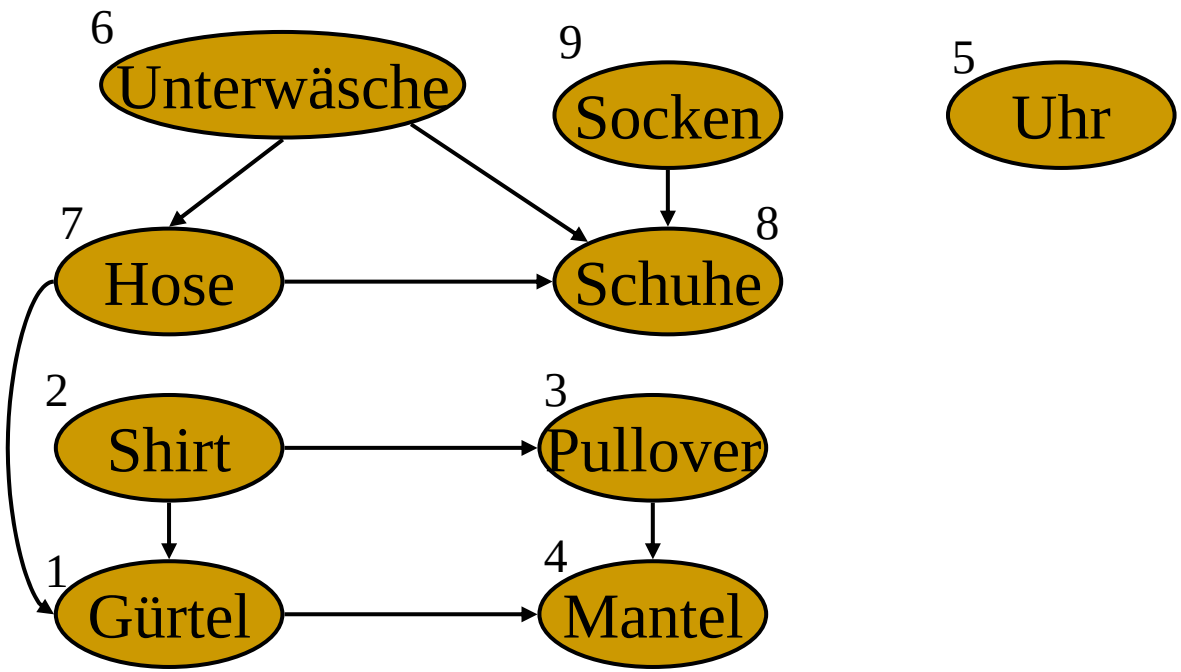
Algorithmus Topol-Sort (DFS-Variante)

Eingabe: DAG D=(V,E) in Adjazenzlistendarstellung

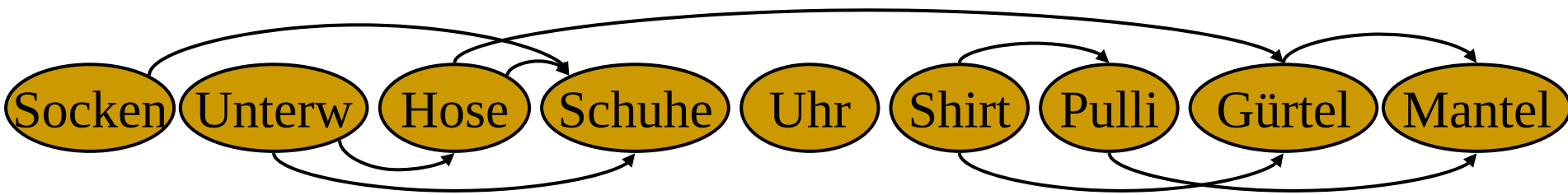
1. Für alle v $\in$ V
   1. pred[v] $\leftarrow$ nil; f[v]=0;
2. i $\leftarrow$ n; S $\leftarrow$ new Stack;
3. while (es gibt unbesuchten Knoten s)
   1. s $\leftarrow$ minimaler unbesuchter Knoten; S.Push(s);
   2. While (S.Isempty() $\neq$ TRUE)
      1. v $\leftarrow$ S.Pop();
         1. if ($\exists$ u $\in$ Γ(v) mit pred[u] = nil) then
            1. S.Push(v); S.Push(u);
            2. pred[u] $\leftarrow$ v
         2. else if (f[v]=0) then
            1. f[v] $\leftarrow$ i; i $\leftarrow$ i-1;

Ausgabe: pred[v], f[v] für alle v $\in$ V

Ankleideproblem



Topologische Sortierung



Relationen vs. Graphen

- Sei D=(V,A).
  - $A \subseteq V \times V$ ist Relation auf V.
- Sei $R \subseteq S \times S$ Relation auf S.
  - D=(S, R) definiert einen Graphen.

Poset $(S, \preceq)$: reflexiv, antisymmetrisch, transitiv

- Definiert DAG (Weglassen der Selbstkanten {v,v}).
- Topologische Sortierung liefert lineare Erweiterung $(S, \preceq_L)$:

  $x \preceq y \Rightarrow x \preceq_L y$ für alle $x,y \in S$

  $\Leftrightarrow d[x] \leq d[y]$ in topologischer Sortierung

Starker Zusammenhang

Sei D=(V,E) ein gerichteter Graph.

Def: G=(V,E') mit {u,v}$\in$ E' $\Leftrightarrow$ (u,v) $\in$ E oder (v,u) $\in$ E
ist zugrundeliegender Graph von D.

Def: D ist stark zusammenhängend
$\Leftrightarrow \exists$ u-v-Pfad für alle u,v $\in$ V
D ist (schwach) zusammenhängend
$\Leftrightarrow$ zugrundeliegender Graph G ist zusammenhängend

Algorithmen zum Testen von starkem Zusammenhang:

- Führe DFS für alle Startknoten s$\in$ V durch: $\mathcal{O}$(n(n+m))
- Verbesserter Algorithmus mit Laufzeit $\mathcal{O}$(n+m):
  - Sei $D^r=(V,E^r)$ mit (u,v) $\in E^r \Leftrightarrow$ (v,u) $\in$ E
  - Anwendung von DFS auf D und D^r genügt (Beweis nicht-trivial).

Transitive Hülle

Algorithmus Simple-Transitiv

Eingabe: Relation R auf S

1. while ($\exists$ x,y,z mit (x,y),(y,z) $\in$ R und (x,z)$\notin$R) do
   1. R $\leftarrow$ R $\cup$ {(x,z)}

Ausgabe: Transitive Hülle R^+ = R

Korrektheit: klar

Laufzeit $\mathcal{O}(|S|^5)$:

- $|R^+| \leq |S|^2$
- In jeder Iteration wird eine „Kante“ (x,z) hinzugefügt.
- Überprüfung der while-Bedingung in $\mathcal{O}(|S|^3)$:
  - R in Adjazenzmatrixdarstellung (r_{ij})
  - Überprüfe für jedes x,y,z ob r_{xy}=1 und r_{yz}=1 und r_{xz}=0.

Mittels Dynamischer Programmierung

Sei D=(V,E). Transitive Hülle $D^+=(V,E^+)$ von D hat die Kantenmenge

$$E^+ = \{(u,v) \in V \times V \mid \exists \text{ u-v-Pfad in D}\}.$$

Definieren $W_k[i,j]$ =

- 1, falls $\exists$ i-j-Pfad mit inneren Knoten aus $\{1,\ldots,k\}$
- 0, sonst

- Initiale Werte: $W_0[i,j] = 1 \Leftrightarrow (i,j) \in E$
- Finale Werte: $W_n[i,j] = 1 \Leftrightarrow (i,j) \in E^+$

Iterative Berechnung

Berechnung von $W_k[i,j]$ aus $W_{k-1}[i,j]$:

$W_k[i,j] = 1$:

- $\exists$ i-j-Pfad mit inneren Knoten aus {1,…,k-1}
- $\exists$ i-k-Pfad und k-j-Pfad mit inneren Knoten aus {1,…,k-1}

$\Rightarrow W_k[i,j] = \max\{W_{k-1}[i,j], W_{k-1}[i,k] * W_{k-1}[k,j]\}$

Algorithmus von Warschall

Algorithmus von Warschall

Eingabe: D=(V,E)

1. for i=1 to n
    1. for j=1 to n
        1. if (i,j)$\in$ E then W[i,j] $\leftarrow$ 1;
        2. else W[i,j] $\leftarrow$ 0;
2. for k=1 to n
    1. for i=1 to n
        1. for j=1 to n
            1. W[i,j] $\leftarrow$ max{W[i,j], W[i,k]*W[k,j]}

Ausgabe: $E^+ \leftarrow$ {(i,j) | W[i,j]=1}

Analyse von Warschalls Algorithmus

Satz: Warschalls Algorithmus berechnet die transitive Hülle eines Graphen in Zeit $\mathcal{O}(n^3)$.

Korrektheit:

- $W_k[i,j] = \max\{W_{k-1}[i,j], W_{k-1}[i,k] * W_{k-1}[k,j]\}$ und
- $W_{k-1}[i,k]=1 \Leftrightarrow W_k[i,k]=1$ bzw. $W_{k-1}[k,j]=W_{k,j}$:

  „$\Rightarrow$“: klar

  „$\Leftarrow$“: $\exists$ $p=(i,v_1,\ldots,v_r=k)$ mit inneren Knoten aus $\{1,\ldots,k\}$.
  Pfadeigenschaft: $v_i \neq v_r=k$ für $i<r$
  $\Rightarrow$ p ist i-j-Pfad mit inneren Knoten $v_1,\ldots,v_{r-1} \in \{1,\ldots,k-1\}$

Laufzeit: $\mathcal{O}(n^3)$

Wurzelbäume

Sei T=(V,E) ein Baum.

- T_v bezeichne Baum mit Wurzel v.
- Von jedem Knoten $u \in V$ gibt es genau einen u-v-Pfad = $(u,v_1,\ldots,v_k=v)$
  - $v_1,\ldots,v_k$ sind Vorgänger von u
  - v_1 ist Elternknoten bzw. Vaterknoten von v.
  - u hat Tiefe k im Baum T_v
  - Höhe $h(T_v) = \max_{u \in V}\{\text{Tiefe von u in } T_v\}$
- Knoten mit gleichem Elternknoten heißen Geschwisterknoten
- Knoten w mit w-u-Pfad heissen
  - Nachfolger von u
  - Falls (w,u) in E, nennt man w ein Kind von u.

Binärbäume

- Binärbaum: Jeder Knoten hat höchstens zwei Kinder.
- Vollständiger Binärbaum:
  Jedes Nichtblatt hat zwei Kinder, Blätter haben gleiche Tiefe.

Realisierung vollständiger Binärbäume als Array:

- Knoten in Tiefe t erhalten Indizes $2^t, \ldots, 2^{t+1}-1$
- Knoten i:
  - linkes Kind: 2i
  - rechtes Kind: 2i+1
  - Vater: $\lfloor i/2 \rfloor$
- Sei h die Höhe des Baums.
  - $n = 2^{h+1}-1$ bzw. $h = \Theta(\log n)$

Verwendung von Arraydarstellung z.B. bei Heapsort.

Binäre Suchbäume

Sei T=(V,E) mit $V \subseteq \mathbb{Z}$ beliebig.

Def: Ein Binärbaum T_w heisst Suchbaum, wenn für alle $v \in V$ gilt:

- **Für alle Knoten u_l im linken Teilbaum von v: $u_l \leq v$.**
- **Für alle Knoten u_r im rechten Teilbaum von v: $u_r \geq v$.**

Algorithmus Suche-Element

Eingabe: Suchbaum T_w, u

1. while ($w \neq u$ und w ist kein Blatt)
   1. if $u \leq w$ then w$\leftarrow$ linkes Kind von w.
   2. else w $\leftarrow$ rechtes Kind von w.

Ausgabe: „u gefunden“, falls w=u und „u nicht gefunden“ sonst

Laufzeit: $\mathcal{O}(h(T_w))$

- Verschiedene Strategien, um $h(T_w)=\mathcal{O}(\log n)$ zu erzwingen
  - Höhen- und gewichtsbalancierte Bäume, Rot-Schwarz-Bäume, etc.