

NP-Vollständigkeit Spezialfälle

Algorithmen und Datenstrukturen
VU 186.866, 5.5h, 8 ECTS, 2024S

Letzte Änderung: 13. Mai 2024

Vorlesungsfolien

ac  ALGORITHMS AND
COMPLEXITY GROUP



Informatics

NP-Vollständigkeit meistern (Wiederholung)

Frage: Angenommen, wir müssen ein NP-vollständiges Problem lösen. Wie sollen wir vorgehen?

Antwort: Die Theorie besagt, dass es unwahrscheinlich ist, einen polynomiellen Algorithmus zu finden.

Man muss eine der gewünschten Eigenschaften opfern:

- Löse Problem optimal
→ Approximationsalgorithmen, Heuristische Algorithmen
- Löse Problem in Polynomialzeit
→ Algorithmen mit exponentieller Laufzeit
- Löse **beliebige** Instanzen des Problems
→ Identifiziere effizient lösbare Spezialfälle

Überblick

Finden von kleinen Vertex Covers

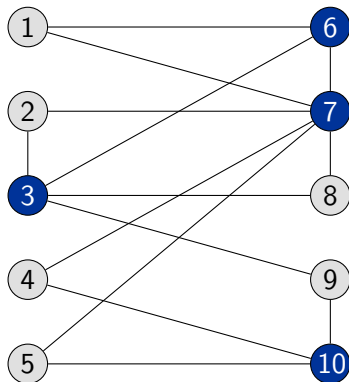
NP-vollständige Probleme auf Bäumen

Knotenfärben in Intervallgraphen

Finden von kleinen Vertex Covers

Vertex Cover (Knotenüberdeckung)

VERTEX COVER: Gegeben sei ein Graph $G = (V, E)$ mit $|V| = n$ Knoten und eine ganze Zahl k . Existiert eine Teilmenge von Knoten $S \subseteq V$, sodass $|S| \leq k$, und für jede Kante $(u, v) \in E$ gilt, dass $u \in S$ oder $v \in S$ (oder beides)?



$$k = 4$$

$$S = \{3, 6, 7, 10\}$$

Finden kleiner Vertex Covers

Frage: Was ist möglich, wenn k klein ist?

Brute-Force. $O(kn^{k+1})$.

- Probiere alle $\binom{n}{k} = O(n^k)$ Teilmengen der Größe k aus.
- Benötigt $O(kn)$ Zeit um zu überprüfen, ob eine Teilmenge ein Vertex Cover ist.

Ziel: Beschränke die exponentielle Abhängigkeit von k .

Beispiel: $n = 1000$, $k = 10$.

Brute-Force: $kn^{k+1} = 10^{34} \Rightarrow$ undurchführbar.

Besser: $2^k kn \approx 10^7 \Rightarrow$ durchführbar.

Anmerkung: Wenn k eine Konstante ist, dann ist der Algorithmus polynomiell. Wenn k eine kleine Konstante ist, dann ist er auch praktikabel.

Finden kleiner Vertex Covers

Behauptung: Sei (u, v) eine Kante von G . G hat ein Vertex Cover der Größe $\leq k$ genau dann, wenn zumindest einer der Graphen $G - \{u\}$ und $G - \{v\}$ ein Vertex Cover der Größe $\leq k - 1$ hat.

■ Lösche u und alle inzidenten Kanten.

Beweis: \Rightarrow

- Angenommen G hat ein Vertex Cover S der Größe $\leq k$.
- S enthält entweder u oder v (oder beide). Angenommen, S enthält u .
- Dann ist $S - \{u\}$ ein Vertex Cover von $G - \{u\}$.

Beweis: \Leftarrow

- Angenommen S ist ein Vertex Cover von $G - \{u\}$ der Größe $\leq k - 1$.
- Dann ist $S \cup \{u\}$ ein Vertex Cover von G . □

Finden kleiner Vertex Covers

Behauptung: Wenn G ein Vertex Cover der Größe k hat, dann hat $G \leq k(n - 1)$ Kanten.

Beweis: Jeder Knoten überdeckt höchstens $n - 1$ Kanten. □

Finden kleiner Vertex Covers: Algorithmus

Behauptung: Der folgende Algorithmus bestimmt mit einer Laufzeit in $O(2^k kn)$, ob G ein Vertex Cover der Größe $\leq k$ hat.

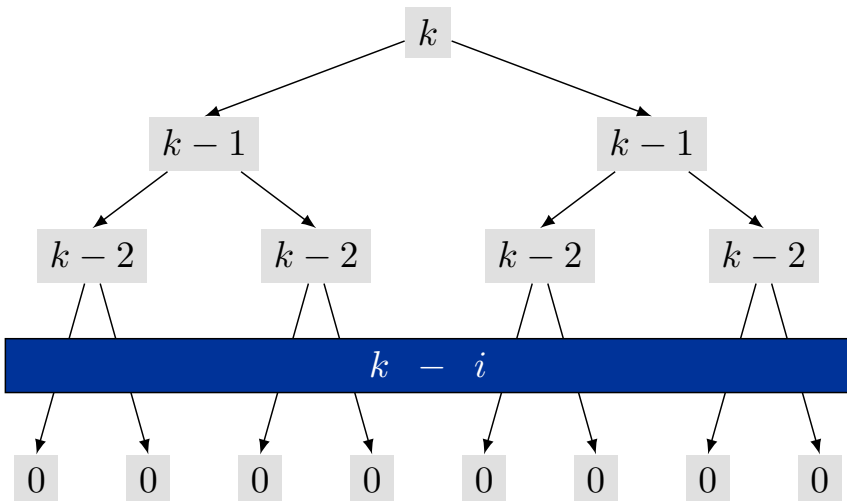
```
Vertex-Cover( $G, k$ ):  
if  $G$  enthält keine Kanten  
    return true  
if  $G$  enthält  $> k(n - 1)$  Kanten  
    return false  
Sei  $(u, v)$  eine beliebige Kante von  $G$   
 $a \leftarrow$  Vertex-Cover( $G - \{u\}, k - 1$ )  
 $b \leftarrow$  Vertex-Cover( $G - \{v\}, k - 1$ )  
return  $a$  OR  $b$ 
```

Beweis:

- Korrektheit folgt aus den zwei vorherigen Behauptungen.
- Es existieren $\leq 2^{k+1}$ Knoten im Rekursionsbaum. Jeder Aufruf benötigt $O(kn)$ Zeit. □

Finden kleiner Vertex Covers: Rekursionsbaum

$$T(n, k) \leq \begin{cases} c & \text{falls } k = 0 \\ cn & \text{falls } k = 1 \\ 2T(n-1, k-1) + ckn & \text{falls } k > 1 \end{cases} \Rightarrow T(n, k) \leq 2^k ckn$$



Lösen NP-vollständiger Probleme auf Bäumen

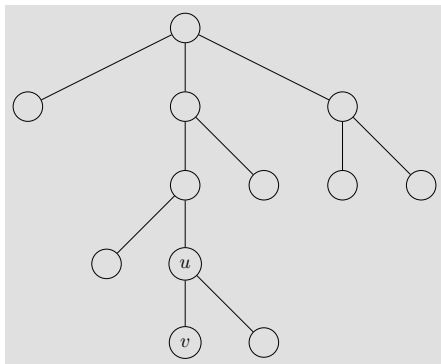
Independent Set auf Bäumen

Independent Set auf Bäumen: Gegeben sei ein **Baum**. Finde die größte Teilmenge von Knoten, sodass keine zwei Knoten durch eine Kante verbunden werden.

Tatsache: Ein Baum mit zumindest zwei Knoten hat zumindest zwei **Blätter**.

□ $\text{Grad} = 1$

Beobachtung: Wenn v ein Blatt ist, dann existiert ein maximales Independent Set, das v enthält.



Beweis: (Austauschargument)

- Wir gehen von einem maximalen Independent Set S aus.
- Wenn $v \in S$, dann ist man fertig.
- Wenn $v \notin S$ und $u \notin S$, dann ist $S \cup \{v\}$ unabhängig $\Rightarrow S$ ist nicht maximal.
- Wenn $v \notin S$ und $u \in S$, dann ist $S \cup \{v\} - \{u\}$ unabhängig.

□

Independent Set auf Bäumen: Greedy-Algorithmus

Theorem: Der nachfolgende Greedy-Algorithmus findet ein maximales Independent Set in einem **Wald** $T = (V, E)$ (jede **Zusammenhangskomponente** des Graphen ist ein **Baum**).

```
Independent-Set-In-A-Forest( $T$ ):  
 $S \leftarrow \emptyset$   
while  $T$  hat zumindest eine Kante  
    Sei  $e = (u, v)$  eine Kante, sodass  $v$  ein Blatt ist  
    Füge  $v$  zu  $S$  hinzu  
    Lösche aus  $V$  die Knoten  $u$  und  $v$  (und alle  
    zu diesen Knoten inzidenten Kanten)  
 $S \leftarrow S \cup V$   
return  $S$ 
```

Beweis: Korrektheit folgt aus der vorherigen Beobachtung. □

Anmerkung: Kann man in $O(n)$ Zeit implementieren, indem man die Knoten in Postorder durchmustert.

Gewichtetes Independent Set auf Bäumen

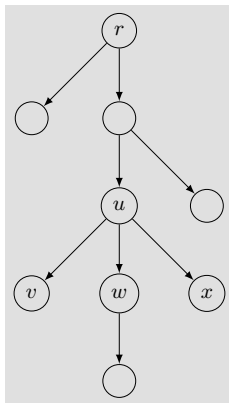
Gewichtetes Independent Set auf Bäumen: Gegeben sei ein Baum und Knotengewichte $w_v > 0$. Finde ein Independent Set S , das $\sum_{v \in S} w_v$ maximiert.

Beobachtung: Wenn (u, v) eine Kante ist, sodass v ein Blatt ist, dann beinhaltet die Lösung entweder u oder sie enthält alle Blattknoten inzident zu u .

Gewichtetes Independent Set auf Bäumen

Lösung mit dynamischer Programmierung: Wähle einen Knoten, z.B. r , als Wurzel aus.

- $OPT_{in}(u)$ = maximales Gewicht eines Independent Sets des Unterbaums mit Wurzel u , das u enthält.
- $OPT_{out}(u)$ = maximales Gewicht eines Independent Sets des Unterbaums mit Wurzel u , das u nicht enthält.



$$\text{Nachfolger}(u) = \{v, w, x\}$$

Formulierung:

$$\begin{aligned} OPT_{in}(u) &= w_u + \sum_{v \in \text{Nachfolger}(u)} OPT_{out}(v) \\ OPT_{out}(u) &= \sum_{v \in \text{Nachfolger}(u)} \max \{OPT_{in}(v), OPT_{out}(v)\} \end{aligned}$$

Gewichtetes Independent Set auf Bäumen

Formulierung:

$$\begin{aligned}OPT_{\text{in}}(u) &= w_u + \sum_{v \in \text{Nachfolger}(u)} OPT_{\text{out}}(v) \\OPT_{\text{out}}(u) &= \sum_{v \in \text{Nachfolger}(u)} \max \{OPT_{\text{in}}(v), OPT_{\text{out}}(v)\}\end{aligned}$$

Erklärung:

- $OPT_{\text{in}}(u)$ addiert das Gewicht von u (u ist enthalten) und die maximalen Gewichte aller Unterbäume, bei denen die Wurzeln (Nachfolger von u) nicht enthalten sind. Wäre eine Wurzel enthalten, dann wäre es kein Independent Set mehr, da dann diese Wurzel mit u eine Kante gemeinsam hätte.
- $OPT_{\text{out}}(u)$ addiert die maximalen Gewichte aller Unterbäume. Bei einem Unterbaum kann die Wurzel v enthalten sein oder nicht. Zwei Wurzeln zweier Unterbäume von u können niemals miteinander verbunden sein.

Gewichtetes Independent Set auf Bäumen: Implementierung

```
Weighted-Independent-Set-In-A-Tree( $T$ ):  
Wähle eine Wurzel  $r$  aus  
foreach Knoten  $u$  von  $T$  in Postorder  
  if  $u$  ist ein Blatt  
     $M_{\text{in}}[u] \leftarrow w_u$   
     $M_{\text{out}}[u] \leftarrow 0$   
  else  
     $M_{\text{in}}[u] \leftarrow w_u + \sum_{v \in \text{Nachfolger}(u)} M_{\text{out}}[v]$   
     $M_{\text{out}}[u] \leftarrow \sum_{v \in \text{Nachfolger}(u)} \max\{M_{\text{in}}[v], M_{\text{out}}[v]\}$   
return  $\max\{M_{\text{in}}[r], M_{\text{out}}[r]\}$ 
```

- *Stellt sicher, dass ein Knoten nach seinen Nachfolgern besucht wird.*

Gewichtetes Independent Set auf Bäumen: Dynamische Programmierung

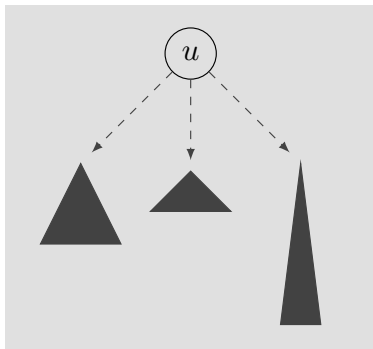
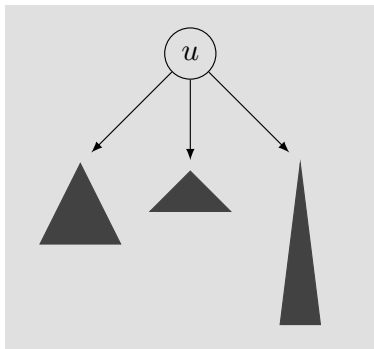
Theorem: Der Algorithmus auf der vorherigen Folie findet ein maximal gewichtetes Independent Set in einem Baum mit einer Laufzeit in $O(n)$.

Kann auch das Independent Set (und nicht nur den Wert) finden.

Beweis: Erfordert $O(n)$ Zeit, da wir Knoten in Postorder durchmustern und jede Kante genau einmal überprüfen. □

Kontext

Independent Set auf Bäumen: Dieser strukturierte Spezialfall ist handhabbar, weil wir einen Knoten finden können, der die **Verbindung** zwischen den Subproblemen in verschiedenen Subbäumen unterbrechen kann.



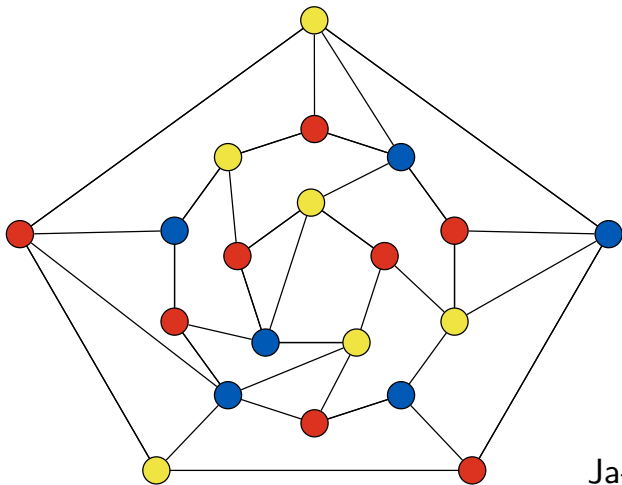
Graphen mit beschränkter Baumweite (elegante Generalisierung von Bäumen):

- Erfassen eine reichhaltige Klasse von Graphen, die in der Praxis auftritt.
- Erlauben die Aufteilung in unabhängige Teile.

Knotenfärben in Intervallgraphen

Knotenfärbeprobem

Erinnerung 3-COLOR: Gegeben sei ein ungerichteter Graph G . Kann man die Knoten des Graphen mit den Farben Rot, Gelb und Blau so einfärben, dass benachbarte Knoten nicht die gleiche Farbe besitzen?



Ja-Instanz

Knotenfärbeprobem

Erinnerung 3-COLOR: Gegeben sei ein ungerichteter Graph G . Kann man die Knoten des Graphen mit den Farben Rot, Gelb und Blau so einfärben, dass benachbarte Knoten nicht die gleiche Farbe besitzen?

k -COLOR: Gegeben sei ein ungerichteter Graph G . Kann man die Knoten des Graphen mit $k \geq 3$ Farben so einfärben, dass benachbarte Knoten nicht die gleiche Farbe besitzen?

k -COLOR ist NP-vollständig für $k \geq 3$

Als Optimierungsproblem:

OPT-COLOR: Gegeben sei ein ungerichteter Graph G . Färbe die Knoten des Graphen mit einer minimalen Anzahl Farben so, dass benachbarte Knoten nicht die gleiche Farbe besitzen.

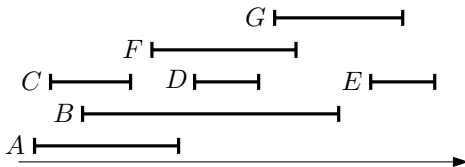
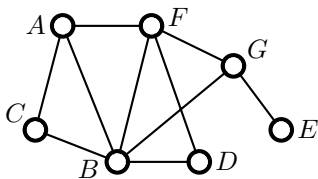
Ziel: Betrachte effizient lösbaren Spezialfall **Intervallgraphen**

Intervallgraphen

Definition: Intervallgraphen sind Graphen, die sich wie folgt als Schnittgraph von Intervallen in \mathbb{R} repräsentieren lassen:

- ungerichteter Graph $G = (V, E)$
- Intervallmenge $\mathcal{I} = \{I_v \subset \mathbb{R} \mid v \in V\}$
- Kante $(u, v) \in E \Leftrightarrow I_u \cap I_v \neq \emptyset$

Beispiel:

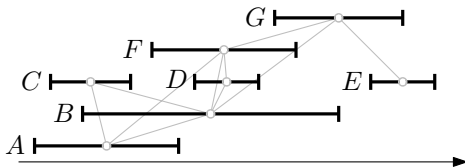
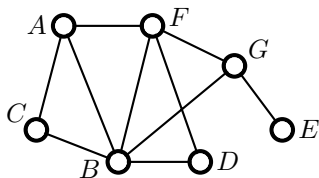


Intervallgraphen

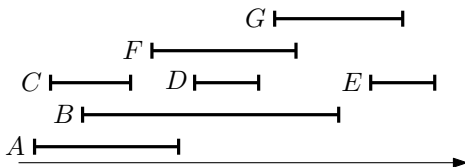
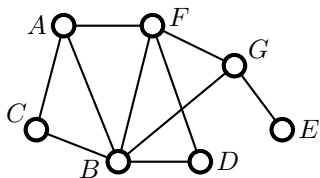
Definition: Intervallgraphen sind Graphen, die sich wie folgt als Schnittgraph von Intervallen in \mathbb{R} repräsentieren lassen:

- ungerichteter Graph $G = (V, E)$
- Intervallmenge $\mathcal{I} = \{I_v \subset \mathbb{R} \mid v \in V\}$
- Kante $(u, v) \in E \Leftrightarrow I_u \cap I_v \neq \emptyset$

Beispiel:



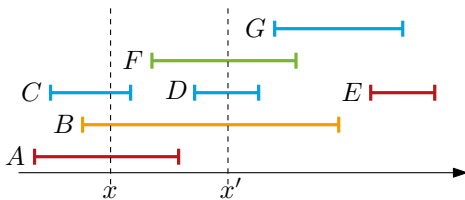
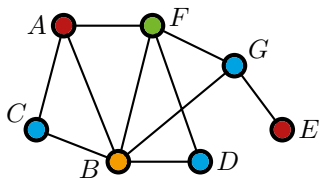
Färben von Intervallgraphen



Färben mit einer Farbe:

- äquivalent zu Maximum Independent Set
- äquivalent zu Interval Scheduling (Kap. Greedy-Algorithmen)
- Greedy-Algorithmus EDF (earliest deadline first)?

Färben von Intervallgraphen: Untere Schranke



Beobachtungen:

- Liegt ein Punkt $x \in \mathbb{R}$ in k Intervallen, braucht man mindestens k Farben
- Für **Tiefe** $d := \max_{x \in \mathbb{R}} \{|\{I \in \mathcal{I} \mid x \in I\}|\}$ gilt Färbung von G benötigt $\geq d$ Farben

Frage: Geht es auch immer mit d Farben?

Färben von Intervallgraphen: Algorithmus

Interval-Coloring(\mathcal{I}):

Sortiere Intervallgrenzen aufsteigend in Liste \mathcal{L}

$\text{col}_{\max} \leftarrow 0$

Initialisiere leere Queue Q

foreach Punkt a von \mathcal{L}

if a ist ein Startpunkt

if Q ist leer

 färbe $I = [a, b]$ mit Farbe col_{\max}

$\text{col}_{\max} \leftarrow \text{col}_{\max} + 1$

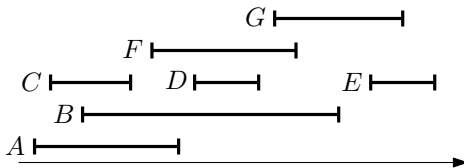
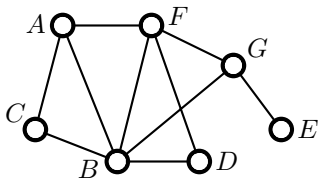
else

 entferne $c = Q.\text{head}$ und färbe $I = [a, b]$ mit Farbe c

else // a ist ein Endpunkt

 füge Farbe von $I = [b, a]$ in Q ein

return Färbung von \mathcal{I} und Anzahl col_{\max}



Färben von Intervallgraphen: Algorithmus

Interval-Coloring(\mathcal{I}):

Sortiere Intervallgrenzen aufsteigend in Liste \mathcal{L}

$\text{col}_{\max} \leftarrow 0$

Initialisiere leere Queue Q

foreach Punkt a von \mathcal{L}

if a ist ein Startpunkt

if Q ist leer

 färbe $I = [a, b]$ mit Farbe col_{\max}

$\text{col}_{\max} \leftarrow \text{col}_{\max} + 1$

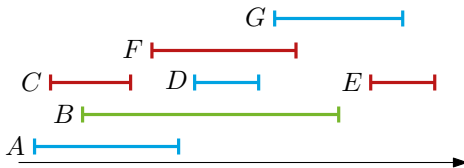
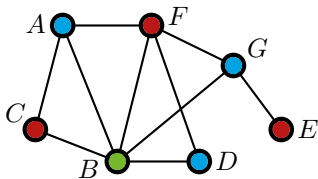
else

 entferne $c = Q.\text{head}$ und färbe $I = [a, b]$ mit Farbe c

else // a ist ein Endpunkt

 füge Farbe von $I = [b, a]$ in Q ein

return Färbung von \mathcal{I} und Anzahl col_{\max}



Färben von Intervallgraphen: Analyse

Theorem: Der Algorithmus Interval-Coloring berechnet eine minimale Färbung einer Intervallmenge \mathcal{I} und des zugehörigen Intervallgraphen G mit n Knoten in $O(n \log n)$ Zeit.

Beweis:

- Q enthält zu jedem Zeitpunkt nur "freie" Farben
- Wenn keine Farbe frei ist, wird eine neue Farbe gewählt
- Am Ende jedes Intervalls wird dessen Farbe wieder freigegeben
- Färbung gültig, da überlappende Intervalle verschiedene Farben erhalten, also mindestens d
- Es werden genau d Farben $0, \dots, d - 1$ verwendet:
 - sonst wären zum Zeitpunkt der ersten Wahl der Farbe d die Queue Q leer und damit die Tiefe $d + 1 \rightarrow$ Widerspruch
- Färbung selbst erfolgt in Zeit $O(n)$, wird allerdings dominiert durch Sortierung der Intervallgrenzen in $O(n \log n)$ Zeit □

Ergänzende Literatur

J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson, 2005. Kapitel 10.