

Diskrete Mathematik

Arne Dür Kurt Girstmair Simon Legner
Georg Moser Harald Zankl

Fakultät für Mathematik, Informatik und Physik @ UIBK
Sommersemester 2011



Zusammenfassung der letzten LV

Satz

jede rekursive Menge ist rekursiv aufzählbar, aber nicht jede rekursiv aufzählbare Menge ist rekursiv

Satz

es kann kein Testprogramm für "hello, world" Programme geben

Satz

*die folgenden Probleme sind **unentscheidbar**:*

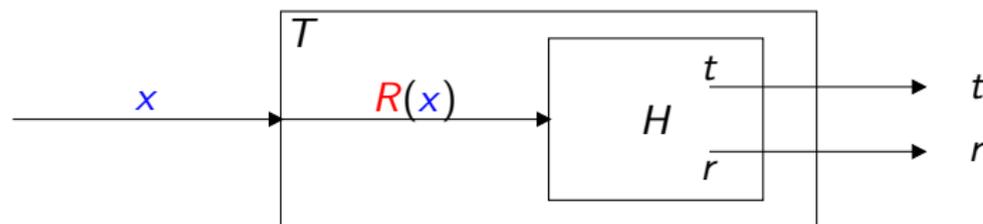
- 1** *das Postsche Korrespondenzproblem*
- 2** *ist eine beliebige Sprache regulär?*

Reduktionen im Bild

angenommen

- L, M Sprachen über Σ
- $L \leq_m M$ mit $R: \Sigma^* \rightarrow \Sigma^*$
- die Reduktion R wird von TM T berechnet

$$x \in L \iff R(x) \in M$$



Lemma

wenn $L \leq_m M$ und M rekursiv, dann ist L rekursiv

Übersicht

Endliche Automaten

Automaten, reguläre Sprachen und Grammatiken, (nicht)-deterministische endliche Automaten, Teilmengenkonstruktion, ϵ -NEAs, Umwandlung endlicher Automaten in reguläre Ausdrücke, Pumpinglemma, Minimierung

Berechenbarkeitstheorie

Einführung in die Berechenbarkeitstheorie, Turingmaschinen, Entscheidungsprobleme, Universelle Maschinen und Diagonalisierung

Komplexitätstheorie

Einführung in die Komplexitätstheorie, die Klassen P und NP, logarithmisch platzbeschränkte Reduktionen, Speicherplatzkomplexität

Übersicht

Endliche Automaten

Automaten, reguläre Sprachen und Grammatiken, (nicht)-deterministische endliche Automaten, Teilmengenkonstruktion, ϵ -NEAs, Umwandlung endlicher Automaten in reguläre Ausdrücke, Pumpinglemma, Minimierung

Berechenbarkeitstheorie

Einführung in die Berechenbarkeitstheorie, Turingmaschinen, Entscheidungsprobleme, Universelle Maschinen und Diagonalisierung

Komplexitätstheorie

Einführung in die Komplexitätstheorie, die Klassen P und NP, logarithmisch platzbeschränkte Reduktionen, Speicherplatzkomplexität

Einführung in die Komplexitätstheorie

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Ressourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Einführung in die Komplexitätstheorie

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Ressourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Ressourcen

- Speicherplatz
- Rechenzeit

Einführung in die Komplexitätstheorie

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Ressourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Ressourcen

- Speicherplatz
- Rechenzeit

Problem & Algorithmus

wir unterscheiden zwischen

- der Komplexität eines Algorithmus
- der Komplexität eines Problems

Einführung in die Komplexitätstheorie

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Ressourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Ressourcen

- Speicherplatz
- Rechenzeit

Problem & Algorithmus

wir unterscheiden zwischen

- der Komplexität eines Algorithmus quicksort hat $O(n \cdot \log n)$
- der Komplexität eines Problems MAZE ist in P

Einführung in die Komplexitätstheorie

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Ressourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Ressourcen

- Speicherplatz
- Rechenzeit

Problem & Algorithmus

wir unterscheiden zwischen

- der Komplexität eines Algorithmus quicksort hat $O(n \cdot \log n)$
- der Komplexität eines Problems MAZE ist in P

(n ist die Länge der zu sortierenden Liste)

Problem MAZE

Definition (MAZE)

gegeben

- gerichteten Graph G mit Eckenmenge E
- Ecken $s, t \in E$

\exists Weg zwischen s und t ?

Problem MAZE

Definition (MAZE)

gegeben

- gerichteten Graph G mit Eckenmenge E
- Ecken $s, t \in E$

\exists Weg zwischen s und t ?

Algorithmus

Nachfolgersuche

DM1

Problem MAZE

Definition (MAZE)

gegeben

- gerichteten Graph G mit Eckenmenge E
- Ecken $s, t \in E$

\exists Weg zwischen s und t ?

Algorithmus

Nachfolgersuche

DM1

Komplexität

- **Zeitkomplexität:** $O(n^2)$
- **Platzbedarf:** $O(n)$

wobei n Anzahl der Ecken

Problem “Traveling Salesperson Problem” (TSP)

Definition (TSP)

gegeben

- n Städte
- Distanz $d_{ij} > 0$, sodass $d_{ij} = d_{ji}$

Problem “Traveling Salesperson Problem” (TSP)

Definition (TSP)

gegeben

- n Städte
- Distanz $d_{ij} > 0$, sodass $d_{ij} = d_{ji}$

gesucht: Minimum der Kostensumme:

$$\sum_{i=1}^n d_{\pi(i), \pi(i+1)}$$

$\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ ist Permutation mit der Erweiterung
 $\pi(n+1) := \pi(1)$; wir bezeichnen diese Permutation auch als **Tour**

Problem “Traveling Salesperson Problem” (TSP)

Definition (TSP)

gegeben

- n Städte
- Distanz $d_{ij} > 0$, sodass $d_{ij} = d_{ji}$

gesucht: Minimum der Kostensumme:

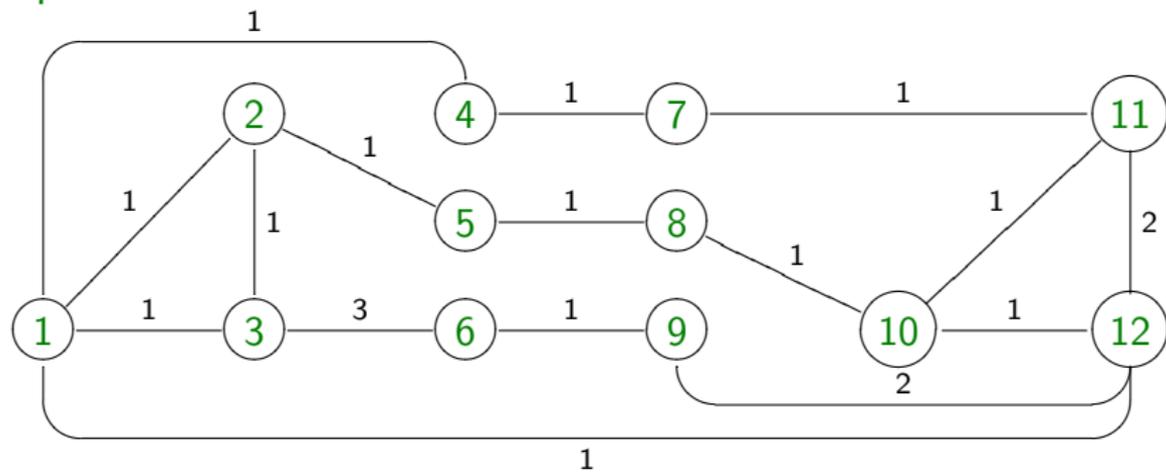
$$\sum_{i=1}^n d_{\pi(i), \pi(i+1)}$$

$\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ ist Permutation mit der Erweiterung
 $\pi(n+1) := \pi(1)$; wir bezeichnen diese Permutation auch als **Tour**

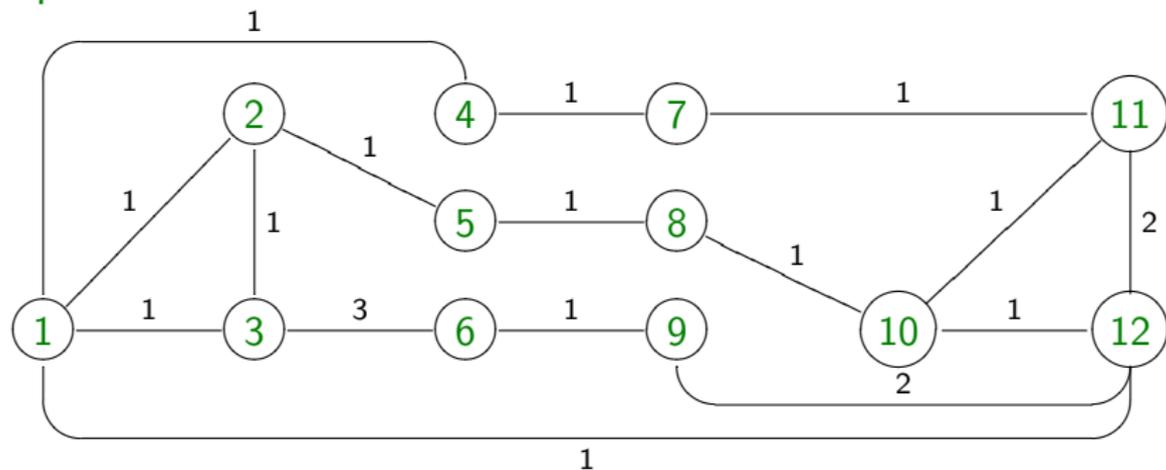
Bemerkung

Tour kann in jeder beliebigen Stadt beginnen, muss aber wieder zum Ausgangspunkt zurück

Beispiel

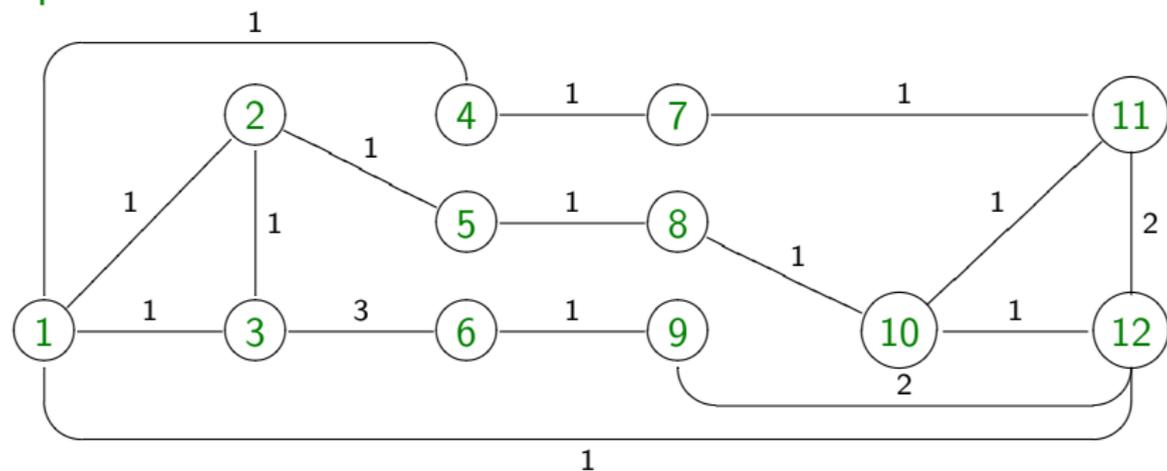


Beispiel



- 1 \exists Tour die alle Städte verbindet?
- 2 \exists Tour mit Budget ≤ 10 ?
- 3 \exists Tour mit Budget ≤ 20 ?

Beispiel



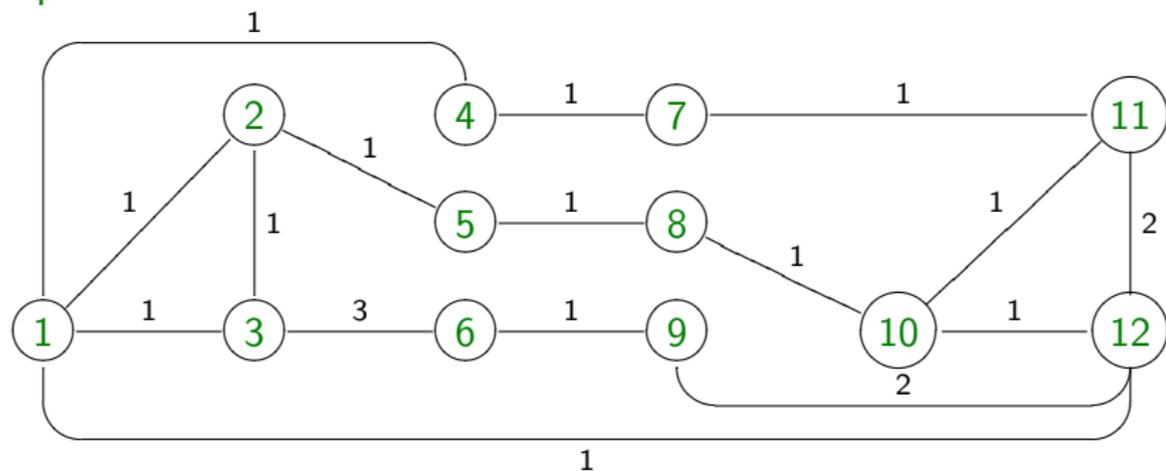
1 \exists Tour die alle Städte verbindet?

Ja

2 \exists Tour mit Budget ≤ 10 ?

3 \exists Tour mit Budget ≤ 20 ?

Beispiel



1 \exists Tour die alle Städte verbindet?

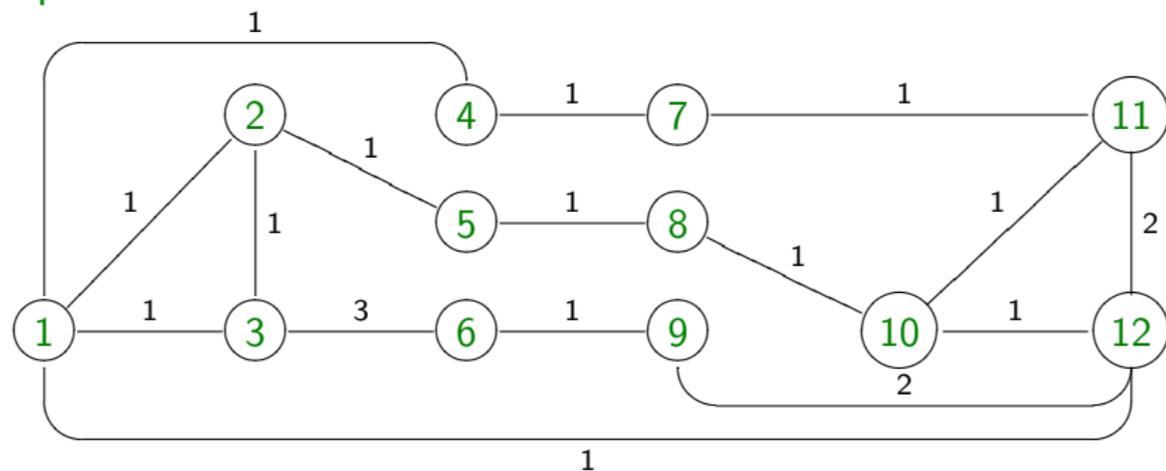
Ja

2 \exists Tour mit Budget ≤ 10 ?

Nein

3 \exists Tour mit Budget ≤ 20 ?

Beispiel



1 \exists Tour die alle Städte verbindet?

Ja

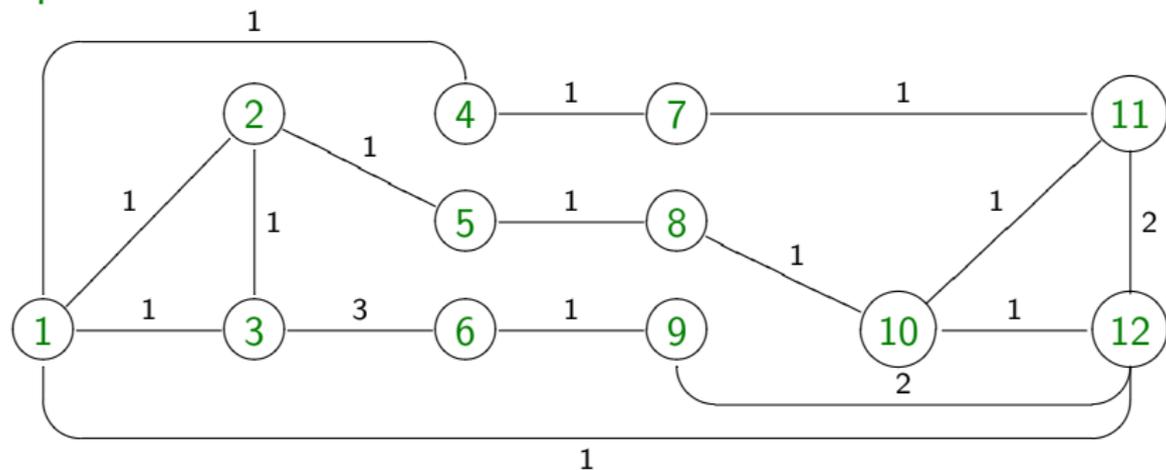
2 \exists Tour mit Budget ≤ 10 ?

Nein

3 \exists Tour mit Budget ≤ 20 ?

Ja

Beispiel



1 \exists Tour die alle Städte verbindet?

Ja

2 \exists Tour mit Budget ≤ 10 ?

Nein

3 \exists Tour mit Budget ≤ 20 ?

Ja

4 minimale Tour:

(1, 4, 7, 11, 10, 8, 5, 2, 3, 6, 9, 12, 1)

Budget = 15

Algorithmus

- 1 Aufzählen aller möglichen Lösungen
- 2 berechne die Kosten
- 3 wähle die Tour mit den geringsten Kosten

Algorithmus

- 1 Aufzählen aller möglichen Lösungen
- 2 berechne die Kosten
- 3 wähle die Tour mit den geringsten Kosten

Komplexität

- 1 **Zeitkomplexität:** $O(n!)$

hier bezeichnet n die Anzahl der Städte

Algorithmus

- 1 Aufzählen aller möglichen Lösungen
- 2 berechne die Kosten
- 3 wähle die Tour mit den geringsten Kosten

Komplexität

- 1 **Zeitkomplexität:** $O(n!)$
- 2 **Zeitkomplexität** kann auf $2^{O(n)}$ verbessert werden

hier bezeichnet n die Anzahl der Städte

Algorithmus

- 1 Aufzählen aller möglichen Lösungen
- 2 berechne die Kosten
- 3 wähle die Tour mit den geringsten Kosten

Komplexität

- 1 **Zeitkomplexität:** $O(n!)$
- 2 **Zeitkomplexität** kann auf $2^{O(n)}$ verbessert werden

hier bezeichnet n die Anzahl der Städte

Bemerkung

- wenn ein polynomieller Algorithmus existiert, dann gilt $P = NP$
- die Frage ob $P = NP$ ist eines der 7 Millenniumprobleme des Clay Institutes
- wenn ein polynomieller Algorithmus existiert, sind online Transaktionen (RSA-Verschlüsselung) unsicher

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

- 1 gerichteter Graph G und Startknoten s

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

- 1 gerichteter Graph G und Startknoten s
- 2 Anna beginnt im Land s sie
wählt ein erreichbares (unmarkiertes) Land t , welches sie markiert

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

- 1 gerichteter Graph G und Startknoten s
- 2 Anna beginnt im Land s sie
wählt ein erreichbares (unmarkiertes) Land t , welches sie markiert
- 3 Otto zieht von t weiter und markiert wieder

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

- 1 gerichteter Graph G und Startknoten s
- 2 Anna beginnt im Land s sie wählt ein erreichbares (unmarkiertes) Land t , welches sie markiert
- 3 Otto zieht von t weiter und markiert wieder
- 4 derjenige Spieler, der nicht mehr ziehen kann verliert

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

- 1 gerichteter Graph G und Startknoten s
- 2 Anna beginnt im Land s sie
wählt ein erreichbares (unmarkiertes) Land t , welches sie markiert
- 3 Otto zieht von t weiter und markiert wieder
- 4 derjenige Spieler, der nicht mehr ziehen kann verliert

∃ **Gewinnstrategie** für Anna?

Problem: “Generalised Geography” (GG)

Definition (GG)

das Spiel **verallgemeinerte Länderkunde** ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

- 1 gerichteter Graph G und Startknoten s
- 2 Anna beginnt im Land s sie
wählt ein erreichbares (unmarkiertes) Land t , welches sie markiert
- 3 Otto zieht von t weiter und markiert wieder
- 4 derjenige Spieler, der nicht mehr ziehen kann verliert

∃ **Gewinnstrategie** für Anna?

Beispiel

Länderkunde

Algorithmus A Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft

Algorithmus A

Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
- 2 eliminiere Knoten b und alle wegführenden Kanten; bezeichne den neuen Graphen mit G'

Algorithmus A

Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
- 2 eliminiere Knoten b und alle wegführenden Kanten; bezeichne den neuen Graphen mit G'
- 3 \forall unmittelbaren Nachfolger b' von b , rufe A rekursiv mit (G', b') auf

Algorithmus A

Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
- 2 eliminiere Knoten b und alle wegführenden Kanten; bezeichne den neuen Graphen mit G'
- 3 \forall unmittelbaren Nachfolger b' von b , rufe A rekursiv mit (G', b') auf
- 4 wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie

Algorithmus A Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
- 2 eliminiere Knoten b und alle wegführenden Kanten; bezeichne den neuen Graphen mit G'
- 3 \forall unmittelbaren Nachfolger b' von b , rufe A rekursiv mit (G', b') auf
- 4 wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie
- 5 wenn ein Aufruf nicht akzeptiert, hat Spieler 1 eine Gewinnstrategie; also akzeptiert A

Algorithmus A

Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
- 2 eliminiere Knoten b und alle wegführenden Kanten; bezeichne den neuen Graphen mit G'
- 3 \forall unmittelbaren Nachfolger b' von b , rufe A rekursiv mit (G', b') auf
- 4 wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie
- 5 wenn ein Aufruf nicht akzeptiert, hat Spieler 1 eine Gewinnstrategie; also akzeptiert A

Komplexität

- Platzkomplexität: $O(n)$

wobei n die Anzahl der Ecken von G

Algorithmus A

Eingabe (G, b)

- 1 sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
- 2 eliminiere Knoten b und alle wegführenden Kanten; bezeichne den neuen Graphen mit G'
- 3 \forall unmittelbaren Nachfolger b' von b , rufe A rekursiv mit (G', b') auf
- 4 wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie
- 5 wenn ein Aufruf nicht akzeptiert, hat Spieler 1 eine Gewinnstrategie; also akzeptiert A

Komplexität

- **Platzkomplexität:** $O(n)$
- nur der Rekursionstack braucht Platz, die Rekursionstiefe ist n

wobei n die Anzahl der Ecken von G

Laufzeitkomplexität

Definition

sei M eine totale Mehrband-DTM

Laufzeitkomplexität

Definition

sei M eine totale Mehrband-DTM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$,

Laufzeitkomplexität

Definition

sei M eine totale Mehrband-DTM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$,
 T misst maximale Anzahl der Schritte (Konfigurationszüge)
auf allen Eingaben

Laufzeitkomplexität

Definition

sei M eine totale Mehrband-DTM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$,
 T misst maximale Anzahl der Schritte (Konfigurationszüge)
auf allen Eingaben
- $T(n)$ is die **Laufzeit** von M , wenn n die Länge der Eingabe

Laufzeitkomplexität

Definition

sei M eine totale Mehrband-DTM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$,
 T misst maximale Anzahl der Schritte (Konfigurationszüge)
auf allen Eingaben
- $T(n)$ is die **Laufzeit** von M , wenn n die Länge der Eingabe
- M heißt **$T(n)$ -Zeit-Turingmaschine**

Laufzeitkomplexität

Definition

sei M eine totale Mehrband-DTM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$,
 T misst maximale Anzahl der Schritte (Konfigurationszüge)
auf allen Eingaben
- $T(n)$ is die **Laufzeit** von M , wenn n die Länge der Eingabe
- M heißt **$T(n)$ -Zeit-Turingmaschine**

Definition

sei $T: \mathbb{N} \rightarrow \mathbb{N}$ eine numerische Funktion; die deterministische
Zeitkomplexitätsklasse **DTIME**($T(n)$):

$$\text{DTIME}(T(n)) = \{L(M) \mid M \text{ ist eine totale DTM mit Laufzeit in } O(T(n)) \}$$

Beispiel

betrachte MAZE as Sprache:

$$\text{MAZE} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt $\text{MAZE} \in \text{DTIME}(n^2)$

Beispiel

betrachte MAZE as Sprache:

$$\text{MAZE} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt $\text{MAZE} \in \text{DTIME}(n^2)$

Definition

$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Beispiel

betrachte MAZE as Sprache:

$$\text{MAZE} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt $\text{MAZE} \in P$

Definition

$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Beispiel

betrachte MAZE as Sprache:

$$\text{MAZE} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt $\text{MAZE} \in P$

Definition

$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Definition

- ein **Verifikator** einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass

$$L = \{x \in \Sigma^* \mid \exists c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$

Beispiel

betrachte **MAZE** as Sprache:

$$\mathbf{MAZE} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt **MAZE** \in P

Definition

$$\mathbf{P} = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Definition

- ein **Verifikator** einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus **V** sodass

$$L = \{x \in \Sigma^* \mid \exists c, \text{ sodass } \mathbf{V} \text{ akzeptiert Eingabe } (x, c)\}$$
- ein **polytime Verifikator** ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$

Beispiel

betrachte MAZE as Sprache:

$$\text{MAZE} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt $\text{MAZE} \in P$

Definition

$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Definition

- ein **Verifikator** einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass

$$L = \{x \in \Sigma^* \mid \exists c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$

- ein **polytime Verifikator** ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$
- Wort c wird **Zertifikat** genannt

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\mathbf{TSP} = \{ (G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\mathbf{TSP} = \{ (G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\text{TSP} = \{(G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

- (G, b, B) , die **Instanz** des Problems

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\mathbf{TSP} = \{ (G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

- (G, b, B) , die **Instanz** des Problems
- Tour H , das **Zertifikat**

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\mathbf{TSP} = \{ (G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

- (G, b, B) , die **Instanz** des Problems
- Tour H , das **Zertifikat**

1 **V** prüft ob H eine Tour ist und Bewertung $\leq B$

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\text{TSP} = \{(G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

- (G, b, B) , die **Instanz** des Problems
- Tour H , das **Zertifikat**

1 **V** prüft ob H eine Tour ist und Bewertung $\leq B$

2 wenn ja, akzeptiert **V**, sonst verwirft **V**

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\mathbf{TSP} = \{ (G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

- (G, b, B) , die **Instanz** des Problems
- Tour H , das **Zertifikat**

1 **V** prüft ob H eine Tour ist und Bewertung $\leq B$

2 wenn ja, akzeptiert **V**, sonst verwirft **V**

3 **V** läuft in polynomieller Zeit

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte **TSP** als Sprache

$$\mathbf{TSP} = \{ (G, b, B) \mid \exists \text{ Tour in } G \text{ hat mit Bewertung } \leq B \}$$

definiere Verifikator **V** für **TSP** mit Eingabe:

- (G, b, B) , die **Instanz** des Problems
- Tour H , das **Zertifikat**

1 **V** prüft ob H eine Tour ist und Bewertung $\leq B$

2 wenn ja, akzeptiert **V**, sonst verwirft **V**

3 **V** läuft in polynomieller Zeit

also gilt **TSP** \in NP

Definition

sei M eine totale Mehrband-NTM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$
 T misst die maximale Anzahl von Schritten von M
auf allen Eingaben in jedem möglichen Berechnungspfad

Definition

sei N eine totale Mehrband-NTM

- die **Laufzeitkomplexität** von N ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$
 T misst die maximale Anzahl von Schritten von N
 auf allen Eingaben in jedem möglichen Berechnungspfad

Definition

sei $T: \mathbb{N} \rightarrow \mathbb{N}$ eine numerische Funktion; die nichtdeterministische Zeitkomplexitätsklasse **NTIME**($T(n)$) ist definiert als:

$$\text{NTIME}(T(n)) = \{L(N) \mid N \text{ ist eine totale NTM mit Laufzeit in } O(T(n))\}$$

Definition

sei N eine totale Mehrband-NTM

- die **Laufzeitkomplexität** von N ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$
 T misst die maximale Anzahl von Schritten von N
 auf allen Eingaben in jedem möglichen Berechnungspfad

Definition

sei $T: \mathbb{N} \rightarrow \mathbb{N}$ eine numerische Funktion; die nichtdeterministische Zeitkomplexitätsklasse **NTIME**($T(n)$) ist definiert als:

$$\text{NTIME}(T(n)) = \{L(N) \mid N \text{ ist eine totale NTM mit Laufzeit in } O(T(n))\}$$

Definition

$$\text{NP}' = \bigcup_{k \geq 1} \text{NTIME}(n^k)$$

Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Beweis.

Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Beweis.

1 sei V ein Verifikator für L

Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Beweis.

- 1 sei V ein Verifikator für L
- 2 V läuft in polynomieller Zeit auf Eingabe (x, c) , das heißt V läuft in Zeit $O(\ell(x)^k)$ für geeignetes k

Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Beweis.

- 1 sei V ein Verifikator für L
- 2 V läuft in polynomieller Zeit auf Eingabe (x, c) , das heißt V läuft in Zeit $O(\ell(x)^k)$ für geeignetes k
- 3 definiere NTM N mit Eingabe x
 - rate (nichtdeterministisch) String c mit $\ell(c) = \ell(x)^k$
 - simuliere V auf (x, c)
 - wenn V akzeptiert, dann akzeptiert N , sonst verwirft V

Algorithmus ist nichtdeterministisch

Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Beweis.

- 1 sei V ein Verifikator für L
- 2 V läuft in polynomieller Zeit auf Eingabe (x, c) , das heißt V läuft in Zeit $O(\ell(x)^k)$ für geeignetes k
- 3 definiere NTM N mit Eingabe x
 - rate (nichtdeterministisch) String c mit $\ell(c) = \ell(x)^k$
 - simuliere V auf (x, c)
 - wenn V akzeptiert, dann akzeptiert N , sonst verwirft V

Algorithmus ist nichtdeterministisch



Lemma

wenn $L \in \text{NP}$, dann $L \in \text{NP}'$

Beweis.

- 1 sei V ein Verifikator für L
- 2 V läuft in polynomieller Zeit auf Eingabe (x, c) , das heißt V läuft in Zeit $O(\ell(x)^k)$ für geeignetes k
- 3 definiere NTM N mit Eingabe x
 - rate (nichtdeterministisch) String c mit $\ell(c) = \ell(x)^k$
 - simuliere V auf (x, c)
 - wenn V akzeptiert, dann akzeptiert N , sonst verwirft V

Algorithmus ist nichtdeterministisch

Beispiel

TSP $\in \text{NP}'$

Lemma

wenn $L \in \text{NP}'$, dann $L \in \text{NP}$

Lemma

wenn $L \in \text{NP}'$, dann $L \in \text{NP}$

Beweis.

Lemma

wenn $L \in NP'$, dann $L \in NP$

Beweis.

1 sei $L \in NP'$; \exists NTM N , sodass $L = L(N)$

Lemma

wenn $L \in \text{NP}'$, dann $L \in \text{NP}$

Beweis.

- 1 sei $L \in \text{NP}'$; \exists NTM N , sodass $L = L(N)$
- 2 N hat polynomielle nichtdeterministische Zeitkomplexität

Lemma

wenn $L \in NP'$, dann $L \in NP$

Beweis.

- 1 sei $L \in NP'$; \exists NTM N , sodass $L = L(N)$
- 2 N hat polynomielle nichtdeterministische Zeitkomplexität
- 3 definiere einen polytime Verifikator V mit Eingabe (x, c) , wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
 - simuliere N auf x
 - verwende Zertifikat c um Nichtdeterminismus aufzulösen

Algorithmus ist deterministisch

Lemma

wenn $L \in NP'$, dann $L \in NP$

Beweis.

- 1 sei $L \in NP'$; \exists NTM N , sodass $L = L(N)$
- 2 N hat polynomielle nichtdeterministische Zeitkomplexität
- 3 definiere einen polytime Verifikator V mit Eingabe (x, c) , wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
 - simuliere N auf x
 - verwende Zertifikat c um Nichtdeterminismus aufzulösen

Algorithmus ist deterministisch

- 4 wenn N akzeptiert, akzeptiert V , ansonsten verwirft V

Lemma

wenn $L \in NP'$, dann $L \in NP$

Beweis.

- 1 sei $L \in NP'$; \exists NTM N , sodass $L = L(N)$
- 2 N hat polynomielle nichtdeterministische Zeitkomplexität
- 3 definiere einen polytime Verifikator V mit Eingabe (x, c) , wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
 - simuliere N auf x
 - verwende Zertifikat c um Nichtdeterminismus aufzulösen

Algorithmus ist deterministisch

- 4 wenn N akzeptiert, akzeptiert V , ansonsten verwirft V



Lemma

wenn $L \in NP'$, dann $L \in NP$

Beweis.

- 1 sei $L \in NP'$; \exists NTM N , sodass $L = L(N)$
- 2 N hat polynomielle nichtdeterministische Zeitkomplexität
- 3 definiere einen polytime Verifikator V mit Eingabe (x, c) , wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
 - simuliere N auf x
 - verwende Zertifikat c um Nichtdeterminismus aufzulösen

Algorithmus ist deterministisch

- 4 wenn N akzeptiert, akzeptiert V , ansonsten verwirft V

Satz

$NP = NP'$

Reduktionen (in polynomieller Zeit)

Definition

1 \exists deterministische, totale TM T mit Eingabealphabet Σ

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
- 2 T läuft in polynomieller Zeit

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
- 2 T läuft in polynomieller Zeit
- 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
 - 2 T läuft in polynomieller Zeit
 - 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band
- dann heißt $f: \Sigma^* \rightarrow \Sigma^*$ **in polynomieller Zeit berechenbar**

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
- 2 T läuft in polynomieller Zeit
- 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band
dann heißt $f: \Sigma^* \rightarrow \Sigma^*$ in polynomieller Zeit berechenbar

Definition

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
- 2 T läuft in polynomieller Zeit
- 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band
dann heißt $f: \Sigma^* \rightarrow \Sigma^*$ in polynomieller Zeit berechenbar

Definition

- 1 $\exists R: \Sigma^* \rightarrow \Sigma^*$

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
- 2 T läuft in polynomieller Zeit
- 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band
dann heißt $f: \Sigma^* \rightarrow \Sigma^*$ in polynomieller Zeit berechenbar

Definition

- 1 $\exists R: \Sigma^* \rightarrow \Sigma^*$
- 2 R berechenbar in polynomieller Zeit

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
- 2 T läuft in polynomieller Zeit
- 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band
dann heißt $f: \Sigma^* \rightarrow \Sigma^*$ in polynomieller Zeit berechenbar

Definition

- 1 $\exists R: \Sigma^* \rightarrow \Sigma^*$
- 2 R berechenbar in polynomieller Zeit
- 3 für $L \subseteq \Sigma^*$, $M \subseteq \Sigma^*$ gilt $x \in L \iff R(x) \in M$

Reduktionen (in polynomieller Zeit)

Definition

- 1 \exists deterministische, totale TM T mit Eingabealphabet Σ
 - 2 T läuft in polynomieller Zeit
 - 3 bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band
- dann heißt $f: \Sigma^* \rightarrow \Sigma^*$ in polynomieller Zeit berechenbar

Definition

- 1 $\exists R: \Sigma^* \rightarrow \Sigma^*$
 - 2 R berechenbar in polynomieller Zeit
 - 3 für $L \subseteq \Sigma^*$, $M \subseteq \Sigma^*$ gilt $x \in L \iff R(x) \in M$
- dann ist L (in polynomieller Zeit) auf M **reduzierbar**; kurz: $L \leq_m^p M$

Vollständigkeit von Komplexitätsklassen

Definition

- 1 \mathcal{C} eine beliebige Komplexitätsklasse
- 2 M eine Sprache über Σ und
- 3 \forall Sprachen $L \in \mathcal{C}$ gilt: $L \leq_m^P M$

Vollständigkeit von Komplexitätsklassen

Definition

- 1 \mathcal{C} eine beliebige Komplexitätsklasse
- 2 M eine Sprache über Σ und
- 3 \forall Sprachen $L \in \mathcal{C}$ gilt: $L \leq_m^P M$

dann ist M \leq_m^P -hart für \mathcal{C}

Vollständigkeit von Komplexitätsklassen

Definition

- 1 \mathcal{C} eine beliebige Komplexitätsklasse
- 2 M eine Sprache über Σ und
- 3 \forall Sprachen $L \in \mathcal{C}$ gilt: $L \leq_m^P M$

dann ist M \leq_m^P -hart für \mathcal{C}

Beispiel

TSP ist \leq_m^P -hart für NP

Vollständigkeit von Komplexitätsklassen

Definition

- 1 \mathcal{C} eine beliebige Komplexitätsklasse
- 2 M eine Sprache über Σ und
- 3 \forall Sprachen $L \in \mathcal{C}$ gilt: $L \leq_m^P M$

dann ist M \leq_m^P -hart für \mathcal{C}

Beispiel

TSP ist \leq_m^P -hart für NP

Definition

für eine Sprache M , sei

- 1 $M \leq_m^P$ -hart für \mathcal{C} und
- 2 $M \in \mathcal{C}$

Vollständigkeit von Komplexitätsklassen

Definition

- 1 \mathcal{C} eine beliebige Komplexitätsklasse
- 2 M eine Sprache über Σ und
- 3 \forall Sprachen $L \in \mathcal{C}$ gilt: $L \leq_m^P M$

dann ist M \leq_m^P -hart für \mathcal{C}

Beispiel

TSP ist \leq_m^P -hart für NP

Definition

für eine Sprache M , sei

- 1 $M \leq_m^P$ -hart für \mathcal{C} und
- 2 $M \in \mathcal{C}$

dann ist M \leq_m^P -vollständig für \mathcal{C} oder (kurz) \mathcal{C} -vollständig

NP-Vollständigkeit von TSP

Satz

TSP ist \leq_m^P -vollständig für NP

NP-Vollständigkeit von TSP

Satz

TSP ist \leq_m^P -vollständig für NP

Folgerung

- wenn \exists polynomieller Algorithmus für TSP, dann
- \exists polynomieller Algorithmus \forall Probleme in NP

NP-Vollständigkeit von TSP

Satz

TSP ist \leq_m^P -vollständig für NP

Folgerung

- wenn \exists polynomieller Algorithmus für TSP, dann
- \exists polynomieller Algorithmus \forall Probleme in NP

Beispiele

- MAZE \in NP, aber MAZE ist nicht hart für NP

NP-Vollständigkeit von TSP

Satz

TSP ist \leq_m^P -vollständig für NP

Folgerung

- wenn \exists polynomieller Algorithmus für TSP, dann
- \exists polynomieller Algorithmus \forall Probleme in NP

Beispiele

- MAZE \in NP, aber MAZE ist nicht hart für NP
- definiere GG als Sprache

$$\text{GG} = \{(G, s) \mid \text{gerichteter Graph } G, \text{ Startknoten } s, \exists \text{ Gewinn-} \\ \text{strategie für Anna}\}$$

GG ist hart für NP, aber GG \notin NP