Spiele in der Informatik

Martin Lange

Lehr- und Forschungseinheit Theoretische Informatik

Informatik-Schnupperstudium an der LMU, 29.3.2010

Übersicht

- Teil 1
  - Schokoladenessen für Spieltheoretiker
  - ein kleines bisschen Spieltheorie
- Teil 2
  - Sinn und Zweck der Informatik: Probleme lösen
  - das größte Problem der theoretischen Informatik
  - wo Spiele helfen können

Schokoladenessen für Spieltheoretiker

betrachte folgendes Spiel für 2 Personen

gegeben rechteckige Tafel Schokolade

Stück in der linken, oberen Ecke ist aber aus Seife

Spieler ziehen abwechselnd: rechts Spalten oder unten Zeilen abbrechen und aufessen

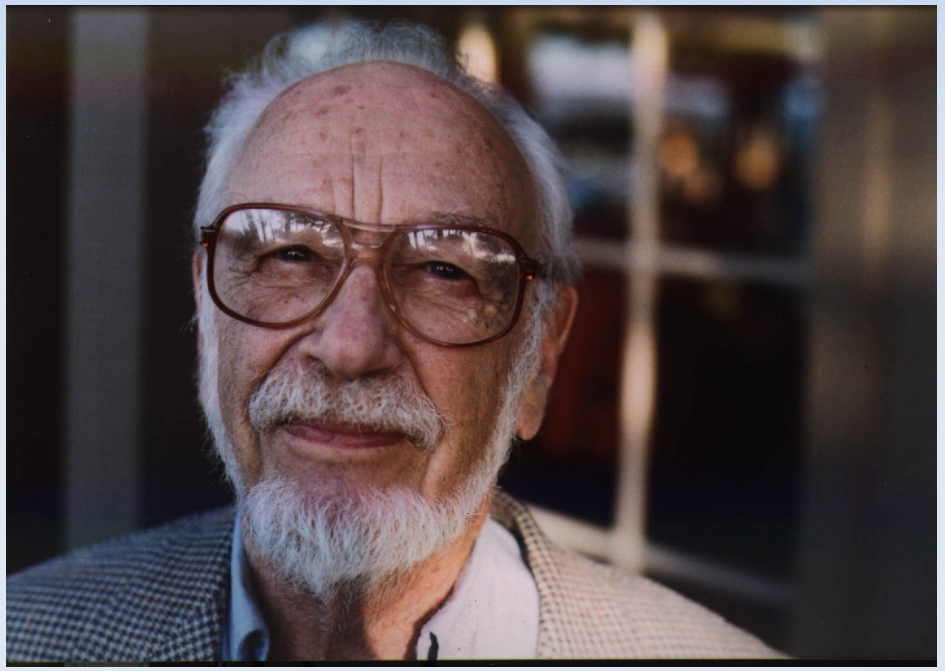
Verlierer ist der, der die Seife isst

Chomp

Spiel bekannt unter dem Namen *Chomp*

erfunden von

- Fred Schuh 1952
- David Gale 1974



Wer traut sich?

Ein paar Begriffe aus der Spieltheorie

aus spieltheoretischer Sicht handelt es sich bei *Chomp* um ein Spiel mit folgenden Eigenschaften

- *2-player game*
- *perfect information*, man sieht immer die ganze Situation
- *turn-based*, Spieler ziehen abwechselnd
- *finite*, es gibt nur endlich viele Situationen
- *finite duration*, keine unendlichen Partien
- *zero-sum*, Spieler 1 gewinnt gdw. Spieler 2 verliert

i.A. Spiele für alle möglichen Kombinationen denkbar, jeweils mit anderen Eigenschaften

Gewinnstrategien

Definition

Sei S die Menge aller Situationen in einem Spiel, $S=S_1 \cup S_2$, S_i die Menge der Situation, in den Spieler i zieht.
Eine Strategie für Spieler i ist eine Funktion $\sigma : S_i \rightarrow S$

Eine Gewinnstrategie für Spieler i ist eine Strategie, die ihn/sie jede Partie gewinnen lässt.

Determiniertheit

Satz: (Zermelo, 1926)

Sei $\mathcal{G}$ ein Spiel mit den Eigenschaften *2-player game, zero-sum, finite duration* und *perfect information.* Dann hat genau einer der beiden Spieler eine Gewinnstrategie für $\mathcal{G}$.

Klar: Hat ein Spieler Gewinnstrategie, dann der andere nicht. Aber wieso muss einer eine haben, nur weil anderer keine hat?

Gewinnstrategien in *Chomp*

Wir wissen, dass jeweils ein Spieler *Chomp* gewinnen kann.
Aber welcher und wie sieht die Gewinnstrategie aus?

Spieler unterscheiden sich nur darin, dass Spieler 1 anfängt.

Beachte:

- Jedes (n x m)-Spiel wird von einem Spieler gewonnen.
- ~~Ein Spieler gewinnt jedes (n x m)-Spiel.~~

Vorschläge?

Gewinnstrategien in *Chomp*

Satz:

Sei $\mathcal{G}$ das Chomp-Spiel der Größe (n x m)

- Spieler 2 gewinnt $\mathcal{G}$, falls $n = m$
- Spieler 1 gewinnt $\mathcal{G}$, falls $n \neq m$

Gewinnstrategien in *Chomp*

Satz:

Sei $\mathcal{G}$ das Chomp-Spiel der Größe (n x m)

- Spieler 2 gewinnt $\mathcal{G}$, falls $n = m$
- Spieler 1 gewinnt $\mathcal{G}$, falls $n \neq m$

Beweis durch Induktion über $n \cdot m$

- Spieler 2 gewinnt, falls $n \cdot m = 1$ bzw. $n = m = 1$
- Sei $n \cdot m > 1$. Durch Abbrechen wird Tafel kleiner.
  - Von Quadrat auf Quadrat unmöglich.
  - Von echtem Rechteck auf Quadrat immer möglich

Gewinnstrategien in *Chomp*

Gewinnstrategie kann also explizit angegeben und damit auch leicht implementiert werden:

Stelle Quadrat her, falls möglich. Falls nicht, dann resigniere.

Eine andere Version von *Chomp*

leicht veränderte Spielregeln

Spieler beißen Rechteck von rechts unten ab.

Satz von Zermelo: Gewinnstrategien existieren immer noch.

Aber für wen und wie sehen sie aus?

Gewinnstrategien in $Chomp_2$

Satz von Zermelo ist sogar konstruktiv: Gewinnstrategien lassen sich für gegebenes *n,m* berechnen.

Aber Spiel hat $k = \binom{n+m}{m}$ verschiedene Situationen.

z.B. $n = 10$, $m = 14$

$k = 1.961.256$

Gewinnstrategien in $Chomp_2$

Satz von Zermelo ist sogar konstruktiv: Gewinnstrategien lassen sich für gegebenes *n,m* berechnen.

Aber Spiel hat $k = \binom{n+m}{m}$ verschiedene Situationen.

z.B. $n = 20$, $m = 24$:

$k = 1.761.039.350.070$

Gewinnstrategien in $Chomp_2$

Computer auch ungeeignet, um Gewinnstrategien für allgemeines n,m anzugeben.

Wer gewinnt eigentlich $Chomp_2$?

Schlauer Kopf wird gebraucht.

Gewinnstrategien in *Chomp*

Satz:

Für jedes $Chomp_2$-Spiel hat Spieler 1 eine Gewinnstrategie.

Beweis durch Widerspruch (nicht konstruktiv)

Ang. Spieler 1 hätte keine Gewinnstrategie. Nach Satz von Zermelo hätte dann Spieler 2 eine. Wenn Spieler 1 zu Anfang nur 1 Stück abbeißt, dann muss Spieler 2 darauf mit einem Zug antworten können zu einer Situation, von der aus er gewinnen kann. Zu dieser Situation kann aber auch Spieler 1 im ersten Zug kommen und dieselbe Strategie verwenden.

Teil 2

Wo Spiele in der Informatik helfen können

Probleme lösen

Informatiker entwickeln Methoden, die Probleme lösen.

Informatiker sind Problemlöser.

Wann hat ein Informatiker gute Arbeit gemacht?

Klar, wenn seine Lösung gut ist.

Aber wann ist eine Lösung gut / besser / optimal?

Kommt auf das Problem an ...

Probleme lösen

Bsp.: Gegeben Telefonbuch mit *n* Einträgen und ein Name. Finde die dazugehörige Telefonnummer.

Lösung 1: Blättere von vorne nach hinten.

Lösung 2: In der Mitte aufschlagen, dann rechts oder links genauso weitermachen.

Lösungen vergleichen

Bsp.: Gegeben Telefonbuch mit n Einträgen und ein Name. Finde die dazugehörige Telefonnummer.

Lösung 1: Blättere von vorne nach hinten.

im Mittel ca. $O(n/2)$ Schritte

Lösung 2: In der Mitte aufschlagen, dann rechts oder links genauso weitermachen.

im Mittel ca. $O(\log n)$ Schritte

Lösung 2 besser als Lösung 1

Probleme klassifizieren

Beachte Unterschied zwischen Problem und Lösung.

Zwei Teilgebiete (von vielen) in der Informatik:

- Algorithmik
  Suche nach möglichst guten Lösungen
- Komplexitätstheorie
  Klassifiziert Probleme bzgl. der Güte möglicher Lösungen

Wir betrachten exemplarisch zwei Klassen von Problemen: **P** und **NP**.

Die Komplexitätsklassen **P** und **NP**

Intuitiv:

- **P** = Menge aller effizient lösbaren Probleme
- **NP** = Menge aller Probleme, deren Lösungen sich effizient nachvollziehen lassen

Formal: Sei *L* ein Problem.

> $L \in \mathbf{P}$ gdw. es gibt Polynom *p* und Methode *A*, so dass *A* Instanzen von *L* der Größe *n* in höchstens *p(n)* Schritten löst.

Bsp.: Suche im Telefonbuch.

Die Komplexitätsklassen **P** und **NP**

Intuitiv:

- **P** = Menge aller effizient lösbaren Probleme
- **NP** = Menge aller Probleme, deren Lösungen sich effizient nachvollziehen lassen

Formal: Sei *L* ein Problem.

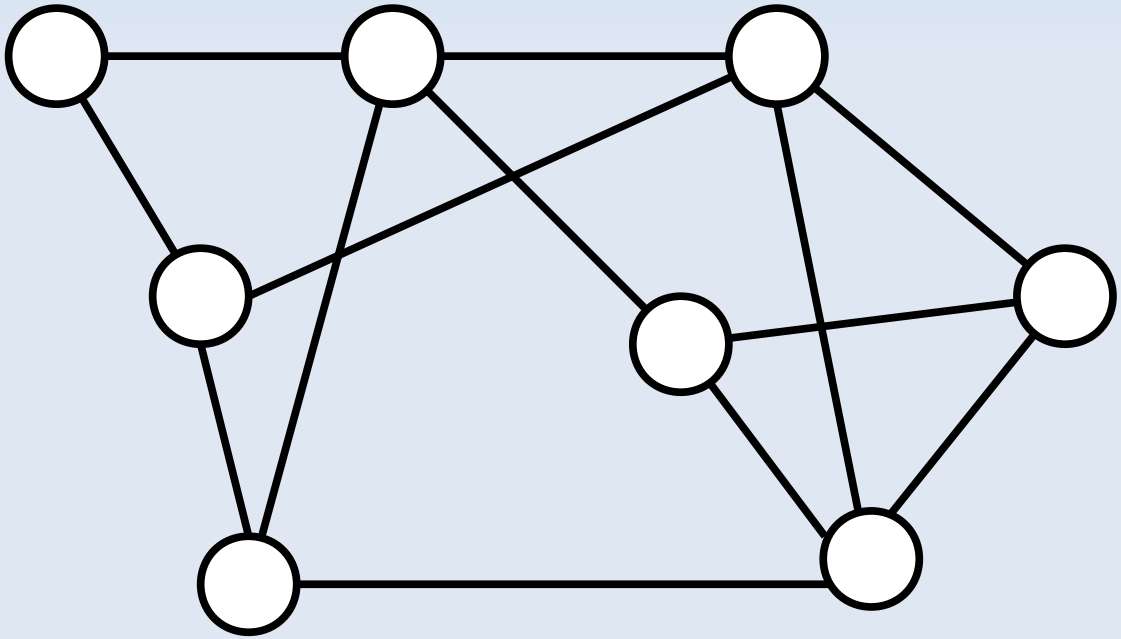
> $L \in \mathbf{NP}$ gdw. es gibt $L' \in \mathbf{P}$, wobei eine Instanz von L' aus einer Instanz von L und einer möglichen Lösung davon besteht

Bsp.: Faktorzerlegung einer natürlichen Zahl

Graphen

Was hat das eigentlich mit Spielen zu tun? ... Geduld!

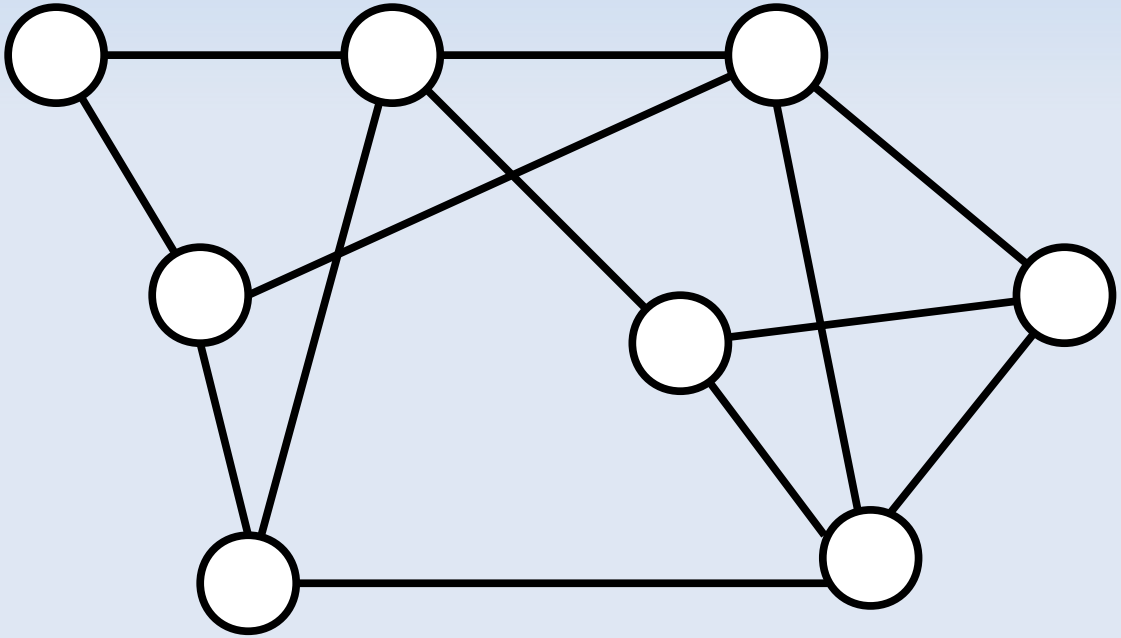
Def.: Ein Graph ist ein ...



Bezeichnung: *V* Knotenmenge, *E* Kantenmenge

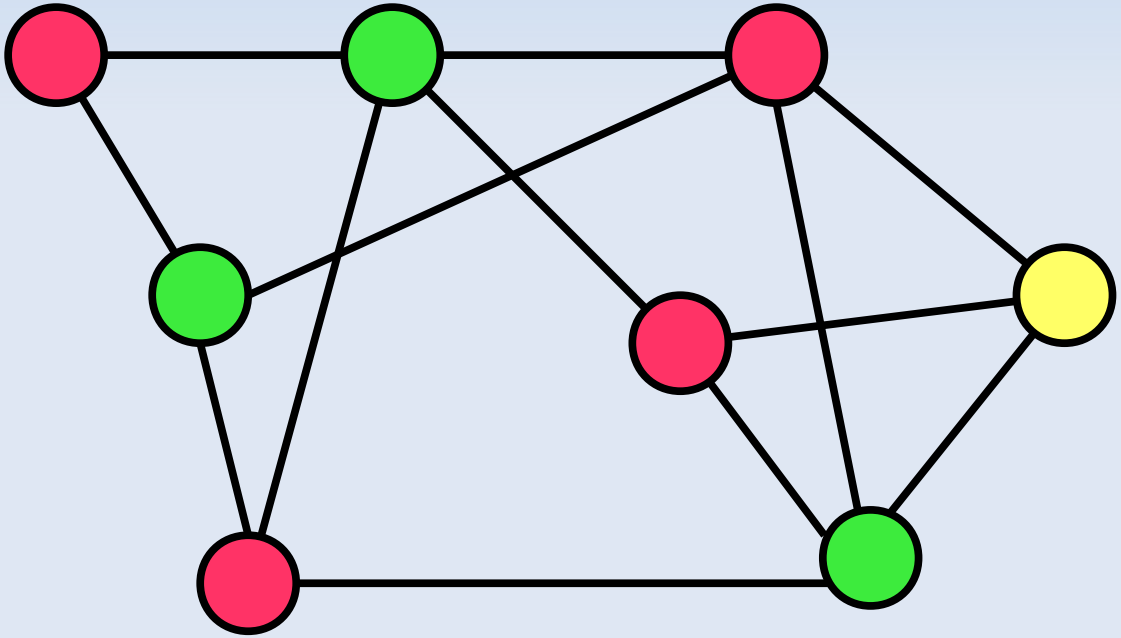
Graphfärbungen

Def.: Eine *k*-Färbung eines Graphen ist eine Funktion $\Omega: V \rightarrow \{1,..,k\}$, sodass für alle Kanten (v,w) gilt: $\Omega(v) \neq \Omega(w)$



Graphfärbungen

Def.: Eine *k*-Färbung eines Graphen ist eine Funktion $\Omega: V \rightarrow \{1,..,k\}$, sodass für alle Kanten (v,w) gilt: $\Omega(v) \neq \Omega(w)$



Def.: Ein Graph heisst *k*-färbbar, wenn es eine k-Färbung gibt

Das *k*-Färbbarkeitsproblem ...

... ist das folgende für jede feste, natürliche Zahl *k*:

Eingabe: ein Graph
Ausgabe: ”Ja”, falls dieser k-färbbar ist, ”Nein” sonst.

Das *k*-Färbbarkeitsproblem ...

... ist das folgende für jede feste, natürliche Zahl *k*:

Eingabe: ein Graph
Ausgabe: ”Ja”, falls dieser k-färbbar ist, ”Nein” sonst.

Satz: Das 0-Färbbarkeitsproblem ist in ...

Das *k*-Färbbarkeitsproblem ...

... ist das folgende für jede feste, natürliche Zahl *k*:

Eingabe: ein Graph
Ausgabe: ”Ja”, falls dieser k-färbbar ist, ”Nein” sonst.

Satz: Das 0-Färbbarkeitsproblem ist in $\mathbf{P}$.

Satz: Das 1-Färbbarkeitsproblem ist in ...

Das *k*-Färbbarkeitsproblem ...

... ist das folgende für jede feste, natürliche Zahl *k*:

Eingabe: ein Graph
Ausgabe: ”Ja”, falls dieser k-färbbar ist, ”Nein” sonst.

Satz: Das 0-Färbbarkeitsproblem ist in $\mathbf{P}$.

Satz: Das 1-Färbbarkeitsproblem ist in $\mathbf{P}$.

Satz: Das 2-Färbbarkeitsproblem ist in ...

Das *k*-Färbbarkeitsproblem ...

... ist das folgende für jede feste, natürliche Zahl *k*:

Eingabe: ein Graph
Ausgabe: ”Ja”, falls dieser k-färbbar ist, ”Nein” sonst.

Satz: Das 0-Färbbarkeitsproblem ist in $\mathbf{P}$.

Satz: Das 1-Färbbarkeitsproblem ist in $\mathbf{P}$.

Satz: Das 2-Färbbarkeitsproblem ist in $\mathbf{P}$.

(Graph ist 2-färbbar gdw. es keinen Zykel ungerader Länge darin gibt)

Das 3-Färbbarkeitsproblem

Es ist nicht bekannt, ob 3-Färbbarkeit in **P** ist.

Satz: Das 3-Färbbarkeitsproblem ist in **NP**.

(Man kann leicht feststellen, ob eine gegebene 3-Färbung korrekt ist.)

D.h. man kennt bis dato keine guten Verfahren, um 3-Färbbarkeit im Allgemeinen zu lösen.

Was tun?

Das 3-Färbbarkeitsproblem

Es ist nicht bekannt, ob 3-Färbbarkeit in **P** ist.

Satz: Das 3-Färbbarkeitsproblem ist in **NP**.

(Man kann leicht feststellen, ob eine gegebene 3-Färbung korrekt ist.)

D.h. man kennt bis dato keine guten Verfahren, um 3-Färbbarkeit im Allgemeinen zu lösen.

Was tun?

Wenn es im Allgemeinen nicht geht, dann vielleicht im Speziellen, z.B. nicht über beliebigen Graphen.

Das 3-Färbbarkeitsproblem über speziellen Graphen

Trivialbeispiel:

Satz: Das 3-Färbbarkeitsproblem über Graphen ohne Kanten ist in **P**.

(Jeder kantenlose Graph ist k-färbbar für jedes $k>0$.)

Der Satz ist zwar richtig, aber vollkommen nutzlos.

Das 3-Färbbarkeitsproblem über speziellen Graphen

Trivialbeispiel:

Satz: Das 3-Färbbarkeitsproblem über Graphen ohne Kanten ist in **P**.

(Jeder kantenlose Graph ist *k*-färbbar für jedes *k*>0.)

Der Satz ist zwar richtig, aber vollkommen nutzlos.
Genauso richtig, aber wesentlich nützlicher ist:

Satz: Das 3-Färbbarkeitsproblem über Graphen von *beschränkter Baumweite* ist in **P**.

Baumweite

Satz sieht schön aus, aber was meint er?

Def.: Eine Menge *M* von Graphen ist von beschränkter Baumweite, wenn es ein *b* gibt, sodass die Baumweite eines jeden $G \in M$ höchstens *b* ist.

Jetzt müssen wir nur noch wissen, was die Baumweite eines Graphen ist.

Kann doch nicht so schwer sein ...

Baumweite

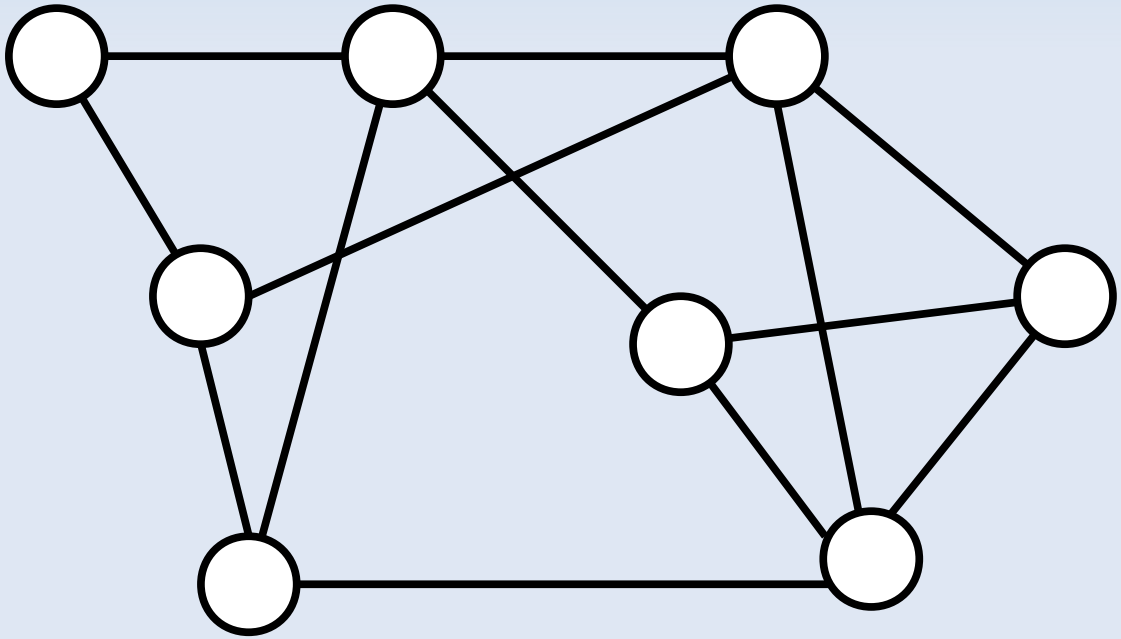
Definition 6.1: Let *k* be a positive integer.

- A graph H is a *k-tree* if either H is a K_{k+1} -clique or there are a k-tree G, nodes $a_1, \ldots, a_k$ in G forming a $\mathbf{K_k}$-clique, and a node *b* in H - G such that H is obtained from G by connecting *b* to each of the nodes $a_1, \ldots, a_k$ (thus, forming a $\mathbf{K_{k+1}}$-clique).
- A graph is a *partial k-tree* if it is a subgraph (not necessarily induced) of a k-tree.
- The *treewidth* of a graph H, denoted by tw(H), is the smallest *k* such that H is a partial k-tree.
- We write $\mathcal{T}(k)$ to denote the class of all graphs H such that tw(H) < *k*.

Clearly, if T is a tree, then tw(T) = 1.

Beispiel Baumweite

Was ist also die Baumweite von ...



Das *k*-Cops-and-Robber-Spiel

gespielt auf einem Graph zwischen Cops-Spieler 1 und Robber-Spieler 2

- Spieler 1 lässt *k* Cops in Hubschraubern über Knoten patrouillieren
- Spieler 2 hält sich in einem Knoten auf

Cop kann Robber durch Landen fangen

in jeder Runde:

- Cops fliegen irgendwo hin oder landen
- Robber läuft entlang der Kanten irgendwo hin

Das *k*-Cops-and-Robber-Spiel

Und wozu ist das gut?

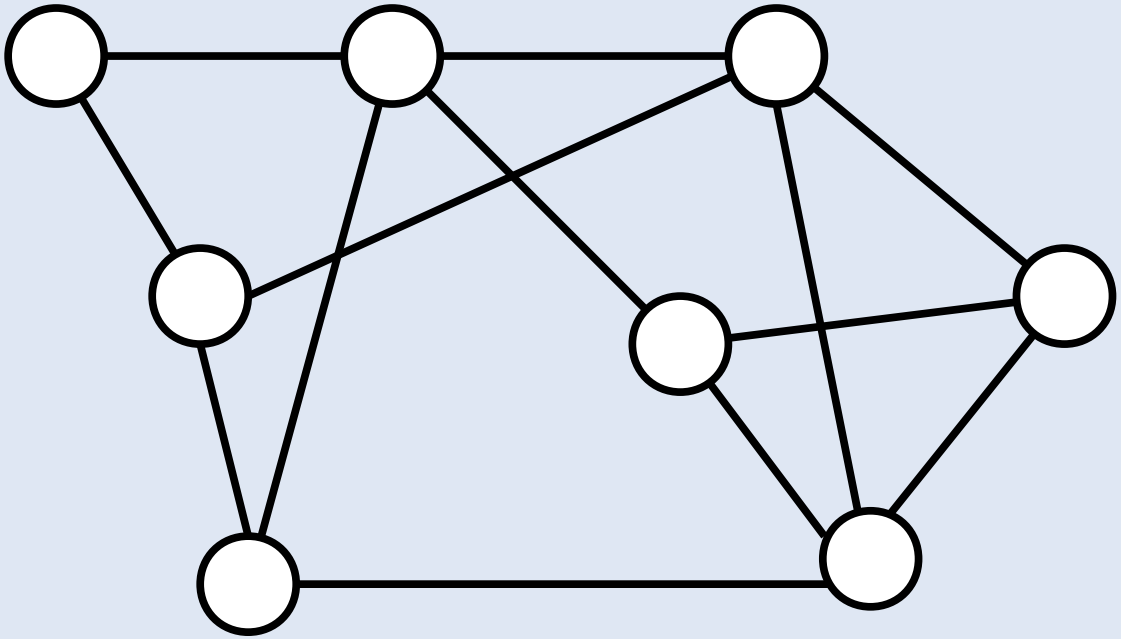
Satz: (Downey/Fellows)
Ein Graph *G* hat Baumweite $< k$ gdw. der Cops-Spieler eine Gewinnstrategie für das *k*-Cops-and-Robber-Spiel hat.

Baumweite von *G* ist also das kleinste *k*, sodass man den Robber mit *k*+1 Cops fangen kann.

Beispiel Baumweite

Dann mal los ...

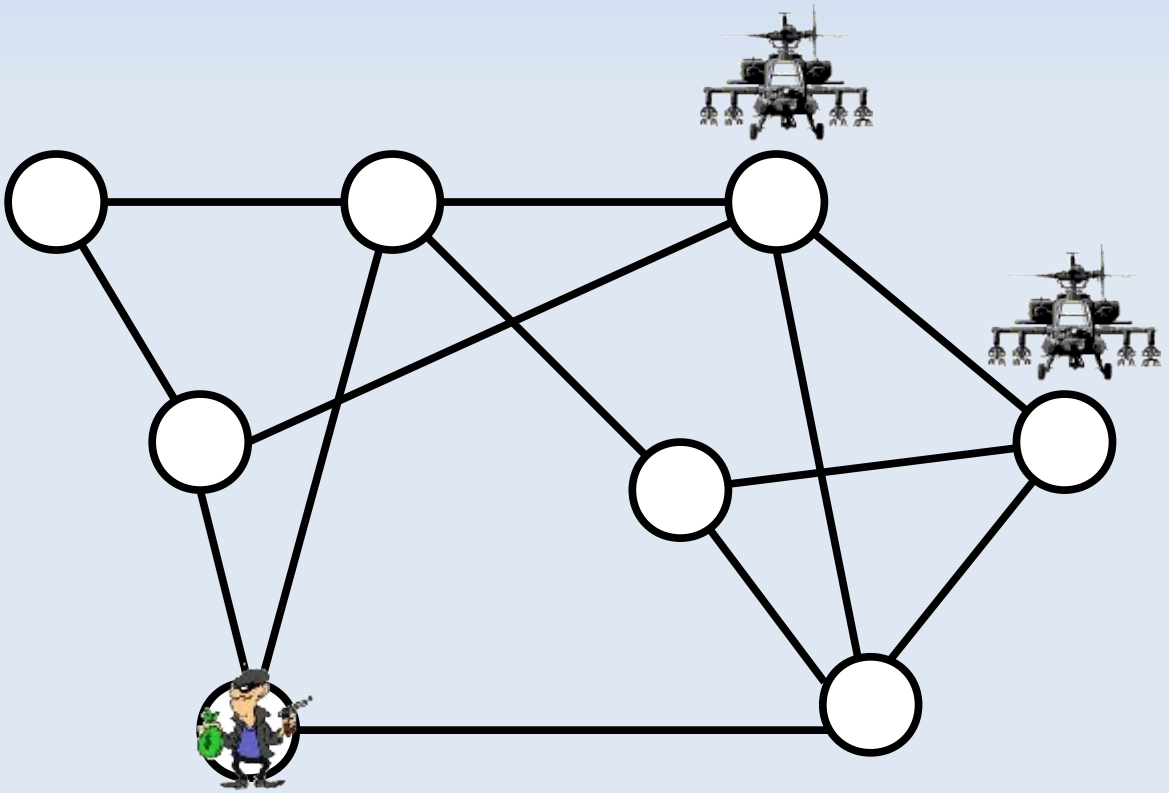
Mit einem geht's nicht.



Beispiel Baumweite

Dann mal los ...

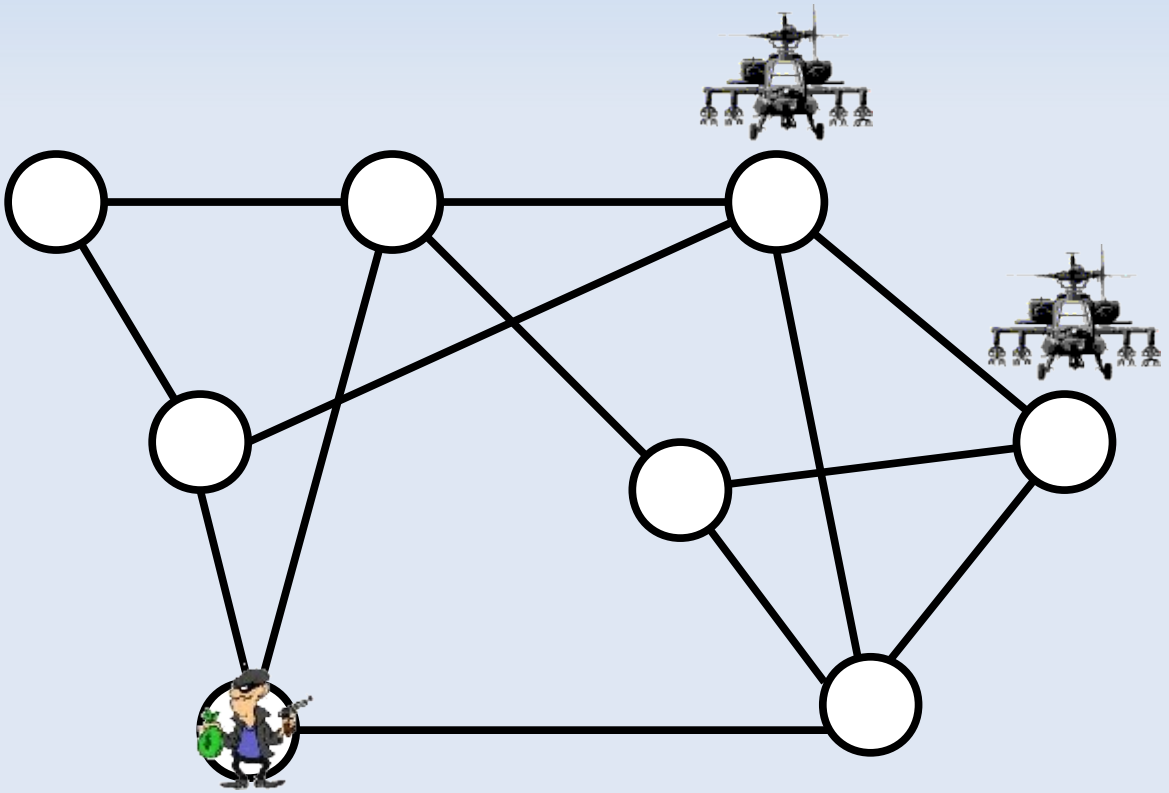
Mit zweien geht's auch nicht.



Beispiel Baumweite

Dann mal los ...

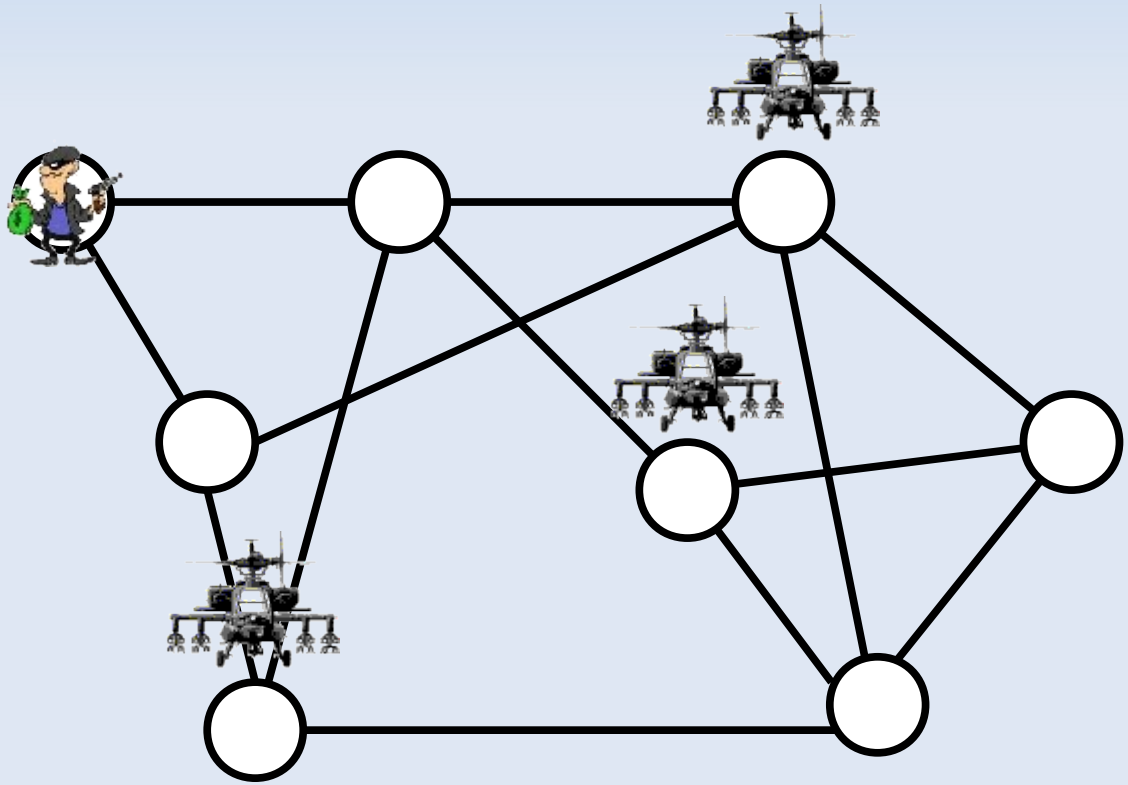
Mit zweien geht's auch nicht.



Beispiel Baumweite

Dann mal los ...

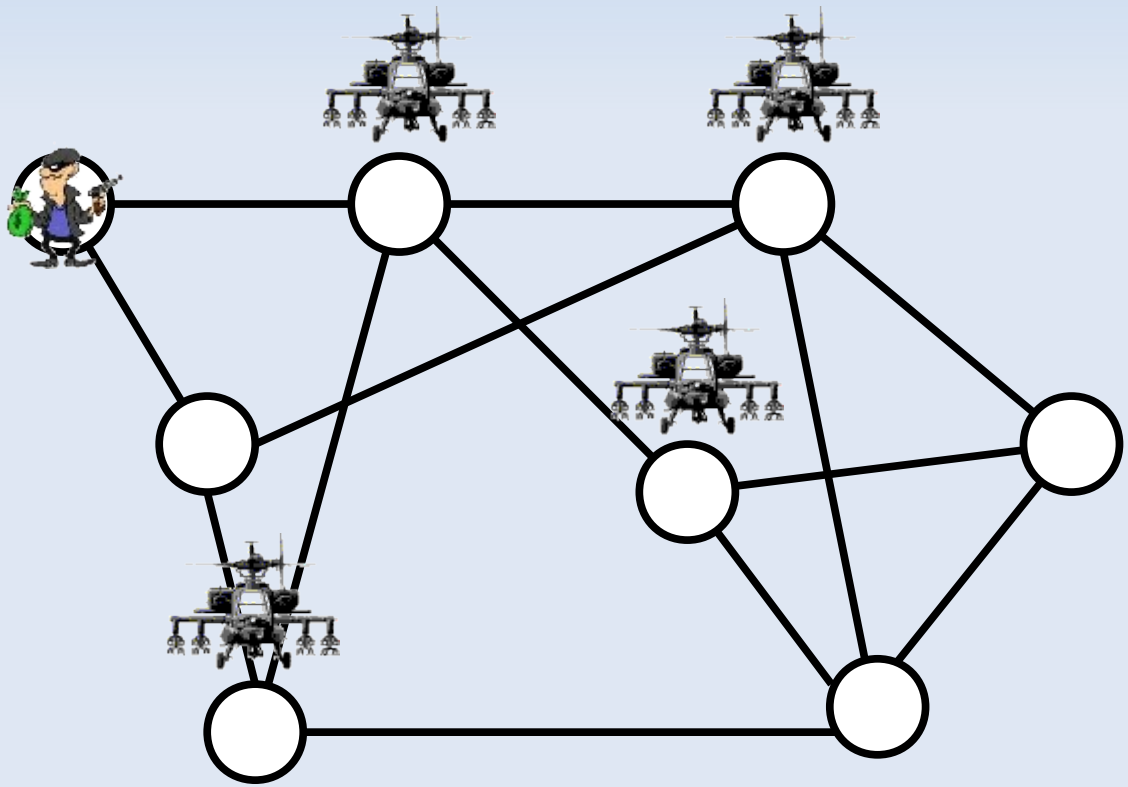
Mit dreien geht's auch nicht.



Beispiel Baumweite

Dann mal los ...

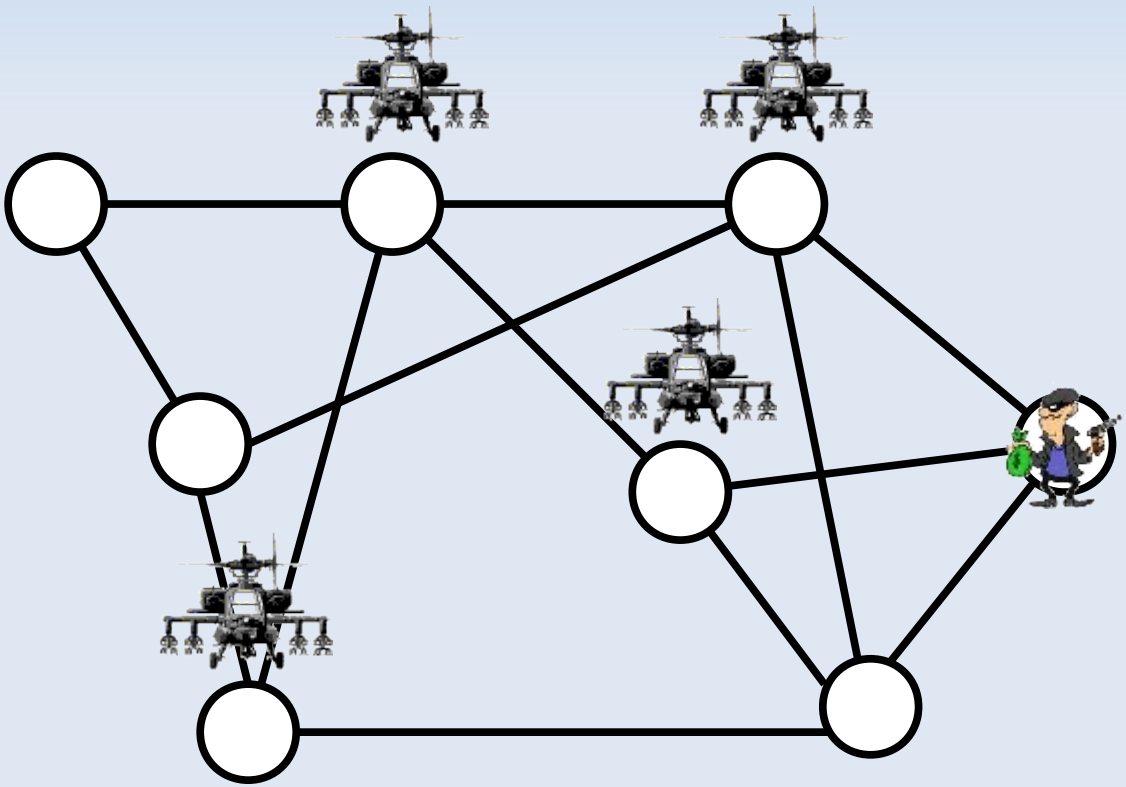
Aber mit vieren geht's!



Beispiel Baumweite

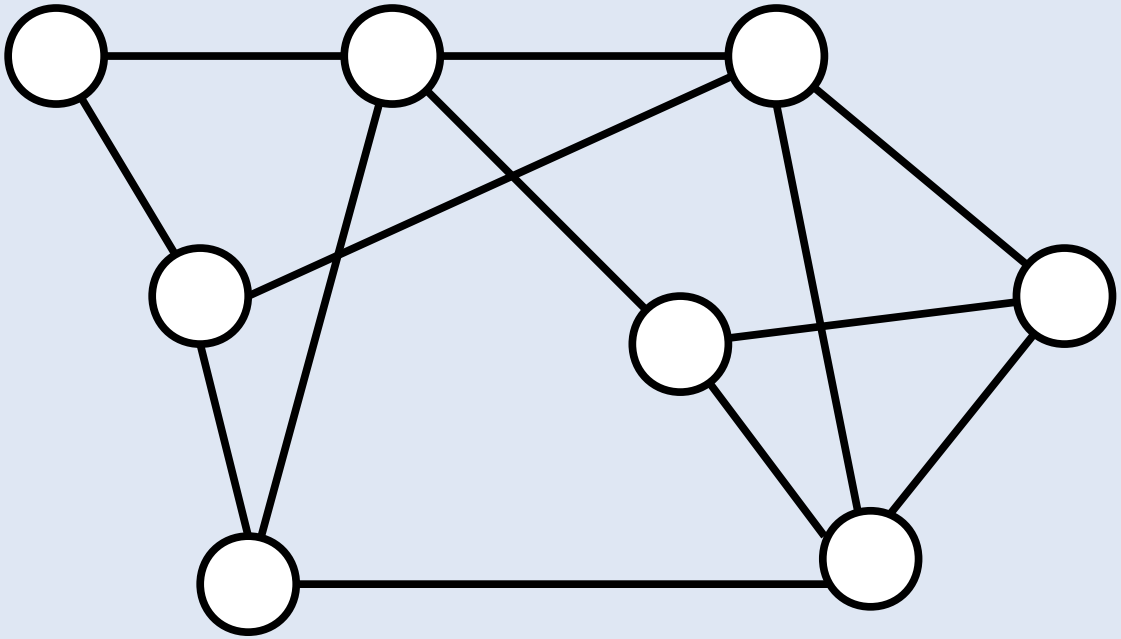
Dann mal los ...

Aber mit vieren geht's auf jeden Fall!



Beispiel Baumweite

Also ist Baumweite dieses Graphen 3.



Fazit

Fazit

... in Bezug auf Spiele, u.a.

- Spiele auch interessant, um mit Methoden der Mathematik und der Informatik untersucht zu werden
- Spiele oft gutes Hilfsmittel zum Verstehen komplexer Sachverhalte

... in Bezug auf Informatik, u.a.

- auf keinen Fall nur Programmieren
- Entwickeln von Methoden, mit denen Computer dann Probleme automatisch lösen können

Ende