Complexiteit 2023 — college 9

11 april 2023

De klasse $\mathcal{NP}$

Vorige keer

- handelbare en onhandelbare problemen: wel/geen *deterministisch* polynomiaal algoritme
- $\mathcal{P}$: oplossing in polynomiale tijd te *vinden*, handelbaar; $\mathcal{EXP} \setminus \mathcal{P}$: oplossing niet in polynomiale tijd te vinden; onhandelbaar
- $\mathcal{NP}$: *niet-deterministisch* polynomiaal algoritme; oplossing in polynomiale tijd te *controleren*
- $\mathcal{NPC}$: moeilijkste problemen in $\mathcal{NP}$; aantal welbekende voorbeelden; handelbaarheid onbekend
- beslissingsproblemen en optimalisatieproblemen: zelfde handelbaarheid

Vandaag

- de klasse $\mathcal{NP}$, hoe deze formeel te definiëren en hoe aan te tonen dat een probleem erin zit
- een voorproefje van reducties en de klasse $\mathcal{NPC}$

Voorbeeldproblemen

Vorige week besproken:

- CNF-satisfiability (SAT)
  Gegeven een logische formule ϕ in CNF. Bestaat er een waardering van de voorkomende variabelen die ϕ waar maakt?
- Subset Sum (SUM)
  Gegeven een eindige verzameling $S \subset \mathbb{N}$ en een getal $t \in \mathbb{N}$. Bestaat er een deelverzameling $S' \subseteq S$ zodat $\sum_{s \in S'} s = t$?
- Hamiltonkring (HC)
  Gegeven een graaf $\mathcal{G} = (V, E)$. Heeft $\mathcal{G}$ een Hamiltonkring?
- Handelsreizigersprobleem (TSP)
  Gegeven een volledige, ongerichte graaf $\mathcal{G} = (V, E)$ met gewichten op de takken, en een geheel getal $k \geq 0$. Bestaat er in $\mathcal{G}$ een Hamiltonkring met totaalgewicht $\leq k$?

Voorbeeldproblemen

Vorige week besproken:

- Kliek
  Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal k $(0 \leq k \leq |V|)$. Bestaat er in $\mathcal{G}$ een kliek ter grootte $\geq k$?
  (of equivalent: ter grootte k?)
- Graafkleuring (Kleur)
  Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal $k > 0$. Bestaat er een kleuring van $\mathcal{G}$ met $\leq k$ kleuren?
  (of equivalent: met k kleuren?)

Gemeenschappelijk

1. voor alle zes voorbeeldproblemen is eenvoudig een exponentieel algoritme op te schrijven.
2. in alle gevallen kan in polynomiale tijd worden gecontroleerd of een kandidaatoplossing een echte oplossing (= "dat wat je zoekt") is.
3. voor al deze problemen geldt: het lijkt extreem moeilijk (exponentieel) te zijn om voor gegeven invoer x te bepalen of het antwoord "ja" of "nee" moet zijn.
4. echter, *als* x een ja-instantie is, dan is er een eenvoudige (polynomiale) manier om iemand daarvan te overtuigen.

Gemeenschappelijk

5. voor ja-instanties bestaat er een zogenaamd certificaat dat kan worden gebruikt om te laten zien dat het antwoord inderdaad "ja" is. In de voorbeelden is dit steeds de gezochte oplossing, d.w.z. dat wat de invoer zou moeten bezitten.
6. bovendien is dit certificaat kort (polynomiaal) en kan verifiëren / controleren ervan in polynomiale tijd.
7. eigenschap 2 t/m 6: de genoemde problemen zitten in $\mathcal{NP}$. Later wordt alles formeler gemaakt.
8. ja-instanties zijn eenvoudig te verifiëren met de juiste hint (certificaat). Hoe zit het met nee-instanties?
9. Er zijn ook makkelijke instanties / ingeperkte versies van het probleem.

$\mathcal{NP}$ informeel

$\mathcal{P}$: beslissingsproblemen die polynomiaal oplosbaar zijn.

$\mathcal{NP}$: beslissingsproblemen die polynomiaal verifieerbaar zijn,

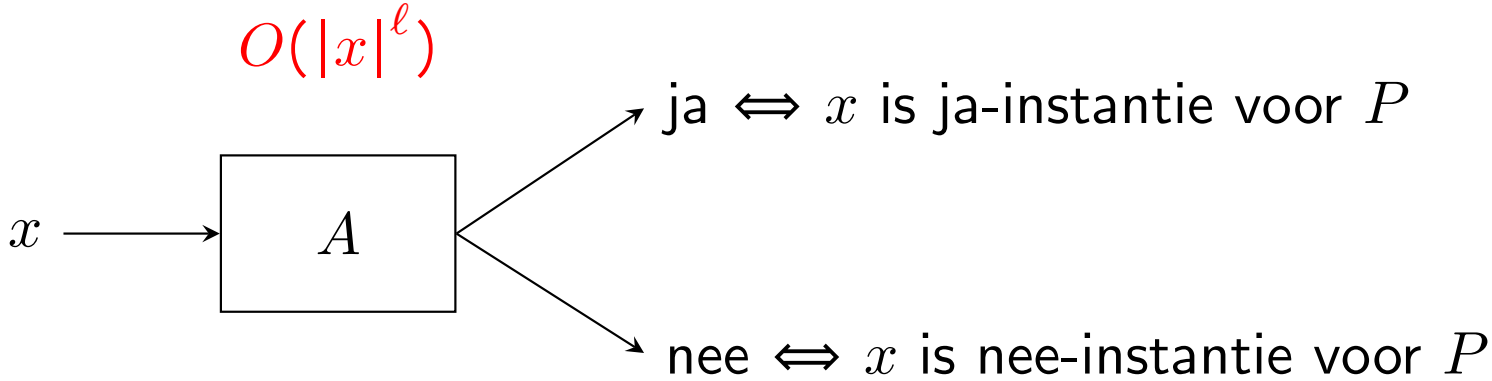
- althans, de ja-instanties
- en met de juiste hint/certificaat
- voor ja-instanties bestaat er altijd zo'n certificaat

$\mathcal{NPC}$: bevat de moeilijkste problemen uit $\mathcal{NP}$.

Voor problemen in $\mathcal{NP}$ duurt het *vinden* van een oplossing lang (tenzij ze in $\mathcal{P}$ zitten); het *verifiëren/controleren* kan echter in polynomiale tijd.

$P \in \mathcal{P}$

Gegeven een willekeurig beslissingsprobleem $P \in \mathcal{P}$. Dan is er een polynomiaal deterministisch algoritme A dat P oplost voor elke invoer x.



Het algoritme is polynomiaal in de lengte van de invoer, dus worst case is $O(|x|^{\ell})$ ($\ell \geq 0$). Een algoritme is deterministisch als het elke keer dat het wordt uitgevoerd op dezelfde invoer (instantie) hetzelfde doet, en dus dezelfde uitvoer oplevert.

Certificaat

NP-problemen: de *ja-instanties* zijn eenvoudig (polynomiaal) te verifiëren met behulp van het juiste certificaat, de nee-instanties niet.

Veel beslissingsproblemen zijn geformuleerd als "is-er-een"-vragen. In dat geval kun je bij een certificaat denken aan een oplossing voor het probleem, dus datgene wat wordt gezocht en wat een ja-instantie een ja-instantie maakt.

Daarvan moet dan onder andere worden gecontroleerd dat deze voldoet aan de criteria van het probleem, dus inderdaad een *echte* oplossing is, en derhalve een ja-antwoord rechtvaardigt.

Als deze verificatie/controle in polynomiale tijd kan, zit het probleem in $\mathcal{NP}$.

Voorbeeld

Gegeven een logische formule ϕ in CNF, bestaat er een waardering die ϕ waarmaakt?

In geval van een ja-instantie kunnen we als certificaat een waarmakende waardering nemen. Er kan nu in polynomiale tijd worden geverifieerd dat deze waardering ϕ inderdaad waarmaakt, en dus dat ϕ een ja-instantie is.
Probleem SAT zit derhalve in $\mathcal{NP}$ (en overigens ook in $\mathcal{NPC}$).

Ook voor de andere vijf voorbeeldproblemen geldt: als x een ja-instantie is voor het probleem, dan bestaat er een certificaat waarmee je dat eenvoudig (polynomiaal) kunt aantonen.

Certificaten

Probleem	Invoer	Certificaat
HC	$\langle \mathcal{G} \rangle$	Hamiltonkring
SAT	$\langle \phi \rangle$	waarmakende waardering
Kliek	$\langle \mathcal{G}, k \rangle$	kliek met ($\geq$) k knopen
Kleur	$\langle \mathcal{G}, k \rangle$	kleuring met ($\leq$) k kleuren
Sum	$\langle S, t \rangle$	deelverzameling met som t
TSP	$\langle \mathcal{G}, k \rangle$	Hamiltonkring met totaalgewicht $\leq k$

Lineair?

1. Invoer: een array A met $n > 0$ gehele getallen
   Algoritme:

```
1 for i := 1 to n do
2 |   print A[i];
3 od
```

   Complexiteit: $\Theta(n)$

2. Invoer: een geheel getal $n > 0$
   Algoritme:

```
1 for i := 1 to n do
2 |   print '1';
3 od
```

   Complexiteit: $\Theta(n) = \Theta(2^{\lg n})$

En nu nog even dit...

Vraag: wat is eigenlijk de lengte van de invoer?

Voorbeeldprobleem

Gegeven een geheel getal $n > 1$. Heeft n echte delers, m.a.w., is $n = a \times b$ voor zekere $a, b > 1$?

Algoritme:

```
// gewoon alle mogelijke delers proberen
1 gevonden := False;
2 m := 2;
3 while not gevonden and m < n do
4 |   if n mod m = 0 then
5 |   |   gevonden := True;
6 |   else
7 |   |   m := m + 1;
8 |   fi
9 od
// m is nu de kleinste deler > 1 van n
```

Lengte invoer

De worst case-complexiteit van dit algoritme is $\Theta(n)$ (indien we de berekening van $n \bmod m$ tellen als één stap, anders $O(n^2)$).

De lengte van de invoer is het aantal karakters van de codering, in dit geval van het getal n. Als we de unaire codering gebruiken is het algoritme dus lineair in de lengte van de invoer. Nemen we de binaire codering, of in het algemeen de ℓ-aire codering met $\ell > 1$, dan is het algoritme exponentieel in de lengte van de invoer (voor elke $\ell > 1$ dus).

Vraag: is het algoritme polynomiaal of exponentieel?

Primes is in $\mathcal{P}$

In 2002 is bewezen door Manindra Agrawal en zijn PhD-studenten Neeraj Kayal en Nitin Saxena dat het probleem in $\mathcal{P}$ zit. De zogenaamde AKS-test is een priemgetaltest die polynomiaal is in het aantal cijfers (dus de lengte) van n.

In 2006 ontvingen zij hiervoor de Fulkerson Prize (discrete wiskunde) en de Gödel Prize (theoretische informatica).

Ander voorbeeld

Knapzakprobleem

Gegeven een knapzak met capaciteit S (een geheel getal > 0) en n objecten met gewichten $s_1, s_2, \ldots, s_n$ en met waarde $w_1, w_2, \ldots, w_n$. (Alle s_i en w_i zijn geheel en > 0.)
Gevraagd een deelverzameling van de objecten met totaalgewicht $\leq S$ en maximale totaalwaarde.

Het knapzakprobleem kan worden opgelost met een algoritme met complexiteit $O(n \times S)$ (dynamisch programmeren, zie Algoritmiek).

Dit is niet polynomiaal, maar exponentieel als functie van de lengte van de invoer!

$\mathcal{NP}$ iets formeler

$\mathcal{NP}$ is de klasse van beslissingsproblemen waarvoor een niet-deterministisch polynomiaal algoritme bestaat:

- niet-deterministisch:
  je mag een oplossing gokken → certificaat
- polynomiaal:
  deze kan daarna in polynomiale tijd worden geverifieerd (gecontroleerd)
- ja-instantie:
  voor een ja-instantie bestaat er een oplossing, en dus een certificaat waarmee je aantoont dat het inderdaad een ja-instantie is

Niet-deterministisch: parallel met Automata Theory

Voor een niet-deterministische eindige automaat (NFA) $\mathcal{A}$ geldt: een woord w zit in de taal van $\mathcal{A}$ d.e.s.d.a. er minstens één executie van $\mathcal{A}$ bestaat waarin w wordt geaccepteerd.

M.a.w.:

- er mogen ook executies bestaan waarin w wordt verworpen.
- een woord zit *niet* in de taal van $\mathcal{A}$ d.e.s.d.a. er *geen enkele* executie van $\mathcal{A}$ bestaat waarin w wordt geaccepteerd.

Hetzelfde geldt voor niet-deterministische algoritmen:

- voor ja-instanties moet minstens één geldige executie bestaan die een oplossing vindt, maar er mogen ook allerlei ongeldige executies bestaan.
- voor nee-instanties mag geen enkele geldige executie bestaan.

Niet-deterministisch algoritme

We kunnen een niet-deterministisch algoritme omschrijven als bestaande uit drie fases: gokken, verifiëren en uitvoer. We gebruiken een dergelijke omschrijving in de (precieze) definitie van $\mathcal{NP}$.

We bekijken de drie fasen in detail:

1. Niet-deterministische gokfase
   Er wordt een willekeurige string s in het geheugen geschreven. Elke keer dat het algoritme executeert kan dit een andere string zijn.

   Deze string is vaak een soort gok van de oplossing van het probleem: het beoogde certificaat; s kan echter ook een onzinstring blijken te zijn.

Niet-deterministisch algoritme

2. Deterministisch verificatiefase
   Zowel de invoer x van het probleem als de string s mogen hier worden gebruikt. Er wordt True of False geretourneerd, of het programma stopt nooit (het kan bijvoorbeeld belanden in een oneindige loop).

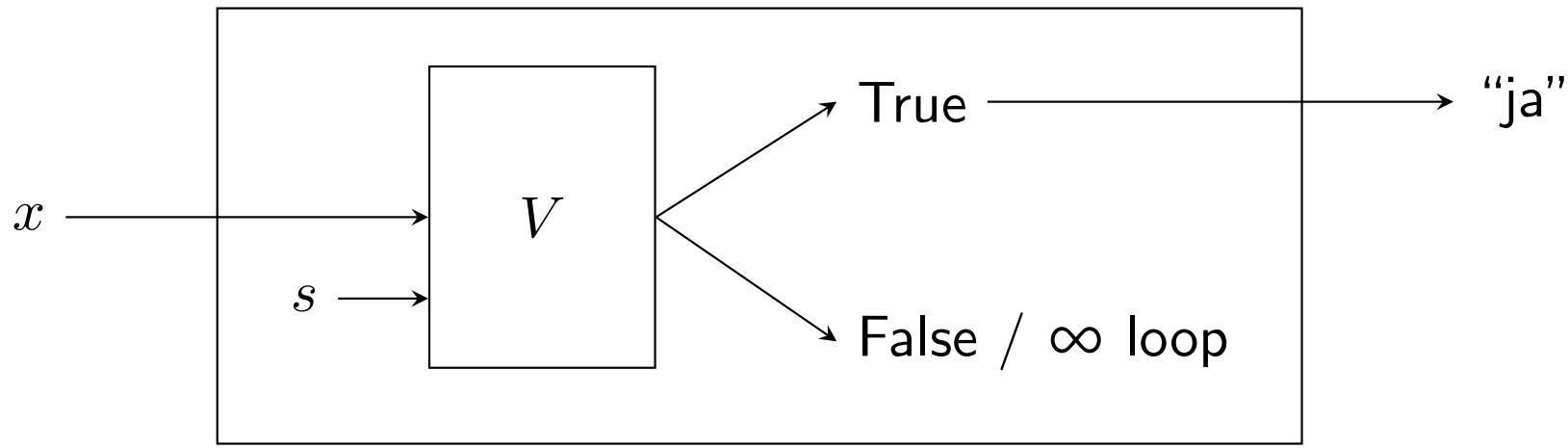
   Hier wordt gecontroleerd of s een oplossing is van het probleem bij de gegeven invoer x, m.a.w. er wordt gecontroleerd of s een ja-antwoord rechtvaardigt.

Niet-deterministisch algoritme

3. Uitvoerfase
   Als fase 2 (verificatie) True retourneert, geeft het algoritme antwoord "ja". Anders is er geen uitvoer.

Het aantal stappen dat een niet-deterministisch algoritme doet is het aantal stappen dat in de gokfase nodig is om s te schrijven (dus het aantal karakters waaruit s bestaat) plus het aantal stappen dat wordt gedaan in de verificatiefase (plus één stap voor de uitvoerfase).

Schematisch: een executie



fase 1: s wordt gegenereerd (niet-deterministisch)

fase 2: verificatiefase (deterministisch)

fase 3: uitvoerstap

Vragen

Bij een niet-deterministisch algoritme zijn verschillende executies mogelijk voor dezelfde invoer x, afhankelijk van de gegokte s. Dus:

(a) Wat is *het* antwoord (ja/nee) van het algoritme voor invoer x?

(b) Wanneer noemen ze zo'n algoritme polynomiaal?

Antwoord

Definitie

Het antwoord van een niet-deterministisch algoritme A voor invoer x is "ja" $\iff$ er is een executie* van A die "ja" geeft als uitvoer $\iff$ er is een string s waarvoor fase 2 True oplevert. Het antwoord van A is "nee" als er voor geen enkele executie, dus voor geen enkele string s, een uitvoer is.

Er geldt dus: het antwoord van zo'n niet-deterministisch algoritme A voor invoer x is "ja"† $\iff$ er bestaat een string s (certificaat) waarmee je kunt aantonen (in fase 2) dat x een ja-instantie is.

*dus een string s

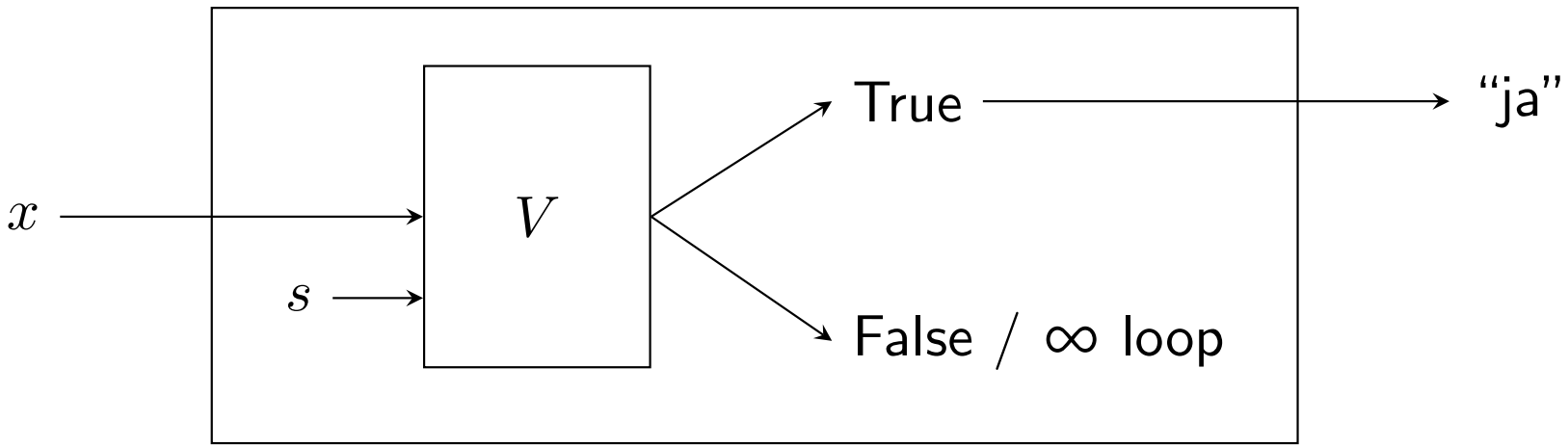
†ofwel: x is een ja-instantie

De klasse $\mathcal{NP}$

- een niet-deterministisch algoritme heet polynomiaal begrensd als voor elke invoer x waarvoor het antwoord "ja" is, er een executie is van het algoritme die "ja" oplevert in hooguit $O(|x|^{\ell})$ stappen (voor zekere $\ell \geq 0$).
  Dientengevolge mag in dat geval de string s niet te lang zijn (polynomiaal in $|x|$), en het verificatiealgoritme uit fase 2 moet polynomiaal zijn begrensd in $|x|$ (en $|s|$).
- $\mathcal{NP}$ is nu de klasse van beslissingsproblemen waarvoor er een polynomiaal begrensd niet-deterministisch algoritme bestaat.
- $\mathcal{NP}$ betekent: $\mathcal{N}$on-deterministisc $\mathcal{P}$olynomial time

$P \in \mathcal{NP}$

Niet-deterministisch en polynomiaal:



voor zekere ja-executie:

fase 1: $O(|s|) \subseteq O(|x|^{\ell})$

fase 2: $O(|s|^{p} \times |x|^{q}) \subseteq O(|x|^{r})$

fase 3: $O(1)$

Voorbeeldproblemen

Stelling. Alle zes voorbeeldproblemen zitten in $\mathcal{NP}$.

Voorbeeld 1: Kleur
Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal $k > 0$. Bestaat er een kleuring van $\mathcal{G}$ die hooguit k kleuren gebruikt (ofwel, is $\mathcal{G}$ k-kleurbaar)?

Voorbeeld 2: HC
Gegeven een (gerichte of ongerichte) graaf $\mathcal{G} = (V, E)$. Heeft deze graaf een Hamiltonkring?
Laat hierna $V = \{1, 2, \ldots, n\}$ en dus $|V| = n$.

Voorbeeld 3: SAT
Gegeven een logische formule ϕ in CNF. Bestaat er een waardering van de in ϕ voorkomende logische variabelen die ϕ waar maakt?

Kleur $\in \mathcal{NP}$

Laat $V = \{1, 2, \ldots, n\}$ en de mogelijke kleuren $1, 2, \ldots, k$. Een polynomiaal begrensd *niet-deterministisch algoritme* A voor Kleur, met als invoer $x = \langle \mathcal{G}, k \rangle$, is:

1. Fase 1 (gokfase)
   Er wordt een string s gegenereerd, hierna te interpreteren als een rij gehele getallen.

2. Fase 2 (verificatiefase)
   Er wordt gecontroleerd of s een goede kleuring voorstelt (s_i wordt geïnterpreteerd als de kleur van knoop i):

   (1) controleer of er precies $n = |V|$ integers staan (elke knoop een kleur): kan polynomiaal, $O(|s|)$

   (2) controleer of elke integer tussen 1 en k is (er worden k kleuren gebruikt): kan polynomiaal, $O(|s|)$

   (3) controleer of aangrenzende knopen verschillend zijn gekleurd. Takken (v, w) aflopen en in s de kleur van v en w opzoeken en vergelijken: $O(|E| \times |s|) \subseteq O(|\mathcal{G}| \times |s|) \subseteq O(|x| \times |s|)$.

Kleur $\in \mathcal{NP}$

Als de drie tests positief zijn wordt True geretourneerd. Zodra een test negatief uitvalt wordt False geretourneerd (of wordt een oneindige loop begonnen).

3. Fase 3 (uitvoerfase)
   Als fase 2 True oplevert wordt "ja" uitgevoerd, anders geen uitvoer.

Merk op. Tests (1) en (2) controleren of s een kleuring van de knopen voorstelt, test (3) controleert of de kleuring correct is.

Er geldt: het antwoord van A op invoer $x = \langle \mathcal{G}, k \rangle$ is "ja" $\Longleftrightarrow$ er bestaat een goede string s waarop fase 2 True oplevert $\Longleftrightarrow$ er bestaat een correcte kleuring van de knopen van $\mathcal{G} \Longleftrightarrow x = \langle \mathcal{G}, k \rangle$ is een ja-instantie voor Kleur.

Verder: voor een ja-executie, dus met s een correcte kleuring, is $|s| \in O(|V|) \subseteq O(|x|)$. Ergo: A is polynomiaal begrensd.

HC $\in \mathcal{NP}$

Een polynomiaal begrensd *niet-deterministisch algoritme* voor HC (invoer $\mathcal{G}, n = |V|, V = \{1, 2, \ldots, n\}$) (zie ook dictaat):

1. Fase 1 (gokfase)
   Er wordt een string s gegenereerd, hierna te interpreteren als een rij gehele getallen.
2. Fase 2 (verificatiefase)
   Er wordt gecontroleerd of s een Hamiltonkring voorstelt:
   (1) controleer dat er precies n integers staan: $O(|s|)$
   (2) controleer dat elke integer tussen 1 en n is: $O(|s|)$
   (3) controleer dat alle knopen uit s verschillen: $O(|s|^2)$*
   (4) controleer dat tussen opeenvolgende knopen uit s een tak zit in de graaf (en tussen de eerste en de laatste): $O(|s| \times |E|) \subseteq O(|s| \times |\mathcal{G}|)$.

*kan ook in bijvoorbeeld $O(n \lg n)$, maar dat maakt nu niet uit

HC $\in \mathcal{NP}$

Als de vier tests positief zijn wordt True geretourneerd. Zodra een test negatief uitvalt wordt False geretourneerd (of wordt een oneindige loop begonnen).

3. Fase 3 (uitvoerfase)
   Als fase 2 True oplevert wordt "ja" uitgevoerd, anders geen uitvoer.

Merk op. Tests (1) t/m (3) controleren dat s een rij van n verschillende knopen van $\mathcal{G}$ voorstelt (syntactische controle), test (4) controleert dat het een Hamiltonkring is.

Voor ja-instanties bestaat er een string s die een Hamiltonkring voorstelt, en dus een executie die "ja" oplevert, waarbij $|s| \in O(|\mathcal{G}|)$. Zo'n ja-executie is dus polynomiaal in $|\mathcal{G}|$. Voor nee-instanties bestaat zo'n string s niet.

SAT $\in \mathcal{NP}$

SAT: Gegeven een logische formule ϕ in CNF. Bestaat er een waardering van de in ϕ voorkomende logische variabelen zodat ϕ de waarde True krijgt (dus een waardering die ϕ waar maakt)?

Een logische formule ϕ staat in CNF (conjunctieve normaalvorm) als hij bestaat uit een conjunctie ($\wedge$) van clausules (disjuncties, $\vee$).

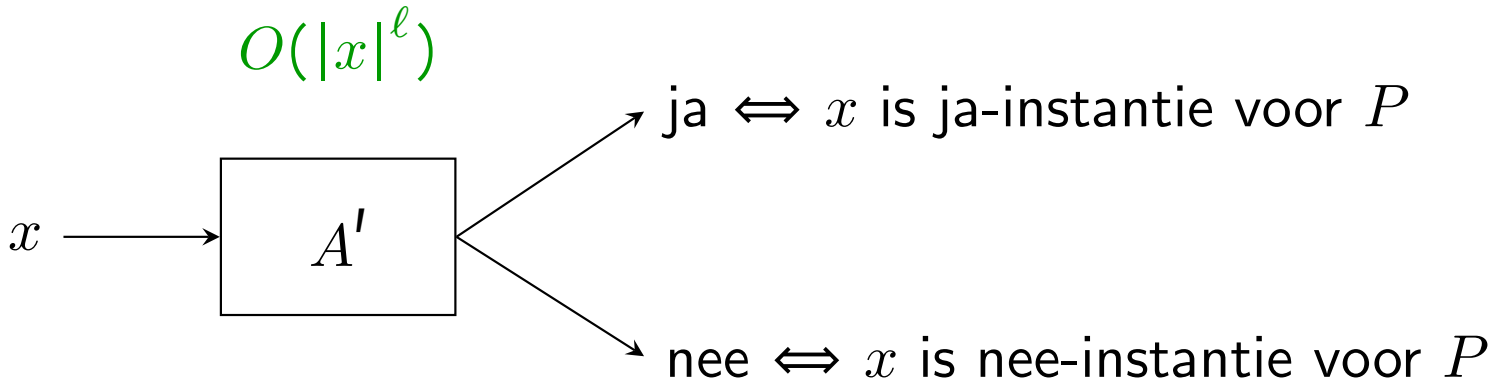
Voorbeeld: $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_1$.

Opgave
Geef een niet-deterministisch algoritme voor SAT.

$\mathcal{P} \subseteq \mathcal{NP}$

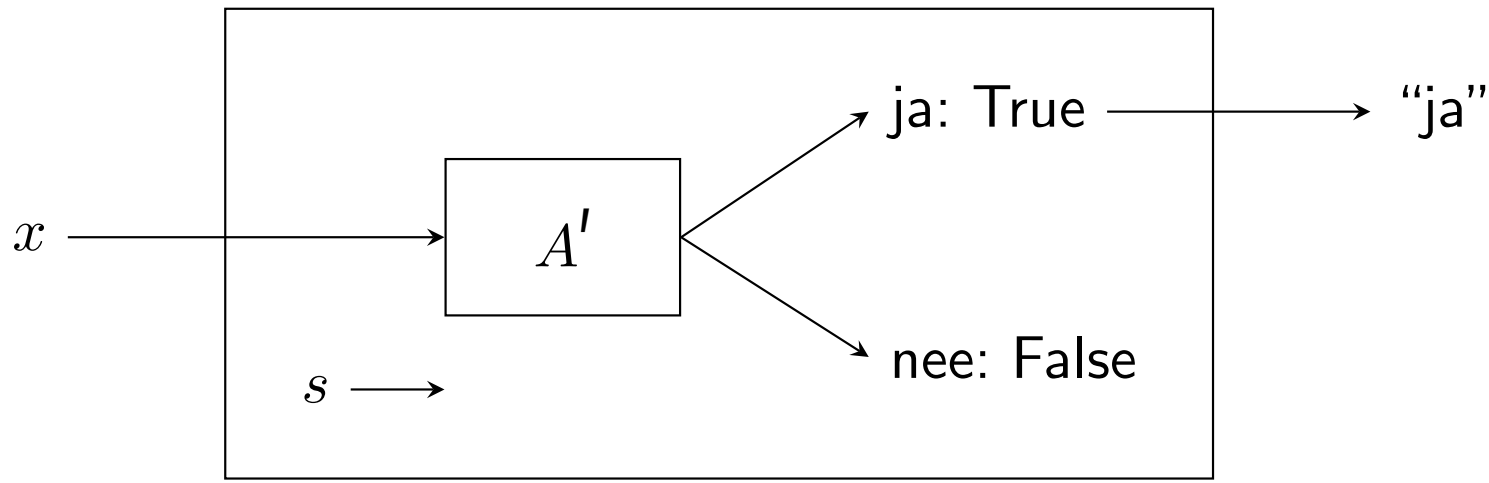
Stelling: $\mathcal{P} \subseteq \mathcal{NP}$

Bewijs:
Neem een willekeurig probleem $P \in \mathcal{P}$. Dan is er een polynomiaal deterministisch algoritme A' dat P oplost.



Het volgende niet-deterministische algoritme A is dan een polynomiaal begrensd algoritme voor P.

$\mathcal{P} \subseteq \mathcal{NP}$



fase 1: s wordt gegenereerd; $O(|s|)$

fase 2: negeer s en voer A' uit op x; $O(|x|^{\ell})$

fase 3: $O(1)$

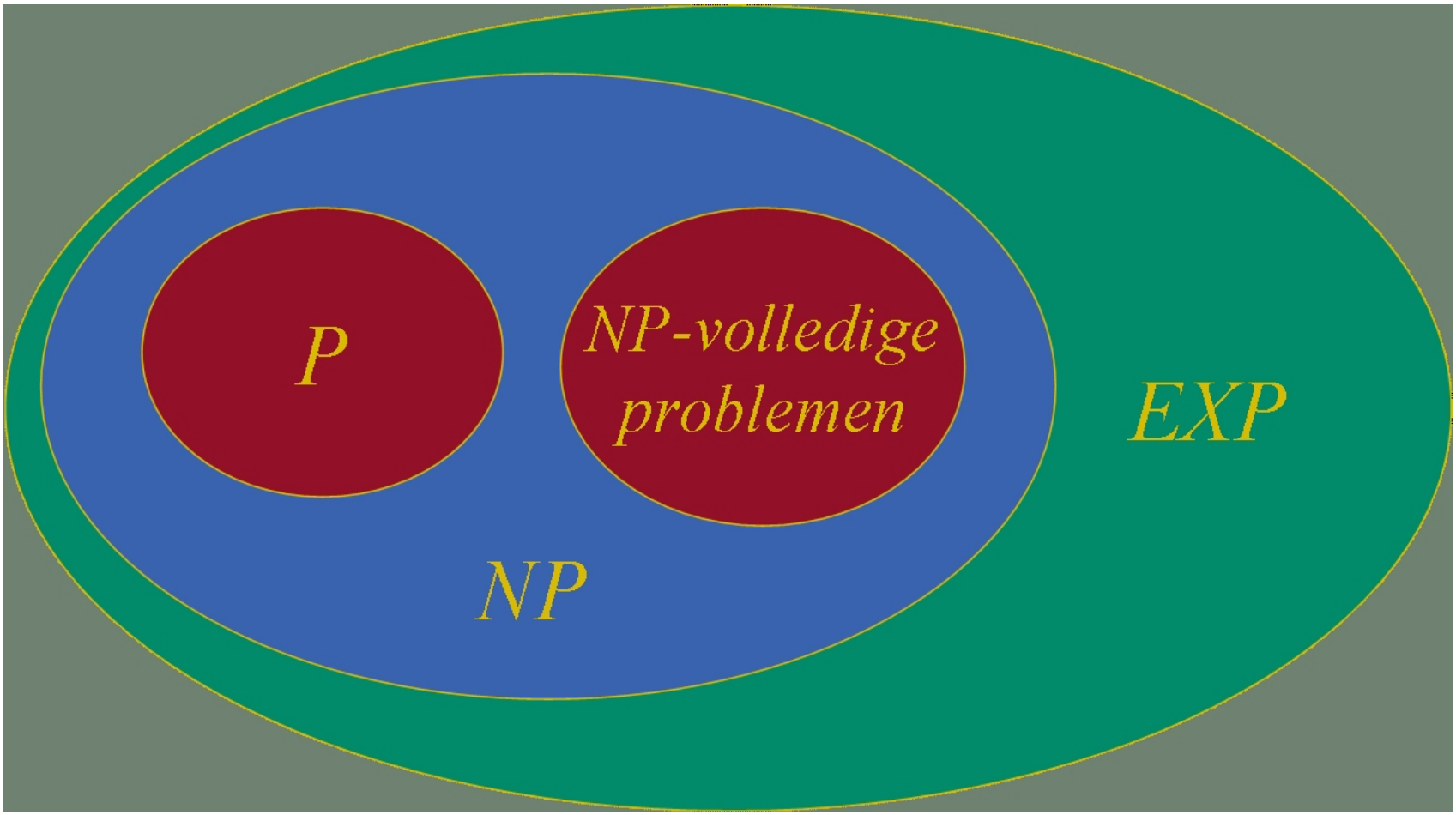
$\mathcal{NPC}$

De klasse van NP-volledige problemen $\mathcal{NPC}$ (Engels: NP-complete) bevat de *moeilijkste* problemen in $\mathcal{NP}$ en heeft de volgende interessante eigenschap:

1. als er een polynomiaal algoritme bestaat voor ook maar één NP-volledig probleem, dan is meteen elk NP-volledig probleem oplosbaar in polynomiale tijd.*
2. omgekeerd: als er van één enkel NP-volledig algoritme wordt bewezen dat het onhandelbaar is, dan zijn alle NP-volledige problemen onhandelbaar.

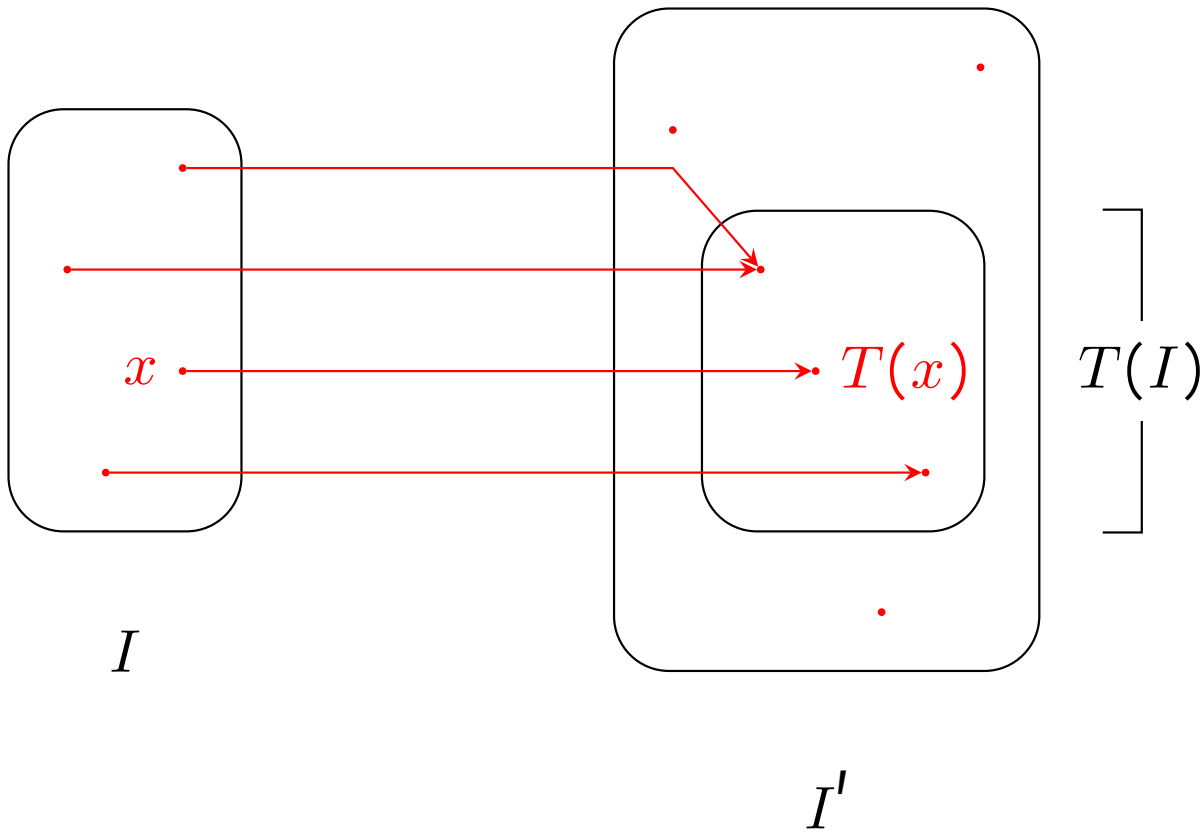
*sterker nog: dan is elk probleem in $\mathcal{NP}$ oplosbaar in polynomiale tijd, dus $\mathcal{P} = \mathcal{NP}$

De meest waarschijnlijke relatie



Reducties

Zij T een functie van de invoerverzameling I van een beslissingsprobleem P naar de invoerverzameling I' van een beslissingsprobleem Q. T beeldt dus elke $x \in I$ af op een $T(x) \in I'$.



Reducties

Definitie

T heet een polynomiale reductie (of *polynomiale transformatie*) van P naar Q als geldt:

1. T kan worden berekend in polynomiaal begrensde tijd (als functie van $|x|$). D.w.z.: de constructie van $T(x)$ uit x kan in $O(|x|^k)$ stappen in de worst case ($k \geq 0$).
2. voor elke x uit I geldt: als x een ja-instantie is voor P dan is $T(x)$ een ja-instantie voor Q.
3. voor elke x uit I geldt: als x een nee-instantie is voor P dan is $T(x)$ een nee-instantie voor Q.

3'. voor elke x uit I geldt: als $T(x)$ een ja-instantie is voor Q dan is x een ja-instantie voor P. (Dit is equivalent met 3.)

Reduceerbaar

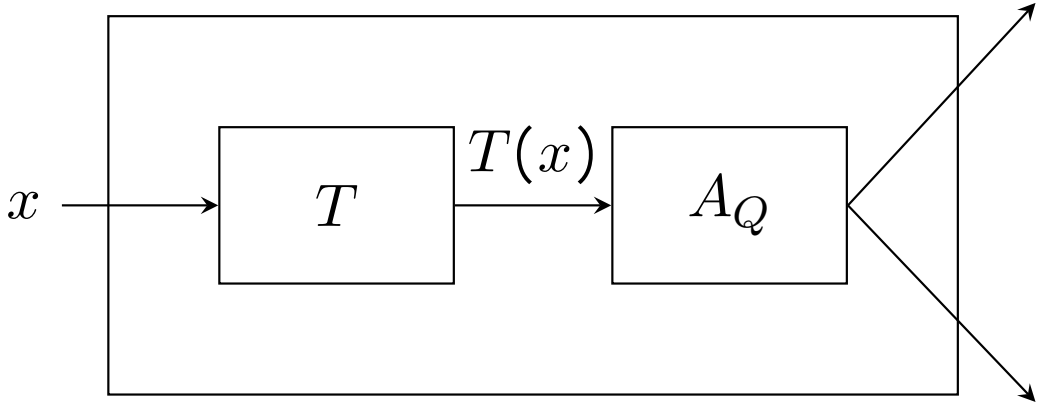
Definitie
Een probleem P is polynomiaal reduceerbaar (of polynomiaal transformeerbaar) naar Q als er een polynomiale reductie bestaat van P naar Q.

Notatie: $P \leq_{\mathsf{P}} Q$.

Stelling
Als $P \leq_{\mathsf{P}} Q$ en $Q \in \mathcal{P}$, dan ook $P \in \mathcal{P}$.

Bewijs stelling

Laat A_Q een polynomiaal deterministisch algoritme zijn voor Q.
Een polynomiaal deterministisch algoritme voor P is dan:



ja $\Leftrightarrow$ $T(x)$ is ja-instantie van Q $\Leftrightarrow$ x is ja-instantie van P

nee $\Leftrightarrow$ $T(x)$ is nee-instantie van Q $\Leftrightarrow$ x is nee-instantie van P

$O(|x|^k) \cdot O(|T(x)|^\ell)$

$O(|x|^{k\ell})$

reductie-eigenschap

Samengevat: reducties

$P \leq_{\mathsf{P}} Q$ betekent dat er een polynomiale reductie T bestaat van P naar Q:

1. T beeldt elke invoer x van beslissingsprobleem P af op een invoer $T(x)$ van beslissingsprobleem Q.
2. de constructie van $T(x)$ uit x is polynomiaal ($O(|x|^k)$).
3. reductie-eigenschap: voor elke x uit I (de invoerverzameling van P) geldt: x is een ja-instantie voor $P \Leftrightarrow T(x)$ is een ja-instantie voor Q.

NP-hard en NP-volledig

Definitie
Een probleem Q is NP-hard (ook wel: NP-moeilijk) als elk probleem P in $\mathcal{NP}$ polynomiaal reduceerbaar is tot Q: dus $P \leq_{\mathsf{P}} Q$ voor alle $P \in \mathcal{NP}$.

Definitie
Een probleem Q is NP-volledig als

1. $Q \in \mathcal{NP}$
2. Q is NP-hard

Notatie De klasse van NP-volledige problemen geven we aan met $\mathcal{NPC}$ (NP-complete).

$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

Stelling

Als een of ander willekeurig NP-volledig probleem in $\mathcal{P}$ zit, dan is $\mathcal{P} = \mathcal{NP}$.

Dit betekent dus: als één enkel NP-volledig probleem P polynomiaal begrensd is, dan zijn alle problemen uit $\mathcal{NP}$ polynomiaal begrensd.

Omgekeerd: als *een* willekeurig probleem in $\mathcal{NP}$ zeker niet polynomiaal begrensd is, dan zijn *alle* NP-volledige problemen niet polynomiaal begrensd.

Om te onthouden van vandaag

- de klasse $\mathcal{NP}$: beslissingsproblemen waarvoor een niet-deterministisch polynomiaal algoritme bestaat
  - niet-deterministisch: je mag een oplossing gokken
  - polynomiaal: oplossing kan in polynomiale tijd worden geverifieerd (gecontroleerd)
  - ja-instantie: alle invoeren waarvoor *een* executie bestaat die "ja" uitvoert
- niet-deterministisch polynomiaal algoritme:
  1. gokfase: genereer string s
  2. verificatiefase: controleer of s een correcte oplossing is in $O(|s|^p \times |x|^q)$
  3. uitvoerfase: antwoord "ja" als de controler slaagt, anders geen uitvoer
  4. als "ja": $|s| \in O(|x|^k)$ en fase 2 dus in $O(|x|^r)$

(Werk)college

Volgende college: dinsdag 18 april, 9u00–10u45, zaal 412 (Snellius)

Werkcollege: zodadelijk van 11u00 tot 12u45, computerzalen 302–304 en 303 (Snellius, onderaan de trap)
Opgaven uit het dictaat: 54, 59, 61

Huiswerk

Huiswerk 2

Is nagekeken, cijfers en feedback staan op Brightspace.

Huiswerk 3

Komt volgende week.

Huiswerk 4

Vervalt, wordt samengevoegd met huiswerk 3.