Inhoudsopgave

[9.1 De complexiteitsklasse P 2](#_Toc92023628)

[9.2 Reducties 3](#_Toc92023629)

[9.3 Compleetheid en de klasse NP 4](#_Toc92023630)

Complexiteitstheorie

Dit is de theorie die problemen classificeert volgens hun moeilijkheidsgraad. Deze eerste klasse van problemen die besproken wordt is de klasse P => de eenvoudigste problemen. Niet alle problemen horen tot de klasse P, er zijn ook problemen zoals Turning’s stopprobleem, die onbeslisbaar zijn.

Reducties geven een manier om aan te tonen dat een bepaald probleem minstens zo moeilijk is als een ander probleem. Dit leidt tot het begrip compleetheid voor een bepaalde klasse.

De klasse NP wordt geïntroduceerd als de klasse waarvan oplossingen eenvoudig te verifiëren zijn. De klasse NP is zeer groot en bevat ook een heel aantal NP-complete problemen.

Het is een open vraag of de klasse NP strikt groter is dan de klasse P, en dit is ook de belangrijkste open vraag binnen de informatica.

# 9.1 De complexiteitsklasse P

In de complexiteitstheorie is men geïnteresseerd om gemakkelijke van moeilijke problemen te onderscheiden en dit op basis van hun tijdscomplexiteit.

**Definitie**: De klasse P is de verzameling van alle problemen die in polynomiale\* tijd kunnen opgelost worden door een algoritme.

**\*polynomiaal:** we bedoelen dat de uitvoeringstijd van het algoritme  is waarbij n de grootte van de invoer voorstelt en k een constante is (onafhankelijk van n).

De meeste van de problemen die we in deze cursus hebben gezien behoren tot de klasse P

* Vinden van een korste pad tussen twee knopen in een gewogen graaf (Dijkstra)
* Vinden dan een minimale kost opspannende boom (Prim en Kruskal)

Turing’s stopprobleem:

Schrijf een algoritme A dat bepaalt of een willekeurig programma P met als input I stopt of niet.

* Het blijkt dat het stopprobleem een onbeslisbaar probleem is: het algoritme A bestaat niet!

**DUS** Turing’s stopprobleem is onbeslisbaar.

**Bewijs:** we moeten aantonen dat er geen algoritme A bestaat dat voor alle programma’s P en elke mogelijke invoer I kan bepalen of het programma P stopt voor de invoer I of niet.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Beide gevallen leiden tot een contradictie die een logisch gevolg is van het bestaan van het programma A. Dit toont aan dat het programma A niet kan bestaan en dat het stopprobleem inderdaad onbeslisbaar is.

Turning’s stopprobleem toont aan dat er wel degelijk grenzen zijn aan wat er kan berekend worden mbv een computer.

# 9.2 Reducties

Reducties zijn een fundamenteel begrip in de informatica en worden in de praktijk ook vaak gebruikt.

**Voorbeeld:** om de mediaan van een rij getallen te vinden kunnen we eenvoudigweg deze rij sorteren en dan het middelste getal of het gemiddelde van de twee middelste getallen teruggeven. Het vinden van de mediaan reduceert tot sorteren, of andere gezegd sorteren kan gebruikt worden om het probleem van het vinden van de mediaan op te lossen.

We maken dit nu duidelijker adhv een definitie:

**Definitie:** een probleem *π*1 reduceert tot een probleem *π*2 wanneer een polynomiaal algoritme voor π2 kan gebruikt worden om probleem π1 op te lossen in polynomiale tijd.



Afbeelding met tekst

Automatisch gegenereerde beschrijving

Met andere woorden: π1 reduceert tot π2 dan is π2 minstens zo moeilijk als π1, want je kan π1 niet alleen gebruiken om π1 op te lossen maar eventueel ook om nog andere zaken te doen.

# 9.3 Compleetheid en de klasse NP

Veronderstel dat C een of andere klasse van problemen is. We wensen nu een definitie die zegt dat her probleem π het moeilijkste probleem is van deze klasse of juister gezegd dat het minstens zo moeilijk is als alle problemen uit deze klasse. Dit is de definitie van C-compleetheid.

**Definitie:** als C een klasse van problemen is dan is een probleem π C-compleet als en slechts als π tot de klasse C behoort en alle problemen uit de klasse C reduceren naar π.

Als we van een bepaald probleem willen aantonen dat het een moeilijk probleem is, dan kunnen we proberen aan te tonen dat het een C-compleet probleem is voor een zeer grote klasse van problemen C.

Als we wensen aan te tonen dat het handelsreizigersprobleem moeilijk is, dan zouden we proberen om alle klasse C alle computatsionele problemen te nemen. Dit werkt niet want Turing’s probleem is strikt moeilijker dan het handelsreizigersprobleem. Dit probleem kan opgelost worden mbv een algoritme met exponentiele uitvoeringstijd terwijl voor het stopprobleem helemaal geen algoritme bestaat.

Het handelsreizigersprobleem kan opgelost worden met een brute kracht algoritme. We wensen nu aan te tonen dat het probleem minstens zo moeilijk is als alle problemen die kunnen opgelost worden met een brute kracht algoritme:

**Definitie:** de klasse NP bestaat uit de problemen waarvoor oplossingen een lengte hebben die hoogstens polynomiaal is in de lengte van de invoer en waarvoor de correctheid van een oplossing geverifieerd kan worden in polynomiale tijd.

De klasse NP bestaat uit problemen waarvoor het eenvoudig is om te herkennen dat je een oplossing hebt gevonden. Dit is anders dan de klasse P waarvoor het eenvoudig is om te oplossing te vinden.

Voorbeeld: veronderstel dat men zich in het handelsreizigersprobleem afvraagt of er een rondreis bestaat met kost K.

* Wanneer iemand je een voorstel van een rondreis geeft, dan is het eenvoudig om te verifiëren of de totale kost van deze rondreis hoogstens K is. Je moet enkel maar controleren of de som van de kosten van de bogen hoogstens K is. Dit kan gebeuren in lineaire tijd.

Alles wat nodig is om tot de klasse NP te behoren is dat men op een efficiënte manier oplossingen kan herkennen.

Wanneer een probleem NP-compleet is, dan betekent dit dat het probleem minstens even moeilijk is als alle andere problemen binnen NP.

* Dwz dat het zeer moeilijk/onmogelijk is een algoritme te bedenken dat kan uitgevoerd worden in polynomiale tijd.
  + Het vinden van zo een algoritme zou betekenen dat alle NP problemen opgelost zouden kunnen worden in polynomiale tijd.

Om aan te tonen dat een probleem NP-compleet is, toont men eerst aan dat het probleem tot de klasse NP behoort. In een tweede klas reduceert men een gekend NP-compleet probleem tot het nieuwe probleem.

**Voorbeeld:** gegeven is een gewogen complete ongerichte graag G en een reëel getal d, bestaat er een rondreis die alle knopen juist eenmaal bezoekt zodanig dat de kost van de rondreis hoogstens d is. Dit noemen we TSP probleem.

Het is duidelijk dat dit beslissingsprobleem tot de klasse NP behoort: als iemand je een voorstel doet van een rondreis, dan is het eenvoudig om te concluderen of het een geldige rondreis is en of de kost inderdaad hoogstens d is.

We veronderstellen nu dat we reeds weten dat het Hamiltoniaanse cykel probleem een NP-compleet probleem is. Gegeven is een ongewogen graaf G, bestaat er een Hamilioniaanse cykel die alle knopen van de graaf bevat? Dit probleem noemen we HC.

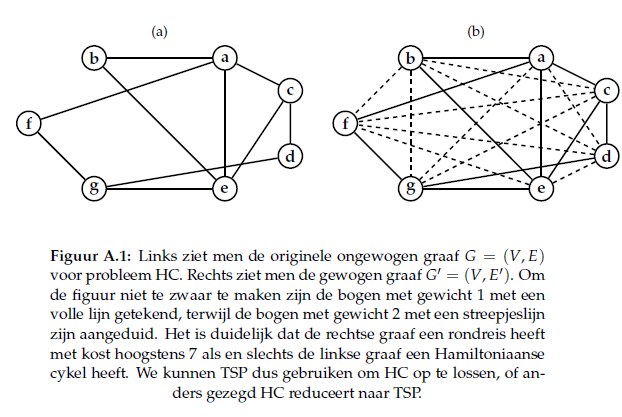
We tonen aan dat HC reduceert tot TSP. Gegeven een graaf G waarop we HC willen toepassen. Construeer nu een nieuwe (gewogen) en complete graaf G’ als volgt:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We tonen nu aan dat het antwoord op HC voor G “ja” is als en slechts als het antwoord op TSP voor G’ met d = n “ja” is.

Veronderstel dat G een hamiltoniaanse cykel bezit, dan kunnen we in G’ al de bogen met gewicht 1 gevolgd worden om een rondreis te bekomen met gewicht gelijk aan n. En dus als HC “ja” heeft, is TSP ook “ja”.



# 9.4 De klasse P versus NP

Het is duidelijk dat de klasse P tot de klasse NP behoort, wanneer men efficiënt een oplossing kan berekenen (klasse P) kan men ook efficiënt controleren of een gegeven oplossing correct is (klasse NP).

De cruciale vraag is echter of de twee klassen samenvallen of niet. Tot op heden is het antwoord hierop nog onbekend, al gaan de meeste computerweterschappers er wel van uit dat de klasse NP strikt groter is dan de klasse P.

Wanneer men in de praktijk te maken heeft met een NP-compleet probleem, dan kan men dit probleem meestal slechts voor kleine instanties exact oplossen binnen een redelijke tijd.