1 Complexiteit

Het college van vandaag gaat over complexiteit van algoritmes. In het boek hoort hier hoofdstuk 8.1 -8.5 bij. Bij complexiteitstheorie is de belangrijkste kernvraag: **Hoe goed is een algoritme?**

Wat bedoelen we met die vraag? Een algoritme voor een bepaald probleem is een stappenplan dat een probleem kan oplossen. D.w.z. voor iedere instantie van het probleem geeft het algoritme het juiste antwoord.

Voor sommige problemen bestaan er algoritmes die **snel** de optimale oplossing geven. Er zijn ook problemen die waarvoor (nog) geen snelle algoritmes bestaan die de optimale oplossing geven. Een alternatief is dan een benaderingsalgoritme dat weliswaar snel is, maar niet de optimale oplossing geeft.

'makkelijk' probleem	'moeilijk' probleem
optimaal en snel	optimaal en traag of benadering en snel

Het is wenselijk om uit te kunnen drukken hoe snel een algoritme is; de tijdsduur die nodig is om het eindantwoord te geven. Een maat daarvoor is de **tijdscomplexiteit** (of complexiteit), het aantal **elementaire stappen** dat maximaal nodig is in het algoritme. Elementaire stappen zijn optelling, vermenigvuldiging, toekenning, vergelijking, enz. We tellen in elemtaire stappen omdat de tijdsduur zelf sterk afhankelijk is van de gebruikte computer.

De complexiteit van een algoritme hangt af van en wordt uitgedrukt in de **invoergrootte** van het probleem. De invoer is bijv. een graaf, een verz. getallen, een matrix enzovoorts. Afhankelijk van de codering heeft deze input een bepaalde grootte (meestal noteren we dat als n). Wat de invloed is van de manier van coderen is, daar gaan we later op in.

De tijdcomplexiteit van een algoritme A wordt gegeven door:

$$f_A(n) = \max_{\text{instanties } I \text{ van grootte } n} \{\text{tijd nodig voor } I\}$$

We zien hier dat dit een functie van n is. Om deze complexiteit te kunnen vergelijken voeren we de volgende notatie in voor **groeisnelheden**.

- $f(n) = \mathcal{O}(g(n))$ als er n_0 en c bestaan zodat voor alle $n > n_0$ geldt dat $f(n) \leq c \cdot g(n)$. We zeggen dat f niet sneller groeit dan g.
- $f(n) = \Omega(g(n))$ als er n_0 en c bestaan zodat voor alle $n > n_0$ geldt dat $f(n) \geq c \cdot g(n)$. We zeggen dat f niet langzamer groeit dan g.
- $f(n) = \Theta(g(n))$ als er n_0 en c_1, c_2 bestaan zodat voor alle $n > n_0$ geldt dat $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. We zeggen dat f en g even snel groeien.

Wanneer noemen we een algoritme voldoende snel? In de praktijk geldt dit voor algoritmes met een **polynomiale groeisnelheid**.
Een algoritme A is polynomiaal als er een $k > 0$ is zodat $f_A(n) = \mathcal{O}(n^k)$.
Elk algoritme waar dit niet voor geldt heet exponentieel.

De groei van polynomiale en exponentiele functies:

functie			
n	10	100	1000
n^2	10^2	10^4	10^6
$10^8 n^4$	10^{12}	10^{16}	10^{20}
2^n	$\approx 10^3$	$\approx 10^{30}$	$\approx 10^{300}$
$n!$	$\approx 10^6$	$\approx 10^{158}$	$\approx 10^{2567}$

Polynomiale functies hebben meer voordeel van snellere technologie:

functie	grootte van oplosbare instantie	grootte van oplosbare instantie in 10x snellere computer
n	10^{12}	10^{13}
n^2	10^6	3.16×10^6
$10^8 n^4$	10	18
2^n	40	43
$n!$	14	15

De grootte van een instantie die in een vaste tijd kan worden opgelost wordt bij een 10x snellere computer vermenigvuldigd door een constante bij polynomiale algoritmes. Dit terwijl exponentiele algoritmes slechts een additieve toename hebben.

Een algoritme met orde n^{100} zal natuurlijk niet naar tevredenheid werken, maar in de praktijk blijkt dat als er eenmaal een polynomiaal algoritme is gevonden, dan is het vaak ook mogelijk om dan een algoritme van orde n^4 of lager te vinden.

De grootte van instantie is de lengte (het aantal symbolen) in een 'redelijke' codering. Alle redelijke coderingen zijn polynomiaal gerelateerd.

Voorbeeld 1. $G = (V, E)$
Codering 1: Adjecency matrix (structuurmatrix): de lengte van de codering is dan $|V|^2$.
Codering 2: Adjecency lijst (burenlijst): de lengte van de codering is dan $|V| + 2|E|$.
Afhankelijk van de toepassing zal de ene of de andere codering efficienter zijn. Beide codering zijn polynomiaal gerelateerd, want $|E| < |V|^2$.

Voorbeeld 2: Een (groot) getal P.
Codering 1: Decimaal getalstelsel: de lengte van de codering is ${}^{10}\log P$.
Codering 2: Binair getalstelsel: de lengte van de codering is ${}^{2}\log P$.
Codering 3: Unair getalstelsel: (dus 111111111=9) de lengte van de codering is P.
Als we de lengtes vergelijken zien we dat ${}^{2}\log P / {}^{10}\log P = {}^{2}\log 10$. Codering 1 en 2 verschillen slechts met een constante factor in lengte en zijn dus zeker polynomiaal gerelateerd.
Codering 3 is exponentieel veel langer, want $P = 2^{{}^{2}\log P}$.
Getallen coderen doen we daarom dus altijd in getalstelsel met basis ≥ 2.

Voorbeeld 3. Een Lineair Programmeringprobleem. Hierbij nemen we aan dat A, b en c bestaan uit gehele getallen.
Om dit te coderen moeten we dus alle getallen in deze vectoren en matrix noteren in binaire (of decimale) code. De lengte van de codering wordt dan van de orde $\Theta(mn + log|P|)$, waarbij P het product is van alle nietnul coefficienten.

Analyse van de complexiteit van Dijkstra's algoritme. Input is een gerichte graaf met met een kostenvector c op de pijlen. Noem het aantal knopen $|V|$. Algoritme:

Algoritme	# stappen	#stappen
Initialisering: $W = s,\ \rho(s) = 0.$	$\mathcal{O}(1)$	
begin: zet $\rho(i) := c_{(s,i)}, \forall i \in V \backslash \{s\}$	$\mathcal{O}(\|V\|)$	
while $W \neq V$	$\mathcal{O}(\|V\|)$	
bepaal x zodat $\rho(x) = \min\{\rho(y) \| y \neq W\}$		$\mathcal{O}(\|V\|)$
zet $W := W \cap x$		$\mathcal{O}(1)$
zet $\rho(i) := \min\{\rho(i), \rho(x) + c_{(x,i)}\}\ \forall i \neq W$		$\mathcal{O}(\|V\|)$

Er zijn orde $|V|$ operaties nodig in de eerste twee regels. De 'while'-lus wordt $|V|$ keer uitgevoerd. In die lus worden maximaal orde $|V|$ elementaire operaties uitgevoerd. Het totale algoritme neemt dus tijd $\mathcal{O}(|V|+|V|^2) = \mathcal{O}(|V|^2)$ in beslag.

Voor het kortste pad probleem bestaat dus een polynomiaal algoritme. Dit is dus een 'makkelijk' probleem. Ook LP is een makkelijk probleem waar een polynomiaal algoritme voor bestaat.

Daartegenover staan bijvoorbeeld het handelsreizigersprobleem, of ILP wat 'moeilijke' problemen zijn, waarschijnlijk bestaan hier geen polynomiale algoritmes voor die het probleem optimaal oplossen.

Drie soorten problemen We kunnen 3 verschillende soorten problemen onderscheiden. De zogenaamde optimaliseringsproblemen, evaluatieproblemen en de herkenningsproblemen. Op volgorde van moeilijk naar minder moeilijk.

1. Bij een **optimaliseringsprobleem** wordt een $f \in F$ gezocht, zodat $c(f) \leq c(y)$ for all $y \in F$.
2. Bij een **evaluatieprobleem** wordt de beste waarde gezocht: bepaal $\min\{c(f) | f \in F\}$.
3. Bij een **herkenningsprobleem** wordt de vraag gesteld is er een $f \in F$ zodat $c(f) \leq L$.

Vaak bestaat er een hele directe manier om deze problemen aan elkaar te koppelen. Bijvoorbeeld bij het kortste pad probleem:

1. Wat is het korste pad?

2. Wat is de waarde van het kortste pad?

3. Is er een pad korter dan L?

$\mathcal{P} = \{$herkenningsproblemen die oplosbaar zijn met een polynomiaal algoritme$\}$

Als een poly-tijd algoritme gevonden is behoort het probleem zeker tot klasse $\mathcal{P}$. Andersom als het algoritme nog niet gevonden is, zou het wel *kunnen* bestaan. Alle problemen waarvoor er een polynomiaal algoritme is (gevonden of niet) behoren tot $\mathcal{P}$.

$\mathcal{NP} = \{$herkenningsproblemen waarvoor bij iedere ja-instantie een certificaat bestaat, waarmee in polynomiale tijd nagegaan kan worden dat dit antwoord juist is.$\}$

Een certificaat is een recept/stappenplan waarmee kan worden nagegaan dat het antwoord van een ja-instantie juist is.

Wat we typisch zien is dat alle problemen die wij bespreken in $\mathcal{NP}$ zitten. In het bijzonder volgt uit de definitie dat $\mathcal{P} \subseteq \mathcal{NP}$. Want een polynomiaal algoritme voor een probleem is zelf een polynomiaal certificaat. Een van de belangrijkste openstaande vragen in complexiteitstheorie is of **geldt dat $\mathcal{P} = \mathcal{NP}$?** Over het algemeen wordt het waarschijnlijk gevonden dat ze niet gelijk zijn, dit omdat er al zoveel moeite en denkwerk in zit om polynomiale algoritmes te vinden.