

Formelt set opfattes et problem som en binær relation mellem en mængde af instanser og en mængde af løsninger. For et givet problem arbejder man i praksis med et tilhørende afgørlighedsproblem. Fordelen er, at man for afgørlighedsproblemet kan nøjes med at holde styr på de "korrekte" instanser (dvs. de instanser, der kan accepteres); afgørlighedsproblemet kan således opskrives på mængdeform. For at tydeliggøre forskelle og sammenhænge mellem generelt problem og afgørlighedsproblem forsøges i det følgende også at opskrive de generelle problemer på mængdeform; alle problemer beskrives derfor med følgende notation (jvf. notationen i [CLRS] p. 976):

$$\text{PROBLEM} = \left\{ \langle \text{instans} \rangle \left| \begin{array}{l} \text{præcision af en instans} \\ \text{præcision af tilhørende løsnings-} \\ \text{rum og løsning(er)} \end{array} \right. \right\}$$

34.1-1

Lad $|p_{uv}|$ betegne antallet af kanter i en simpel vej $p : u \rightsquigarrow v$. Vi definerer optimeringsproblemet

$$\text{LONGEST-PATH-LENGTH} = \left\{ \langle G, u, v \rangle \left| \begin{array}{l} G = (V, E) \text{ ikke-orienteret} \\ u, v \in V \\ \exists p : u \rightsquigarrow v \\ \max |p_{uv}| \end{array} \right. \right\}$$

og afgørlighedsproblemet

$$\text{LONGEST-PATH} = \left\{ \langle G, u, v, k \rangle \left| \begin{array}{l} G = (V, E) \text{ ikke-orienteret} \\ u, v \in V \\ k \geq 0 \text{ heltallig} \\ \exists p : u \rightsquigarrow v \\ |p_{uv}| \geq k \end{array} \right. \right\}$$

$\text{LONGEST-PATH-LENGTH} \in \text{P} \Leftrightarrow \text{LONGEST-PATH} \in \text{P}$

\Rightarrow Antag, at $\text{LONGEST-PATH-LENGTH} \in \text{P}$.

Vi søger et svar på $\text{LONGEST-PATH}\langle G, u, v, k \rangle$. Antag, at resultatet af $\text{LONGEST-PATH-LENGTH}\langle G, u, v \rangle$ er \tilde{k} . Svaret på $\text{LONGEST-PATH}\langle G, u, v, k \rangle$ vil nu være det samme som svaret på om $k \leq \tilde{k}$ eller ej.

Det tager polynomiel tid at bestemme \tilde{k} og konstant tid at lave sammenligningen $k \leq \tilde{k}$, hvorfor den samlede køretid vil være polynomiel.

\Leftarrow Antag, at $\text{LONGEST-PATH} \in \text{P}$.

For $k = |V| - 1, \dots, 0$ (man kan også bruge binær søgning) køres $\text{LONGEST-PATH}\langle G, u, v, k \rangle$ indtil svaret er bekræftende. Herefter returneres k som svar på $\text{LONGEST-PATH-LENGTH}\langle G, u, v \rangle$.

Der skal højst laves $|V|$ kald til LONGEST-PATH . Hvert kald tager polynomiel tid i længden af inddata $\langle G, u, v, k \rangle$ som oplagt er mindst $|V|$ bit stor (pga. G). Den samlede køretid vil derfor stadig være polynomiel.

34.1-2

Problemet med at finde den længste simple cykel i en graf kan formelt set beskrives som en binær relation mellem instanser (mængden af grafer) og løsninger (mængden af simple cykler i den givne graf), således at en given graf relateres til den længste simple cykel (el. cykler, idet der godt kan være flere løsninger). Hvis der ikke findes en simpel cykel i den givne graf, er løsningsmængden tom.

Lad $|c|$ betegne antallet af kanter i en simpel cykel c . Vi definerer optimeringsproblemet

$$\text{LONGEST-SIMPLE-CYCLE} = \left\{ \langle G \rangle \left| \begin{array}{l} G = (V, E) \text{ ikke-orienteret} \\ \exists c \\ \max |c| \end{array} \right. \right\}$$

og afgørlighedsproblemet

$$\text{SIMPLE-CYCLE} = \left\{ \langle G, k \rangle \mid \begin{array}{l} G = (V, E) \text{ ikke-orienteret} \\ k \geq 0 \text{ heltallig} \\ \exists c : |c| \geq k \end{array} \right\}$$

34.1-4

Køretiden for 0–1-knapsack-problemet med n ting (med heltallig vægt) og en øvre vægt på W er $O(nW)$ hvis man baserer sig på dynamisk programmering. Da W ikke afhænger af n , vil køretiden ikke være polynomiel i n . Mere præcist skal vi bruge $\lg W$ bit til at repræsentere W , så hvis n f.eks. holdes konstant vil køretiden være eksponentiel i størrelsen af inddata: $W = O(2^{\lg W})$. Algoritmen ligger altså *ikke* i P.

34.1-6

I det følgende antages det, at $L, L_1, L_2 \in P$. Lad desuden $x \in \{0, 1\}^*$ være givet og antag, at x har længden n således, at x kan skrives som $x = \langle x_1, \dots, x_n \rangle$.

Idet vi betragter P som en mængde af sprog, skal vi se, at P er lukket under forskellige sproglige konstruktioner:

Foreningsmængde For $L_1, L_2 \in P$ skal vi se, at $L_1 \cup L_2 \in P$:

Da $L_1, L_2 \in P$ kan vi i polynomiel tid afgøre om $x \in L_1$ eller $x \in L_2$. Ligger x i bare én af disse vil også $x \in L_1 \cup L_2$.

Fællesmængde For $L_1, L_2 \in P$ skal vi se, at $L_1 \cap L_2 \in P$:

Analogt med ovenstående.

Sammensætning For $L_1, L_2 \in P$ skal vi se, at $L_1 L_2 \in P$:

Der er $n - 1$ muligheder for at opdele x : $\langle x_1 | x_2, \dots, x_n \rangle, \dots, \langle x_1, \dots, x_{n-1} | x_n \rangle$. For hver mulighed kan vi i polynomiel tid afgøre om $\langle x_1, \dots, x_i \rangle \in L_1$ og $\langle x_{i+1}, \dots, x_n \rangle \in L_2$ eller ej.

Komplement For $L \in P$ skal vi se, at $\bar{L} \in P$:

Det bemærkes, at $x \in \bar{L}$ er ensbetydende med, at $x \notin L$, hvilket jo kan afgøres i polynomiel tid.

Afslutning (Kleene stjerne) For $L \in P$ skal vi se, at $L^* \in P$:¹

Vi kan tillade os at nøjes med at undersøge, om x ligger i $\{\epsilon\} \cup L \cup L^2 \cup \dots \cup L^n$ (hvor n er længden af x). Med henblik på at løse problemet med dynamisk programmering indfører vi nu delproblemet $T[i, j]$, som angiver, om de første j elementer af x kan accepteres af et sprog "til og med" L^i eller ej; dvs. $T[i, j] \in \{\text{TRUE}, \text{FALSE}\}$ og

$$T[i, j] = \langle x_1, \dots, x_j \rangle \in \{\epsilon\} \cup L \cup L^2 \cup \dots \cup L^i$$

$T[n, n]$ vil altså udtale sig om hvorvidt $x \in L^*$ eller ej. Resultatet af $T[i, j]$ baserer sig på resultaterne af $T_{(i-1)k}$, $k = 1, \dots, j$ afhængig af om delstreng accepteres af "mindre" sprog:

$$T[i, j] = \bigvee \left\{ \begin{array}{l} T[i-1, j] \\ \bigvee_{k=1}^{j-1} (T[i-1, k] \wedge \langle x_{k+1}, \dots, x_j \rangle \in L) \end{array} \right.$$

For hvert af de n^2 delproblemer skal vi bruge polynomiel tid på at afgøre, hvorvidt $\langle x_{k+1}, \dots, x_j \rangle \in L$ eller ej. Den samlede køretid bliver altså også polynomiel, hvormed $L^* \in P$.

¹Vi har allerede set, at sprog som udspringer af operationerne sammensætning og forening kan afgøres i polynomiel tid og derfor ligger i P. Det er imidlertid ikke til megen nytte her, da vi skal foretage uendeligt mange sådanne operationer for at danne L^* .

34.2-1

Vi betragter sproget

$$\text{GRAPH-ISOMORPHISM} = \left\{ \langle G_1, G_2 \rangle \mid \begin{array}{l} G_1 \text{ og } G_2 \text{ er isomorfe grafer:} \\ f : V_1 \rightarrow V_2 \text{ bijektiv} \\ (u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2 \end{array} \right\}$$

Vi skal se, at GRAPH-ISOMORPHISM \in NP. Det betyder, at vi skal kunne godtgøre en løsning i polynomiell tid: Antag derfor, at der foreligger et forslag til en en-til-en afbildning mellem knuderne i de to grafer. For hver knude i G_1 undersøger vi, om den tilsvarende knude i G_2 har præcis de samme kanter. Køretiden for dette vil være $O(V + E)$, hvilket vil være polynomielt i inddatas størrelse, idet vi skal bruge $O(V + E)$ bit til at repræsentere grafen.

34.2-2

Da grafen er bipartit (to-delelig), kan alle kanter opfattes som gående fra den ene del til den anden del. Antallet af kanter i en hamilton-cykel vil være det samme som antallet af knuder i grafen. Følger man et ulige antal kanter, vil man imidlertid ende i den anden del end den, hvor man startede. Det vil derfor ikke være muligt at konstruere en simpel cykel, der gennemløber alle knuderne.

34.2-4

I det følgende antages det, at $L, L_1, L_2 \in$ NP med tilhørende algoritmer A, A_1 og A_2 . Lad desuden $x \in \{0, 1\}^*$ være givet og antag, at x har længden n således, at x kan skrives som $x = \langle x_1, \dots, x_n \rangle$.

Idet vi betragter NP som en mængde af sprog, skal vi se, at NP er lukket under forskellige sproglige konstruktioner. For at godtgøre, at de forskellige konstruktioner ligger i NP, skal vi altså påvise eksistensen af certifikater, y , så $L = \{x \in \{0, 1\}^* \mid \exists y : A(x, y) = 1\}$, dvs. L præcis udgøres af de elementer, der — med et (korrekt) tilhørende certifikat — kan accepteres af A .

Det bemærkes, at en algoritme som *accepterer* $x \wedge y$ i polynomiell tid kan udvides til at *afgøre* $x \wedge y$ i polynomiell tid, idet man — hvis man ikke har fået et svar, når den forventede (polynomielle) tid er gået — kan stoppe og svare afvisende. I det følgende betegnes sådant udvidede algoritmer med \tilde{A} .

Foreningsmængde For $L_1, L_2 \in$ NP skal vi se, at $L_1 \cup L_2 \in$ NP:

Hvis $x \in L_1 \cup L_2$ kan vi som certifikat bruge enten y_1 eller y_2 : Algoritmen

$$A(x, y) = \tilde{A}_1(x, y_1) \vee \tilde{A}_2(x, y_2)$$

svarer til det ønskede.

Fællesmængde For $L_1, L_2 \in$ NP skal vi se, at $L_1 \cap L_2 \in$ NP:

Hvis $x \in L_1 \cap L_2$ skal vi som certifikat bruge både y_1 og y_2 : Algoritmen

$$A(x, y) = \tilde{A}_1(x, y_1) \wedge \tilde{A}_2(x, y_2)$$

svarer til det ønskede.

Sammensætning For $L_1, L_2 \in$ NP skal vi se, at $L_1 L_2 \in$ NP:

Der er $n - 1$ muligheder for at opdele x : $\langle x_1 | x_2, \dots, x_n \rangle, \dots, \langle x_1, \dots, x_{n-1} | x_n \rangle$. Ligger $\langle x_1, \dots, x_i \rangle$ i L_1 vil det kunne accepteres med et certifikat y_1 . For $\langle x_{i+1}, \dots, x_n \rangle$ ligeledes i L_2 . Algoritmen

$$A(x, y) = \tilde{A}_1(\langle x_1, \dots, x_i \rangle, y_1) \wedge \tilde{A}_2(\langle x_{i+1}, \dots, x_n \rangle, y_2)$$

svarer til det ønskede.

Komplement For $L \in \text{NP}$ skal vi overveje, om $\bar{L} \in \text{NP}$:

Hvis det gælder, vil $\text{NP} = \text{co-NP}$ (hvilket stadig er uafklaret); hvis $\text{NP} \neq \text{co-NP}$ vil $\text{P} \neq \text{NP}$ (jvf. 34.2-10)

Afslutning (Kleene stjerne) For $L \in \text{NP}$ skal vi se, at $L^* \in \text{NP}$:

Vi kan tillade os at nøjes med at undersøge, om x ligger i $\{\epsilon\} \cup L \cup L^2 \cup \dots \cup L^n$. Med henblik på at løse problemet med dynamisk programmering indfører vi nu delproblemet $T[i, j]$, som angiver, om de første j elementer af x kan accepteres af et sprog "til og med" L^i eller ej; dvs. $T[i, j] \in \{\text{TRUE}, \text{FALSE}\}$ og

$$T[i, j] = \langle x_1, \dots, x_j \rangle \in \{\epsilon\} \cup L \cup L^2 \cup \dots \cup L^i$$

$T[n, n]$ vil altså udtale sig om hvorvidt $x \in L^*$ eller ej. Resultatet af $T[i, j]$ baserer sig på resultaterne af $T_{(i-1)k}$, $k = 1, \dots, j$ afhængig af om delstrengene accepteres af "mindre" sprog. Ligger $\langle x_{k+1}, \dots, x_j \rangle$ i L vil det kunne accepteres med et certifikat y :

$$T[i, j] = \bigvee \left\{ \begin{array}{l} T[i-1, j] \\ \bigvee_{k=1}^{j-1} (T[i-1, k] \wedge \tilde{A}(\langle x_{k+1}, \dots, x_j \rangle, y)) \end{array} \right.$$

For hvert af de n^2 delproblemer skal vi bruge polynomiel tid på at afgøre $\langle x_{k+1}, \dots, x_j \rangle \wedge y$. Den samlede køretid bliver altså også polynomiel, hvormed $L^* \in \text{NP}$.

34.2-6

Lad $p : u \xrightarrow{H} v$ betegne en hamilton-vej fra u til v .

$$\text{HAM-PATH} = \left\{ \langle G, u, v \rangle \left| \begin{array}{l} G = (V, E) \\ u, v \in V \\ \exists p : u \xrightarrow{H} v \end{array} \right. \right\}$$

Vi skal se, at $\text{HAM-PATH} \in \text{NP}$: Antag derfor, at der foreligger en mulig hamilton-vej fra u til v . Det skal så godtgøres, at vejen starter i u , slutter i v og iøvrigt besøger alle andre knuder under vejs. Dette kan oplagt gøres i polynomiel tid, hvorfor $\text{HAM-PATH} \in \text{NP}$.

34.2-8 (\leftarrow 34.3-7 \leftarrow 34.4-4)

$$\text{TAUTOLOGY} = \left\{ \langle \phi \rangle \left| \begin{array}{l} \phi \text{ er et boolsk udtryk} \\ \text{bestående af } \neg, \vee, \wedge, () \\ \text{og variableerne } x_1, \dots, x_k \\ x_i \in \{0, 1\}, i = 1, \dots, k \\ \hline \forall (x_1, \dots, x_k) \in \{0, 1\}^k : \\ \phi(x_1, \dots, x_k) = 1 \end{array} \right. \right\}$$

Vi skal se, at $\text{TAUTOLOGY} \in \text{co-NP}$; det kommer pr. definition ud på, at $\overline{\text{TAUTOLOGY}} \in \text{NP}$:

$$\overline{\text{TAUTOLOGY}} = \left\{ \langle \phi \rangle \left| \begin{array}{l} \phi \text{ er et boolsk udtryk} \\ \text{bestående af } \neg, \vee, \wedge, () \\ \text{og variableerne } x_1, \dots, x_k \\ x_i \in \{0, 1\}, i = 1, \dots, k \\ \hline \exists (x_1, \dots, x_k) \in \{0, 1\}^k : \\ \phi(x_1, \dots, x_k) = 0 \end{array} \right. \right\}$$

Foreligger der en mulig tildeling af x_i 'erne kan denne godtgøres ved at sætte ind i ϕ ; dette kan gøres i lineær tid i udtrykkets længde. For senere brug bemærkes, at $\overline{\text{TAUTOLOGY}} = \text{NON-SAT} \in \text{NP}$.

34-3**a)**

En mulig to-farvning kan bestemmes ved først at køre bredde-først-søgning og dernæst tildele alle knuder med lige afstande den ene farve og alle knuder med ulige afstande den anden farve. Afslutningsvis undersøges det, om alle kanter forbinder knuder af forskellig farve. Køretiden vil være $O(E)$.

b)

$$\text{GRAPH-COLOURING} = \left\{ \langle G, k \rangle \left| \begin{array}{l} G = (V, E) \text{ ikke-orienteret} \\ c : V \rightarrow \{1, \dots, k\} \\ c(u) \neq c(v), \forall (u, v) \in E \end{array} \right. \right\}$$

$\text{MINIMUM-GRAPH-COLOURING} \in \text{P} \Leftrightarrow \text{GRAPH-COLOURING} \in \text{P}$

\Rightarrow Antag, at $\text{MINIMUM-GRAPH-COLOURING} \in \text{P}$.

For et givent $k \geq 0$ sammenlignes resultatet af $\text{MINIMUM-GRAPH-COLOURING}\langle G \rangle$ med k , hvorpå der kan svares af- eller bekræftende på $\text{GRAPH-COLOURING}\langle G, k \rangle$.

\Leftarrow Antag, at $\text{GRAPH-COLOURING} \in \text{P}$.

For $k = 1, \dots, |V|$ køres $\text{GRAPH-COLOURING}\langle G, k \rangle$ indtil svaret er bekræftende. Herefter returneres k som svar på $\text{MINIMUM-GRAPH-COLOURING}\langle G \rangle$.