ALGORITMUSELMÉLET

A P és NP osztályok
NP-teljes problémák

13. előadás

A bonyolultság-elmélet alapfogalmai

Az előadás során az algoritmusok (Turing-gépek) idő- és tárigény szerinti hatékonyságát vizsgáljuk. Két problémaosztállyal (nyelvosztállyal) fogunk foglalkozni: a $\mathbf{P}$ és az $\mathbf{NP}$ osztályokkal.

13.1 definíció: ($TIME(t(n))$ nyelvosztály)

$TIME(t(n)) := \{\ \mathrm{L} \subseteq \Sigma^*$; L felismerhető egy $O(t(n))$ időkorlátos determinisztikus M Turing-géppel $\}$

A $TIME(t(n))$ nyelvosztályba tehát azok az L nyelvek tartoznak, amelyekhez létezik $ct(n)$ időkorlátos M Turing-gép. A c konstans függhet L-től.

A definíció lényeges eleme, hogy n jelből álló w inputokon az M Turing-gép legfeljebb $ct(n)$ számú lépésben *mindig megáll*, tekintet nélkül arra, hogy $w \in \mathrm{L}$ igaz-e. Ennek következményeként a $TIME(t(n))$ nyelvosztály **rekurzív nyelvek**et tartalmaz.

13.1 példa:

$TIME(n) := \{ \ L \subseteq \Sigma^* ; \ L$ felismerhető $O(n)$ időkorlátos determinisztikus M Turing-géppel $\}$

A $TIME(n)$ nyelvosztályba a lineáris időben felismerhető nyelvek tartoznak. Algoritmikus időigény szempontjából ezek a nyelvek tekinthetők a legkönnyebbeknek. Szokásos még a $TIME(n^2)$ nyelvosztályt a négyzetes, a $TIME(n^3)$ nyelvosztályt pedig a köbös időben felismerhető nyelvek összességének nevezni.

13.2 definíció: ($\mathbf{P}$ nyelvosztály)

$P = \bigcup_{k \geq 1} TIME(n^k)$, a **polinomiális időben felismerhető nyelvek osztálya**

13.2 példa:

$L = \{0^n 1^n ; n \geq 1\} \in TIME(n) \subseteq P$.

Könnyen szerkeszthető olyan M determinisztikus Turing-gép, amely ezt a nyelvet lineáris időben ismeri fel.

Az időosztályokkal analóg módon kapjuk a tárosztályok definícióját:

13.3 definíció: ($SPACE(s(n))$ nyelvosztály)

$SPACE(s(n)) := \{$ $\mathrm{L} \subseteq \Sigma^*$; L felismerhető $O(s(n))$ tárkorlátos determinisztikus M Turing-géppel $\}$

A $SPACE(s(n))$ nyelvosztályba tehát azok az L nyelvek tartoznak, amelyekhez létezik $cs(n)$ tárkorlátos M Turing-gép. A c konstans függhet L-től.

13.3 példa:

$SPACE(\log n) := \{$ $\mathrm{L} \subseteq \Sigma^*$; L felismerhető $O(\log n)$ tárkorlátos determinisztikus M Turing-géppel $\}$

A $SPACE(\log n)$ nyelvosztályba a logaritmikus tárban felismerhető nyelvek tartoznak.

A nyelvosztályokhoz hasonló módon kaphatunk idő-, illetve tárkorlátokkal meghatározott függvényosztályokat. Csupán annyit kell tennünk, hogy a definíciókban nyelveket felismerő Turing-gépek helyett függvényeket kiszámoló Turing-gépeket szerepeltetünk.

13.4 definíció: ($FTIME(t(n))$ függvényosztály)

$FTIME(t(n)) :=$ { $f : \Sigma^* \to \Sigma^*$; f kiszámítható $O(t(n))$ időkorlátos determinisztikus M Turing-géppel }

13.5 definíció: ($FSPACE(s(n))$ függvényosztály)

$FSPACE(s(n)) :=$ { $f : \Sigma^* \to \Sigma^*$; f kiszámítható $O(s(n))$ tárkorlátos determinisztikus M Turing-géppel }

Igen fontos osztály a **P** nyelvosztállyal analóg **FP** függvényosztály, a polinomiális időben kiszámítható függvények osztálya:

13.6 definíció: (**FP** függvényosztály)

$\mathrm{FP} := \bigcup_{k \geq 1} FTIME(n^k)$, a **polinomiális időben kiszámítható függvények osztálya**

Tár-idő tétel

A következő állítás egy alapvető összefüggést rögzít a tár- illetve időkorlátokkal definiált osztályok között. Ha egy nyelv felismerésére (függvény kiszámítására) van tárkorlátos algoritmus, akkor van időkorlátos algoritmus is. Az adódó időkorlát a tárkorlát exponenciális függvényével becsülhető.

13.1 tétel: (Tár-idő tétel nyelvekre)

Ha $\mathrm{L} \in SPACE(s(n))$, akkor létezik olyan L-től függő c konstans, mellyel $\mathrm{L} \in TIME(c^{s(n)})$ teljesül.

A bizonyítás minden nehézség nélkül átvihető függvényekre. Egyetlen módosítást kell csak eszközölni: végtelen ciklus esetén, amikor a w input szóra az f függvény nincs értelmezve, legyen

$$f(w) = *$$

13.2 tétel: (Tár-idő tétel függvényekre)

Ha $f \in FSPACE(s(n))$, akkor létezik olyan f-től függő c konstans, mellyel $f \in FTIME(c^{s(n)})$ teljesül.

13.7 definíció: (komplementer nyelvosztály)

Legyen X egy Σ ábécé feletti nyelveket tartalmazó nyelvosztály ($X \subseteq 2^{\Sigma^*}$). A coX **komplementer nyelvosztály** az X-beli nyelvek komplementereiből áll:

$$\mathrm{co}X := \{ L \subseteq \Sigma^*;\ \Sigma^* \setminus L \in X \}$$

13.3 tétel:

Érvényes, hogy $TIME(t(n)) = \mathrm{co}TIME(t(n))$

Bizonyítás:

Legyen $\mathrm{L} \subseteq \Sigma^*$ nyelv a $TIME(t(n))$ nyelvosztály tetszőleges eleme. Megmutatjuk, hogy a $\Sigma^* \setminus \mathrm{L}$ nyelv is benne van ebben a nyelvosztályban.

Mivel $\mathrm{L} \in TIME(t(n))$, ezért létezik olyan $O(t(n))$ időkorlátos determinisztikus M Turing-gép, amely felismeri az L nyelvet. Ha megcseréljük az M gép elfogadó és elutasító állapotait, akkor a kapott M' Turing-gép éppen $\Sigma^* \setminus \mathrm{L}$ nyelvet fogja felismerni. Mivel az M' gép is $O(t(n))$ időkorlátos, ezért $\Sigma^* \setminus \mathrm{L} \in TIME(t(n))$.

□

13.8 definíció: (**PSPACE** nyelvosztály)

$\mathrm{PSPACE} := \bigcup_{k\geq 1} SPACE(n^k)$, a **polinomiális tárban felismerhető nyelvek osztálya**

13.9 definíció: (**EXPTIME** nyelvosztály)

$\mathrm{EXPTIME} := \bigcup_{k\geq 1} TIME(2^{n^k})$, az **exponenciális időben felismerhető nyelvek osztálya**

Az **EXPTIME** nyelvosztályra tekinthetünk úgy, mint a gyakorlatban előforduló nyelvek (eldöntési problémák) univerzumára. Ezt úgy értjük, hogy nemigen van olyan gyakorlati probléma, ami ezen nyelvosztályon kívül eső, még nehezebb nyelvhez vezetne.

A **P** osztályba tehát a polinomiális időben *megoldható* problémák tartoznak. Ezeket tekintjük többnyire jól kezelhető problémáknak. Néhány érv ennek alátámasztására:

- A gyakorlati tapasztalat azt mutatja, hogy egy probléma első polinomiális idejű megoldását gyakran követik hatékonyabb algoritmusok.
- Ha egy probléma polinomiális időben megoldható egy számítási modellben, akkor az egy másikban is polinomiális.
- A **P** osztálynak szép zártsági tulajdonságai vannak, mivel a polinomok zártak az összeadásra és a szorzásra nézve. Például továbbra is polinomiális marad az az algoritmus, amely polinomiális számú lépésén kívül konstansszor hív meg egy polinomiális szubrutint.

Az **időkorlátos** ill. **tárkorlátos** NTG definíciója alapján a $TIME$ ill. $SPACE$ nyelvosztályok mintájára definiálhatjuk a nemdeterminisztikus időkorlátokkal ill. tárkorlátokkal kijelölt nyelvosztályokat.

13.10 definíció: ($NTIME(t(n))$ nyelvosztály)

$NTIME(t(n)) := \{$ $\mathrm{L} \subseteq \Sigma^*$; L felismerhető $O(t(n))$ időkorlátos NTG-vel $\}$

13.11 definíció: ($NSPACE(s(n))$ nyelvosztály)

$NSPACE(s(n)) := \{$ $\mathrm{L} \subseteq \Sigma^*$; L felismerhető $O(s(n))$ tárkorlátos NTG-vel $\}$

Az NTG-k segítségével megadható a számításelmélet egyik legérdekesebb nyelvosztályának, az **NP**-nek a definíciója:

13.12 definíció: (**NP** nyelvosztály)

$\mathrm{NP} = \bigcup_{k\geq 1} NTIME(n^k)$, az **NTG-vel polinomiális időben felismerhető nyelvek osztálya**

Az $\mathbf{NP}$ nyelvosztály ugyanúgy épül fel az $NTIME(n^k)$ nyelvosztályokból, mint a $\mathbf{P}$ nyelvosztály a $TIME(n^k)$ nyelvosztályokból. A definíciók közötti nyilvánvaló formai hasonlóság alapján az $\mathbf{NP}$ nyelvosztályt a $\mathbf{P}$ nyelvosztály nemdeterminisztikus megfelelőjének tekinthetjük.

Az $\mathbf{NP}$ nyelvosztály a nemdeterminisztikus Turing-gépekkel polinomális időben felismerhető nyelvekből áll. A determinisztikus Turing-gépek felfoghatók NTG-nek is, sőt egy $t(n)$ időkorlátos DTG tekinthető $t(n)$ időkorlátos NTG-nek is. Így azonnal adódik, hogy

$$TIME(n^k) \subseteq NTIME(n^k),$$

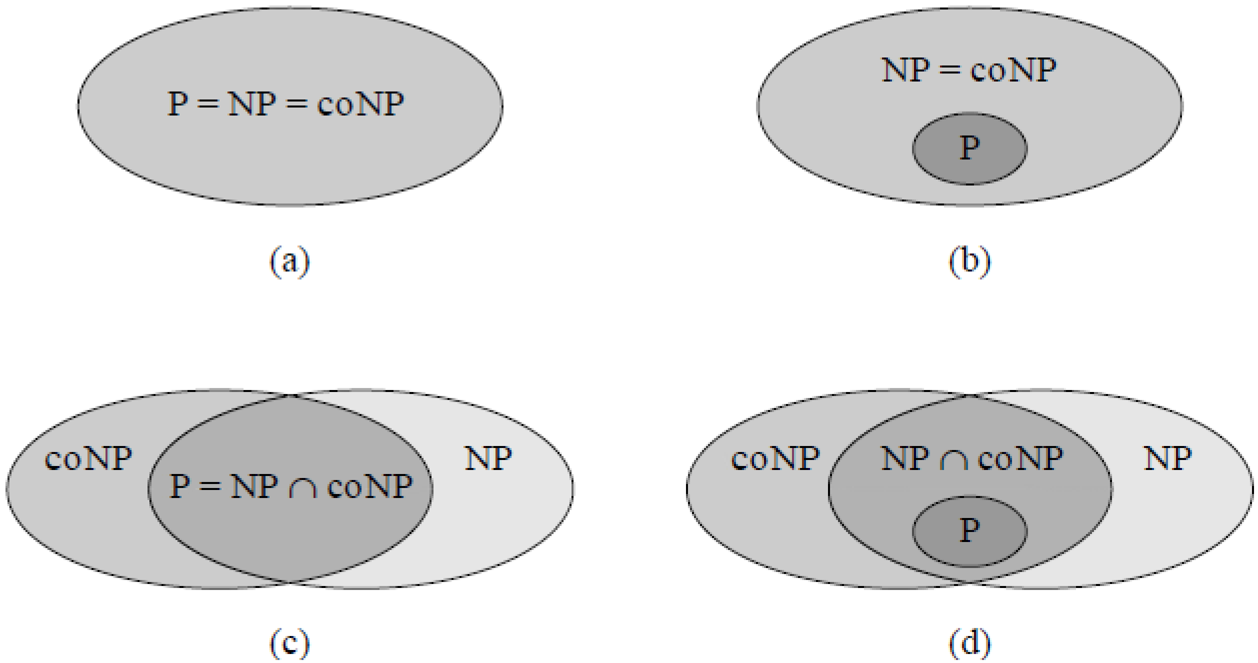
amiből kapjuk, hogy:

13.4 tétel:

Érvényes, hogy $\mathbf{P} \subseteq \mathbf{NP}$

Megjegyzés: A $\mathbf{P} = \mathbf{NP}$ kérdés a számításelmélet egyik legfontosabb megoldatlan problémája.

A **P** és **NP** osztályok közötti négy lehetséges kapcsolat:



Az **NP** osztályba olyan problémák tartoznak, amelyek polinomiális időben *ellenőrizhetők*. Ez azt jelenti, hogy létezik olyan polinomiális idejű algoritmus, amely **ellenőrzi, hogy egy adott szó benne van-e az adott nyelvben**.

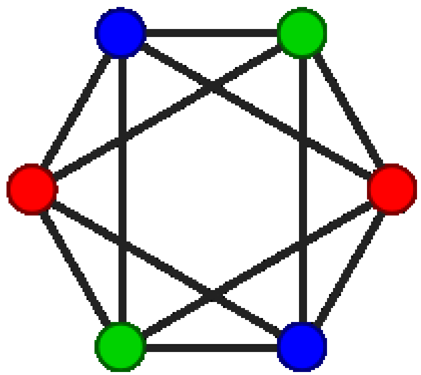
Néhány NP-beli probléma

Az n csúcsot tartalmazó gráf szomszédsági mátrixa tárolható n^2 hosszúságú bitsorozatként. Egy ilyen kódolás eredményeként a gráfokat $\{0,1\}^*$-beli szavak reprezentálják.

Háromszín probléma

Rendeljünk színeket a gráf csúcsaihoz úgy, hogy az éllel összekötött csúcsok eltérő színeket kapjanak.

Legyen 3-SZÍN a 3 színnel színezhető gráfok kódjaiból álló nyelv.



13.5 tétel:

3-SZÍN $\in$ **NP**

Bizonyítás:

Elegendő megmutatni, hogy a három színnel való színezhetőségnek van rövid és hatékonyan ellenőrizhető tanúja. Egy n csúcspontú, 3 színnel színezhető G gráf jó színezése alkalmas tanú lesz. Egy ilyen színezés leírható $2n$ bittel (pl. legyen **01** = piros, **10** = kék, **11** = zöld). Ez a feladat polinomiális időben megoldható.

A **tanú-tétel** alkalmazásakor a G gráf felel meg az x inputnak, a színezés pedig az y tanú. Az A algoritmusnak csak ellenőriznie kell, hogy a javasolt színezés megfelelő-e:

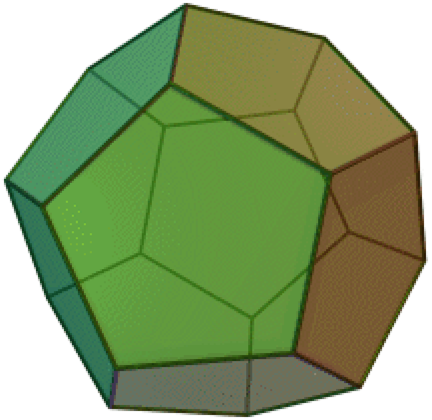
- ha G benne van a 3-SZÍN nyelvben, akkor $A(G, \textit{színezés}) = 1$
- ha G nincs benne a 3-SZÍN nyelvben, akkor $A(G, \textit{színezés}) = 0$

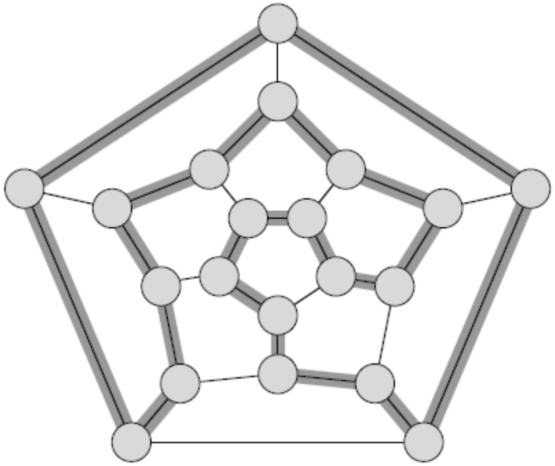
Mivel létezik ilyen A algoritmus, ezért a 3-SZÍN **NP**-beli nyelv. □

Hamilton-kör probléma

Egy gráf **Hamilton-köre** olyan kör, amely a gráf minden csúcsán pontosan egyszer halad át.

Pl. a dodekaéder gráf-reprezentációjának van Hamilton-köre.





Legyen Hamilton-kör probléma formális nyelvként az alábbi módon definiálva:

$$\mathrm{HAM} = \{ \langle G \rangle ;\ G \text{ tartalmaz Hamilton-kört} \}$$

13.6 tétel:

$\mathrm{HAM} \in \mathbf{NP}$

Bizonyítás:

Elegendő megmutatni, hogy a Hamilton-kör problémának van rövid és hatékonyan ellenőrizhető tanúja. Egy n csúcspontból álló G gráf Hamilton-köre alkalmas tanú lesz, ez leírható a csúcsoknak a kör mentén való bejárási sorrendjével. Elegendő megvizsgálni, hogy a megadott csúcssorozat permutációja-e a G gráf csúcsainak, és hogy az egymás után következő csúcsok (valamint az első és az utolsó) szomszédosak-e. Ez az eljárás $O(n^2)$ lépésben végrehajtható.

Létezik tehát A polinomiális idejű algoritmus, amely polinomiális időben bizonyítja a HAM nyelvet, ezért ez $\mathbf{NP}$-beli nyelv.

$\square$

Az számelméletben és az algebrában is igen sok probléma tartozik az $\mathbf{NP}$ osztályba. Minden természetes számra tekinthetünk úgy, mint egy $\{0,1\}^*$-beli szóra, elegendő a számot kettes számrendszerben felírni.

Összetett szám probléma

Egy természetes szám akkor összetett szám, ha létezik valódi osztója. Legyen ÖSSZ a kettes számrendszerben felírt összetett természetes számokat tartalmazó nyelv.

13.7 tétel:

$\text{ÖSSZ} \in \mathbf{NP}$

Bizonyítás:

Elegendő megmutatni, hogy az összetett szám problémának van rövid és hatékonyan ellenőrizhető tanúja. Ha m összetett szám, akkor ezt egy d valódi osztója tanúsítja. A d ismeretében az m szám összetettsége az $m:d$ osztás elvégzésével igazolható.

Az alapiskolából ismert osztási algoritmussal az $m : d$ osztás $O(n^2)$ bit-művelettel elvégezhető.

Létezik tehát A polinomiális idejű algoritmus, amely polinomiális időben bizonyítja az $\mathrm{ÖSSZ}$ nyelvet, ezért ez $\mathbf{NP}$-beli nyelv.

$\square$

Az NPC osztály

Az előadás hátralévő részében az **NPC** osztállyal fogunk foglalkozni, amelybe az **NP**-teljes problémák tartoznak. Ezekre a problémák megoldására a mai napig nem sikerült polinomiális idejű algoritmust adni.

Az **NP**-teljes problémák egyik különösen kellemetlen tulajdonsága, hogy számosan közülük ránézésre igen hasonlóak olyan problémákhoz, melyekre létezik polinomiális algoritmus. A következő problémapárok mindegyikében az egyik probléma polinomiális időben megoldható, míg a másik **NP**-teljes, annak ellenére, hogy a köztük lévő különbség csekélynek tűnik.

Euler és Hamilton-körök irányított gráfban:

Legyen G összefüggő, irányított gráf.

A G gráf **Euler-köre** olyan kör, amely a gráf minden élén pontosan egyszer halad át, egy csúcson azonban többször is átmehet.

A G gráf **Hamilton-köre** olyan kör, amely a gráf minden csúcsán pontosan egyszer halad át.

Annak eldöntése, hogy egy adott gráf tartalmaz-e Euler-kört, elvégezhető polinomiális időben, sőt polinomiális időben meg is található az Euler-kör (feltéve hogy létezik).

Annak eldöntése, hogy egy adott gráf tartalmaz-e Hamilton-kört, $\mathbf{NP}$-teljes.

2-CNF és 3-CNF kielégíthetőség:

Egy **Boole-formula** 0 vagy 1 értékű változókból, egy- vagy kétváltozós Boole-függvényekből és zárójelekből áll.

Egy Boole-formula *kielégíthető*, ha létezik a változóknak olyan behelyettesítése, amelyre a formula értéke 1.

Egy Boole-formula k-konjunktív normálformájú (k-CNF), ha olyan zárójeles tagok ÉS művelettel való összekapcsolásából áll, melyek pontosan k darab, VAGY művelettel összekapcsolt változót vagy negált változót tartalmaznak.

Pl. 2-CNF formula: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$

Annak eldöntése, hogy egy 2-CNF formula kielégíthető-e, létezik polinomiális idejű algoritmus.

A 3-CNF formulák kielégíthetőségének problémája viszont **NP**-teljes.

Hogyan mutatjuk meg, hogy egy probléma NP-teljes?

Az **NP**-teljesség bizonyításakor nem azt mutatjuk meg egy problémáról, hogy mennyire könnyű, hanem azt, hogy mennyire nehéz. Vagyis nem azt próbáljuk meg bebizonyítani, hogy megoldására létezik hatékony algoritmus, hanem épp ellenkezőleg: azt, hogy *valószínűleg* egyáltalán nem létezik hatékony algoritmus.

Az **NP**-teljesség fogalmát kizárólag **döntési problémák**ra alkalmazzuk, melyeknél az output 0 vagy 1.

A bizonyításkor gyakran alkalmazunk **visszavezetést**, amely során egy problémát egy másik problémára vezetünk vissza.

Egy P probléma visszavezethető a Q problémára, ha P bármely x esete könnyen átfogalmazható a Q egy olyan y esetévé, melynek megoldásából következik x megoldása. Például az elsőfokú egyismeretlenes egyenlet megoldásának problémája visszavezethető a másodfokú egyenlet megoldására. Az $ax+b=0$ egyenlet ugyanis a $0y^2+ay+b=0$ egyenletté alakítható, s ennek megoldása kielégíti az $ax+b=0$-t is.

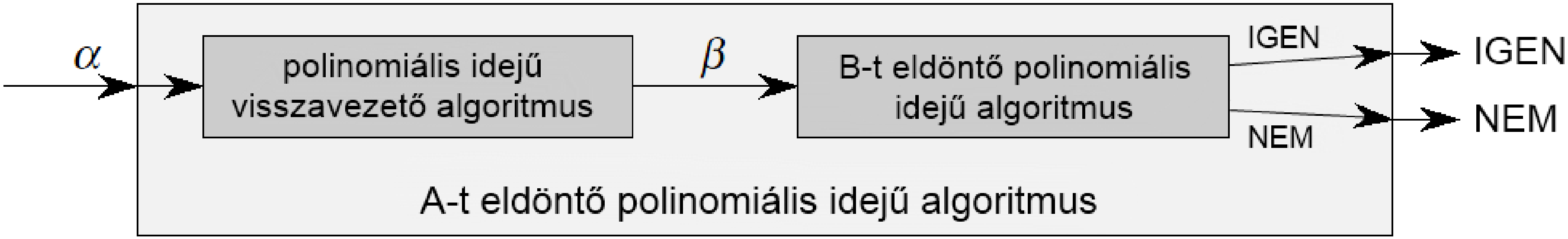
Ez azt is jelenti, hogy ha a P probléma visszavezethető a Q problémára, akkor P-t bizonyos értelemben nem nehezebb megoldani, mint Q-t.

Általánosan fogalmazva tekintsünk egy $\mathbf{A}$ döntési problémát, amelyet szeretnénk polinomiális időben megoldani. Tegyük fel, hogy van egy $\mathbf{B}$ döntési problémánk, melyről már tudjuk, hogyan lehet polinomiális időben megoldani.
Végül tegyük fel, hogy rendelkezésünkre áll egy eljárás, melynek segítségével az $\mathbf{A}$ probléma egy α esetét átalakíthatjuk a $\mathbf{B}$ probléma egy β esetévé úgy, hogy teljesüljenek az alábbiak:

1) az átalakítás polinomiális ideig tart,
2) a válaszok azonosak, azaz a válasz az α esetre akkor és csakis akkor IGEN, ha a válasz a β esetre IGEN.

Az ilyen eljárást polinomiális idejű **visszavezető algoritmus**nak nevezzük.

A visszavezető algoritmus az alábbi módon szolgáltat polinomiális idejű megoldást az $\mathbf{A}$ döntési problémára:



1) az $\mathbf{A}$ probléma adott α esetét a visszavezető algoritmussal átalakítjuk a $\mathbf{B}$ probléma egy β esetévé,
2) lefuttatjuk a megoldjuk a $\mathbf{B}$ problémát megoldó algoritmust a β esetre,
3) a β esetre kapott választ adjuk meg az $\mathbf{A}$ probléma α esetéhez tartozó válaszként.

Amennyiben mindhárom lépés megvalósítható polinomiális időben, akkor ezek együttesen is megvalósíthatók polinomiális időben, vagyis az $\mathbf{A}$ probléma polinomiális időben eldönthető.

Az **NP**-teljesség bizonyítására a polinomiális idejű visszavezetés módszerét fordítva fogjuk alkalmazni:

Tekintsünk egy **B** döntési problémát, melyről meg akarjuk mutatni, hogy nem létezik rá polinomiális idejű algoritmus. Tegyük fel, hogy létezik egy **A** döntési probléma, melyről tudjuk, hogy nem oldható meg polinomiális időben. Tegyük fel továbbá, hogy rendelkezésünkre áll egy polinomiális idejű visszavezető algoritmus, amely az A probléma eseteit a **B** probléma eseteivé alakítja át.

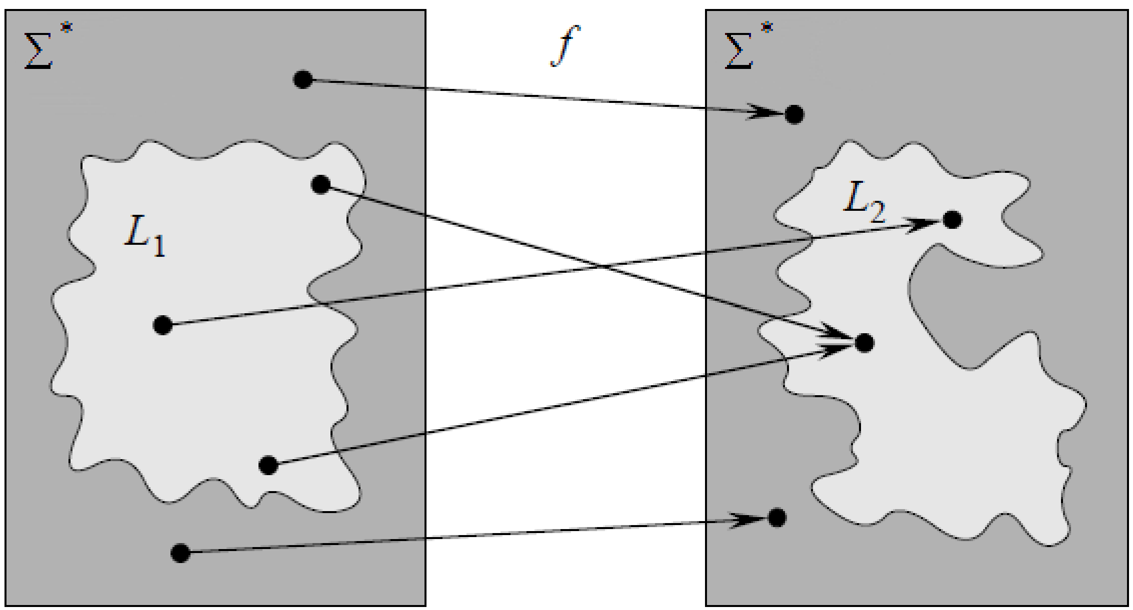
Ekkor igazolható, hogy a **B** probléma nem oldható meg polinomiális idejű algoritmussal.

13.13 definíció: (polinomiális visszavezethetőség)

Az $\mathrm{L}_1 \subseteq \Sigma^*$ nyelv **polinomiálisan visszavezethető** az $\mathrm{L}_2 \subseteq \Sigma^*$ nyelvre, ha létezik $f : \Sigma^* \to \Sigma^*$ polinomiális időben kiszámítható függvény, melyre minden $w \in \Sigma^*$ szó esetén

$w \in \mathrm{L}_1$ pontosan akkor teljesül, ha $f(w) \in \mathrm{L}_2$.

Jelölés: $\mathrm{L}_1 \prec \mathrm{L}_2$



Az f függvényt **visszavezető függvény**nek v. **Karp-redukció**nak nevezzük, az f-et kiszámító polinomiális idejű algoritmust pedig **visszavezető algoritmus**nak.

A visszavezetési módszer jól használható nyelvek $\mathbf{P}$ nyelvosztályba való tartozására.

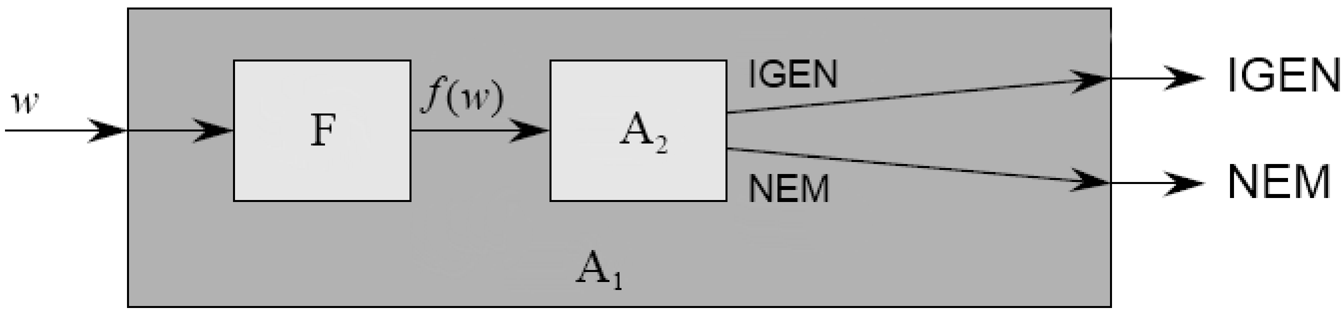
13.8 tétel:

Legyen $L_1, L_2 \subseteq \Sigma^*$. Ha $L_1 \prec L_2$ teljesül és $L_2 \in P$, akkor $L_1 \in P$.

Bizonyítás:

Legyen A_2 polinomiális algoritmus, amely eldönti az L_2 nyelvet, és legyen F az L_1 nyelvet L_2-re visszavezető f függvényt kiszámító polinomiális algoritmus. Megadunk egy olyan polinomiális idejű A_1 algoritmust, amely eldönti az L_1 nyelvet.

Az A_1 algoritmus konstrukcióját az alábbi ábra szemlélteti:



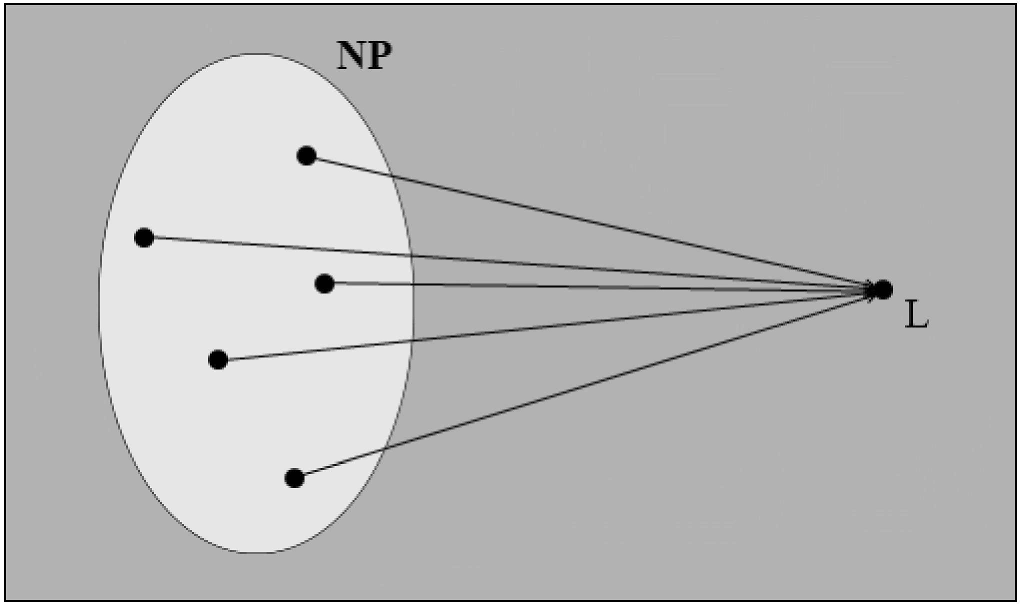
A $w \in \Sigma^*$ inputból az A_1 algoritmus először F segítségével előállítja $f(w)$-t, majd az A_2 algoritmus felhasználásával eldönti, hogy $f(w)$ benne van-e az L_2 nyelvben, és az erre kapott választ adja ki outputként.

Az A_1 algoritmus polinomiális, mivel az F és A_2 algoritmusok is azok. Létezik tehát polinomiális idejű algoritmus amely eldönti az L_1 nyelvet, ezért $L_1 \in P$.

$\square$

13.14 definíció: (**NP**-nehéz nyelv)

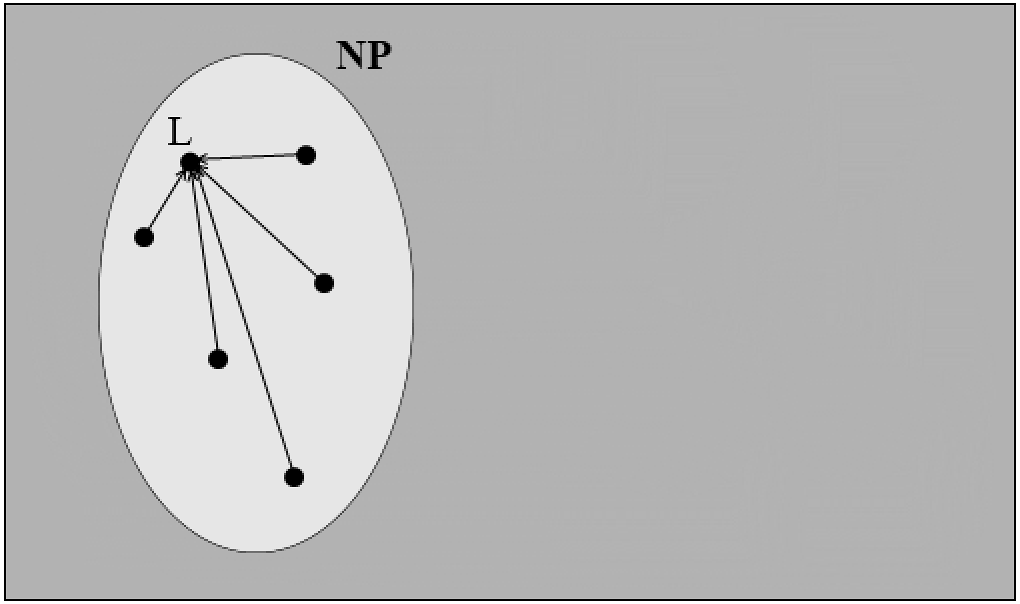
Az $L \subseteq \Sigma^*$ nyelv **NP-nehéz**, ha minden $L' \in NP$ nyelv polinomiálisan visszavezethető az L nyelvre, azaz létezik $L' \prec L$ Karp-redukció.



13.15 definíció: (**NP**-teljes nyelv)

Az $L \subseteq \Sigma^*$ nyelv **NP-teljes**, ha

1) $L \in NP$,
2) minden $L' \in NP$ nyelv polinomiálisan visszavezethető az L nyelvre, azaz létezik $L' \prec L$ Karp-redukció.



Amint azt a következő tétel mutatja, az **NP**-teljességnek döntő szerepe van a **P** és **NP** nyelvosztályok viszonyának eldöntésében.

13.9 tétel:

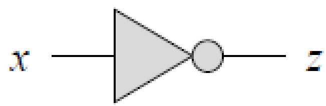
Ha létezik polinomiális időben megoldható **NP**-teljes probléma, akkor **P** = **NP**. Más szóval: ha létezik az **NP** nyelvosztályban polinomiális időben nem megoldható probléma, akkor egyetlen **NP**-teljes probléma sem polinomiális.

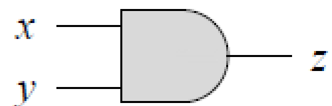
Érthető tehát, hogy (**P** = **NP**?) kérdéssel kapcsolatos kutatások középpontjában az **NP**-teljes problémák állnak. Amennyiben sikerülne megtalálni egy **NP**-teljes probléma polinomiális idejű megoldását, akkor a kérdésre választ kapnánk (**P** = **NP**). Mivel mindez idáig nem történt meg, és nem is nagyon várható ennek bekövetkezése, így egy probléma **NP**-teljessége jó ok annak feltételezésére, hogy a probléma algoritmussal nehezen kezelhető.

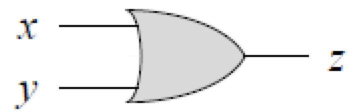
A Boole-hálózatok kielégíthetőségének problémája

A Boole-hálózatok huzallal összekötött logikai áramkörökből, ún. **logikai kapuk**ból állnak, amelyek egyszerű Boole-függvényeket számítanak ki.

A három alapvető kapu: NEM, ÉS, VAGY





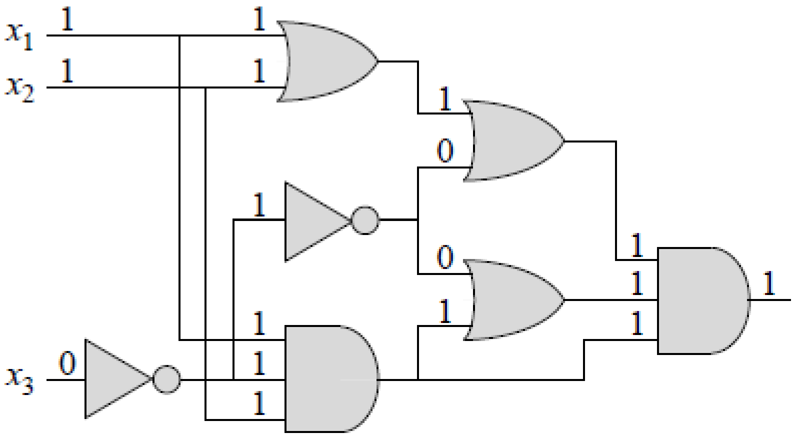


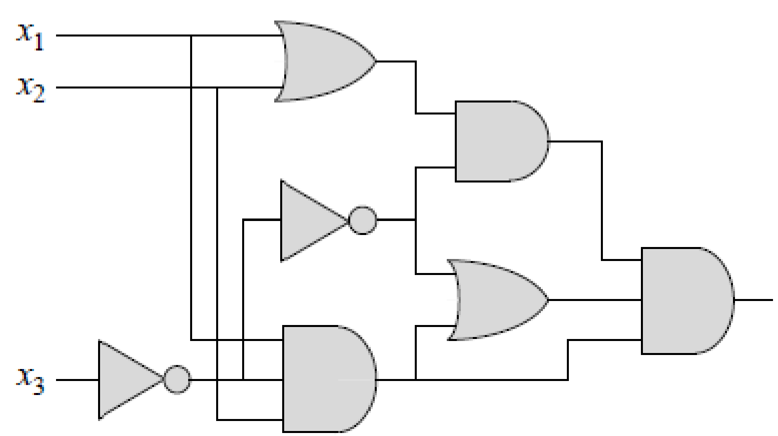
x	$\neg x$
0	1
1	0

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Az ÉS és VAGY kapukat általánosíthatjuk úgy, kettőnél több bemenetet is elfogadjanak, az ilyen ÉS kapu kimenete pontosan akkor 1, ha minden bemenete 1; az ilyen VAGY kapu kimenete pontosan akkor 1, ha legalább egy bemenete 1.





A Boole-hálózat **behelyettesítése** a Boole-változók egy, a bemenetek számával megegyező elemszámú sorozata. Egy egy kimenetű Boole-hálózat **kielégíthető**, ha van **kielégítő behelyettesítése**, azaz olyan behelyettesítése, melyre a kimenet 1.

C-SAT: (Circuit Satisfiability – a Boole-hálózatok kielégíthetősége)

Adott – NEM, ÉS és VAGY kapukból álló **– Boole-hálózat kielégíthető-e?**

A probléma formalizálásához meg kell adnunk a Boole-hálózatot valamilyen kódolásban. A Boole-hálózatok reprezentálhatók irányított gráfokkal: legyenek a kapuk a gráf csúcsai, a huzaloknak pedig feleljenek meg irányított élek úgy, hogy az u csúcsból a v csúcsba mutasson irányított él, ha a huzal kimenete az u kapunak és bemenete a v kapunak.

Most már definiálhatjuk a problémát formális nyelvként:

$$\text{C-SAT} = \{\langle \text{C} \rangle;\ \text{C kielégíthető Boole-hálózat}\}$$

A C-SAT probléma jelentős szerepet játszik a hardver optimalizációban. Ha egy hálózat minden bemenetre pl. 0-t ad, akkor helyettesíthető egy konstans 0-t adó kapuval. Egy polinomiális algoritmusnak tehát komoly gyakorlati haszna lenne.

13.10 tétel:

Ha L olyan nyelv, melyhez létezik olyan $\mathrm{L}' \in \mathbf{NPC}$ nyelv, hogy $\mathrm{L}' \prec \mathrm{L}$ akkor az L nyelv $\mathbf{NP}$-nehéz. Ha $\mathrm{L} \in \mathbf{NP}$ is teljesül, akkor $\mathrm{L} \in \mathbf{NPC}$.

A **13.10 tétel** szerint az alábbi módon lehet belátni egy L nyelv $\mathbf{NP}$-teljességét:

1) Belátjuk, hogy $\mathrm{L} \in \mathbf{NP}$.

2) Választunk egy ismert $\mathrm{L}' \in \mathbf{NPC}$ nyelvet.

3) Megadunk egy F algoritmust, amely kiszámít egy olyan f függvényt, amely az L' nyelv szavaihoz az L nyelv szavait rendeli.

4) Bebizonyítjuk, hogy tetszőleges $w \in \Sigma^*$ esetén $w \in \mathrm{L}'$ akkor és csakis akkor teljesül, amikor $f(w) \in \mathrm{L}$.

5) Igazoljuk, hogy az F algoritmus polinomiális.

NP-teljes problémák

SAT: (Satisfiability Problem – a Boole-formulák kielégíthetősége)

Adott Boole-formula kielégíthető-e?

A Boole-formulák az alábbi elemekből épülnek fel:

- Boole-változók ($x_1, x_2, x_3, \ldots$),
- Boole-operátorok ($\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow, \ldots$),
- zárójelek.

Azt mondjuk, hogy egy φ Boole-formula **kielégíthető**, ha a benne szereplő változók értékeit meg tudjuk adni úgy, hogy φ értéke 1 legyen.

A probléma formális nyelvként megadva:

$$\mathrm{SAT} = \{ \langle \varphi \rangle ;\ \varphi \text{ kielégíthető Boole-formula} \}$$

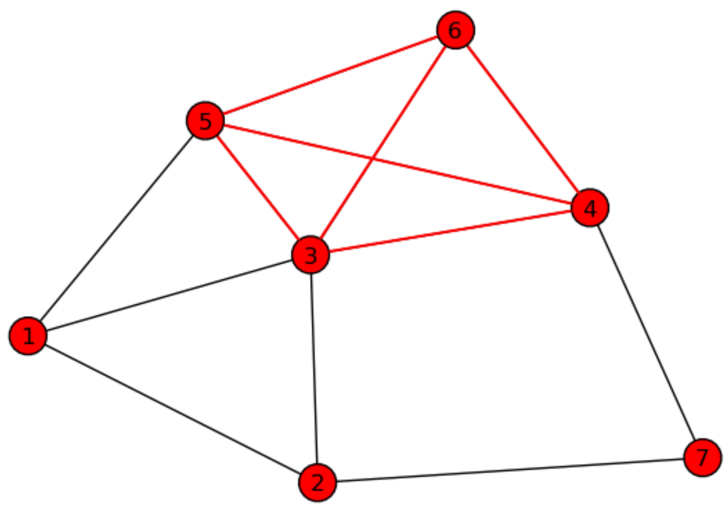
3-SAT: (3-CNF-beli Boole-formulák kielégíthetősége)

Adott 3-CNF-beli Boole-formula kielégíthető-e?

KLIKK: (Klikk-probléma)

Mekkora a legnagyobb klikk a gráfban?

A **klikk** egy gráf olyan részgráfja, amelyben bármely két csúcs között van él (más szóval olyan részgráf, amely teljes gráf).



Ez egy optimalizálási probléma, alakítsuk át döntésivé: Létezik-e a G gráfban k csúcsot tartalmazó klikk?

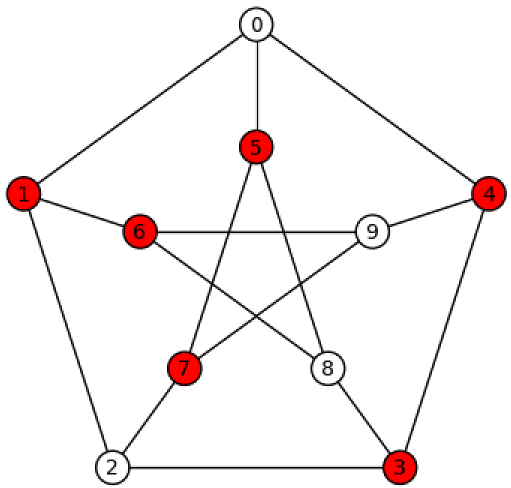
A probléma formális nyelvként megadva:

$$\mathrm{KLIKK} = \{\langle \mathrm{G}, k \rangle;\ \text{ a G gráfban van } k \text{ csúcsot tartalmazó klikk } \}$$

LEFEDÉS: (Minimális lefedő csúcshalmaz probléma)

Adjuk meg a gráf minimális lefedő csúcshalmazát!

Egy $G=(V,E)$ gráf **lefedő csúcshalmaz**ának nevezzük a $V' \subseteq V$ csúcs-halmazt, ha minden $(u,v) \in E$ él esetén $u \in V'$ vagy $v \in V'$. A G gráf egy csúcshalmaza akkor lefedő, ha minden élt legalább egy eleme lefed.



Ez egy optimalizálási probléma, alakítsuk át döntésivé: Létezik-e a G gráfnak k elemű lefedő csúcshalmaza?

A probléma formális nyelvként megadva:

$$\text{LEFEDÉS} = \{\langle G, k\rangle;\ \text{a } G \text{ gráfnak van } k \text{ méretű lefedő csúcshalmaza}\}$$

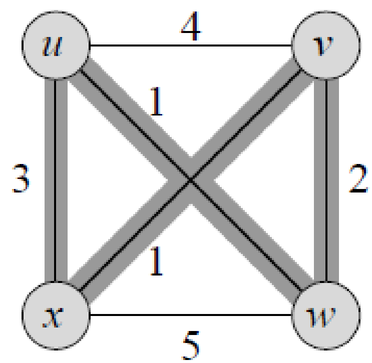
HAM: (Hamilton-kör probléma)

Van-e a gráfban Hamilton-kör?

TSP: (Traveling Salesman Problem – Utazóügynök probléma)

Az ügynöknek n várost kell meglátogatnia tetszőleges sorrendben, de minél kisebb utazási költség mellett. Az egyes városok közötti utazási költségek ismertek.

Adott egy n csúcspontú teljes gráf, melynek éleihez egész számokat rendelünk. A feladat olyan Hamilton-kör megtalálása, melynek összköltsége minimális!



Ez egy optimalizálási probléma, alakítsuk át döntésivé: Létezik-e a G gráfban k legfeljebb költségű Hamilton-kör?

A probléma formális nyelvként megadva:

$\mathrm{TSP} = \{ \langle G, c, k \rangle ;\ c : V \times V \to Z$, a $G = (V, E)$ teljes gráfnak van legfeljebb k költségű Hamilton-köre $\}$

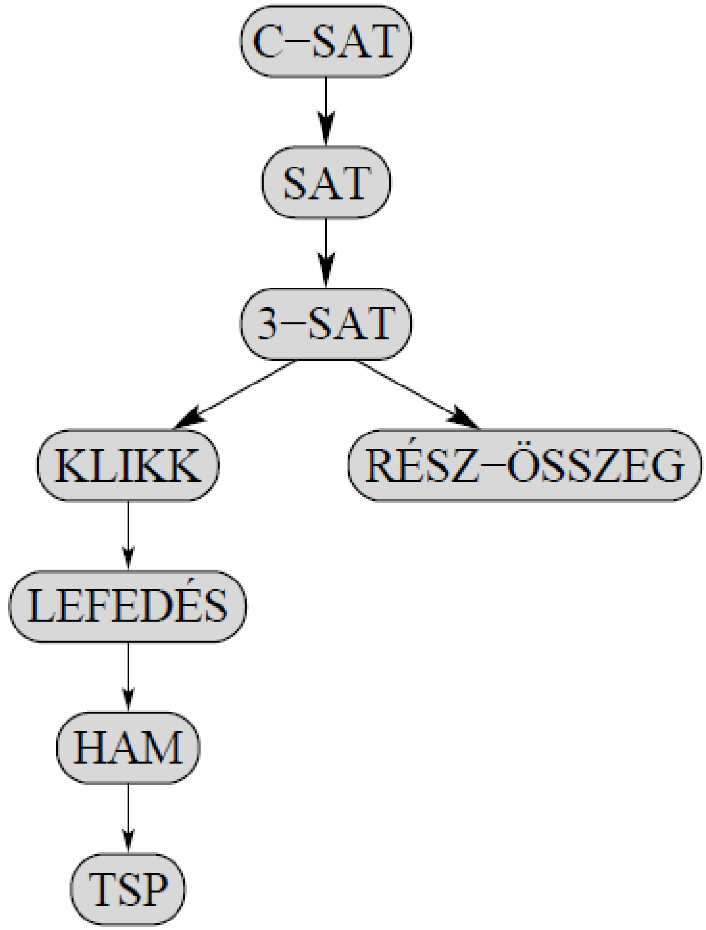
RÉSZ-ÖSSZEG: (Részletösszeg probléma)

Legyen adott egy $\mathrm{S} \subseteq \mathrm{N}$ véges halmaz és egy $t \in \mathrm{N}$ szám.
Létezik-e olyan $\mathrm{S}' \subseteq \mathrm{S}$ halmaz, melyben az elemek összege t?

Pl. ha $\mathrm{S} = \{1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993\}$ és $t = 138457$, akkor az $\mathrm{S}' = \{1, 2, 7, 98, 343, 686, 2409, 17206, 117705\}$ részhalmaz ilyen.

A probléma formális nyelvként megadva:

$$\text{RÉSZ-ÖSSZEG} = \{\langle \mathrm{S}, t\rangle;\ \text{létezik } \mathrm{S}' \subseteq \mathrm{S} :\ \sum_{s \in \mathrm{S}'} s = t\ \}$$


C–SAT
SAT
3–SAT
KLIKK
RÉSZ–ÖSSZEG
LEFEDÉS
HAM
TSP