ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

4. Előadás

Előadó: Hajnal Péter
Jegyzetelő: Hajnal Péter

2013. február 28.

Ebben az előadásban az eddig megismert bonyolultsági osztályok hierarchiájában helyezünk el ismert matematikai problémákat.

1. $\mathcal{D}$-n kívül

Enlítettük, hogy a bonyolultságelmélet témája a $\mathcal{D}$-beli nyelvek vizsgálata, összehasonlításuk, bonyolultság szerint struktúrálásuk. A $\mathcal{D}$-n kívüli nyelvek is igen aktívan vizsgáltak. Kutatásuk módszerei és motivációja inkább a matematikai logikához köthető.

Egy új probléma esetén az első kérdés (döntési problémák esetén), hogy $\mathcal{D}$-hez tartozik-e. Nagyon sok matematikailag fontos, központi kérdés esetén kiderült, hogy nem $\mathcal{D}$-beli kérdésről van szó. Egy ilyen matematikai tétel jelentése az, hogy amíg a Church-tézis jól leírja a kiszámíthatóság fogalmát, addíg tudjuk, hogy a probléma általánosságban számítógéppel NEM kezelhető. Természetesen speciális inputokra, különböző feltételek mellett elképzelhető a kiszámíthatóság. Ilyen problémáknál a matematikai kutatásoknak ebbe az irányba kell tartaniuk.

Most néhány ilyen problémát sorolunk fel. Az első Hilbert X. problémája. Ennek történeti jelentősége van. Ez nagyban hozzájhárult a kiszámíthatóság fogalmának tisztázásához, ami elvezetett a Turing-gép definíciójához.

Példa (Hilbert X. Problémája). Legyen

$$DIOPHANTOSZ = \{\lceil p(x) \rceil : p \in \mathbb{Z}[x_1, x_2, \ldots, x_n],\ p\text{-nek van egész gyöke}\}.$$

Hilbert problémájának modern értelmezése, hogy $DIOPHANTOSZ$ nyelv $\mathcal{D}$-hez tartozik-e. (A probléma kitűzésének idejében $\mathcal{D}$ fogalma még nem született meg.) A klasszikus nyelven a probléma az, hogy van-e olyan algoritmus, ami egy adott egész együtthatós polinomról eldönti, hogy van-e egész gyöke.

Két lehetőség volt. Vagy valaki ad egy algoritmust, ami megoldja Hilbert problémáját (azaz DIOPHANTOSZ$\in \mathcal{D}$), a metematikusok közössége pedig megérti, ellenőrzi és elfogadja az algoritmust. A másik lehetőség: nincs ilyen algoritmus. Ebben az esetben ezt bizonyítani kell. Ez nem megy $\mathcal{D}$ definíciója nélkül. Kiderült, hogy a második lehetőség az igazság.

Hilbert X. problémájának megoldásának története: 1900 Hilbert előadja a problémát, 1935 Church megfogalmazza a Church-tézist, 1936 Turing bevezeti a Turing-gép fogalmát, 1950-es és 60-as évek a diophantikus halmazok bevezetése és vizsgálata Davies és Robinson vezetésével, 1970 Matijaszevics megteszi az utolsó (legnehezebb) lépéseket, bebizonyítja, hogy DIOPHANTOSZ nem tartozik $\mathcal{D}$-hez.

Természetesen DIOPHANTOSZ$\in \mathcal{S}$ (miért?).

Egy változó illetve lineáris eset könnyen megoldható. A kvadratikus kétváltozós eset is megoldható, de már komoly számelméleti vzisgálatok szükségesek.

Példa (Turing megállási problémája). A fenti matematikai motivációjú problema után megadunk egy eldönthetetlen nyelvet, ami a Turing-gépek definiációjával kapcsolatos. Be is bizonyítjuk eldönthetetlenségét. A példa Turing nevéhez fűződik, az első kiszámíthatatlansági eredmény.

Definíció.

$$\text{MEGÁLLÁS} = \{\ulcorner T, \omega \urcorner : T \text{ leáll } \omega\text{-n, azaz STOP vagyis ELVET/ELFOGAD állapotba kerül}\}.$$

1. Tétel (Turing-tétel). *(i)* $MEGÁLLÁS \in \mathcal{S}$,

(ii) $MEGÁLLÁS \notin \mathcal{D}$.

Bizonyítás. A tétel első része következik az univerzális Turing-gép leírásából. A szimuláló gép leállását úgy kell módosítani, hogy ne a leálló állapotnak megfelelő állapotba jusson, hanem a leállás tényét bejelentő ELFOGAD állapotba kerüljön.

A második állítás bizonyítása indirekten történik, azaz tegyük fel, hogy létezik I Turing-gép, amely eldönti a MEGÁLLÁS nyelvet. Továbbiakban az indirekt feltevés I gépére alapítva egy kissé módosított gépet írunk le.

Definíció. Legyen $\widetilde{I}$ egy Turing-gép, amely eldönti, hogy i input kódol-e Turing-gépet. Amennyiben nem a gép ELVET állapotba kerül. Amennyiben $i = \ulcorner T \urcorner$ az I MEGÁLLÁS nyelvet eldöntő Turing-gépet futtatja (i, i)-n. Ha a futtatás ELFOGAD állapottal ér véget, akkor jobbra-balra lépegető végtelen ciklusba megy át, ha ELVET állapottal ér véget, akkor STOP állapottal leáll.

Kérdés: Mit csinál $\widetilde{I}$ gép $\ulcorner\widetilde{I}\urcorner$-n? Azaz mi történik, ha a saját kódján futtatjuk az $\widetilde{I}$ gépet?

A definíció alapján „kibontja magát", és a MEGÁLLÁS nyelvet eldöntő géppel eldönti, hogy a megadott inputon (ami esetünkben saját kódján) hogyan dolgozik. Ha I ELFOGAD állapotba kerül (a $\widetilde{I}$ gép $\ulcorner\widetilde{I}\urcorner$-n megáll), akkor nem áll le (mármint $\widetilde{I}$ az $\ulcorner\widetilde{I}\urcorner$-n), és ha I ELVET állapotba kerül (a $\widetilde{I}$ gép $\ulcorner\widetilde{I}\urcorner$-n nem áll meg), akkor leáll. Mindenféleképpen ellentmondásra jutunk. ■

A bizonyítás lényege hasonlít Cantor bizonyítására, hogy $[0, 1]$ nem felsorolható/megszámlálhatóan végtelen halmaz (átlós módszer), csak ebben az esetben számok és sorszámok helyett gépek, illetve inputok kódjai szerepelnek.

Példa (Szóprobléma). SZÓPROBLÉMA inputja tartalmaz egy G csoportot. G-re multiplikatív írásmódot használva hívatkozunk. Mielőtt leírnánk a teljes problémát tisztáznunk kell, hogyan kódolhatunk csoportokat?

Egy lehetséges megoldást ad a kombinatorikus csoportelmélet. Legyen G egy csoport egy B generátorhalmazzal. Ekkor B elemeiből kifejezéseket épithetünk fel, amik a csoport egy-egy elemét írják le. Ha $B = \{a, b, c\}$, akkor $abbaca^{-1}ba^{-1}$ egy ilyen kifejezés. 1, az előző betűkészletből felírt üres szorzat is egy kifejezés, ami a csoport egységelemét írja le. Tehát a kifejezéseink, szakzsargonnal *szavaink*, B elemeiből és B elemeinek inverzéből szorzásokkal felépített kifejezések. Persze különböző szavak írhatják le ugyazt az elemet. A csoportszámtan garantálja, hogy $aa^{-1}b$ és b ugyanazt az elemet írja le.

Egy szó elemi egyszerűsítése az xx^{-1}, illetve $x^{-1}x$ egymásutáni két karakter kihúzása. Ha egy $w_1, w_2, w_3, \ldots, w_n$ szósorozatban bármely két egymásutáni szó közül egyik a másik elemi egyszerűsítése, akkor a sorozat bármely két eleme ugyanazt a csoportelemet írja le. Azt mondjuk w_1 és w_n ekvivalens. Ez egy ekvivalenciareláció a B-ből felírható csoportkifejezések halmazán. Az ekvivalenciaosztályok között könnyű szorzást, inverzet, egységosztályt definiálni. Így egy csoporthoz jutunk. Ez a B generátorhalmazhoz tartozó „legbővebb" generált csoport. A neve a B által szabadon generált csoport.

A B által szabadon generált csoport esetén könnyű tervezni egy algoritmust, amely két adott szóról eldönti, hogy ugyanazt a csoportbeli elemet írják-e le. Jóval általánosabb csoportok is leírhatók a fenti módszer általánosításával: Adjunk meg elemi egyszerűsítésekkel (és persze elemi bonyolításokkal) nem levezethető szóegyenlőségeket. Ha ilyen összefüggések egy halmazát adjuk meg, akkor ehhez is tartozik egy csoport: az elemi egyszerűsítés/elemi bonyolítás fogalmát ki kell terjeszteni az egyenlőség egyik oldalán szereplő kifejezés átírásával a másik oldalon szereplő kifejezésre. Így ha adott egy B halmaz és T egyenlőségek egy halmaza (ezek bal és jobb oldalán egy-egy szó szerepel), akkor egy $G = \langle B; T\rangle$ csoportot írtunk le.

Amennyiben B és T véges az így leírt csoportok a végesen prezentált csoportok. Például $\langle a, b; ab = ba\rangle$ egy csoport. Könnyen ellenőrizhető, hogy ez $(\mathbb{Z}, +) \times (\mathbb{Z}, +)$.

Ezekután a problémánk: Legyen adva egy B véges generátorhalmaz, egy véges T összefüggés halmaz (így adva van egy $G = G(B; T)$ végesen prezentált csoport). Adott még két B-re épített szó. Döntsük el, hogy azonos csoportbeli elemet írnak-e le.

Definíció.

$$\text{SZÓPROBLÉMA} = \{\ulcorner B, T; w_1 = w_2\urcorner : \text{a } \langle B; T\rangle \text{ csoportban a } w_1 \text{ és } w_2 \text{ csoportelemek megegyeznek}\}$$

A probléma eldönthetetlen,

$$\text{SZÓPROBLÉMA} \notin \mathcal{D}.$$

azaz

Igazából létezik olyan egyetlen végesen generált csoport, amely olyan komplex, hogy az erre vonatkozó szóprobléma (a csoport most nem része az inputnak) is eldönthetetlen.

Példa (Homeomorfizmus probléma). HOMEOMORF inputja két topológikus tér. Azt kell eldöntenünk, hogy homeomorfak-e.

Ismét a lényeges kérdés: Hogyan kódolunk topológikus tereket? A legegyszerűbb megoldás a rekurzió: Egyszerű, jól ismertnek vett topológikus terekből egyszerű operációkkal „felépítünk" további, bonyolultabbakat. Talán a legkombinatorikusabb lehetőség, ha szimplexekből indulunk ki. Szimplexek a pontok, szakaszok, háromszögek, tetraéderek. Ezek pontosan a legfeljebb három-dimenziós szimplexek. Minden d természetes szám esetén definiálható egy d-dimenziós szimplex, például a $\mathbb{R}^d$ origója és e_i standard báziselemeinek konvex burka. A felépítés lehet a lap-menti ragasztás. A Könnyű igazolni, hogy csak a kiinduló szimplexek dimenziója és a ragasztásnál használt lapok ismerete elég a leírt topológikus tér homomorfiatípusának ismeretéhez. Ennek leírásához a szimplexeket és lapjaikat azonosítjuk csúcsaik

halmazával. A szimpliciális komplexus egy halmazrendszer lesz egy véges V halmaz felett. A szimpliciális komplexus egyetlen tulajdonsággal jellemezhető: minden hozzátartozó halmaz összes részhalmaza is hozzátartozik (egy szimplex csúcsainak tetszőleges csúcshalmaza egy jól meghatározott lapja — ami szintén egy szimplex — csúcshalmaza).

A HOMEOMORF probléma (pontosabban a SZIMPLICIÁLIS-KOMPLEXUSOK-HOMEOMORFIZMUSA probléma) nem eldönthető. Azaz

$$\text{HOMEOMORF} \notin \mathcal{D}.$$

Példa (Post megfeleltetési problémája). A POST problémában adott Σ véges ábécé. Az input egy dominó készlet: Véges sok dominótípus, ahol egy típus egy alsó és egy felső minta, ami egy-egy Σ^*-beli szó. Minden típusból végtelen sok dominónk áll rendelkezésünkre. Azt kell eldönteni, hogy ki tudunk-e rakni dominóinkból egy sort úgy, hogy az alsó és felső minták összeolvasva (konkatenálva) ugyanaz a szót adják.

A probléma a mi elemi tárgyalásunk helyett a félcsoportok nyelvén is elmondható. Az irodalomban legtöbbször félcsoportokra vonatkozó problémaként ismertetik ezt a nyelvet.

A probléma nem eldönthető.

$$\text{POST} \notin \mathcal{D}.$$

A kurzus továbbiakban részében a $\mathcal{D}$ halmaz nyelveivel dolgozunk. Célunk az eldöntési problémák összehasonlítása, a döntési feladatok nehézségének mérése.

2. $\mathcal{EXPSPACE}$-ben

Példa. IDEÁL-ELEM-TESZT inputja egy végesen generált ideál a $\mathbb{Q}[x_1, x_2, \ldots, x_n]$ polinomgyűrűben és egy p polinom. Az ideál $g_1, g_2, \ldots, g_N$ generáló polinomokkal adott. A kérdés, hogy p az ideálhoz tartozik-e.

Könnyű leírni az ideált: az $\alpha_1 g_1 + \alpha_2 g_2 + \ldots + \alpha_N g_N$ alakú polinomok, ahol az α_i együtthatók is polinomok. Hogy egy hatékony nem-determinisztikus algoritmust adjunk ez alapján kellene egy becslés az ideálhoz tartozást bizonyító együtthatókra (fokaikra és együtthatóikra). Ez nem egyszerű.

A Gröbner-bázisok elméletén alapulva $\mathcal{EXPSPACE}$ bonyolultságú algoritmus adható a problémára. Azaz

$$\text{IDEÁL-ELEM-TESZT} \in \mathcal{EXPSPACE}.$$

Példa. IDEÁL-TELJESSÉG inputja egy végesen generált ideál a $\mathbb{Q}[x_1, x_2, \ldots, x_n]$ polinomgyűrűben. Az ideál $g_1, g_2, \ldots, g_N$ generáló polinomokkal van leírva. A kérdés, hogy az ideál a teljes gyűrű-e, azaz 1 az ideálhoz tartozik-e.

Ez nyilván az előző probléma egy speciális esete. Bonyolultsága legfeljebb akkora mint az előző kérdésé. A Gröbner-bázisok elméletén alapulva $\mathcal{PSPACE}$ bonyolultságú algoritmus adható a problémára. Azaz az algoritmuselmélet ki tudja használni a specialitását a problémának (az előző kérdéshez képest).

$$\text{IDEÁL-ELEM-TESZT} \in \mathcal{EXPSPACE}.$$

Példa. SLIDING-BLOCK-PUZZLE inputja egy $n \times m$ táblázatban (mint alappályán) elhelyezett egymást át nem fedő téglalapok. A téglalapok a pályát nem fedik le teljesen, így lehetőség van tologatásukra. El kell döntenünk, hogy az input/kiinduló konfigurációból tologatásokkal el tudunk-e jutni egy célkonfigurációba. Azaz elérhető-e egy célkonfiguráció-halmaz egy eleme (mondjuk az egyik téglalapot egy adott pozícióba vihetjük-e)?

1. ábra.

Könnyű becsülni a megfelelő konfiguráció-gráf méretét és ez alapján igazolni, hogy

$$\text{SLIDING-BLOCK-PUZZLE} \in \mathcal{PSPACE}.$$

3. $\mathcal{NP}$-ben

Példa. HAMILTON probléma inputja egy gráf. El kell döntenünk, hogy van-e benne Hamilton-kör.

Formálisan:

Definíció. HAMILTON=$\{\lceil G \rceil : G$ egyszerű gráfnak létezik Hamilton-köre$\}$

Igen válasz esetén a tanúszalagon elvárhatjuk a csúcsok egy olyan felsorolását, ami egy Hamilton-kör bejárásából nyerhető. Ellenőriznünk kell, hogy az egymásutáni csúcsok szomszédosak a gráfban és az első, illetve utolsó csúcs is összekötött. Ellenőrizni kell azt az „ígéretet" is, hogy minden csúcsot pontosan egyszer soroltunk fel. Tezteink nyilván polinomiális időben megvalósíthatók. Ha ezen tesztek mindegyike stimmel, akkor a tanú bizonyítja, hogy inputgráfunkban van Hamilton-kör. Másrészt nyilván minden Hamilton-körrel rendelkező gráfhoz található bizonyító tanú. Kaptuk, hogy

$$\text{HAMILTON} \in \mathcal{NP}.$$

coNP-beliséghez a Hamilton-kör hiányát kellene hatékonyan megindokolnunk. Könnyű egy polinomiális Turing-gépet tervezni, ami teszteli, hogy a tanúszalag tartalma egy U csúcshalmaz-e és $G - U$ komponenseinek száma nagyobb-e mint U

elemszáma. Ha igen, akkor biztosak lehetünk, hogy gráfunkban nincs Hamilton-kör. Valóban U elhagyása után a Hamilton-kör megmaradt ívei garantálnák, hogy $|U|$-nál nem több komponensünk van. A fent leírt gép azonban NEM bioznyíta a HAMILTON nyelv $co\mathcal{NP}$beliségét. Nem igaz, hogy Hamilton-kör hiányát ilyen módon biztos igazolni tudjuk. A Petersen-gráfban nincs Hamilton-kör. A fenti gép nem fogadná el.

Igazából nem ismert, hogy HAMILTON a $co\mathcal{NP}$ nyelvhez tartozik-e.

A Hamilton-kör probléma $\mathcal{NP}$ egy „nehéz" példánya. Ezt a matematikailag pontatlannak tűnő állítást a későbbiekben pontosan megfogalmazzuk es bebizonyítjuk.

$\star$

Definíció. FAKTORIZÁCIÓ$= \{\lceil(n,t)\rceil : n$-nek van t-nél kisebb valódi osztója$\}$.

A FAKTORIZÁCIÓ egy számelmélethez kapcsolódó nehéz nyelv. A nehéz jelző a több évszázados vizsgálatokon alapul, matematikailag bizonyítható nehézség nem ismert. Azaz a jelző egy „hit". Korunk több titkosító algoritmusa esetén az, hogy megbízunk benne, ezen a hiten alapul.

4. $\mathcal{P}$-ben

Példa. TELJES-PÁROSÍTÁS-TESZTELÉS inputja egy egyszerű gráf. El kell döntenünk, hogy az input tartalmaz-e teljes párosítást.

Az input kódolását nem tárgyaljuk. Azonban azt megjegyezzük, hogy a fenti értelemben v, a csúcsszám is vehető az input méretének (kódja hossza helyett).

Először egy nemdeterminisztikus algoritmust írunk le. A nemdeterminizmus második értelmezését használjuk. Azaz egy tanúszalag tartalma segítségével döntünk az elfogadásról. A tanúszalag tartalma csúcspárok egy M halmaza lesz.

A T gép azt teszteli, hogy a csúcspárok éllel összekötött párok-e, és minden csúcs pontosan egy párban szerepel-e. Ha mindkétszer igen a válasz, akkor ELFOGAD állapotba kerülünk. Ha valamelyik teszten elbukik a tanú, akkor NEM-STIMMEL állapotba kerülünk.

Egy teljes párosítás létezése esetén könnyű bizonyító tanút megadnunk. Ha nincs teljes párosítás, akkor mindegyik tanú elbukik.

A tesztek polinom időben könnyen elvégezhetők. Így kaptuk, hogy

$$\text{TELJES-PÁROSÍTÁS-TESZTELÉS} \in \mathcal{NP}.$$

A feladatunk nem annyira egyszerű, ha a teljes párosítás nem létét szeretnénk nem determinisztikusan bizonyítani. Tutte-tétel ismeretében azonban ekkor is egyszerű dolgunk van: A tanúszalag tartalma legyen egy T ponthalmaz. A gép az $\omega \equiv G$ gráf és $\tau \equiv T$ ponthalmaz esetében meghatározza $G-T$ komponenseit, megszámolja páratlan pontszámúakat és ezt a számot összehasonlítja $|T|$-vel. Amennyiben T elemszáma kisebb a páratlan pontszámú komponensek számánál a gép ELFOGAD állapotba kerül (a komplementer nyelvhez definiáljuk a gépet; az elfogadás azt jelenti, hogy a komplementer nyelv eleme, azaz nincs benne teljes párosítás). Valóban T bizonyítja ezt: $G-T$ minden páratlan pontszámú komponensében lesz olyan csúcs, ami a komponensen belülről nem kaphat párt (nyilvánvaló számelméleti okok miatt). Ezek a csúcsok csak T-beli párral rendelkezhetnek. A teljes párosításhoz azonban

nincs elég csúcs T-ben. Gépünk minden más esetben NEM-STIMMEL állapotba kerül. A gép polinomiális megvalósíthatóságának igazolása az olvasó feladata. Az algoritmus korrektsége (G-ben akkor és csak akkor nincs teljes párosítás, ha alkalmas T tanú ezt bizonyítja) éppen Tutte-tételének állítása. Így kapjuk a következőt

$$\text{TELJES-PÁROSÍTÁS-TESZTELÉS} \in co\mathcal{NP}.$$

Az Edmonds-algoritmus Turing-gép megvalósítása egy polinomiális algoritmus. Ez (az igen összetett) algoritmus az előző két eredménynél erősebb állításhoz vezet:

$$\text{TELJES-PÁROSÍTÁS-TESZTELÉS} \in \mathcal{P}.$$

Példa. PRÍM-TESZTELÉS probléma inputja egy n pozitív egész (mondjuk 10-es számrendszerben kódolva). El kell döntenünk, hogy príme-e.

A PRÍM-TESZTELÉS-sel kapcsolatban az egyszerű feladat a nem prímség bizonyítása. Ehhez csak egy valódi osztót kell előhoznunk tanúként. Könnyű ellenőrizni az oszthatóságot (és a valódiságot is). Kapjuk, hogy

$$\text{PRÍM-TESZTELÉS} \in co\mathcal{NP}.$$

A prímség $\mathcal{NP}$-bizonyítása már fogósabb kérdés. Páros számok esetén könnyű dolgunk van, a „prímség" megegyezik a „kettővel egyenlő" fogalommal. Feltehető, hogy n páratlan. Könnyű látni, hogy n akkor és csak akkor prím, ha $(\mathbb{Z}_n - \{0\}, \cdot)$ egy ciklikus csoport, azaz alkalmas $1 < g < n$ számra a $g, g^2, g^3, \ldots, g^{n-1}$ $\mathbb{Z}_n - \{0\}$ elemeit sorolja fel ($\bmod n$ aritmetikában számolunk). Könnyű látni, hogy ez ekvivalens azzal, hogy a sorozatban g^{n-1} az első 1 érték. Persze ha $g^{n-1} = 1$, akkor $g^\nu = 1$ esetén $\nu | n-1$. Tehát, ha a g hatványai között a g^{n-1}-nél korábbi 1-es előfordulást ki akarjuk zárni, akkor elég $g^{n-1/p}$ értékeket ellenőrizni. Ha ezek egyike sem 1 ($g^{n-1} = 1$ mellett), akkor g bizonyítja $(\mathbb{Z}_n - \{0\}, \cdot)$ azon tulajdonságát, ami mellett biztosak lehetünk n prímségében. A tanúszalagra g-t nem elég felírnunk. Szükségünk van $n-1$ prímtényezőire is. Így elvárjuk, hogy a tanúszalagon ott legyen $n-1$ prímtényezős felírása is. Azt könnyű ellenőriznünk, hogy a felsorolt számok (multiplicitásukkal) összeszorozva $n-1$-et adják. Az algoritmus korrektsége azonban azt jelenti, hogy nem lehet „hamis tanúkat elő'llítani". Hogy ebben bizonyosak legyünk, azt is tudnunk kell, hogy a tanúszalagon felírt prímtényezők valóban prímek. Ehhez meg kell követelnünk, hogy $n-1$ prímosztóiról a fent leírt sémát rekurzíven alkalmazva bizonyítást lássunk prím mivoltára (így persze csak a páratlanokkal van gondunk). Azaz mindegyik p-hez kell egy g_p szám és $p-1$ prímtényezős felbontása. Az input szalag tartalma egy prímséget állító tétel. A tanúszalag tételek (lemmák, segédlemmák, ...) sorozatát adja. Ezek az állítások egy fastruktúrába rendezhetők. Az input n szám (a főtétel) a gyökérrel van kapcsolatban. Ez alatt vannak $n-1$ páratlan prímosztóira vonatkozó lemmák. Ezek mindegyikének értéke legfeljebb $n - 1/2$. Azaz a fa mélységére az input hoszával arányos felső becslést adhatunk. Minden szinten a szereplő számok szorzata n-nél kisebb. Így a szükséges tanú hossza is kezelhető, az elfogadás polinom időben megtehető, azaz

$$\text{PRÍM-TESZTELÉS} \in \mathcal{NP}.$$

Agrawal—Kayal—Saxena-prímteszt a következő tételhez vezet:

$$\text{PRÍM-TESZTELÉS} \in \mathcal{P}.$$

Megjegyezzük, hogy $co\mathcal{NP}$-hez tartozás a prímség definíciójából közvetlenül adódik, az ókori matematikához kapcsolódik. Az $\mathcal{NP}$-beliség Pratt 1975-ben publikált eredménye. A polinomiális algoritmus a 2002-es bejelentés után 2004-ben jelent meg a matematika egyik legrangosabb folyóiratában.

Példa. LP-TESZTELÉS probléma inputja egy $A_{m\times n}$ mátrix és egy $b_{m\times 1}$ (oszlop)vektor. Kódolhatósági megfontolásokból racionális számok fölött dolgozunk. El kell döntenünk, hogy az $Ax = b$ egyenletrendszernek ($x = (x_1, x_2, \ldots, x_n)^T$) van-e nem negatív megoldása.

Valójában az egészek felett is dolgozhatunk. Az inputban szereplő számok nevezőinek legkisebb közös többszörösével megszorozhatjuk egyenleteinket. Az eredetivel ekvivalens egyenletrendszer együtthatói leírásának összhossza az eredeti inputméret polinomjával (négyzetével) becsülhető.

Az $\mathcal{NP}$-beliség egyszerűnek tűnik. A tanúszalagra fel kell írni egy megoldást. A gép csak ellenőrzi ezt. A probléma, hogy az ellenőrzés csak a tanúszámok méretében lesz polinomiális (szemben az inputszámokkal). Azaz vigyáznunk kell, hogy tanúnk ne legyen lényegesen hosszabb az input méreténél. Ilyen tanú létezik. Ennek indoklását itt nem végezzük el.

$$\text{LP-TESZTELÉS} \in \mathcal{NP}.$$

Egy egyenletrendszer nem negatív számok körében való meg nem oldhatóságára ismertetünk egy módszert. Az egyenleteink számszorosa, ezek összege a kiinduló rendszer egy következménye. Ha ezt a következtetést úgy végezzük, hogy a bal oldalon szereplő kikombinált lineáris kifejezésben minden együttható nem negatív legyen, míg a jobb oldalon egy negatív szám adódjon, akkor nagyon transzparens lesz, hogy a következtett egyenletnek nincs nem negatív megoldása. Így az eredeti egyenletrendszernek sincs. Az előzőekben nem ismertetett gondolatmenethez hasonlóan belátható, hogy a bizonyító következmények között olyan is van, ami kezelhető együtthatókkal kikombinálható. Így a tanúszalagról leolvasható és tesztelhető polinomiális időben. A fent ismertetett stratégia akkor vezet $\mathcal{NP}$ algoritmushoz, ha igaz, hogy nem megoldható inputrendszer esetén ilyen bizonyítás is található rá. Ez a jól-ismert Farkas-lemma. Tehát

$$\text{LP-TESZTELÉS} \in co\mathcal{NP}.$$

Jelenleg több olyan lineáris programozási algoritmus is van, ami polinomiális időben fut. Azaz

$$\text{LP-TESZTELÉS} \in \mathcal{P}.$$

Megjegyezzük, hogy az LP-TESZTELÉS a lineáris programozás optimalizálási probléma egyik döntési változata. $\mathcal{NP}$-belisége klasszikus becsléseken alapul. $co\mathcal{NP}$-belisége a Farkas-lemmán alapul, amit 1902-ben publikált Farkas Gyula. A LP optimalizálás mind a mai napig ünnepelt szimplex algoritmusát 1947-ben jelent meg (Dantzig). 1972-ben Klee és Minty bizonyította hogy az algoritmus nem polinomiális (valójában exponenciális futási idejű). Az első polinomiális algoritmust Kachian adta 1979-ben.

5. $\mathcal{NL}$-ben

Példa (Irányított elérhetőség).

Definíció. $\overrightarrow{st}$-ELÉRHETŐSÉG$= \{\lceil \overrightarrow{G}, s, t\rceil : s, t \in V(\overrightarrow{G}),$ létezik $\overrightarrow{st}$ út$\} \subseteq \Sigma^*$.

A fenti definícióban szereplő nyelv a matematikailag leírt hármasok kódjait tartalmazza. A kódolást azonban nem részleteztük. Ez a matematikai „pongyolaság" technikai részletek/megállapodások hosszadalmas leírását hagyja ki. A megállapodásokban lévő esetlegességek lényegtelenek. A „természetes" kódolások mind megfelelők. Nézzünk néhány lehetőséget egy G egyszerű gráf ($|V(G)| = n$, az input gráf csúcsszáma) kódolására:

$\lceil G\rceil$, a szomszédsági mátrix sorfolytonos elolvasása ($\Sigma = \{0, 1\}$). G kódjának hossza n^2.

Egy kicsit spórolhatunk, ha csak a főátló feletti elemeket olvassuk el. Ekkor a kód hossza $\binom{v}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$ lesz. Mindkét esetben a csúcsok a mátrix soraival azonosítottak. Azaz az i-edik sorral azonosított csúcs neve/kódja lehet i vagy $i-1$. Ezt a számot kódolhatjuk a kettes számrendszerbeli alakjával. Technikailag jobb megállapodás, ha mindegyik csúcs kódja ugyanolyan hosszú. Ekkor a kódok $\lceil \log_2 n\rceil$ hosszú 0-1 sorozatok, az előbbi kettes számrendszerbeli felírások 0-kkal kiegészítve az elején.

Egy másik lehetséges kódolás egy csúcsot egy $\lceil \log_2 n\rceil$ hosszú 0-1 sorozattal ír le és minden csúcs kódja után „:" jelet követve a szomszédjai sorozatát sorolja fel (az elemek „," jelel elválasztva, a teljes sor „;" jellel lezárva). Az összes csúcs — mindegyik követve a szomszédai listájával — sorozata adja a teljes kódot. (A felhasznált ábécé $\{0, 1, :, ;\} \cup \{,\}$.) Ennek hossza

$$\sum_{v:v\in V(G)} (\lceil \log_2 n\rceil + 1 + d(v)(\lceil \log_2 n\rceil + 1)) = n(\lceil \log_2 n\rceil + 1) + 2|E(G)|(\lceil \log_2 n\rceil + 1).$$

Egyszerű gráfok esetén n pontú gráfok között a leghoszabb kód nagyságrendileg $n^2 \log_2 n$ hosszú. Kevés élű gráfoknál ez nagyságrendileg kevesebb mint n^2. Sok élű gráfoknál nagyságrendileg elhanyagolható a többlet.

Ami közös, hogy a kódszó hossza polinomiális n-ben. Egy csúcs kódjának hossza $\mathcal{O}(\log n)$. Azaz a kód hosszában polinomiális, logaritmikus, exponenciális az ugyanaz mint n-ben polinomiális, logaritmikus, exponenciális. Egy n pontú egyszerű gráf kódjának mérete a bonyolultságelmélet szempontjából tekinthető n-nek. Habár ez nem a kód mérete. Egy logaritmikus tárat használó algoritmusra tekinthetünk úgy, mint egy olyan eljárásra, amely munkaszalagja a csúcsok számának és még — mondjuk — 100 csúcsnak tárolására alkalmas hellyel rendelkezik.

2. Tétel. *$\overrightarrow{st}$-ELÉRHETŐSÉG$\in \mathcal{NL}$.*

Bizonyítás. $\overrightarrow{st}$-ELÉRHETŐSÉG$\in \mathcal{NL}$. Az $\mathcal{NL}$-beliséget bizonyító Turing-gép csupán két csúcs nevét és egy számlálót tárol. $|V(G)| = n$ a számláló határszáma, ha a számláló ennél több lesz, megszakítjuk a gép futását és NEM-STIMMEL állapotot adunk ki. A számláló egy irányított séta által meglátogatott csúcsok számát tárolja, ha ennyi lépésből nem érjük el a kijelölt csúcsot, akkor az nem érhető el a kiindulási helyről.

A Turing-gépet a nemdeterminisztikus gépek első definíciójával konstruáljuk. A gép az alábbi műveletsort hajtja végre a (G, a, z) inputon.

1. Bemásolja a-t a munkaszalagra elejére $v \leftarrow a$.

2. A számlálót 0-ra állítja.

3. A munkaszalagra v a mögé felírja a következőnek elért v^+ csúcsot (ez egy nem-determinisztikus tippelés).

4. Ellenőrzi, hogy létezik-e $\overrightarrow{vv^+}$ él. Amennyiben nem, úgy NEM-STIMMEL eredményt, ad. Ha igen, akkor az v csúcs helyére bemásolja a v^+-t, 1-gyel megnöveli a számlálót.

5. A számlálót összehasonlítja n-nel.

6. Ha a számláló nagyobb mint n, akkor a gép NEM-STIMMEL eredménnyel leáll.

7. Ha a számláló nem nagyobb mint n, akkor megnézi, hogy a eredeti helyén álló csúcs z-e. Ha igen, akkor ELFOGAD állapottal leáll. Ha nem, akkor a harmadik lépéshez tér vissza.

Nyilván az algoritmus tárigénye $3 \cdot \lceil \log_2 |V| \rceil$, amennyiben bináris ábécé-t használunk. Az is egyszeru:en látszik, hogy pontosan az elfogadandó inputok esetén van ELFOGAD álpotba vezető futás, akkor egy $\overrightarrow{az}$ út pontjait kell sorban megtippelni a gépnek a harmadik lépések során. ■

6. Hierarchia tételek

A fentiekben konkrét problémák bonyolultságára adtunk felső becslést. Ez egy algoritmus megadásával történik. A fenti érvelések korábbi algoritmusok előhívása alapján történtek. Ezen algoritmusok ismertetése a legkülönbözőbb kurzusokon történt meg: algoritmuselmélet, kombinatorika, diszkrét matematika, algebra, numerikus analízis, operációkutatás, analízis, optimalizálás...

Jó lenne az algoritmusok által adott felső becslést alsó becsléssel párosítani. Sajnos ilyen matematikai tételek egyelőre elérhetetlennek tűnnek. Egy alsó becslés arra is alkalmas, hogy két boznyolultsági osztály között valódi tartalmazást igazoljon.

A Cantor-féle átlós módszer alkalmazható szeparációs tételek igazolására. Mi csak két tételt mondunk ki bizonyítás nélkül.

3. Tétel. *Legyen $s(n)$ egy szép tár-függvény, és $\widehat{s}(n)$ egy függvény, amelyre*

$$\widehat{s}(n)/s(n) \to \infty,$$

ha $n \to \infty$. Ekkor

$$\cup_{\alpha \in \mathbb{N}} \mathcal{SPACE}(\alpha s(n)) \subsetneq \cup_{\alpha \in \mathbb{N}} \mathcal{SPACE}(\alpha \widehat{s}(n)).$$

4. Tétel. *Legyen $t(n)$ egy szép tár-függvény, és $\widehat{t}(n)$ egy függvény, amelyre*

$$\widehat{t}(n)/t(n) \log t(n) \to \infty,$$

ha $n \to \infty$. Ekkor

$$\cup_{\alpha \in \mathbb{N}} \mathcal{TIME}(\alpha t(n)) \subsetneq \cup_{\alpha \in \mathbb{N}} \mathcal{TIME}(\alpha \widehat{t}(n)).$$

5. Következmény. *Az*

$$\mathcal{L} \subset \mathcal{NL} \subset \mathcal{P} \subset \mathcal{NP} \subset \mathcal{PSPACE}$$

tartalmazás első és utolsó osztálya biztos nem egyenlő. Azaz a szomszédos osztályok valamelyikének párja két különböző osztályt alkot.

Konkrétan bizonyítani egy szigorú tartalmazást valamelyik szomszédos bonyolultsági osztálypárra, az hatalmas áttörés lenne a bonyolultságelméletben.