ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

4. Előadás: Példák

Előadó: Hajnal Péter 2015. tavasz

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 példa Turing nevéhez fűződik, az első kiszámíthatatlansági eredmény (a Turing-gép definícióját megadó cikkben szerepel). Be is bizonyítjuk eldönthetetlenségét.

Példa (Megállási probléma). A megállási problémában adott egy T Turing-gép és egy ω input. El kell döntenünk, hogy T leáll-e ω-n. Formálisan:

Definíció.

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

1. Tétel (Turing-tétel, 1936). *(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.

A következő (könnyen, de technikai módon megoldható) feltevéssel élünk. A Turing-gépeket azonsítjuk a természetes számokkal. Azaz minden i terméeszetes szám egy T Turing-gép kódja ($i = \lceil T \rceil$). Továbbá i-ből dekódolható T. Hasonlóan azonosítjuk Σ^*-ot és $\mathbb{N}$-et. $j = \lceil \omega \rceil$ esetén j-ből dekódolható ω.

Képzeljünk el egy $\mathbb{N} \times \mathbb{N}$ típusú táblázatot, amely (i, j) pozíciójában ($i = \lceil T \rceil$, $j = \lceil \omega \rceil$) ∞ áll, ha T az ω-n végtelen ciklusba kerül és 0 különben (T az ω-n leáll).

Indirekt feltevésünk az, hogy van olyan I Turing-gép, amely „kiszámolja" ezt a táblázatot.

Az ellentmondást Cantor átlós módszere adja. Megadunk egy E Turing-gépet a következő módon. Beolvas egy i inputot és kiszámolja a fenti táblázat i indexű ($\lceil T_i \rceil = i$, $\lceil \omega_i \rceil = i$) átlós elemét: Ha ez ∞ ha T_i nem áll le ω_i-n. Ekkor E gépünk STOP állapotba kerül. Ha ez 0 ha T_i leáll ω_i-n. Ekkor E gépünk jobbrabalra „lépeget", végtelen ciklusba kerül. Azaz E éppen a kiolvasott információval ellentétes viselkedést végez.

Tegyük fel, hogy $\lceil E \rceil = k$. Mit csinál E a k inputot olvasva?

A definíció alapján „kibontja k"-t mint Turing-gép és ekkor magát/E-t találja. E hogyan működik ω-n ($\lceil \omega \rceil = k$)? Akár leáll, akár végtelenségig fut E definíciója ellentmondáshoz vezet.

■

A bizonyítás lényege hasonlít Cantor bizonyítására, hogy $[0, 1]$ nem felsorolható/megszámlálhatóan végtelen halmaz (átlós módszer). Csak most valós számok helyett gépek, illetve tizedesvessző utáni pozíciók helyett inputok kódjai szerepelnek.

Példa. Ebben a példában a kiszámíthatóság elméletének egy nagy alakja, Emil Post által bevezetett problémát vizsgáljuk meg.

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 (Post 1946).

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

Példa (Szóprobléma). SZÓPROBLÉMA inputja tartalmaz egy G csoportot. G-re multiplikatív írásmódot használva hívatkozunk. Mielótt 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 (igy 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 kérdést Dehn 1911-ben vetette fel és ettől kedve központi problémának számított. A probléma eldönthetetlen (Novikov, 1955),

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

azaz nem várható egy univerzális eljárás a kérdés tisztázására.

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

A következő példa Hilbert X. problémájának megoldásából ered. Ennek történeti jelentősége van. Ez nagyban hozzájárult a kiszámíthatóság fogalmának tisztázásához, ami elvezetett a Turing-gép definíciójához.

Példa (Hilbert X. Problémája). Legyen

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

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. 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

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

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

2. Példák eldönthető nyelvekre

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).

$$\text{IDEÁL-TELJESSÉG} \in \mathcal{PSPACE}.$$

Példa. SLIDINGBLOCKPUZZLE 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 (az alaptábla oldalaival párhuzamosan, az át nem fedés betartásával). 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

$$\text{SLIDINGBLOCKPUZZLE} \in \mathcal{PSPACE}.$$

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

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.

Példa. LP-TESZT 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.

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

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

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

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

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

Megjegyezzük, hogy az LP-TESZT 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.

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

A PRÍM-TESZT-tel 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

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

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 (mod 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 előállítani". 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

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

Ez Pratt (1975) tétele.

Agrawal—Kayal—Saxena-prímteszt (2004) a következő tételhez vezet:

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

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. TELJES-PÁROSÍTÁS-TESZT 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

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

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

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

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

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

Példa. ELÉRHETŐSÉG: Adott $\vec{G}$ egyszerű irányított gráf és s, t két csúcsa. Döntsük el, hogy van-e irányított st séta $\vec{G}$-ben.

Természetesen az $(\vec{G}, s, t)$ inputot kódolnunk kell. ELÉRHETŐSÉG azon kódok halmaz, amelyek gráf komponense tartalmaz st sétát.

A kódolásra az alábbiak leírunk egy példát: Legyen $v = |V|$. Az v szám leírásával kezdjük a kódunkat. A kód olvasójával ezzel azt is közöljük, hogy a csúcsokat $\lceil \log_2 v \rceil$ hosszú 0-1 kódokkal kódoljuk. A csúcskódok ezen bináris sorozatok közül a lexikografikus sorrendben az első v darab. Azaz az utolsó csúcskód $v-1$ bináris számrendszerben felírt alakja. Ha $v = 13$, akkor a csúcskódok hossza 4. A csúcskódok halmaza: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100.

A kezdeti csúcsszám után egy ; következik, majd a csúcsok felsorolva (kódjukkal), mindegyik után :-ot követve a kiszomszédok felsorolása lexikografikus sorrendben ,-kel elválasztva és ;-vel lezárva (kivéve az utolsó csúcs sorozatát, amit .-tal zárunk le). Egy példa egy gráf kódolására: 13; 0000 : 0010, 0101; 0001 : 0000, 1000, 1100; 0010 : 0000, 1001, 1011; 0011 :; 0100 :; 0101 : 0011, 0100; 0110 : 1100; 0111 :; 1000 : 0111, 1100; 1001 : 0000, 0001, 0010, 0011; 1010 : 0000, 0011, 0100; 1011 : 1010; 1100 : 0000, 1001. A kód hossza könnyen becsülhető: legalább $v \log_2 v$ és legfeljebb $(v^2+1)(\log_2 v + 1)$. Az inputméret polinomjával való becsülhetőség ekvivalens v polinomjával való becsülhetőséggel. Az inputméret logaritmusának számszorosával való becsülhetőség ekvivalens v logaritmusának számszorosával való becsülhetőséggel. Így (egy kissé nagyvonalúan) azt is mondhatjuk, hogy az input méretét v, a csúcsszám adja meg.

Az algoritmus, amit adunk, az egy nemdeterminisztikus algoritmus lesz. Azt is mondhatnánk, hogy egy eltévedt sétáló algoritmusa a $\vec{G}$ gráfban. A sétálás folyamán azt nézzük, hogy t-ben vagyunk-e, illetve számoljuk, hány lépést tettünk eddig meg. Ha elértük t-t, akkor ELFOGAD állapottal leállunk. Ha nem értük el t-t, akkor megnézzük, hogy tettünk-e v lépést. Ha igen, akkor NEM-STIMMEL állapottal leállunk. Ha még nem tettünk ennyi lépést, akkor nemdeterminisztikus lépésekkel felírunk egy csúcsot. Ellenőrizzük, hogy az előző csúcsból ide léphetünk-e egy élen keresztül. Ha nem, akkor ismét NEM-STIMMEL állapotba jutunk. Ha igen, akkor az előző csúcsot töröljük (!). Persze a törölt csúcs helyét a séta későbbi részére fenntartjuk. Így elérjük, hogy a tárban a futás minden pillanatában legfeljebb két csúcs van és egy számláló, ami értéke legfeljebb v. A szükséges tárigény $\mathcal{O}(\log v)$. Így kaptuk, hogy

$$\text{ELÉRHETŐSÉG} \in \mathcal{NL}.$$

Példa. Ismét az ELÉRHETŐSÉG nyelvet vizsgáljuk. Egy újabb algoritmust adunk, aminek tárfelhasználása lesz nagyon takarékos:

$$\overrightarrow{st}\text{-ELÉRHETŐSÉG} \in \cup_{\alpha\in\mathbb{N}} \mathcal{SPACE}(\alpha \log_2^2 n) = \mathcal{SPACE}(\log^2 n).$$

Ezt a következő rekurzív algoritmus bizonyítja.

Savitch-algoritmus:

KORLÁTOZOTT-$\overrightarrow{st}$-ELÉRHETŐSÉG$(x, y, 2^\ell)$:

// Adott x és y csúcsok esetén teszteli, hogy van-e köztük legfeljebb 2^ℓ lépéses
// séta. A sétára gondolhatunk úgy, mint egy „lusta" séta. Minden lépésnél
// két lehetőségünk van: vagy egy szomszédba mozgunk, vagy maradunk.
// Lusta sétánál feltehetjük, hogy a hossz pontosan 2^ℓ.

Ha $\ell = 0$, akkor teszteljük, hogy $x = y$ vagy $\overrightarrow{xy}$ egy él. Ha a teszt sikerül, akkor ELFOGAD állapottal, különben ELVET állapottal leállunk.

Ha $\ell > 0$, akkor

Összes $k \in V$ esetén

// k a lusta séta középső pontja, azaz x-ből k-ba $2^{\ell-1}$ lusta lépés vezet és k-ból
// y-ba is $2^{\ell-1}$ lusta lépés vezet. Az összes lehetőséget végigpróbáljuk.

(1) KORLÁTOZOTT-$\overrightarrow{st}$-ELÉRHETŐSÉG$(x, k, 2^{\ell-1})$
ha NEM, akkor következő k és vissza (1)-hez
ha NEM, és nincs következő k (V kimerült) akkor ELVET.
ha IGEN, akkor

(2) KORLÁTOZOTT-ELÉRHETŐSÉG$(k, y, 2^{\ell-1})$

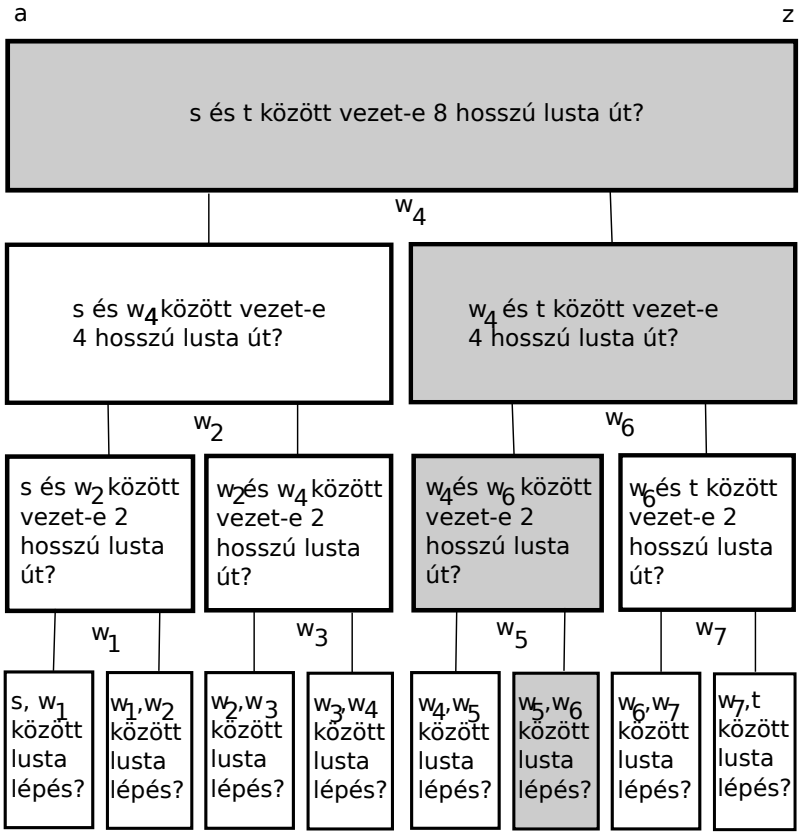
ha IGEN, akkor ELFOGAD állapot és leáll
ha NEM, akkor következő k és vissza (1)-hez
ha NEM, és nincs következő k (V kimerült) akkor vissza (1) NEM ágára.

A fenti algoritmus $\ell = \lceil |V(G)| \rceil$ paraméterrel futtatva megoldja az elérhetőséget. Az algoritmust Savitch Turing-gépen implementálta tár-takarékos módon.

2. Tétel (Savitch-tétel, 1970). *A fenti rekurzív algoritmus Turing-gépen megvalósítható úgy, hogy minden konfigurációban a munkaszalagon legfeljebb ℓ (a rekurzió mélysége sok) darab szakaszt írunk, ahol egy szakasz hossza $\mathcal{O}(\log |V|)$ (véges sok csúcs tárolására alkalmas hely).*

A pontos megvalósításhoz/implementációhoz csak ötleteket adunk:

A rekurzióban felmerülő kérdéseket struktúráját egy fával reprezentálhatjuk. A fa gyökere az ELÉRHETŐSÉG probléma, azaz az, hogy van-e 2^ℓ hosszú lusta séta? Minden p=(u-ból v-be vezet-e 2^ℓ hosszú lusta séta) probléma két részfeladatra bomlik: Egy középső w csúcsra $p_{bal}(w)$= (vezet-e u-ból w-be $2^{\ell-1}$ hosszú lusta séta), illetve $p_{jobb}(w)$= (vezet-e w-ból v-be $2^{\ell-1}$ hosszú lusta séta) a p probléma két részfeladata. A két feladat egymás *testvére*.



2. ábra. Az ábrán $|V| = 8$ esetén látjuk, hogy elfogadó futás esetén milyen részfeladatokat kell megoldani/ellenőrizni. A részfeladatok egy gyökeres bináris fában foglalhatók össze. A munkaszalag tartalma mindig egy feladat (csúcs a fában) a gyökérhez vezetett úttal együtt. Egy példát kiemeltünk sötétítéssel.

Egy új p problémának (ha $\ell \neq 0$) — amennyiben jelöltünk van egy w középső csúcsra — két gyerek-problémája lesz. Ha mindkettőt igenlően eldöntöttük, akkor

tudjuk, hogy p is igaz. Ha valamelyikre nemleges a válasz, akkor a w *rossz* középső csúcs p-re. A V csúcshalmaz elemeit felsoroljuk, a leheséges középső csúcsok eszerint a sorrend szerint következnek. Az aktuális w-nél először a bal-gyerek kerül a munkaszalgra. Eleinte a munkaszalag tartalmaz csak bővül amíg fánkban egy levélhez nem jutunk.

A levélnek megfelelő probléma könnyen ellenőrizhető (akár plusz munkaszalag-igény nélkül, csak az input olvasásával).

- Ha a bal-gyerek problémája IGENlően dől el, akkor a jobb-gyerek feladatával felülírjuk.
- Ha a bal-gyerek problémája NEMlegesen dől el, akkor w *rossz* jelölt volt. A feladatot töröljük a munkaszalagról és az apafeladattal foglalkozunk.
  - A következő w-re térünk át, azt mondjuk w-t *léptetjük*. A továbbiakat a fenteik alapján folytatjuk.
  - Ha nincs rákövetkező w (azt monjuk a megfelelő középső csúcs *kimerült*), akkor tudjuk, hogy p-re/az apafeladatra NEMleges a válasz. Töröljük a munkaszalagról és a továbbiakat a fenteik alapján folytatjuk.
- Ha a jobb-gyerek problémája IGENlően dől el, tudjuk, hogy p-re/az apafeladatra IGENlő a válasz. Töröljük a munkaszalagról és a továbbiakat a fenteik alapján folytatjuk.
- Ha a jobb-gyerek problémája NEMlegesen dől el, akkor w *rossz* jelölt volt. A feladatot töröljük a munkaszalagról és az apafeladattal foglalkozunk, ugyanúgy mint fentebb.

A munkaszalag tartalmaznak szervezése/felülírásának szabályai (idegen szóval update-szabály) megköveteli, hogy minden problémánál tudjuk, hogy ő bal vagy jobb gyerek. Ezt érdemes a probléma leírásába befoglalni (habár az apaproblémával összevetve ez ki is olvasható a tömörebb kódolásból). A fenti update-szabályoknak van egy szokásos értelmezése/interpretációja: A problémákat egy *veremben* tároljuk. A verem szó azért jogos, mert csak a verem tetején lévő feladatot látjuk, amit olvashatunk, kivehetünk a veremből, vagy rápakolhatunk. Fent éppen egy ilyen verem kezelési útmutatóját írtuk le. A verem tartalma (ez lesz amunkaszalagon) mindig egy gyökérből induló út csúcsai. Ahogy a gyökérből indulva végigmegyünk az úton, a veremben növekvő magasságban lesznek a feladatok. A verem tetején lévő probléma az út végén lévő csúcsnak felel meg.

Ha V elemei 0-1 sorozatokkal van kódolva ($\mathcal{O}(\log |V|)$ hosszúakkal) és a sorozataink kódjainak lexikografikus rendezésében az első $|V|$ darabot vesszük csúcskódnak, akkor a LÉPTETÉS lépés lehet eggyel való növelés, a KIMERÜLÉS tesztelése pedig az utolsó csúcs kódjának és a legnagyobb kódnak az összehasonlításából adódik. A leállási szabályok: Ha a gyökér-feladatot igenlően válaszoljuk meg, akkor gépünk ELFOGAD állapottal megáll. Ha a gyökér-feladat középső w csúcsa kimerül, akkor a gépünk ELVET állapottal megáll. A konstruált gép nyilván az ELÉRHETŐSÉG nyelvet fogadja el.

Az átmeneti függvény leírását (a munka-ábécé, állapothalmaz választását beleértve), azaz a technikai részletek kidolgozását nem végezzük el. A programozásban jártas hallgató elvégezheti. Könnyen ellenőrizhető, hogy a munkaszalag tartalma

legfeljebb ℓ probléma leírása, amelyek mindegyike két csúcs kódja, egy $k \leq \ell$ paraméter, és egy bit (apjának — amennyiben nem a gyökér — bal vagy jobb gyereke). A teljes tárigény $\mathcal{O}(\ell \cdot \log n) = \mathcal{O}(\log^2 n)$.