ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

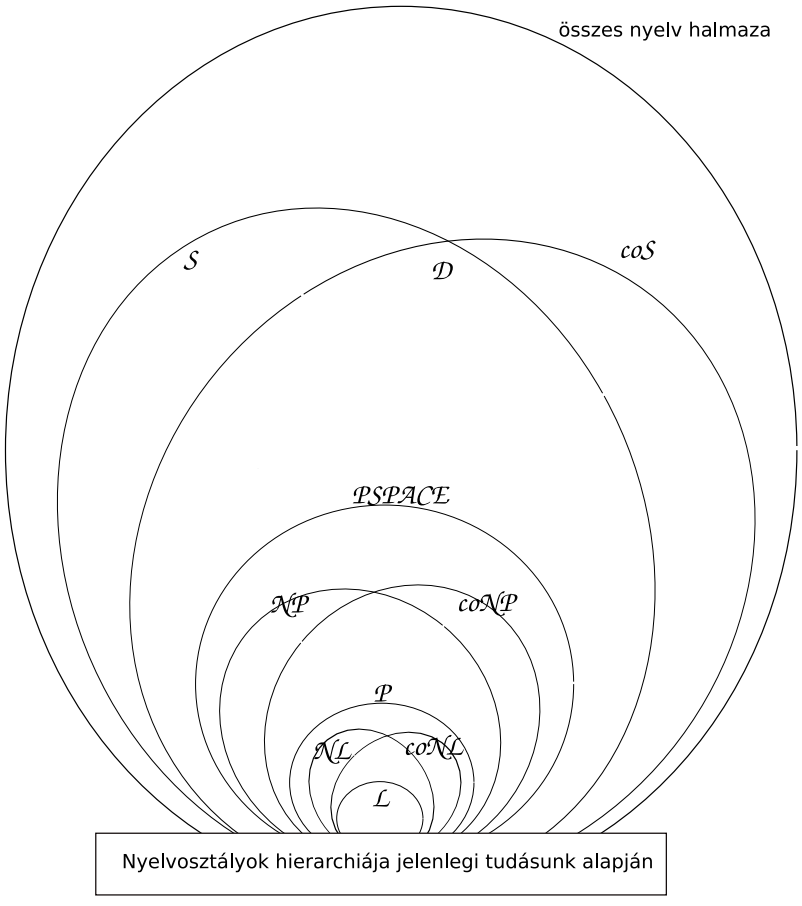
4. Előadás

Előadó: Hajnal Péter
Jegyzetelő: Hajnal Péter 2010. február 22.

Az eddigiek összefoglalása

$\mathcal{L} \subset \mathcal{NL} \subset \mathcal{P} \subset \mathcal{NP} \subset \mathcal{PSPACE} \subset \mathcal{NPSPACE} \subset \mathcal{EXP} \subset \mathcal{NEXP} \subset \mathcal{EXPSPACE}$.

Az alábbiakban Venn-diagrammal szemléltetjük a fenti tartalmazást. Az így kapott képpel óvatosan kell bánnunk. A Venn-diagramok ábráján kialakulhatnak (esetünkben ez valóban így is van, ahogy később látjuk) olyan tartományok, amelyek üresek. Két egybeeső nyelvhalmazt a diagram külön jelezhet.



1. ábra.

Példák

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. TELJESPÁROSÍTÁSTESZTELÉS 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

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

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

$$\text{TELJESPÁROSÍTÁSTESZTELÉS} \in co\mathcal{NP}.$$

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

$$\text{TELJESPÁROSÍTÁSTESZTELÉS} \in \mathcal{P}.$$

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

A PRÍMTESZTELÉS-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

$$\text{PRÍMTESZTELÉS} \in co\mathcal{NP}.$$

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 előallítani”. 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

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

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

$$\text{PRÍMTESZTELÉS} \in \mathcal{P}.$$

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. LPTESZTELÉS 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.

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

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

$$\text{LPTESZTELÉS} \in co\mathcal{NP}.$$

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

$$\text{LPTESZTELÉS} \in \mathcal{P}.$$

Megjegyezzük, hogy az LPTESZTELÉS 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. SLIDINGBLOCKPUZZLE inputja egy $n \times m$ táblázatban (mint alappályán) elhelyezett egymas't 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)?

2. 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. SZÓPROBLÉMA inputja egy G multiplikatív 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. A probléma magoldójának azt kell eldönteni, hogy két adott szó ugyanazt az elemet írja-e le. Egy másik formában egyetlen szó adott és azt kell eldöntenünk, hogy ez a csoport egységelemét írja-e le.

A fenti megfogalmazás nem pontos. Hogyan adunk meg egy csoportot? 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. Ha ezzel a csoport az input csoportja, akkor a szóprobléma egyszerűen eldönthető.

Persze G más is lehet mint ez. Ekkor kell lennie két szónak, hogy azok csak az elemi egyszerűsítésekkel (és persze elemi bonyolításokkal) nem kaphatók meg egymásból mégis egyenlőek. 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 véges halmaz és T egyenlőségek egy véges halmaza (ezek bal és jobb oldalán egy-egy szó szerepel), akkor egy csoportot írtunk le. Az így leírt csoportok a végesen prezentált csoportok. Például $\langle a, b; ab = ba\rangle$ egy csoport. könnyen ellenőrizhető, hogy ez $(\mathbb{Z}, +) \times (\mathbb{Z}, +)$. Ha ez szerepel az input csoportjaként, akkor a szóprobléma egyszerűen eldönthető.

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

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. 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. 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.

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

Példa. ELÉRHETŐSÉG az első problémánk volt. Most egy másik algoritmust ismertetünk, amely azt bizonyítja, hogy

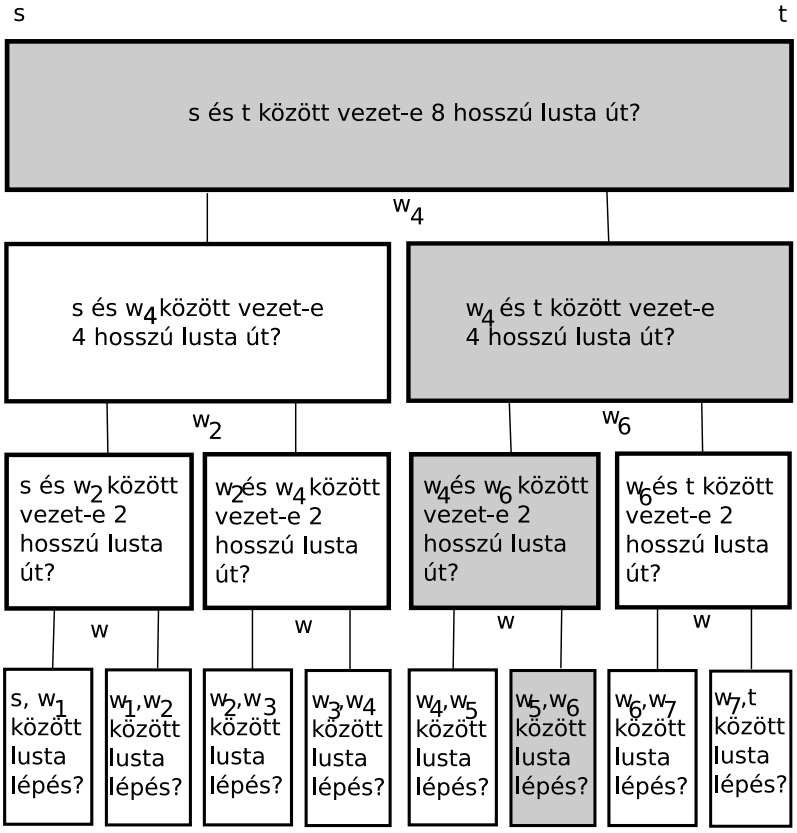
$$\text{ELÉRHETŐSÉG} \in \mathcal{SPACE}(\log^2).$$

Először egy rekurzív elirását adjuk az algoritmusnak. Az alábbi fenti algoritmus jól érthető azoknak, akik a rekurzióval már találkoztak. Azonban most nemcsak értenünk kell az eljárást, hanem azt is látnunk kell, hogy Turing-géppel megvalósítva

milyen tárigényű a program. Ehhez egy kicsit jobban el kell mélyednünk abban, hogy a rekurzió megvalósításához milyennek kell lenni a munkaszalag tartalmának.

```
ELÉRHETŐ-E(u,v,2^ℓ):
%Azt kell eldöntenünk, hogy van-e 2^ℓ hosszú lusta séta
%u és v adott csúcsok között.
%Egy lusta séta olyan módosítása a sétának, hogy egy lépesben
%az aktuális csúcsban is maradhatunk.
HA ℓ = 0, AKKOR
    ELÉRHETŐ-E:=(u=v)VAGY(uv∈ E);
HA ℓ > 0, AKKOR
    MINDEN w csúcsra
    HA ELÉRHETŐ-E(u,w,2^(ℓ−1)) ÉS ELÉRHETŐ-E(w,v,2^(ell−1)), AKKOR ELFOGAD;
HA eddig nem fogadott el, AKKOR ELVET.
%                                                    FŐPROGRAM
k= a minimális szám, amelyre |V| ≤ 2^k
OUTPUT=ELÉRHETŐ-E(u,v,2^k)
```

A Turing-gépes implementáció részletei: A rekurzióban felmerülő kérdéseket struktúráját egy fával reprezentálhatjuk.



3. ábra. A sötétített feladatok foglalják el a munkaszalagot, a levél kiértékelése alapján változtatjuk ezt meg

A fa gyökere az ELÉRHETŐSÉG probléma, azaz az, hogy van-e 2^k 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*. 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. Egy rossz w esetén a következő w-re térünk át, azt mondjuk w-t *léptetjük*. Ha nincs rákövetkező w (azt monjuk a megfelelő középső csúcs *kimerült*), akkor tudjuk, hogy p-re nemleges a válasz. Ha p nem a gyökér csúcs, akkor p-nek is van egy testvére. Őket az apa-feladat hoztalétre egy aktuális középső csúcsra tett tippel. Ezt a középső csúcsot kell lépteni, vagy kimerülés esetén az apa-feladatot nemlegesen kiértékelni

Az algoritmus Turing-implementációjában minden pillanatban a fenti fa egy gyökérből egy levélbe vezető út tartalma van ($\mathcal{O}(\log n)$ csúcs és a bal/jobb fordulók sorozata, azaz $\mathcal{O}(\log^2 n)$ karakter). Azt kell meggondolnunk, hogy ez a tartalom és a levél probléma megoldása ismeretében mekkora munkával/tárigénnyel frissíthető, azaz idegen szóval update-elhető.

Rossz w esetén léptetni kell. Ha V elemei 0-1 dorozatokkal van kódolva ($\mathcal{O}(\log |V|)$ hosszúakkal) és a sorozataink kódjainak lexikografikus rendezésében az első $|V|$ darabot vesszük csúcskódnak, akkor a léptetés lehet eggyel való növelés, a kimerülés pedig az utolsó csúcs kódjánál nagyobb kód elérését. A lépetetés és a kimerülésnek is gyűrűző hatása van. A léptetett csúcsot előhívó probléma alatti részproblémákat újból kell vizsgálni. Kimerülés magasabb szintű csúcs léptetéséhez (vagy kimerüléséhez) vezet. Hasonló gyűrűző hatása van a jó w megtalálásának. Ezek a gyűrűző hatások kezelése már kényelmetlenebb feladat mint a léptetés végrehajtása, illetve kimerülés tesztelése, de még ez is könnyen ellenőrizhető módon $\mathcal{O}(\log^2 n)$-es tárkorlátban maradva megtehető.

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.