ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

5. Előadás

Előadó: Hajnal Péter
Jegyzetelő: Hajnal Péter 2012. március 7.

1. Problémak redukciói

1.1. Emlékeztető

Definíció. 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.

1. Tétel. *Legyen $L \in_T \mathcal{NSPACE}(s(n))$, ahol $s(n) \geq \log n$ szép tárfüggvény. Feltehető, hogy a tartalmazást igazoló Turing-gépnek adott hosszúságú inputokon kétféle leálló konfigurációja van (így elfogadó futások ugyanabban a konfigurációban érnek véget).*

$\omega \in \Sigma^*$-hoz rendelhető a T Turing-gép redukált konfigurációinak (irányított) $\overrightarrow{G} = \overrightarrow{G}_{T,\omega}$ gráfja. Ebben ott van v_0 a kiinduló konfigurációnak megfelelő csúcs és v_1 az elfogadó konfigurációnak megfelelő csúcs. Ez a hozzárendelés olyan, hogy

(i) $\omega \in L$ akkor és csak akkor, ha $\lceil \overrightarrow{G}_{\omega,T}, v_0, v_1 \rceil \in \overrightarrow{st}$-ELÉRHETŐSÉG.

(ii) A hozzárendelés kiszámításának tárigénye $\mathcal{O}(s(n) + \log(n)) = \mathcal{O}(s(n))$.

1.2. Karp-redukció

A múlt óráról idezett tétel lényege furcsa. Egy ω input lényegében egy kérdés: ω hozzátartozik L-hez? A fenti tételben egy olyan hozzárendelést/számolás szerepe, ami a kérdés megválaszolása helyett egy új kérdést számol ki: „Leírok egy $\widetilde{\omega}$ új inputot és megkérdezem, hogy egy új nyelvhez ($\overrightarrow{st}$-ELÉRHETŐSÉG) tartozik-e. Ha valaki megmondja a választ, akkor én meg tudom mondani az eredeti kérdésre a választ. Sőt az ugyanaz lesz mint az én kérdésemre." A gondolat furcsa de fontos, központi fogalomhoz vezet el.

Definíció. Legyen $L, \widehat{L} \subset \Sigma^*$ két nyelv és $\mathcal{C}$ egy bonyolultsági osztály. L redukálható $\widehat{L}$-ra, jelben: $L \preccurlyeq_{\mathcal{C}} \widehat{L}$, ha létezik R kiszámítható Turing-gép, hogy

(i) R egy $\mathcal{C}$ komplexitású gép/eljárás,

(ii) $\omega \in L$ pontosan akkor, ha $\widetilde{\omega} \in \widehat{L}$, ahol $\widetilde{\omega}$ az ω-ból R által kiszámolt jelsorozat.

A bevezetett reláció olvasata: L redukálható $\widehat{L}$-re $\mathcal{C}$-ben. Az L-ben rejlő kérdés visszavezethető az $\widehat{L}$-ben rejlő kérdésre. Jelentése: Az $\widehat{L}$ nyelv eldöntési feladata „legalább olyan nehéz", mint az L-é „modulo $\mathcal{C}$".

Más redukció fogalmak is léteznek. A fenti definíció Karp munkásságában rejlik és általában Karp-redukcióként hivatkkozzák. Ha szükség van ennek hangsulyozására, akkor a $\preccurlyeq_{\mathcal{C}}^{\text{Karp}}$ jelölést használjuk. Ebben a kurzusban legtöbbször ilyen redukciót látunk. Legtöbbször le is hagyjuk a felső indexet. A teljessség kedvéért megemlítünk egy másik fajta redukciót. Ezt Turing nevéhez fűzik.

1.3. Turing-redukció

Definíció. Legyen $L, \widehat{L} \subseteq \Sigma^*$ két nyelv és $\mathcal{C}$ egy bonyolultsági osztály.
$L \preccurlyeq_{\mathcal{C}}^{\text{Turing}} \widehat{L}$ pontosan akkor, ha megadható R eldöntő Turing-gép, amelyre

(i) L-et dönti el és R egy L_2-orákulumos gép.

(ii) R bonyolultsága $\mathcal{C}$-beli.

(i)-ben szerepel egy eddig ismeretlen fogalom, amit tisztáznunk kell.

Definíció. Legyen $O \subset \Sigma^*$ egy nyelv. R egy O-orákulumos gép, ha van egy extra kérdés/orákulum-szalagja. Erre csak írhat a gép (nincs szem a szalag felett, a kéz csak jobbra mozogva írhat). Az írott karakterek $\Sigma \cup \{?\}$ elemei, azaz az O nyelv ábécéjének elemei és egy speciális '?' jel. A kérdésszalagra a ? jel feírása egy kérdés feltételét jelenti. Az előző kérdőjel (vagy szalag-kezdet jel) és közte lévő Σ^*-eli karaktersorozatról kérdezi meg a gép/algoritmus, hogy az orákulum O nyelvéhez tartozik-e. Az állapothalmaz

$$\{\text{ORÁKULUM-IGEN}, \text{ORÁKULUM-NEM}\} \times S_0$$

alakú. Az átmeneti függvény a következő konfigurációnak az állapotában csak a második komponensre ha. Az első komponens csak akkor változik, ha az algoritmus kérdést tesz fel az orákulumhoz. A változást a kérdés karaktersorozat O-hoz való viszonyától függ természetes módon. Azaz a kérdés ára 1 időegység és 0 tár. Ezek után a futás (a kiinduló konfigurációból generált konfigurációsorozat), a kiszámított nyelv értelemszerűen definiálható.

A Karp-redukció nagyon speciális Turing-redukció: Szokásos számolás után egyetlen kérdés hangozhat el az $\widehat{L}$ nyelvhez tartozásról. A kérdésre adott válasz egyben a kiszámított bit is.

A Turing-redukció nyilván sokkal erősebb fogalom. Az $\widehat{L}$-ra úgy gondolhatunk, mint egy megíratlan szubrutin. A redukció lényege, hogy ha a $\widehat{L}$ szubrutint valaki hatékonyan leírja, akkor L hatékonyan eldönthető (feltéve, hogy R hozzájárulása (ami $\mathcal{C}$ bonyolultságú) is hatékonynak tekintható).

1.4. Példák

Az emlékeztetőben felidézett tétel egy példává válik.

Példa. Legyen L egy tetszőleges $\mathcal{NSPACE}(s(n))$-beli nyelv, ahol $s(n) \geq \log n$ szép tárfüggvény. Ekkor

$$L \preccurlyeq_{\mathcal{SPACE}(\mathcal{O}(s(n)))} \text{ELÉRHETŐSÉG}.$$

Egy másik példa a BSc-s Kombinatorika tárgyból.

Példa. Legyen

$$\text{KLIKK} = \{\lceil G, k \rceil : G\text{-ben létézik } k \text{ elemű klikk}\}.$$

Legyen

$$\text{FÜGGETLEN} = \{\lceil G, k \rceil : G\text{-ben létézik } k \text{ elemű független csúcshalmaz}\}.$$

Tudjuk, hogy K akkor és csak akkor klikk, ha K független csúcshalmaz $\overline{G}$-ben, a komplementer gráfban. Tudjuk, hogy F akkor és csak akkor független csúcshalmaz, ha F klikk $\overline{G}$-ben.

Így KLIKK$\preccurlyeq_{\mathcal{P}}$FÜGGETLEN és FÜGGETLEN$\preccurlyeq_{\mathcal{P}}$KLIKK.

A redukció rendkívül egyszerű. Az első esetben a következő lesz: Adott egy $\omega = \lceil G, k \rceil$ inputja a KLIKK problémának. Ekkor G kódjából kiszámoljuk a komplmenterét (annak kódját). $\widetilde{\omega}$ a $\lceil \overline{G}, k \rceil$ karaktersorozat lesz. A korábbi gráfelméleti megjegyzésünk szerin az új „kérdés" ekvivalens az eredetivel. $\widetilde{\omega}$ kiszámításának bonyolultsága nyilván polinomiális.

Megjegyezzük, hogy se a KLIKK, se a FÜGGETLEN nyelvre nem ismert hatékony algoritmus. Ha bármelyikre lenne, akkor az a másik problémára is jelentős kihatással lenne.

1.5. Redukálhatóság tulajdonságai

Végül megemlítünk egy fontos tulajdonságát a redukcióknak.

2. Lemma. *(i)* $\preccurlyeq_{\mathcal{P}}$ *tranzitív.*

(ii) $\preccurlyeq_{\mathcal{L}}$ *tranzitív.*

(iii) Legyen $s(n) \geq \log n$ *szép tárfüggvény. Ha* $L_1 \preccurlyeq_{\mathcal{SPACE}(\mathcal{O}(s(n)))} L_2$ *és* $L_2 \preccurlyeq_{\mathcal{L}} L_3$ *, akkor* $L_1 \preccurlyeq_{\mathcal{SPACE}(\mathcal{O}(s(n)))} L_3$

Bizonyítás. (i) Tegyük fel, hogy $L_1 \preccurlyeq_{\mathcal{P}} L_2$ és $L_2 \preccurlyeq_{\mathcal{P}} L_3$. Továbbá R_1 és R_2 kettő, a két redukciót igazoló algoritmus. Speciálisan R_1 és R_2 is polinomiális. Legyen p_1 és p_2 két polinom, ami R_1 és R_2 időkorlátját adják. p_2-ről feltehetjük, hogy monoton növő.

Egy ω inputon futtassuk R_1-et, ami $\widetilde{\omega}$ karaktersorozatot számolja ki. Majd R_2-t futtasuk $\widetilde{\omega}$-n, ami $\widetilde{\widetilde{\omega}}$ kiszámításához vezet. Az így kapott Turing-gép legyen R. Belátjuk, hogy R a $L_1 \preccurlyeq_{\mathcal{P}} L_3$ redukciót igazolja.

$\omega \in L_1$ akkor és csak akkor teljesül, ha $\widetilde{\omega} \in L_2$. Ami akkor és csak akkor teljesül, ha $\widetilde{\widetilde{\omega}} \in L_3$ban,

Be kel még látni, hogy R polinomiális. ω inputon R időigénye $p_1(|\omega|) + p_2(|\widetilde{\omega}|)$. $\widetilde{\omega}$-t egy p_1 időkorlátú gép számolja ki ω-ból, így $|\widetilde{\omega}| \leq p_1(\omega)$. Így ω-n a futási idejére a következő felső becslés adódik

$$p_1(|\omega|) + p_2(|\widetilde{\omega}|) \leq p_1(|\omega|) + p_2(p_1(|\omega|)).$$

Ez egy polinomiális felső becslés.

(ii) Tegyük fel, hogy $L_1 \preccurlyeq_{\mathcal{L}} L_2$ és $L_2 \preccurlyeq_{\mathcal{L}} L_3$. Továbbá R_1 és R_2 kettő, a két redukciót igazoló algoritmus. Speciálisan R_1 és R_2 is logaritmikus tárigényű.

A két redukcióból az előző módon rakunk össze egy R algoritmust: Egy ω inputon futtassuk R_1-et, ami $\widetilde{\omega}$ karaktersorozatot számolja ki. Majd R_2-t futtasuk $\widetilde{\omega}$-n, ami $\widetilde{\widetilde{\omega}}$ kiszámításához vezet.

Az így kapott algoritmus NEM jó. A közbülsőnek kiszámolt $\widetilde{\omega}$-ra a munkaszalagon an szükség. Ez várhatóan nem fér el logaritmikus tárban. Ennek ellenére ezen R algoritmus futása legyen a fejünkben. Az igazi $\widetilde{R}$ redukcióban felismerjük R futásának töredékeit.

$\widetilde{R}$ munkaszalagjai megfelelnek R_1 munkaszalgjainak plusz R_2 munkaszalgajainak. Lesz két plusz szalagunk a korábbi szalag helyett, ami R_1 output- és a vele közös R_2 inputszalagja volt. A plusz két szalag közül az első R_1 output szalagjának egy pozíciójának indexét (a szalag feletti kéz pozícióját) tartalmazza, míg másik szalag tartalma R_2 inputszalagján egy pozíció indexe (a szalag feletti szem pozíciója).

$\widetilde{R}$ az R_2 szimulációját végzi az $\widetilde{\omega}$ inputszalag tartalom nélkül. Minden olvasási feladatnál meg kell dolgoznunk. Ekkor tudjuk, hogy az R_1 által kiszámolt karaktersorzat hanyadik karakterére vagyunk kiváncsiak. El kezdjük R_1 szimulálását. A szimuláció során a kiszámolt karaktereket nem írjuk le, csupán az output-kéz pozícióját tároljuk. Ha R_1 ír, akkor az új pozíciót összeasonlítjuk az olvasni kívánt pozícióval. Ha megegyezik a két pozíció, akkor a le nem írt karaktert kiolvassuk az állapotból és az R_1-szimulációt leállítjuk, folytatjuk az R_2-szimulációt. Ha a két pozíció különbözik, akkor az R_1-szimulációt folytatjuk.

(iii) Az előző bizonyítás ötletei most is működnek. ■

A fenti bizonyítás egy kis módosítása az alábbi lemmához vezet.

3. Lemma.

(i) $L \preccurlyeq_{\mathcal{P}} \widehat{L}$ *és* $\widehat{L} \in \mathcal{P}$*, akkor* $L \in \mathcal{P}$*.*

(ii) $L \preccurlyeq_{\mathcal{L}} \widehat{L}$ *és* $\widehat{L} \in \mathcal{L}$*, akkor* $L \in \mathcal{L}$*.*

Bizonyítás. Tekintsünk egy A Turing-gépet, amely az L-ről $\widehat{L}$-ra történő redukciót végzi, valamint $\widehat{A}$-ot, amely a $\widehat{L}$ nyelvehz tartozási feladatot dönti el $\mathcal{P}$-ben. Legyen adott az ω input.

Ekkor végezzük el a

$$\omega \to A(\omega) \in \Sigma^{p(n)} \to \widehat{A}(A(\omega)) \in \{\text{ELFOGAD}, \text{ELVET}\}$$

számolást. Az első időigényét az n inputméretben egy p polinom korlátozza. A leghosszabb input, amit kiszámolhatunk $\Sigma^{p(n)}$-ba esik. A második lépés időigényét az inputméretben egy q polinom korlátozza. Az összidő $(p + q \circ p)(n)$, ami n egy polinomiális függvénye.

A két algoritmus együttese a Karp-redukció fogalma alapján éppen az L nyelvet dönti el, vagyis L is eldönthető polinom időben.

(ii) Az (i) rész és az előző lemma ötletei alapján nyilvánvaló. ■

2. Teljesség

Az előadás elején szereplő emlékeztető jóval töb mint egy redukció. Az ott szereplő L nyelv egy nyelvosztály ($\mathcal{NL}$) tetszőleges eleme. Ez az észrevétel

A fenti okoskodás nagyon fontos. Hogy a gondolatmenetet kihasználjuk egy nagyon fontos fogalmat vezetünk be:

Definíció. $\widehat{L}$ nyelv *teljes* a $\mathcal{C}$ osztályban az $\mathcal{R}$ bonyolultságú rekurzióra, ha:

(i) $\widehat{L} \in \mathcal{C}$,

(ii) minden $L \in \mathcal{C}$ esetén $L \preceq_{\mathcal{R}} \widehat{L}$.

Két speciális esetet kiemelünk.

Definíció. Az $\widehat{L}$ nyelv $\mathcal{NP}$-teljes ha teljes $\mathcal{NP}$-ben a $\mathcal{P}$ redukcióra. Azaz $\widehat{L}$ nyelv $\mathcal{NP}$*-teljes*, ha

(i) $\widehat{L} \in \mathcal{NP}$,

(ii) minden $L \in \mathcal{NP}$ esetén $L \preceq_{\mathcal{P}} \widehat{L}$.

A fenti megállapodás egy alternatívája a $\preceq_{\mathcal{L}}$ redukcióval való dolgozás. Ez a szigorúbb értelmezés is igaz lesz a legtöbb későbbi $\mathcal{NP}$-teljességet igazoló redukciókra. Mi azonban csak a redukciók könnyebben ellenőrizhető polinom időbeliségét követeljük meg.

Definíció. Az L nyelv $\mathcal{NL}$-teljes, ha teljes $\mathcal{NL}$-ben az $\mathcal{L}$ redukcióra nézve.

Definíció. $\widehat{L}$ *nehéz* a $\mathcal{C}$ osztályra $\mathcal{R}$ bonyolultságú redukcióval, ha minden $L \in \mathcal{C}$ esetén $L \preceq_{\mathcal{R}} \widehat{L}$.

Azaz a nehézség a teljesség fogalma az (i) feltétel nélkül. Azaz nem követeljük meg az osztályhoz tartozást csak az osztály elemeinek visszavezethetőségét rá. Gyakran kiszámíthatósági feladatokra is mondják, ha $\mathcal{C}$-nehéz, ha van olyan redukciós algoritmus, amely által kiszámolt $\widetilde{\omega}$ karaktersorozatra a feladat olyan értéket ad, amiből ω nyelvhez tartozása könnyen látható.

Az előadás bevezető emlékeztetőjét az alábbi tétel az új fogalmak segítségével tömören fogalmaza meg.

4. Tétel. *ELÉRHETŐSÉG $\mathcal{NL}$-nehéz.*

Ennél több is mondható.

5. Tétel. *ELÉRHETŐSÉG $\mathcal{NL}$-teljes.*

Bizonyítás. 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 $3 \cdot \lceil \log_2 |V| \rceil$, amellyiben bináris ábécé-t használunk. Az is egyszerúen 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. ■

Egy később bizonyítandó tételt is megemlítünk.

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

6. Tétel. *HAMILTON egy $\mathcal{NP}$-teljes nyelv.*

Egy $\mathcal{C}$-teljes nyelv egy olyan nyelv, amely „a $\mathcal{C}$ osztált teljes nehézségét magában hordozza". Ha a teljes nyelvről mondunk valami nem-triviálisat, az az egész osztályról mondunk valamit. Ez általában nagyon meglepő. A fentiek alapján a HAMILTON nyelvről szóló tudásunk az egész $\mathcal{NP}$ nyelvről mond valamit. Speciálisan az alábbi $\mathcal{NP}$-beli nyelvről.

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.

Észrevétel. Ha az L nyelv $\mathcal{NP}$-teljes, és $L \in \mathcal{P}$, akkor $\mathcal{NP} \subseteq \mathcal{P}$, azaz $\mathcal{P} = \mathcal{NP}$.

Valóban, egy korábbi észrevétel ötletét kell megismételnünk: Legyen K egy tetszőleges $\mathcal{NP}$-beli nyelv. Legyen R egy redukáló algoritmus, ami $K \preceq_{\mathcal{P}} L$-et igazolja. Legyen M egy $L \in \mathcal{P}$-t igazoló algoritmus, azaz az L nyelvhez tartozás problémáját polinom időben eldöntő algoritmus. Ekkor K polinom időben eldönthető. ω inputon számoljuk ki $R(\omega)$-t majd futtasuk az L-et eldöntő M algoritmust $R(\omega)$-n. Ezzel a K-hoz tartozást döntjük el. Futási időnk $|R(\omega)|$ egy polinomja, ami egyben $|\omega|$ egy polinomja, hiszen a polinomok zártak a kompozícióra.

3. $\overrightarrow{st}$-ELÉRHETŐSÉG és $\mathcal{NL}$

Tudjuk, hogy az $\overrightarrow{st}$-ELÉRHETŐSÉG $\mathcal{NL}$-teljes. Így a róla szerzett tudásunk egész $\mathcal{NL}$-re kihat. Erre adunk két példát.

(1): ELÉRHETŐSÉG$\in \mathcal{P}$. A legtöbb bejárási algoritmus (például mélységi, szélességi keresés) irányított gráfokra vonatkozó változata polinomiális idejű megoldást ad. Ez alapján adódott, hogy $\mathcal{NL} \subset \mathcal{P}$.

(2): 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-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-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 ELVET.

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

7. Tétel (Savitch-tétel). *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 megvalósítást/implementációt nem végezzük el.

Az $\overrightarrow{st}$-ELÉRHETŐSÉG-ről tudtunk valami „okosat" mondani (Savitch-algoritmus/Savitch-tétel). Mi adódott $\mathcal{NL}$-re?

8. Következmény (Savitch-tétel). *(i)* $\mathcal{NL} \subseteq \mathcal{SPACE}(\log^2 n)$.

(ii) $\mathcal{NPSPACE} \subseteq \mathcal{PSPACE}$, *azaz* $\mathcal{PSPACE} = \mathcal{NPSPACE} = co\mathcal{NPSPACE}$.

Bizonyítás. (i) Legyen L egy tetszőleges nyelv $\mathcal{NL}$-ben. Vezessük vissza az ELÉRHETŐSÉG nyelvre. A visszavezetés által legyárott inputon futtassuk a Savitch-algoritmust. Az eredő algoritmusból kiküszöbölhető a legyárott input leírása (lásd például az $\mathcal{L}$ redukció tranzitivitásának indoklását). Így az eredő algoritmus (amely egy tetszőleges választott $\mathcal{NL}$-beli nyelvet dönt el) $\mathcal{O}(\log^2 n)$ tárigényű.

(ii) Vegyünk egy tetszőleges $L \in_T \mathcal{NPSPACE}$ nyelvet.

Végezzünk el egy redukciót a teljes nyelvre: Gyártsuk le a redukált konfigurációk $(\overrightarrow{G}_{\omega,T}, v_0, v_1)$ gráfját, mint az $\overrightarrow{st}$-ELÉRHETŐSÉG problémá egy inputját.

Alkalmazzuk a teljes nyelvről ismerteket: Az új probléma megoldását a Savitch-algoritmussal végezzük el.

A redukcióhoz polinomiális tár kell. A Savitch-algoritmus a gráf egy csúcshoz szükséges tárigény négyzetét igényli. $(\overrightarrow{G}_{\omega,T}, v_0, v_1)$ egy csúcsának kódolásához ω hosszában polinomiális mező kell. Polinom négyzete is polinom. A $(\overrightarrow{G}_{\omega,T}, v_0, v_1)$ kódjának felírása kiküszöbölhető az algoritmusból. Kapjuk a bizonyítandót. ■

Természetesen a polinomiális tárkorláttal azért tudtunk kényelmesen dolgozni, mert polinom négyzete is polinom. Általában is megfogalmazható a következő tétel:

9. Következmény $^+$**.** *Legyen S szép tárfüggvények egy osztálya, ami zárt a négyzetre emelésre. Ekkor* $\mathcal{NPSACE}(\cup_{s\in S} s(n)) = \mathcal{PSPACE}(\cup_{s\in S} s(n))$.

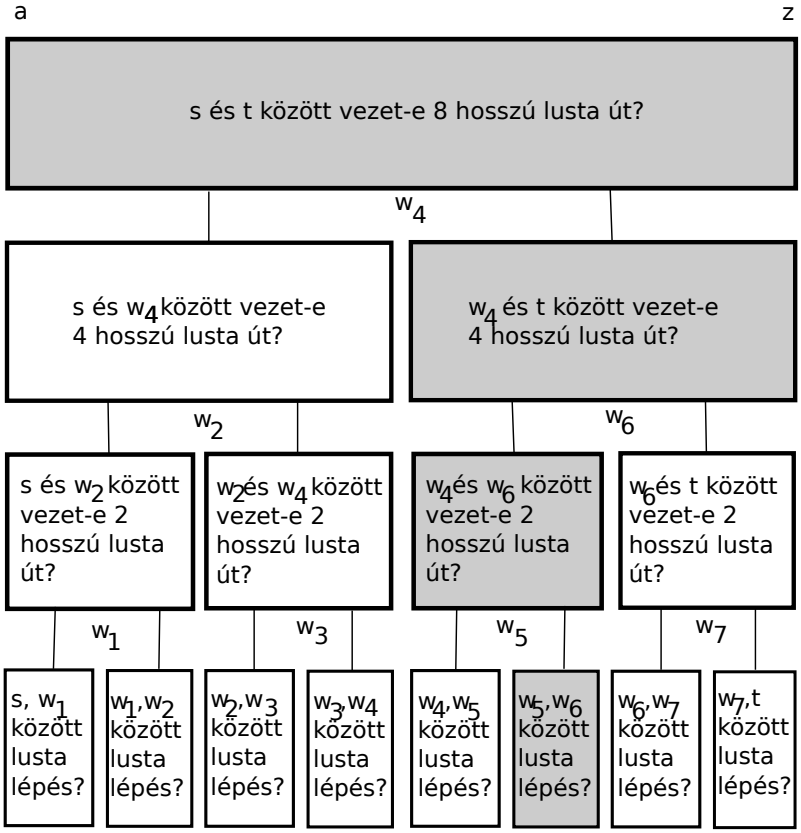
Speciálisan $\mathcal{NPSPACE}(\cup_{s\in S} s(n))$ *zárt a komplementálásra, azaz*

$$\mathcal{NPSPACE}(\cup_{s\in S} s(n)) = co\mathcal{NPSPACE}(\cup_{s\in S} s(n)).$$

Így speciálisan $\mathcal{EXPSPACE} = \mathcal{NEXPSPACE}$.

4. APPENDIX: Savitch-algoritmus implementációja Turing-gépen

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*.



1. 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 $\mathcal{O}(\ell \cdot \log n) = \mathcal{O}(\log^2 n)$.