ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

6. Előadás

Előadó: Hajnal Péter
Jegyzetelő: Hajnal Péter 2012. március 14.

1. $\overrightarrow{st}$-ELÉRHETŐSÉG $co\mathcal{NL}$-ben, Immerman-Szelepcsényi tétel

1. Tétel (Immerman (1988)—Szelepcsényi (1987)).

$$\overrightarrow{st}\text{-}ELÉRHETŐSÉG \in co\,\mathcal{NL}.$$

Megjegyzés. A tétel kifejtve a következőket jelenti. Ha egy rögzített kódolás mellett az input $\omega = \ulcorner(\overrightarrow{G}, a, z)\urcorner$, akkor létezik olyan logaritmikus tárigényű nemdeterminisztikus T Turing-gép, amelynek

- ha nem létezik $\overrightarrow{az}$ irányított út, akkor van elfogadó futása,
- ha létezik $\overrightarrow{az}$ irányított út, akkor minden futása elvető.

Észrevétel. Neil Immerman és Róbert Szelepcsényi egymástól függetlenül igazolták a tételt 1987-ben, Szelepcsényi mindössze 20 éves volt ekkor. A közismert bonyolultsági osztályok gyorsan kialakultak az 50-es, 60-as évek környékén, így ez a tétel viszonylag későinek mondható. A tétel — a bizonyításával együtt — egyszerűsége és fontossága miatt ma már standard tananyagnak számít.

Bizonyítás. A bizonyítás során mindig feltesszük, hogy $\overrightarrow{G}$ gráf az ω inputban kódolt gráf. Vezessük be a következő jelöléseket:

$$\begin{aligned} N_i &= \left\{ v \in V(\overrightarrow{G}) : a \overrightarrow{\sim}_{\leq i} v \right\}, \\ n_i &= |N_i|. \end{aligned}$$

Tehát N_i jelenti azoknak a csúcsoknak a halmazát, melyek a-ból legfeljebb i lépésben irányítottan elérhetők, n_i pedig az N_i számosságát jelöli. Érdemes észrevenni, hogy $N_0 = \{a\}$, $N_1 = \{a\} \cup \{a \text{ ki-szomszédai}\}$, $N_0 \subseteq N_1 \subseteq N_2 \subseteq \ldots \subseteq N_i \subseteq \ldots \subseteq V(G)$. $n_0 = 1$ illetve $n_0 \leq n_1 \leq n_2 \leq \ldots \leq n_i \leq \ldots \leq \left|V(\overrightarrow{G})\right| := n$.

A bizonyítás során az lesz a célunk, hogy az $n_0, n_1, n_2, \ldots$ értékeket kiszámoljuk. A következő észrevétel alapján ez elegendő célunkhoz.

Észrevétel. Akkor és csak akkor nincs $\overrightarrow{G}$-ben egy irányított az út, ha létezik $i \leq |V(G)|$, hogy $n_i = n_{i+1}$ és $z \notin N_i$.

Ez azzal indokolható, hogy ha valamely i-re $n_i = n_{i+1}$, akkor $N_i = N_{i+1}$ és az egyenlőség az $(i+1)$-nél nagyobb indexekre is fennáll, így az N_i az összes a-ból elérhető csúcsot felsorolja. Ilyen $i \leq |V(G)|$ mindig létezik. Az irányított út létezése ekvivalens azzal, hogy z eleme-e N_i-nek.

A bizonyítás során egy nem-determinisztikus rekurzív algoritmust fogunk felírni. A rekurzió feltétele, hogy ismerjük n_i-t. Az algoritmus segítségével n_i-ből meghatározzuk n_{i+1}-et és képesek leszünk N_i elemeinek felsorolására. Ez elegendő lesz számunkra: az ELFOGAD állapot elérése ekvivalens lesz azzal, hogy valamely $i \leq |V(G)|$ esetén n_i megegyezik n_{i+1}-gyel és $z \notin N_i$.

A bizonyítás során szükségünk lesz négy darab munkaszalagtöredékre. Gondolhatunk rájuk úgy is, mint ugyanannak a munkaszalagnak a különböző, de jól meghatározott része, vagy úgy is, mint külön munkaszalagok (ekkor többszalagos Turing-gép modellel dolgozunk).

(1)

$\triangleright$	i	:	n_i	$\triangleleft$

Az (1) munkaszalagtöredéken tároljuk a megfelelő indexekhez tartozó n_i-ket, és biztosnak kell lennünk abban, hogy $n_i = |N_i|$. i és n_i területe is $\mathcal{O}(\log n)$ mezőnyi.

A következő (2)-es munkaszalagok az N_i-beli elemeket sorolják föl.

(2a)

$\triangleright$	N_i-beli elemek helye	:	számláló	$\triangleleft$	...

$\lceil \log V \rceil$, azaz egy csúcsnyi hely

(2b)

$\triangleright$	bizonyító séta csúcsa	következő csúcs	megtett lépések száma	...

$\lceil \log V \rceil$, azaz egy csúcsnyi hely $\lceil \log V \rceil$, azaz egy csúcsnyi hely

A (3) munkaszalagok V tesztelésére szolgálnak, egy V-beli csúcs N_{i+1}-hez tartozását vizsgálja.

(3a)

$\triangleright$	tesztelt csúcs	$\triangleleft$

csúcsnyi hely

(3b)

$\triangleright$	tesztelő séta csúcsa	következő csúcs	megtett lépések száma	...

$\lceil \log V \rceil$, azaz egy csúcsnyi hely $\lceil \log V \rceil$, azaz egy csúcsnyi hely

A (4) pedig a megtalált N_{i+1}-beli elemeket számolja.

(4)

$\triangleright$		$\triangleleft$

$\log |V|$

A (1)-es szalagon kezdetben $0:1$ van, a többi szalag pedig üres. $|N_0| = |\{a\}| = 1$ alapján kiinduló hipotézizunk helyes. A szalagtartalom megbízható.

A rekurzív lépés (n_i-re alapulva, felsoroljuk N_i elemeit, kiszámoljuk n_{i+1}-et) a következő.

1. Felsoroljuk az N_i elemeit, azaz a $(2a)$ szalagon a $v_1, v_2, \ldots, v_{n_i}$ csúcsok sorolódnak. Természetesen oly módon, hogy v_1 leíródik (számláló értéke 1), azt v_2 felülírja (számláló eggeyle növelődik), és ez így megy tovább, míg végül v_{n_i} lesz ráírva (számláló eléri n_i-t).

   Az algoritmusnak ez a része egy nem-determinisztikus fázis. Az összes csúcs „tippelt", nem-determinisztikus lépés eredménye. Erre a felsorolásra úgy is lehet majd gondolni, mint egy szubrutin. Ismételten alkalmazzuk (természetesen ugyanazon tárterület felhasználásával).

   - Azért, hogy elkerüljük, hogy egy csúcsot többször is felsoroljunk, feltesszük, hogy a csúcsok kódjai az indexekkel növekednek, azaz $\ulcorner v_1 \urcorner < \ulcorner v_2 \urcorner < \ldots$
   - Mielőtt felsorolnánk a következő csúcsot, a $(2b)$ részben minden v_j-re legyártunk egy bizonyítást arra, hogy $v_j \in N_i$. Ez is alapvetően nem-determinisztikus része az algoritmusnak. Ezért ha $v_j \in N_i$, akkor van olyan futás/tanúszalag-tartalom, ami ezt bizonyítja, és ezt logtárral le tudjuk ellenőrizni a (2)-es szalag bizonyító részében, mivel a probléma $\mathcal{NL}$-be tartozását tudjuk. A bizonyító rész tartalma a következőképpen alakul: Kezdetben a-ból átlépünk egy szomszédos a^+ csúcsba, majd növeljük a számlálót. Ezután az a felülíródik a szomszédjával (a^+-szal) és most annak egy szomszédja lesz a második csúcs (a^{++}), ami pedig az a^+ csúcsot írja felül. Ez a művelet addig ismétlődik, míg a második csúcs v_j nem lesz vagy a számláló túlcsordul (nagyobb lesz, mint i). Két állapot léphet fel: STIMMEL, illetve NEM-STIMMEL. A STIMMEL állapotban elérjük v_j-t a számláló túlcsordulása nélkül. A komplementer NEM-STIMMEL állapot ekvivalens azzal, hogy elvetjük az egész futást. Ha NEM-STIMMEL állapot nélkül a felsorolás számlálója n_i értéket vesz fel, akkor SIKERES-FELSOROLÁS állapotba jutunk.

     Ha feltevésünk helyes (ha (1) tartalma valóban n_i), továbbá egy „jogkövető" futásról van szó akkor az N_i elemeit tényleg fel tudjuk sorolni.

   2. Lemma. *Ha végigfutunk a SIKERES-FELSOROLÁS állapotba kerülünk, akkor biztosak lehetünk abban, hogy N_i elemeit soroltuk fel.*

   Bizonyítás. Mivel elkerültük a NEM-STIMMEL állapotot, ezért bizonyítottuk, hogy n_i darab különböző elemet soroltunk fel N_i-ből. Mivel n_i-ben megbízunk, így N_i összes elemét felsoroltuk. ■

2. Az algoritmus következő fázisa a V tesztelése, azaz a (3)-as szalagon felsoroljuk V összes elemét, és ellenőrizzük, hogy benne vannak-e N_{i+1}-ben. Legyenek az $u_1, \ldots, u_n$ csúcsok a V elemei. Kezdetben a (4) szalag tartalma 0.

   - Minden u_j csúcs esetében meghívjuk a fenti szubrutint, azaz (nem-determinisztikusan) felsoroljuk az N_i elemeit.
   - Azt teszteljük, hogy az aktuális u_j az fel van-e sorolva N_i-ben, vagy kiszomszédja egy olyan csúcsnak, ami fel van sorolva. Ha ez megtörténik, akkor a (4)-es szalag(töredéken) növeljük eggyel a számlálót és a következő csúcsra térünk át. Ha az N_i felsorolása SIKERES-FELSOROLÁS

állapotba ér, az aktuális V-beli elem nem lett felismerve mint N_{i+1} egy eleme, akkor (a számláló növelése nélkül) a következő csúcsra térünk át.

A következő lemma eddigi észrevételeinket foglalja össze.

3. Lemma. *(i) $x \in N_{i+1}$ akkor és csak akkor, ha $x \in N_i$ vagy alkalmas $y \in N_i$ esetén: $\overrightarrow{yx} \in E$.*

(ii) Ha V összes elemének tesztelése megtörténik a NEM-STIMMEL állapot elkerülésével, akkor a (4) szalag tartalma biztos n_{i+1}

Az előző lemma tulajdonképpen egy trivialitás, a bizonyításához elég egyszerűen meggondolni azt, hogy mit jelent. Ez a lemma biztosítja a rekurziós lépést, és így a rekurzív lépés leírásának vége van.

3. Már csak annak az ellenőrzése van hátra, hogy n_i megegyezik-e n_{i+1}-gyel.

   - Ha nem, akkor az (1)-es szalagon i-t kicseréljük $(i+1)$-re és n_i-t kicseréljük a (4)-es szalagon lévő számra, ami az előző lemma szerint pontosan az n_{i+1}, és visszatérünk az algoritmus 1. részéhez.
   - Ha igen, akkor újból felsoroljuk N_i elemeit, és csak azt vizsgáljuk, hogy z előjön-e közöttük. Ha nincs NEM-STIMMEL állapot a felsorolás során és a z sem jön elő, akkor tudjuk, hogy nem létezik $\overrightarrow{az}$ út, így ELFOGAD állapottal leállunk.

Az algoritmus logaritmikustárat használ fel, a helyes működése a fentiekben bizonyítva lett, ezért az Immerman-Szelepcsényi tétel bizonyítása kész. ■

4. Következmény. $\mathcal{NL} = co\,\mathcal{NL}$

Bizonyítás. Ha L egy $\mathcal{NL}$-beli nyelv, akkor létezik egy $L \prec_{\mathcal{L}} \overrightarrow{st}$-ELÉRHETŐSÉG visszavezetés (mivel az $\overrightarrow{st}$-ELÉRHETŐSÉG $\mathcal{NL}$-teljes), így tudjuk, hogy az L nyelv $co\,\mathcal{NL}$-ben is benne van. Az $L \in co\,\mathcal{NL}$ pontosan azt jelenti, hogy L komplementere benne van $\mathcal{NL}$-ben, azaz $\overline{L} \in \mathcal{NL}$. Ezt a gondolatmenetet megismételve $\overline{L}$-re, azt kapjuk, hogy az $\overline{L} \in \mathcal{NL}$-ből következik, hogy $L \in \mathcal{NL}$. Ebből a két megállapításból pedig azt kapjuk, hogy L akkor és csak akkor van $\mathcal{NL}$-ben, ha a komplementere is ott van, és ez azt jelenti, hogy $\mathcal{NL} = co\,\mathcal{NL}$. ■

Ahogy a bizonyítás is mutatta a tétel lényege, hogy $\mathcal{NL}$ zárt a komplementálásra nézve.

5. Következmény. *Ha $s(n)$ szép tárfüggvény, akkor*

$$\mathcal{NSPACE}(\mathcal{O}(s(n))) = co\,\mathcal{NSPACE}(\mathcal{O}(s(n))).$$

Bizonyítás. A bizonyítás hasonlóan történhet, mint az Immerman-Szelepcsényi tétel bizonyítása, csak itt a blokkok nagyobbak. ■

Megjegyzés. Az eredeti tétel, illetve a két következmény közül mindegyiket szokták Immerman-Szelepcsényi-tételnek nevezni.

ELÉRHETŐSÉG az eddig vizsgált probléma változata irányítatlan gráfokra. A következő tétel azt mutatja, hogy az irányítatlan eset valószínűleg könnyebb az irányítottnál.

6. Tétel (Omer Reingold, 2004). *ELÉRHETŐSÉG* $\in \mathcal{L}$.

Attól függetlenül, hogy klasszikus szélességi és mélységi keresések a problémát $\mathcal{P}$-be rakják, ez egy nagyon nehéz tétel. Omer Reingold bizonyította 2004-ben. Maga az algoritmus nem túl használható, de bonyolultságelméleti szempontból nagy jelentőséggel bír. Viszont az $\overrightarrow{st}$-ELÉRHETŐSÉG és az $\mathcal{L}$ osztály viszonyáról még semmit sem tudunk. Az $\overrightarrow{st}$-ELÉRHETŐSÉG $\in \mathcal{L}$ bizonyítása azt jelentené, hogy $\mathcal{L} = \mathcal{NL}$.

2. Hálózatok és $\mathcal{P}$

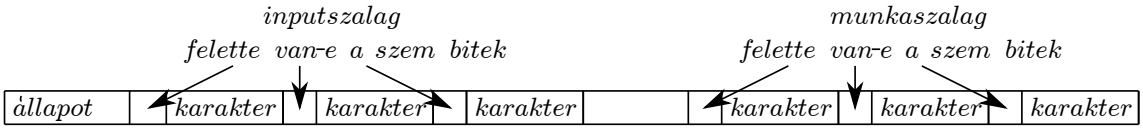
Emlékeztető. $L \in_T \mathcal{TIME}(t(n))/\mathcal{NTIME}(t(n))$ esetén mindig feltesszük, hogy $t(n)$ szép idő függvény, azaz Turing-géppel megvalósítható egy óra, ami n hosszú inputra „$t(n)$ idő után üt".

$\omega \in \Sigma^n$ inputon T futása a

$$\kappa_0(\omega) \to \kappa_1 \to \kappa_2 \to \ldots \to \kappa_\ell,$$

konfigurációsorozat, ahol $\kappa_0(\omega)$ az ω-hoz tartozó kiinduló konfiguráció, κ_{i+1} a κ_i konfiguráció rákövetkezője, és κ_ℓ az első olyan konfiguráció, ahol az állapot ELFOGAD vagy ELVET. Feltehető, hogy $\ell = t(n)$.

Észrevétel 1. κ konfigurációk kódolhatók bitsorozatokkal.



1. ábra.

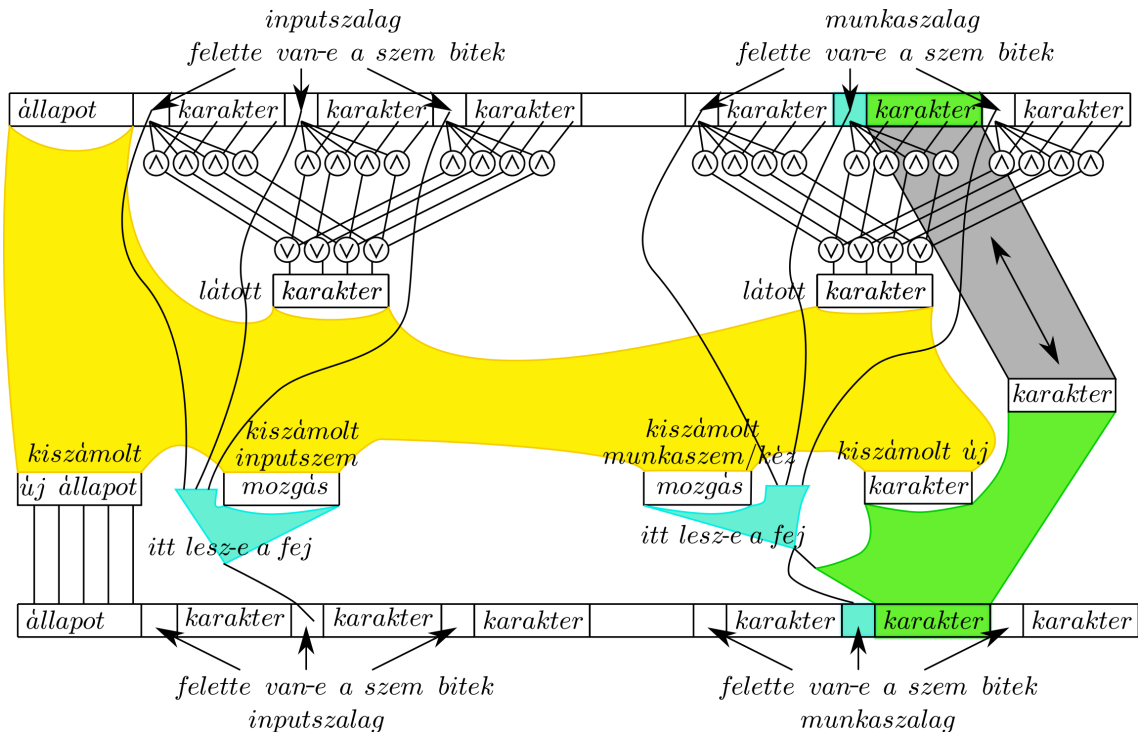
A fenti ábra (megjegyzéseivel) magáért beszél. A fej pozícióját is kell kódolnunk. Ezt „szétszórtuk", minden mező raktunk egy bitet. Így nem miden adott hosszúságú sorozat kódol konfigurációt. Az állapotot és karaktereket kódoló blokkkok hossza függ $|S|$, $|\Sigma|$ és $|\Gamma|$ értékétől. Minden estre konstans sok bit szükséges (ami a Turing-géptől függ). S elemeinek kódolására $\lceil \log_2 |S| \rceil$ bit legendő. Ha $|S|$ nem kettőhatvány, akkor lesznek olyan 0-1 bitsorozatok, amik a megfelelő pozíciókban állnak, de nem kódolnak állapotot. Ennek ellenére könnyű tervezni egy olyan tesztelő hálózatot, ami egy adott (megfelelő hosszú) kódról eldönti, hogy konfigurációt kódol-e.

Megjegyzés (FONTOS). A megállapodát úgy választhatjuk adott n hosszú ω input esetén $\lceil \omega \rceil$ hossza $\alpha_T \cdot n$ legyen. Ha T időigénye legfeljebb $t(n)$, akkor konfigurációk kódjának hossza $\beta_T \cdot t(n)$ legyen. (A konfigurációban a munkaszalag elvágható az első $t(n)$ mező után. Az elhagyott/levágott rész csak érintetlen mezőket tartalmaz.)

A továbbiakban n és a kódolási megállapodás mindig rögzített (ennek megfelelően a megfelelő kódok hossza mindig ismert).

Észrevétel 2. $\lceil \kappa_i \rceil \to \lceil \kappa_i^+ \rceil = \lceil \kappa_{i+1} \rceil$ egyszerű hozzárendelés/függvény.

Az észrevétel állítása matematikailag nem pontos, az „egyszerű" jelző értelmezése nem jól definiált. Ennek ellenére mutatunk egy módot arra, hogy egy konfigurációt kódoló bitsorozatból a rákövetkező konfigurációt kódoló bitsorozat hogyan számolható ki. A konstrukciónk egyszerű, de sok esetlegességet, megállapodást követel. Egy formális leírás helyett egy példán szemléltetjük milyen ötletek vezethetnek el egy megoldáshoz.



2. ábra. ábra

Egy mezőnél a szem/kéz-ott-van-e bitet és a mező tartalmát kódoló bitek mindegyikét össze-és-eljük. A kapott bitsorozat vagy a csupa 0 (ha nincs ott a szem/kéz, mindent 0-val és-eltünk) vagy a látott karakter kódja (ha ott a szem/kéz). Az inputszalag mezőinél így kapott bitsorozatok mindegyikének első, majd második és így tovább karakterét össze-vagy-olva kapjuk az inputszalagon látott karakter kódját. Hasonlóan számolhatunk a munkaszalagnál.

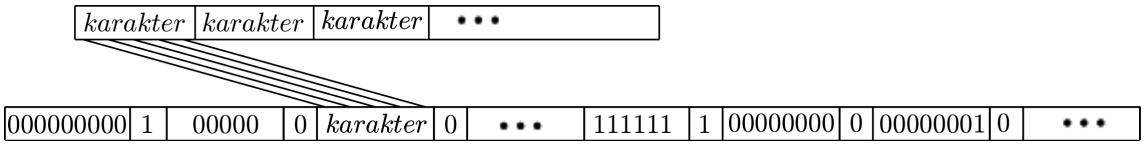
A sárga területben egy összetettebb számolás történik: konstans sok bitból számolunk ki konstans sok bitet (a konstanok függnek a Turing-géptől). Konkrét kidolgozása az átmenetifüggvénytől függ. Nem okozhat problémát akkor sem, ha az egyes bitek függését a nyilvánvaló DNF formula alapján írjuk fel. Ekkor is konstans sok kapuval dolgozva megoldjuk a feladatunkat.

A világos kék területen számolódik kis a fej pozícióját leíró egyik bit. Ez attól függ, hogy a fej ott volt-e vagy valamelyik szomszéd felett állt, illetve merre írja elő mozgását az átmeneti függvény. A hálózatunk ezen részét konkrétan felírhatnánk, de ez a munka felesleges az előző megjegyzésünk alapján. Ez a kék rész minden fej-pozíció-bithez ott van. Az átláthatóság kedvéért rajzoltuk fel csak egyszer az input- és egyszer A munkaszalaghoz. A zöld területen számoljuk ki a munkaszalag új tartalmát. Minden munkaszalag mezőhöz tartozik egy ilyen zöld blokk (egyszerűség kedvéért csak egyet tüntettünk fel). Az új karaktertől, a régitől és attól az

információtól függ, hogy a fej ott áll-e. Ez a rész is könnyen megvalósítható lenne, ha tudnánk Γ elemeinek kódolásához használt bitek ℓ számát ismernénk. A zöld területen az $f(\epsilon, k_0, k_1) = k_\epsilon$ függvényt számoljuk ki, ami $1 + 2\ell$ bitből számol ki ℓ-et.

Észrevétel 3. $\lceil \omega \rceil \to \lceil \kappa_0(\omega) \rceil$ egy egyszerű hozzárendelés/függvény.

Ez a korábbiaknál egyszerűbb észrevétel. Ismét egy ábrára utalunk.



3. ábra.

Feltettük, hogy a START állapot kódja $00\ldots0$ (hossza $\lceil \log_2 |S| \rceil$, példánkban 9). Az inputszalagon $\triangleright$ kódja $00\ldots0$, a $\triangleleft$ kódja $11\ldots1$ (hosszaik $\lceil \log_2 |\Sigma| \rceil$, példánkban 5). Az munkaszalagon $\triangleright$ kódja $0\ldots00$, a szűzkarakter kódja $0\ldots01$, (hossza $\lceil \log_2 |\Gamma| \rceil$, példánkban 8).