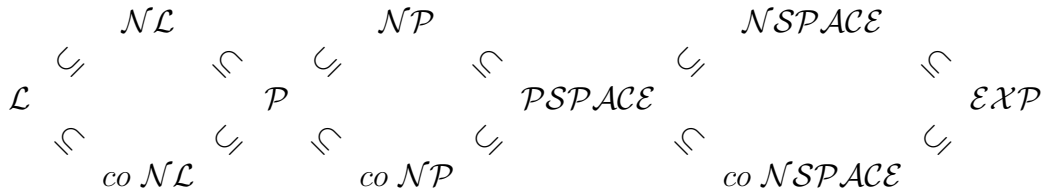
ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

5. Előadás

Előadó: Hajnal Péter
Jegyzetelő: Szőri András 2011. március 1.

Emlékeztető.



A továbbiakban konkrét döntési problémákat, nyelveket helyezünk el a fenti hierarchiában.

1. Példák

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

Példa. Legyen $|V(G)| = n$ az input gráf csúcsszáma. $\lceil G\rceil$, a G gráf kódolt alakjának hossza polinomiális n-ben: A szomszédsági mátrixszal történő kódolás esetén az input kódjának ($\{0,1\}$ ábécé-t használva) hossza n^2. 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 szomszédja 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 kevesebb mint n^3 karakter.

Mi csak a nagyságrendeket nézzük. Ezek szempontjából n és n^3 ugyanaz. Általában egy gráfra úgy gondolunk mintha méretét a csúcsszám adná.

- Teszteljük a $V(G)$ összes lehetséges sorbaállítását. Ez összesen n! (kevesebb mint 2^{n^2}) lehetőség. Ha bármelyik egy Hamilton-kör csúcslátogatási sorrendje, akkor ELFOGAD állapottal leállunk. Ha az összes lehetőség végigvizsgálata során nem találtunk jó sorrendet, akkor ELVET állpottal álunk le. Így $HAMILTON \in \mathcal{EXP}$.

- Ha a naív algoritmus tárigényére koncentrálunk, akkor észrevehetjük, hogy az egyes próbálkozásoknál használt tárnak nem kell mindig újnak lenni. Egy új próbánál a korábbi munkaszalag szakasz tartalmát „felülírhatjuk" az új próbahoz szükséges információkkal. Így kapjuk $HAMILTON \in \mathcal{PSPACE}$.

- $HAMILTON \in \mathcal{NP}$: Egy egyszerű tanúszalagos Turing-gépet készítünk állításunk igazolására. A tanúszalag tartalma egy lehetséges sorbarendezése a csúcsoknak. A gép teszteli, hogy tényleg minden csúcs pontosan egyszer van felsorolva a szalagon és az egymást követő csúcsok valóban szomszédosak, továbbá a lista első és utolsó csúcsa is összekötött. Ha mindegyik ellenőrizendő teljesül, akor a gép ELFOGAD, különben ELVET állapottal áll le.

- Hogy a HAMILTON benne van-e *co* $\mathcal{NP}$-ben vagy $\mathcal{P}$-ben az ismeretlen.

Definíció. TELJES-PÁROSÍTÁS $= \{\lceil G \rceil : G$-nek létezik teljes párosítása$\}$

Példa. A HAMILTON nyelvhez hasonlóan naív/különösebb gondolat nélküli megoldásokkal kezdünk. Ekkor „puszta erővel", az összes lehetőség vizsgálatával döntjük el a problémát.

- Ellenőrizzük, hogy páros sok csúcsunk van-e. Ha nem, akkor ELVET állpottal leállunk. Ha igen, akkor az $|V| := 2n$ csúcs összes párbaállítását végigvesszük ($n!! = n(n-2)(n-4)(n-6)\ldots < n!$ darab) és az összes párra ellenőrizzük, hogy összekötöttek-e. Ha tesztünk valamikor stimmel ez akkor ELFOGAD állapottal, ha sose stimmel, akkor ELVET állapottal állunk le.

- A fentihez hasonlóan belátható, hogy TELJES-PÁROSÍTÁS$\in \mathcal{PSPACE}$.

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

- A HAMILTON-nal ellentétben TELJES-PÁROSÍTÁS viszonyát ismerjük a *co* $\mathcal{NP}$ nyelvosztályhoz: TELJES-PÁROSÍTÁS$\in$ *co* $\mathcal{NP}$ Az indoklás lényege a Tutte-tétel: Ha G-ben nincs teljes párosítás, az ekvivalens azzal, hogy G-ben van Tutte-akadály.

  Vegyük a következő nemdeterminisztikus Turing-gépet: A tanúszalagra egy lehetséges T akadályt írunk. A gép ellenőrzi, hogy ez valóban akadály: $G - T$ komponenseit meghatározza, mindegyik komponensnek megszámolja csúcsait és a páratlan elemszámú komponensék számát összehasonlítja $|T|$-mal. Ha a páratlan pontszámú komponensek száma a nagyobb, akkor ELFOGAD, különben ELVET. Tutte-tétele éppen azt állítja, hogy pontosan azon gráfokra vena elfogadó futás (azokat fogadjuk el nem-determinisztikus értelemben), amelyek nem tartoznak a TELJES-PÁROSÍTÁS nyelvhez.

- Az Edmonds-algoritmus Turing-gépen történő megvalósítás és analízise adja, hogy TELJES-PÁROSÍTÁS$\in \mathcal{P}$.

Definíció. PÁROSÍTÁS$= \{\lceil (G,k) \rceil : G$-nek létezik k elemű párosítása$\}$.

A párosítások kérdése természetes módon egy optimalizáási problémához vezet. Azaz minden G gráf esetén szeretnénk tudni/kiszámítani a legnagyobb párosításának méretét. Esetleg szeretnénk meghatározni egy legnagyobb méretű párosítást. A fenti döntési feladat mindkettő lényegét megfogja. Ha azt hatékonyan meg tudjuk oldani, akkor a legnagyobb párosítás mérete egyszerű bináris kereséssel adódik. Ha a legnagyobb párosítás méretét meg tudjuk határozni, akkor addig hagyhatunk el éleket egy gráfból míg ez a paraméter (értéke legyen ℓ) nem változik. Ha már nem tudunk tovább éleket elhagyni a paraméter változtatása nélkül, akkor gráfunk éppen ℓ független él, egy optimális párosítás az eredeti gráfban.

Példa. A PÁROSÍTÁS nyelvre is felhasználhatóak a fent felsorolt érvek és módszerek (Edmonds-algoritmussal bezárólag). A Tutte-tételnél a Berge-formula alkalmasabb a *co* $\mathcal{NP}$-hez tartozás igazolásához. Kapjuk, hogy PÁROSÍTÁS$\in \mathcal{EXP}$, $\mathcal{PSPACE}$, $\mathcal{NP}$, *co*$\mathcal{NP}$, $\mathcal{P}$.

Definíció. Legyen $V = \{x_1, \ldots, x_n\}$ egy változó halmaz. A literálok halmaza: $L = V \cup \{\overline{x}_1, \ldots, \overline{x}_n\} = V \cup \overline{V}$. Egy C diszjunktív klóz: $C \subset L$. Például $\{x_1, x_2, \overline{x_5}\}$, amit általában diszjunkcióval összekapcsolva írunk fel: $C = x_1 \vee x_2 \vee \overline{x_5}$. Egy φ konjunktív normálformájú formula (röviden φ egy CNF), ha diszjunktív klózok egy halmaza. A klózokat általában konjunkcióval kötjük össze. Egy példa CNF formulára:

$$\varphi = (x_1 \vee x_2 \vee \overline{x_5}) \wedge (x_1 \vee \overline{x_6} \vee \overline{x_7}) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_5 \vee x_6 \vee x_7).$$

$\kappa : V \to \{0, 1\}$ a változók egy kiértékelése. Ez könnyen kiterjeszthető a literálok egy kiértékelésére, diszjunktív klózok kiértékelésére (egy klóz értéke 1, ha valamelyik literáljának 1 az értéke), majd CNF formulák kiértékelésére (egy CNF formula igaz értékű, ha mindegyik klóza igazzá értékelődik). Igazából tetszőleges ítéletkalkulusbeli formula kiértékelését definiálhatjuk a fenti módon.

φ kielégíthető, ha van olyan $V \to \{0, 1\}$ kiértékelése a változóknak, ami φ-t igazzá teszi (értéke 1 lesz).

Definíció. SAT$= \{\lceil\varphi\rceil : \varphi$ kielégíthető$\}$.

Példa. Csak egy állítást igazolunk: SAT$\in \mathcal{NP}$. Valóban: Legyen egy nem-determinisztikus Turing-gép tanúszalagjának tartalma egy $V \to \{0, 1\}$ kiértékelés. A gép terjessze ezt ki φ-re. Ha az 1 értéket kapja ELFOGAD, ha nem, akkor ELVET állpottal álljon le. A fenti gép nyilván polinomiális és az állítást igazolja.

Definíció. LP-SAT a következő nyelv

$$\{\lceil(A, b)\rceil : A \in \mathbb{Z}^{\ell \times n}, b \in \mathbb{R}^{\ell}, \text{van olyan } x \in \mathbb{R}^N, \text{ hogy } Ax \preceq b \text{ teljesüljön}, \}$$

ahol $y \preceq y'$ azt jelenti, hogy az n dimenziós vektrok mindegyik komponensében y' legalább olyan nagy mint y.

Példa. Az LP-SAT probléma a lineáris programozás optimalizálási feladatát megfogó döntési feladat.

- LP-SAT$\in \mathcal{NP}$. A nyelvhez tartozást egy lineáris egyenlőtlenségrendsszer megoldhatósága adja. Könnyű látni, hogy ha az egyenletrendszer megoldható, akkor van racionális megoldás is, azaz kódolható megoldás. Az is nyilvánvaló, ha a tanúszalag egy lehetséges x_0 megoldást tartalmaz, akkor a gép ellenőrizheti, hogy $Ax \preceq b$ teljesül-e. Ennek időigénye nyilván polinomiális $\lceil(A, b, x_0)\rceil$ hosszában. Ez azonban nekünk kevés. Az kell, hogy ez ellenőrizhető legyen $\lceil(A, b)\rceil$ hosszának polinomja által korlátolt időben. Azaz, ha lineáris egyenletrendszerünk megoldható, akkor van az input hosszában kezelhető méretű megoldás is. Ez igaz, bizonyítás nélkül a következő lemmában foglaljuk össze a szükséges állítást.

  1. Lemma. *Ha létezik megoldás, azaz jó tanú, akkor létezik olyan is, amely* $\lceil(A, b)\rceil$ *hosszában legfeljebb polinomiális méretű.*

- LP-SAT $\in co\,\mathcal{NP}$. Ez a Farkas-lemmából következik.

  2. Lemma (Farkas-lemma). *Ha A egy $\ell \times n$-es mátrix, és $b \in \mathbb{R}^\ell$, akkor az $Ax \preceq b$ egyenletrendszer megoldhatatlansága esetén van olyan $y \in \mathbb{R}^\ell_{\geq 0}$, hogy $y^T A \succeq 0$ és $y^T b = -1$.*

  Az egyik irány nyilvánvaló: Megfelelő y nem-negatív súlyokat ad, amelyekkel súlyozva a kiinduló egyenletrendszerünket kapjuk, hogy $0 \leq -1$. Ha lenne megoldás, akkor ellentmondást vezettünk volna le, azaz egyenletrendszerünk nem megoldható. A Farkas-lemma nehéz része a fordított irány.

  A lemma alapján könnyen adódik a $co\,\mathcal{NP}$-hez tartozás: Egy nem-determinisztikus-gép a Farkas-lemmabeli y-t írja a tanúszalagjára és ellenőrzi a lemma állítását. Amennyiben van ELFOGADÓ futás kapjuk, hogy egyenletrendszerünk nem megoldható, azaz LP-SAT nyelv komplementeréből van.

- LP-SAT$\in \mathcal{P}$. A gyakorlatban nagyon jól mükődő szimplex módszer ezt nem bizonyítja. Azonban Khachian ellipszoid módszere ezt igazolja. Az eredmény 1979-ben született, amikor is a New York Times címlapjára került.

Definíció. PRÍM-TESZT$= \{\lceil n \rceil : n \text{ prím}\}$.

Példa.
- PRÍM-TESZT$\in co\,\mathcal{NP}$ egyszerű: egy nem-determinisztikus Turing-gép egy m osztót írjon a tanúszalagra. Ha a tanúszalag tartalmaz 2 és $n-1$ között van és osztja n-et, akkor ELFOGAD állpottal állunk le, különben ELVET állapottal. Gépünk nyilván polinomiális és az összetett számokat fogadja el (PRÍM-TESZT komplementer halmaza).

- PRÍM-TESZT$\in \mathcal{NP}$ már nem olyan egyszerű, Pratt igazolta 1975-ben.

- PRÍM-TESZT$\in \mathcal{P}$ az Agrawal—Kayal—Saxena-algoritmus egy következménye.

Megjegyzés. A PRÍM-TESZT nyelv nem fogja meg a prím tényezős felbontást megkeresését kitűző számítási feladatot. Az erre szolgáló döntési feladat: FAKTORIZÁCIÓ$= \{\lceil (n,t) \rceil : n$-nek van t-nél kisebb valódi osztója$\}$. Erről nem ismert, hogy $\mathcal{P}$-hez tartozik-e. Általában nehéznek gondolják. Korunk több titkosító algoritmusa esetén az, hogy megbízunk benne, ezen a hiten alapul.

Definíció. ELÉRHETŐSÉG$= \{\lceil (\overrightarrow{G}, s, t) \rceil :$ létezik st irányított út $\overrightarrow{G}$-ben$\}$.

Példa. A fenti nyelv nagyon sok helyen nagyon fontos szerepet játszik (egyszerűsége ellenére). Lássuk az alapalgoritmusokat és az általuk bizonyított hozzátartozást egy bonyolultsági osztályhoz.

- ELÉRHETŐSÉG$\in \mathcal{P}$. A legtöbb bejárási algoritmus (például mélységi, szélességi keresés) irányított gráfokra vonatkozó változata polinomiális idejű megoldást ad.

- 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
2. a számlálót 0-ra állítja
3. a mögé bemásolja v-t, a következőnek elért csúcsot a munkaszalagra
4. ellenőrzi, hogy létezik-e $\overrightarrow{av}$ él. Amennyiben nem, úgy NEM STIMMEL eredményt, ad, ha igen, akkor az a 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 nagyobb mint n, akkor megnézi, hogy a eredeti helyén álló csúcs z-e. Ha igen, akkor ELFOGAD állapottal leáll. Ha nem, akkor megismétli a harmadik lépést.

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.

- 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.

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

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

  ha NEM, akkor következő k

  ha IGEN, akkor

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

  ha IGEN, akkor ELFOGAD állpot és leáll

  ha NEM, akkor következő k

  Ha V kimerül, akkor ELVET állapottal leáll.

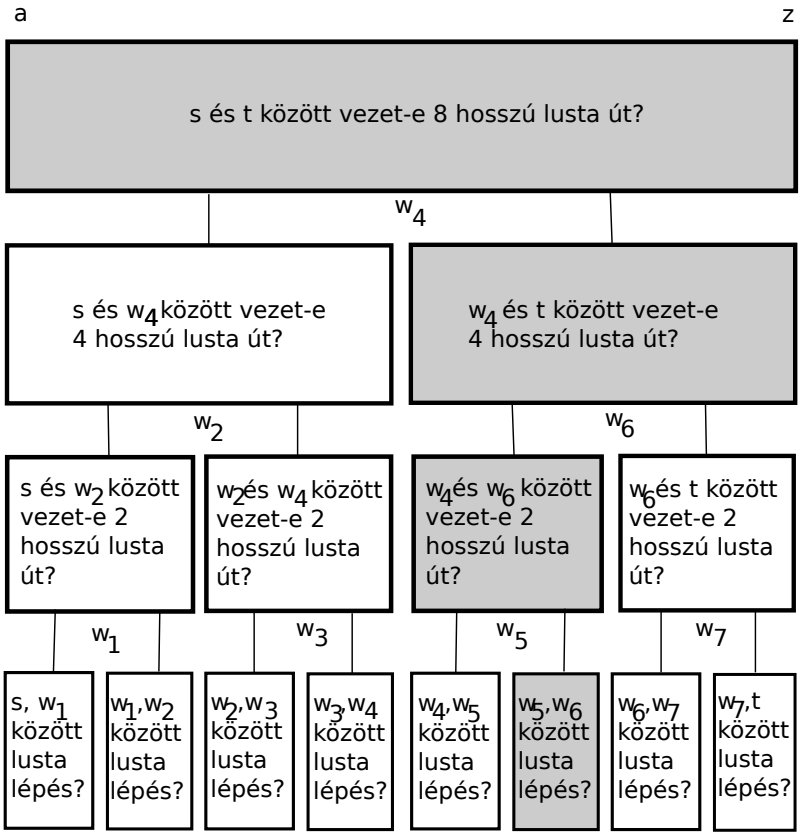
A fenti algoritmus $\ell = \lceil |V(G)| \rceil$ paraméterrel futtatva megoldja az elérhetőséget. Az algoritmus a következő tételle Savitch nevéhez fűzhető.

3. 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 nem végezzük el. A tétel azonban adja a bizonyítandót.

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