ALGORITMUSOK ÉS BONYOLULTSÁGELMÉLET

Matematika MSc hallgatók számára

1. Előadás

Előadó: Hajnal Péter
Jegyzetelő: Hajnal Péter 2012. február 8.

1. Az algoritmus naív fogalma

Az algoritmus egy eljárás, ami az adatok megkapása után egy jól definiált lépéssort elvégezve megadja a probléma megoldását.

Az algoritmus fogalmával együtt kialakult egy az algoritmusokkal kapcsolatos nyelvezet is. Az adatokat *inputnak*, az eredményt *outputnak* nevezzük. Ha adott inputon az algoritmus utasításait követve végezzük az előírt lépéseket, akkor az *algoritmus futásáról* beszélünk.

Már az általános iskolában tanulunk algoritmusokat. Az alapműveletek elvégezésének szokásos módja is egy-egy algoritmus. Az alapszerkesztések, a prímtényezőkre bontás megtanított módja, az Euklideszi-algoritmus mind jól ismertnek kell lenni egy érettségizett számára.

Legyen $\mathcal{I}$ az inputok, $\mathcal{O}$ az outputok halmaza. Tehát a probléma egy $f : \mathcal{I} \to \mathcal{O}$ függvény. Legtöbbször azonban nem vagyunk ennyire formálisak. A FAKTORIZÁCIÓ például egy probléma. A szóhasználat jelentheti a következők bármelyikét.

Példa (FAKTORIZÁCIÓ I). Input: egy pozitív egész szám. Output: prím osztóinak listája a megfelelő multiplicitásokkal.

Példa (FAKTORIZÁCIÓ II). Input: egy pozitív egész szám. Output: egy prím osztója.

Példa (FAKTORIZÁCIÓ III). Input: egy pozitív egész szám. Output: legkisebb prím osztója.

Példa (FAKTORIZÁCIÓ IV). Input: egy pozitív egész szám és egy t érték. Döntsük el van-e 2 és t közötti osztó.

Bármelyik megoldása átalakítható (alap programozási technikák, például rekurzió, bináris keresés ismeretével) a többi megoldásává.

Az inputok és outputok leírásában/leírtak értelmezésben is meg kell egyezniük a számítást követőknek. Ehhez $\mathcal{I}$ és $\mathcal{O}$ elemeit kódolnunk kell. A halmaz elemeit kódszavakkal helyettesítjük. Ehhez nézzünk néhány fontos alapfogalmat.

Definíció. Σ *ábécé* egy nemüres véges halmaz. Elemeire mint *betűk* vagy *karakterek* hívatkozunk.

Definíció. Σ ábécé esetén Σ^n az n hosszú karaktersorozatok, másképpen *szavak* halmaza. Σ^0 egy egyelemű halmaz, egyetlen eleme az ϵ üres (0 hosszú) szó. Σ^* a Σ ábécé-t használó véges szavak halmaza, azaz $\Sigma^* = \cup_{i=0}^{\infty}\Sigma^i$.

Az input/output kódolása az input elemeinek azonosítása kódszavakkal, azaz Σ^* elemeivel.

Példa. $\mathbb{N}^+$ (a pozitív egészek) egy kódolása lehet a következő. Legyen $\Sigma = \{0, 1\}$. Az x szám kódját úgy definiáljuk, hogy felírjuk kettes számrendszerben és a kezdő 1-est elhagyjuk. $1 \mapsto \epsilon$, $9 \mapsto 001$, hiszen $9 = 1001_2$. Ez egy bijekció $\mathbb{N}^+$ és $\{0, 1\}^*$ között. Ezt nevezzük $\mathbb{N}^+$ *standard kódolásának*.

Példa. Persze a tízes számrendszerben való felírás is egy kódolása $\mathbb{N}$-nek. Ekkor $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. A továbbiakban két lehetőségünk van. Vagy minden x természetes számnak egy kódját definiáljuk, a szokásost. Ekkor Σ^* nem minden eleme kódol inputot. 002012 például nem kódol inputot. Egy másik lehetőség, hogy minden számjegysorozatban a kezdő 0-kat elhagyva a maradékot értelmezzük. Ekkor egy számnak több kódja is van. $2012, 02012, 000002012$ kódok mindegyike ugyanazt a számot kódolja. Mindkét megoldás kérdéseket vet fel.

Adott Σ^* egy eleme. Lehet, hogy nem kódol inputot? Ha igen, akkor mit teszünk? Elvárjuk az algoritmust felhasználótól, hogy az általa megadott jelsorozat garantáltan inputot kódóljon és az algoritmussal bármi történhet, ha nem így tesz. Esetleg elvárjuk, hogy ebben az esetben az algoritmus jelezze, hogy „rossz kód”.

Ha több kódja is van egy inputnak, akkor gyakran kijelölhetünk egy standard kódot, amihez ragaszkodunk. Ha több kódja van egy inputnak, akkor elvárjuk-e, hogy két kód esetén el tudjuk dönteni, hogy ugyanazt az inputot kódolják-e?

Ezekel a problémákkal nem foglalkozunk. A szokásos kódolások olyanok, hogy algoritmusaink a legnagyobb követelményt is könnyen teljesítik (esetleg kis többlet munkával).

Példa. $\mathbb{N}$ kódolása az $\Sigma = \{1\}$ ábécé-vel is lehetséges: n kódja legyen n darab 1-es (1^n). Ezt $\mathbb{N}$ *unáris kódolásának* nevezzük.

Példa. $\mathbb{Q}$ szokásos kódolásában $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /, -\}$ A racionális számot kódoló szvak egyetlen „/” jelet tartalmaznak, ami előtt egy természetes szám áll (esetleg egyetlen $-$ előjellel kezdve), míg a törtjel után egy pozitív egész kódja áll. Ebben a kódolásban a „fél” racionális számnak $1/2$ és $1006/2012$ is kódja. $1/2/3$, $12/0$, $-1/-2$ nem kódolnak racionális számot (legalábbis nem a fenti megállapodások alapján).

A kódolás után egy számítási feladat egy $f : \Sigma^* \to \Sigma^*$ függvény.

Példa (FAKTORIZÁCIÓ V). Legyen $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, .\} \cup \{,\}$ Input: egy pozitív egész szám tízes számrendszerben felírva. Output: prím osztóinak növekvő listája, mindegyik osztót multiplicitása követi egy vesszővel elválasztva, majd egy pontos vessző követi, kivéve az utolsó prím osztót és multiplicitását, amit egy pont követ. Azaz:

$$24 \mapsto 2, 3; 3, 1.$$

$$121 \mapsto 11, 2.$$

$$43 \mapsto 43, 1.$$

$$2010 \mapsto 2, 1; 3, 1; 5, 1; 67, 1.$$

$$2011 \mapsto 2011, 1.$$

$$2012 \mapsto 2, 2; 503, 1.$$

Megjegyzés. Megjegyezzük, hogy csak megszámolható $\mathcal{I}$ és $\mathcal{O}$ halmazok kódolhatók.

Megjegyzés. A kódoláshoz elször választani kell egy ábécé-t. Ekkor a $\Sigma = \{0, 1\}$ választáshoz is ragaszkodhatnánk. Néha azonban technikailag egyszerűbb lesz nagyobb ábécé-vel dolgozni.

A $\Sigma = \{1\}$ „minimál ábécé" is egy lehetőség. Ez azonban túlzott. Például az $\{0, 1, 2, ..., n-1\}$ elemű halmaz első kódolásában minden szám kódja legfeljebb $\log_2 n$ hosszú. Bármilyen nagyobb ábécé-t választunk ennél nagyságrendileg jobb kódolást nem találhatunk. (Az ábécé növelése csak konstansszorosára „nyomja össze" a kódokat.) Az egyelemű ábécé viszont rossz, ebben legalább $n - 1$ hosszú kódszó is szükséges bármit teszünk.

Egy kódolás után beszélhetünk az *input méretéről*, az input jelsorozat karaktereinek száma, az input hossza.

Példa. $\mathbb{N}$ standard kódolásában n kódjának hossza $\lceil \log_2 n \rceil$. $\mathbb{N}$ unáris kódolásában n kódjának hossza n.

Példa. Legyen G egy egyszerű gráf a $V = \{1, 2, \ldots, v\}$ csúcshalmazon. Kódolásához legyen $\Sigma = \{0, 1, \}$. G kódjának hossza $\binom{v}{2}$ lesz (az ilyen alakú számokat nevezzük *háromszögszámoknak*). A kód pozíciói V kételemű részhalmazaival vannak azonosítva. Egy karakter/bit azt kódolja, hogy a megfelelő két csúcs össze van-e kötve. Mivel $2^{\binom{v}{2}}$ a kódolandó objektumok száma és $|\Sigma| = 2$ ezért rövidebb kódszavakkal dolgozva nem is lehetséges az összes v csúcsú egyszerű gráf kódolása.

Példa. Legyen G egy e élű egyszerű gráf a $V = \{1, 2, \ldots, v\}$ csúcshalmazon. Ekkor kódja lehet, hogy V elemeit felsoroljuk és mindegyikhez kettőspont után felsoroljuk szomszédait (amelyek sorát pontosvesszővel választjuk el és ponttal zárjuk le). Azaz $\Sigma = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ;, .\}$. Például egy kódolt gráf $1 : 2; 4.2 : 1.3 : 1; 5.4 : 1.5 : 3$. A gráf kódjának hosszát felülről becsülhetjük $(v + 2e)(\log_2 v + 1)$-gyel. Nagyságrendileg tömörebb kódolást nem is remélhetünk, hisz a kódolandó objektumok száma $\binom{\binom{v}{2}}{e}$.

⋆ ⋆ ⋆

Ki kell emelnünk a számítási problémák egy fontos esetét, a döntési problémákat. Ekkor egyetlen bitet számolunk ki (függvényünk két értékű). Azt is mondhatjuk (ez így szokás) hogy egy inputot vagy elfogadunk vagy elvetünk. Tehát egy eldöntési probléma azonosítható Σ^* egy részhalmazával, az elfogadandó inputok halmazával. A szavak egy elfogadandó részhalmazát nyelvnek nevezzük.

Definíció. Egy döntési probléma leírható egy $L \subset \Sigma^*$ nyelvvel. (Illetve a nyelv értelmezhető, mint a „hozzátartozik-e" döntési probléma.)

Az algoritmus fenti leírása matematikailag nem pontos (lényegében az algoritmus idegen szót egy ismerősebb „eljárás" szóval helyettesítettük). Mégis, talán rávilágítottunk arra, hogy egy algoritmus értelmezéséhez sok megállapodás szükséges.

Azt is szeretnénk hangsúlyozni, hogy az algoritmus statikus leírása mellett (amit egy tankönyvben láthatunk) van egy dinamikus kép is (gondoljunk egy filmre, ami

azt mutatja, hogy két háromjegyű szám esetén szorzatukat kiszámoljuk): Az idő telésével a papírunkon számok jelennek meg, egész a számítás végéig szervezzünk munkaterületünket, amikor is leállunk (például az eredmény kétszeres aláhúzásával jelezve a számítás végét). A tankönyvben leírt eljárást akkor értettük meg, ha képesek vagyunk futtani azt különböző inputokon. Illetve, ha jól begyakoroltunk egy algoritmust és a matematikai és nyelvi kifejezés egy érettségi szintjét elértük, akkor képeseknek kell lennünk valamilyen tankönyvi leírását adni az eljárásunknak. A statikus és dinamikus szemlélet együtt jár. Azért életünk/matematikai tanulmányaink során a dinamikus képpel találkozunk először. Ez az az út, amit az alapműveletek elvégzésénél az általános iskola első osztályában mindenkivel követtetnek.

2. Az algoritmus matematikai fogalma

Az algoritmus matematikai fogalma a XX. század első harmadában alakult ki. Egyik ösztönzője a logika fejlődése, illetve Hilbert nevezetes problémái közül a tizedik volt. Ebben azt kérdezte Hilbert, hogy van-e olyan algoritmus, ami egy adott egészegyütthatós polinomról eldönti, hogy van-e egész gyöke (Diophantikus számelmélet alapproblémája). A válasz, mint később kiderült: „nem". Ennek igazolása már lehetetlen az algoritmus fogalmának matematizálása nélkül.

A probléma érdekes. Az algoritmus/eljárás szavak részei mindennapi szóhasználatunknak, de a matematikában nincsenek értelmezve. Egy definíció megadása, akkor sikeres, ha a matematikus társadalom többsége rábólint: „elfogadom korrekt definíciónak". Az a pillanat, amihez ezt kötik az Church egy előadása (amit 1935-ben tartott az Amerikai Matematikai Társulat számára). A felhívást, hogy az algoritmus korrekt definíciójának keresése véget érhet (ott vagyunk a „szent grálnál") *Church-tézisként* hivatkozzák. Érdekes megjegyezni, hogy a tézis bejelentésénél nem volt meg a matematikusok összhangja. Például Gödel nem fogadta el. Turing munkája, majd az, hogy Turing, Church, Gödel és mások próbálkozásai mind ekvivalens fogalomhoz vezetnek a tézist elfogadottá tették. Ennek ellenére, ha valami technológiai áttörés történik, amely gyökeresen átalakítja a mindennapi képünket a számítás fogalmáról, akkor a tézist újra kell értékelni. Például a kvantum számítógépek megvalósításának elméleti lehetősége is felvetette a tézis vizsgálatát. Erre azonban nem volt szükség, kvantum gépekkel is ugyanazon függvények lesznek kiszámíthatók (amennyiben technológialilag megvalósíthatók), mint klasszikus gépekkel.

Mi az alábbiakban Turing definícióját ismertetjük, ami talán a legtranszparensebben próbálja az algoritmus fogalmát megragadni. Ez az algoritmus fogalom a dinamikus szemléletét írja le.

Definíció. Turing-gép egy konfigurációját írjuk le. Ennek „fizikai" részei: input-szalag, munkaszalag, outputszalag, fej. A szalagok mezők sorozatát tartalmazzák. A t szalag mezői $\{M_i^t\}_{i=0}^{\infty}$ (azaz m_{100}^{input} az inputszalag 100 indexű/101-edik mezője, m_0^{munka} a munkaszalag első/0 indexű mezője). A mezőket úgy kell elképzelni mint egy négyzethálós papír egy sorát, amelyik jobb irányban végtelen. A mezők tartalma egy-egy karakter. Az input- és outputszalag mezői a feladat kódolásához használt Σ ábécé elemeit tartalmazzák. Az outputszalagon egy (új) $\smile$ üres/érintetlen karakter is szerepelhet. A munkaszalaghoz egy Γ ábécé-t használunk, emellett itt is szerepelhet az érintetlen karakter. Ezenkívül (a fentiekben leírt karakterektől különböző „határolójeleink" is vannak: $\triangleright$ és $\triangleleft$. Ezek a szalagok „határmezőinek" „bevésett"

tartalma (lásd későbbi formális leírás).

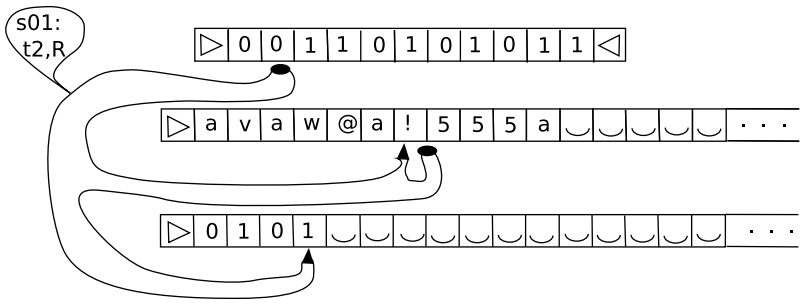
A fej a Turing-gép szalagaival szemmel, illetve kézzel kapcsolódik. A „szem” egy mező értékét olvassa, a kéz a látott mezőt írhatja felül. A gépnek egy input-, munka-szemmel, munka-kézzel és output-kézzel rendelkezik. Ezek helyzetét az írja le, hogy melyik mező felett helyezkednek el. Azaz mindegyik szalaghoz tartozik egy-egy pozíció, amit a fej kontrolál (az inputszalag esetén ez az input-szem által látott mező, a munkaszalag esetén ez a munka-szem által látott és egyben a munka-kéz által írható mező).

A fejnek van egy bizonyos állapota. A lehetséges állapotok egy S véges halmazt alkotnak (S elemeit állapotoknak hívjuk, S az állapothalmaz) A konfiguráció tehát egy n hosszú input mellett egy hetes:

$$\langle \{M_i^{input}\}_{i=0}^{n+1}, \{M_i^{munka}\}_{i=0}^{\infty}, \{M_i^{output}\}_{i=0}^{\infty}, p^{input}, p^{munka}, p^{output}, s\rangle,$$

ahol $M_0^{input} = M_0^{munka} = M_0^{output} = \triangleright$, $M_{n+1}^{input} = \triangleleft$, $p^{input} \in \{0, 1, 2, \ldots, n+1\}$, $p^{munka}, p^{output} \in \mathbb{N}$, $s \in S$.

A mellékelt rajzon egy vizualizációját adjuk a Turing-gép egy állapotának.



1. ábra. Egy képzeletbeli gép képzeletbeli állapota

Megjegyzés. Néhány fontos megállapítást kell tennünk:

1) A fej a konfigurációnak csak egy szűk részét „látja”. Ez a két szem által látott mező tartalma és az állapota (azaz $(\Sigma \cup \{\triangleright, \triangleleft\}) \times (\Gamma \cup \{\triangleright, \smile\}) \times S$ egy eleme.

2) Az algoritmus a konfigurációt csak lokálisan írja felül. A lokalitás miben létét az alábbiakban fogaljuk össze: Csak a munkaszalag látható mezőjére lehet írni (azt mondjuk a munkaszalag olvasható és írható).

   A szemek (és velük a kezek) mozgása „folytonos”. Ez alatt azt értjük, hogy mindegyik szalagon az új pozíció legfeljebb 1-gyel tér el az előzőtől.

3) Az inputszalag funkciója csak az input tárolása. Az outputszalag funkciója csak az output leírása. Ezt a következő megállapodással garantáljuk: Az outpuszalagra akkor írunk majd, ha a kéz egyet jobbra mozdul. Más mozgást nem is engedélyezünk.

   Azt mondjuk: inputszalag csak olvasható, az outputszalag csak írható.

4) A határoló jelek ($\triangleright$ és $\triangleleft$) szerepe a szemek/kezek szalag fölött tartása.

A fenti megjegyzések alapján egy konfiguráció változtatását egy egyszerű, úgy nevezett *átmenetifüggvény* írja le:

$$\delta : (\Sigma \cup \{\triangleright, \triangleleft\}) \times (\Gamma \cup \{\triangleright, \smile\}) \times S \to \{B, ., J\} \times \Gamma \times \{B, ., J\} \times (\{.\} \cup \Sigma) \times S.$$

Az átmenetifüggvény értelmezési tartománya az 1) megjegyzés alatt leírtakat formalizálja.

Az átmeneti függvény értékkészletében minden értéknek öt komponense van. Ezek rendere a következőt írják le:

(i) input-szem mozgása ('B'= bal szomszédra ugrás, '.' = maradás, 'J' = jobb szomszédra ugrás),

(ii) munka-kéz által leírt karakter (a nem írás a már ott lévő karakter újraírása),

(iii) munka-szem mozgása (ami egyben a kéz mozgása is lásd az input-szem mozgására vonatkozó magyarázatot),

(iv) az output-kéz mozgása és írása egybeolvasztva (a kéz vagy marad és nem ír (.), vagy jobbra mozog és ír egy karaktert (Σ egy eleme)),

(v) az új konfigurációban a fej állapota.

Az értelmezés után egyszerű gyakorlat κ-ból kiolvasni a konfiguráció „látott részét", venni az átmeniti függvény ezen felvett értékét és ez alapján módosítnai κ-t. Az így kapott konfiguráció κ *konfiguráció* κ^+ *rákövetkezője.* A formális leírást az olvasóra bízzuk.

Megjegyzés. Korábbi megjegyzéseink feltételként szolgálnak az átmeneti függvényre. Például a 4) megjegyzést külön feltételek fogalmazzák meg:

- Azaz $\delta(\triangleright, karakter, STATE)$-nek olyan értéknek kell lenni, hogy első koordinátája nem B.
- Azaz $\delta(\triangleleft, karakter, STATE)$-nek olyan értéknek kell lenni, hogy első koordinátája nem J.
- $\delta(karakter, \triangleright, STATE)$-nek olyan értéknek kell lenni, hogy harmadik koordinátája nem B, második koordinátája lényegtelen mert a határoló jel nem írható felül, a rákövetkező konfigurációban a munkaszalag tartalma ugyanaz lesz mint korábban.

A teljes feltételrendszer felírását nem végeztük el. Mindenki megteheti, aki úgy érzi, hogy a formalizálás segíti megértését.

Leírunk egy speciális ω inputhoz tratozó konfigurációt. Ebből a konfigurációból indul az ω inputhoz tartozó számítás.

Definíció. Az $\omega \in \Sigma^n$ inputhoz tartozó kezdőkonfiguráció definíciójához szükséges néhány előzetes megállapodás. Legyen $START \in S$ egy speciális állapot (a számítás kezdetét jelző állapota gépünknek) és legyen $\smile$ egy speciális eleme Γ-nak (az üres karakter). Legyen

$$\kappa_0(\omega) = \langle \{M_i^{input}\}_{i=0}^{n+1}, \{M_i^{munka}\}_{i=0}^{\infty}, \{M_i^{output}\}_{i=0}^{\infty}, p^{input}, p^{munka}, p^{output}, s \rangle,$$

ahol $M_0^{input} = M_0^{munka} = M_0^{output} = \triangleright$, $M_{n+1}^{input} = \triangleleft$, $M_i^{input} = \omega_i$ $(i = 1, 2, \ldots, n)$, $M_i^{munka} = M_i^{output} = \smile$ $(i \in \mathbb{N}^+)$, $p^{input} = p^{munka} = p^{output} = 0$, $s = START$.

Ezekután már természetes az algoritmus futásának dinamikus képét leírni.

Definíció. Legyen $\{\kappa_i\}_{i=0}^{\infty}$ konfigurációk azon sorozata, amelyre $\kappa_0 = \kappa_0(\omega)$ (az ω-hoz tartozó kezdőkonfiguráció) és $\kappa_{i+1} = \kappa_i^+$. Ezt a konfiguráció-sorozatot az ω *inputhoz tartozó futásnak* nevezzük.

3. Kiszámító és eldöntő Turing-gépek

Definíció. Legyen $STOP$ egy speciális állapot, azaz $STOP \in S$. Ennek szerepe a számítás végének jelölése. A futás végtelen sorozatát bizonyos esetekben „levágjuk". Legyen $\{\kappa_i\}_{i=0}^{\ell}$ konfigurációk azon sorozata, amelyre $\kappa_0 = \kappa_0(\omega)$ (az ω-hoz tartozó kezdőkonfiguráció) és $\kappa_{i+1} = \kappa_i^+$, továbbá $\ell = \min\{i : \kappa_i$ állapota $STOP\}$. Amennyiben a minimum mögött álló halmaz üres $\ell = \infty$. Ha $\ell < \infty$, akkor azt mondjuk *a futás véges/leáll.* Ezt a futást redukált futásnak nevezzük.

Ha ω-n gépünk futása leáll, akkor κ_ℓ-ben az outputszalag tartalma a $\triangleright$ jel után az output-kézig tartalmazza a *kiszámított outputot.*

Definíció. Egy f függvényt kiszámít egy T Turing-gép, ha minden $\omega \in \Sigma^*$ esetén leáll a Turing-gép és a kiszámított érték $f(\omega)$.

Másképpen is fogalmazhatunk.

Definíció. Minden T Turing-gép kiszámol egy $f_T : \Sigma^* \to \Sigma^* \cup \{\infty\}$ függvényt. $\omega \in \Sigma^*$ esetén $f_T(\omega) = \infty$, ha a futás nem véges, míg a kiszámolt érték, ha a futás véges ω-n.

T akkor számol ki egy f függvényt, ha $f = f_T$.

Definíció. Eldöntési feladat megoldására szolgáló Turing-gép esetén az outputszalag egyetlen bit leírására szolgál. A definíció egyszerűsíthető: A STOP álapotot helyettesítsük ELFOGAD és ELVET állapotokkal. Ebben az esetben az outputszalagra nincs szükségünk. Döntési feladatoknál ezzel a modellel dolgozunk, ezt *eldöntő Turing-gépnek* nevezzük. Ekkor a futás akkor és csak akkor áll le, ha ELVET vagy ELFOGAD állapotba kerül.

Definíció. Egy eldöntő T Turing-gép eldönti az L nyelvet, ha minden $\omega \in L$ esetén ELFOGAD állapottal áll le a gép és minden $\omega \notin L$ esetén ELVET állapottal áll le a gép. (Speciálisan minden inputon leáll a gép.)

Megemlítünk egy további lehetőséget az elfogadás/elvetés „kódolására".

Definíció. Egy eldöntő T Turing-gép *felsorolja* az L nyelvet, ha minden $\omega \in L$ esetén ELFOGAD állapottal áll le a gép és minden $\omega \notin L$ esetén nem áll le.

⋆ ⋆ ⋆

Egy futáshoz természetes módon rendelhetünk idő- és tárigényt.

Definíció. $TIME(\omega; T)$ a T gép ω-n történő redukált futás definíciójában szereplő ℓ.

$SPACE(\omega; T)$ definíciójához ismét vegyük a T gép ω-n történő redukált futását: $\{\kappa_i\}_{i=0}^{\ell}$. T tárigénye ω-n:

$$SPACE(\omega; T) = \sup\{i : M_i^{\text{munka}} \neq \smile \text{ valamely } t \leq \ell \text{ esetén a } \kappa_t \text{ konfigurációban}\}$$

1. Lemma. *Tetszőleges Turing-gép esetén minden ω inputra*

$$SPACE(\omega; T) \leq TIME(\omega; T).$$

4. Példa

Definíció. Legyen

$$PALINDROM = \{\omega = \omega_1, \ldots \omega_n : \text{ahol } \omega_i = \omega_{n+1-i} \\ \text{minden } i \in \{1, 2, \ldots, n\} \text{ esetén}\}$$

a palindrom szavak nyelve.

Tehát döntési problémával állunk szemben. Adott egy szó, el kell döntenünk, hogy előlről és hátulról olvasva ugyanazt olvasuk-e. Így nem kell outputszalag, ezt az S állapothalmaznak ELVET és az ELFOGAD eleme helyettesíti.

Két Turing-gépet/algoritmust vázolunk. Ennek során elmondjuk, hogyan néznek ki a Turing-gépről készített pillanatfelvételek és szemezgetünk az állapothalmazból. Az egyszerűség kedvéért feltesszük, hogy $\Sigma = \{0, 1\}$.

1. algoritmus/Turing-gép

A START állapotból egyet jobbra lép az input szem és a munka szem/kéz (mindkettő a szalaghatároló jel utáni első mező felett lesz), továbbá a gép a MÁSOLOK állapotba jut. Amíg ez lesz az állapot a gép átmásolja az inputot a munkaszalagra. Közben a két szem (ezzel együtt egy kéz) folyamatosan jobbra mozog. Ez akkor áll le, amikor az inputszalagon az $\lhd$ jel nem olvasható (MÁSOLÁS-KÉSZ állapot).

Ezzel az omega input a munkaszalgra is rákerült. A továbbiakban csak ezt a másolatot használjuk. Az input munkaszalagon való elhelyezésének lényege, hogy itt felülírhatjuk/módosíthatjuk a karaktereket. Ha ezt megtehettük volna az inputszalagon is, akkor a másolásra nem lett volna szükség.

A munkaábécé: $\Gamma = \{0, 1, 0^{\checkmark}, 1^{\checkmark}\}$

A Turing-gép további munkájához következő állapotokat használjuk

$$S \supset \{\text{HÁTUL-PIPÁL, HÁTUL-TESZT-0, HÁTUL-TESZT-1,} \\ \text{HÁTUL-TESZT-0-MOST, HÁTUL-TESZT-1-MOST, ELŐL-PIPÁL,} \\ \text{ELŐL-TESZT-0, ELŐL-TESZT-1, ELŐL-TESZT-0-MOST,} \\ \text{ELŐL-TESZT-1-MOST, ELFOGAD, ELVET}\}.$$

MÁSOLÁS-KÉSZ állapotból egyet balra lép a munkaszem/kéz és HÁTUL-PIPÁL állapotba kerül. Ebben az állapotban felülírja a látott karaktert: 0-t $0^{\checkmark}$-re ír át, 1-re $1^{\checkmark}$-et ír rá. A látott karakter alapján ELŐL-TESZT-0 vagy ELŐL-TESZT-1 állapotba kerül és elindul balra. Elmegy az utolsó pipálatlan karakterhez: Persze ezt olvasáskor nem érzékeli, egyet túl kell mennie. Ha $\rhd$ vagy pipált karaktert lát, akkor jobbra (vissza)lép egyet. Ekkor hajtja végre a tesztet, azaz ELŐL-TESZT-0-MOST vagy ELŐL-TESZT-1-MOST állapotba kerül. Ha az olvasott karakter és az állapotban tárolt infomáció nem egyezik meg, akkor a teszten megbukott az input; a gép ELVET állapotba kerül, leáll. Ha a teszten átmegy az input, akkor a karaktert pipálja a gép, egyet jobbra lép a munkaszem/kéz és ELŐL-PIPÁL állapotba

kerül. Új teszt indul. Az olvasott karaktert kipipálja a gép (átírja az ott látott 0-t $O^{\checkmark}$-re, illetve az ott látott 1-t $1^{\checkmark}$-re). A kipipált karaktert megjegyzi állapotában: HÁTUL-TESZT-0 vagy HÁTUL-TESZT-1 állapotba kerül. Jobbra mozgás, amíg meg nem találjuk az utolsó kipipálatlan karaktert.

A továbbiak kidolgozását: az ELFOGAD állapotba jutás tisztázását és a leíró szöveg kódolását az átmeneti függvénnyel az érdeklődő hallgatóra bízzuk.

Könnyű látni, hogy minden ω inputon a fenti gép futása $\mathcal{O}(|\omega|^2)$ Ha ω egy palindrom szó, akkor a fenti T' gép futásának hosszát könnyen kiszámíthatjuk és ennek nagyságrendje $|\omega|^2$ lesz. A konstansokat nem számoltuk ki. Lényegtelen is, az állapotok számának növelésével a futási idő csökkenthető. Például használhatnánk a HÁTUL-TESZT-000, HÁTUL-TESZT-001, HÁTUL-TESZT-010, HÁTUL-TESZT-011, HÁTUL-TESZT-100, HÁTUL-TESZT-101, HÁTUL-TESZT-110, HÁTUL-TESZT-111 állapotokat input bitek hármasainak együttes tesztelésére és kevesebbet kellene „ingáznia” a gépnek.

A fenti algoritmus nem hatékony. A másolás után az inputnak két példánya áll rendelkezésünkre. Mindegyikhez tartozik egy-egy szem. Ezt kihasználhatjuk.

2. algoritmus/Turing-gép

Az állapothalmaz legyen

$$S = \{\text{START, MÁSOLOK, MÁSOLÁS-KÉSZ, INPUTFEJ-ELŐRE, ELŐL-VAGYOK, TESZT, ELFOGAD, ELVET}\}.$$

Ugyanúgy kezdünk mint az előző algoritmus. Az inputot átmásoljuk a munkaszalagra. Ennek befejezésével MÁSOLÁS-KÉSZ állapotba jutunk.

Ebből mindkét szem balra egyet lép (a munka szem/kéz az átmásolt sorozat utolsó karaktere felett lesz), továbbá az INPUTFEJ-ELŐRE állapotba jut a gép. Ekkor a munkaszalag felett a szem/kéz mozdulatlan, az inputszalag feletti szem folyamatosan jobbra mozog. Egész addig, amíg a $\triangleright$ jelet nem látja (ELŐL-VAGYOK állapot). Innen TESZT állapotba jut a gép az input szem eggyel jobbra mozgatása után (ekkor az input szem az input első karaktere fölé kerül, közben az output szem az átmásolt input utolsó karaktere felett maradt végig). A TESZT állapotban mindig ellenőrzi a gép, hogy a két szem ugyanazt látja-e. Ha valamikor ez nem teljesül, akkor ELVET állapotba jut, különben az input szem egyet jobbra, a munka szem egyet balra mozdul. Ez addig történik, amíg az input szem az input végét jelző karaktert nem látja (ekkor szükségszerű, hogy a munka szem a munkaszalag kezdetét jelző karakter felett legyen. Ha ez megtörténik, akkor a gép ELFOGAD állapotba kerül.

A fenti „szöveg” egyszerűen megfogalmazható az átmeneti függvény alkalmas definic'iójával.

Minden ω inputra a futás hossza legfeljebb $3(n+1) = 3|\omega| + 3 = \mathcal{O}(|\omega|)$, ahol $n = |\omega|$, és az $\mathcal{O}$ (olvasd „nagy ordó”) egy felső becslést jelöl rejtett szorzó és additív konstanssal.

Definíció. T Turing-gép időigénye egy $\omega \in \Sigma^*$ inputon $TIME(\omega; T) = \ell$, mely egy „csonkított” konfigurációsorozat hossza (azaz a konfigurációk végtelen sorozatában megállunk az első olyannál, amelyben az állapot a számítás végét jelzi). Ekkor a futás $\{\kappa_i\}_{i=0}^{\ell}$, azaz az ℓ-edik konfigurációban kerül először STOP vagy döntési feladatnál ELFOGAD/ELVET állapotok.

Korábbi megállapítáunk az új jelöléssel kimondva

Minden $\omega \in \Sigma^*$ esetén $TIME(\omega; T) = \mathcal{O}(|\omega|)$, ahol T a fenti ismertetett, a PALINDROM nyelvet elfogadó gép.

Ez az eredmény a nagyságrend szempontjából éles. Pontosabban minden PALINDROM-ot kiszámító T Turing-gépre, van olyan ω input, amin T futása legalább $|\omega|$ hosszú. Feltéve, hogy $0 \in \Sigma$ a 0^n (n darab 0 karakter) esetén a gépnek el kell ezt fogadni, de ezt nem teheti meg az utolsó karakter elovasása nélkül. Ehhez viszont legalább n darab jobbra lépést kell tennie.