Algoritmuselmélet és bonyolultságelmélet MSc hallgatók számára

A Turing-gép fogalma

2017. Előadó: Hajnal Péter

1. Számítási feladat formalizálása, kódolás

A számítási feladat egy $f : \mathcal{I} \to \mathcal{O}$ függvény az inputok halmazából az outputok halmazába. Ezt formalizáljuk. Ehhez az $\mathcal{I}/\mathcal{O}$ halmazokat kódoljuk.

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 egy Σ ábécé választása és az input elemeinek azonosítása kódszavakkal, azaz Σ^* elemeivel.

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

Megjegyzés. Megjegyezzük, hogy csak megszámolható $\mathcal{I}$ és $\mathcal{O}$ halmazok kódolhatók. Azaz halmazelméleti okok miatt a valós számok halmaza nem kódolható.

Példa. $\mathbb{N}$ egy kódolása lehet a következő. Legyen $\Sigma = \{0, 1\}$. Az x szám kódját úgy definiáljuk, hogy felírjuk kettes számrendszerben. $0 \mapsto 0$, $1 \mapsto 1$, $9 \mapsto 1001$, hiszen $9 = 1001_2$. Ez egy-egy, de nem bijektív leképezés $\mathbb{N}$ és $\{0, 1\}^*$ között: ϵ és a 0-val kezdődő, legalább kettő hosszú 0-1 sorozatok nem kódszavak.

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 kettes számrendszerbeli bijektív 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. 002017 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. $2017, 02017, 000002017$ 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 kódoljon 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 $2017/3034$ is kódja. $1/2/3$, $12/0$, $-1/-2$ nem kódolnak racionális számot (legalábbis nem a fenti megállapodások alapján).

$\star \qquad \star \qquad \star$

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

Példa (FAKTORIZÁCIÓ). 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:

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

$$121 \mapsto 11,2.$$

$$43 \mapsto 43,1.$$

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

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

$$2013 \mapsto 3,1;11,1;61,1.$$

$$2016 = 2,5;3,2;7.$$

$$2017 \mapsto 2017,1.$$

A Turing-gép fogalma-2

Példa (FAKTORIZÁCIÓ). Legyen $\Sigma = 0, 1$. Az n input számot kettes számrendszerben kódoljuk. $f(\omega)$ legyen az ω által kódolt szám legkisebb prímtényezőjének kódja.

Példa (FAKTORIZÁCIÓ). Legyen $\Sigma = 0, 1$. Az input egy n, t számpar mondjuk 10 számrendszerben kódolva ($\Sigma = \{0, 1, `,'\}$ a számpár két elemének elhatárolására használjuk a vessző karaktert). Egy bit számítandó ki: Van-e n-nek olyan o osztója, hogy $2 \leq o \leq t$

Talán zavaró, de sok FAKTORIZÁCIÓ problémát formalizáltunk. Ezek mind formálisan különböznek, mégis mind ugyanazon matematikai problémát jelenítik meg. Ezen tegyük túl magunkat. Ha az egyiket algoritmikusan kezelni tudjuk akkor standard trükkökkel (például rekurzió, bináris keresés) a többi is kezelhető. Ahogy az egyik kezelését egy másik formalizált FAKKTORIZÁCIÓ algoritmusára „visszavezetjük" elhanyagolható többlet munkát végzünk. Például, ha kettes számrendszerbeli kódolás esetén meg szeretnénk találni a legkisebb prímtényezőt, és tudjuk a tízes számrendszerbeli kódolásra vonatkozó problémát megoldó algoritmust, akkor csak kettes számrendszerbeli kóddal leírt számunkat tízes számrendszerbe kell átírnunk, futtatni rajta a 10-es számrendszerbeli algoritmust, majd a tízes számrendszerben leírt eredmény át kell írnunk kettes számrendszerbe. A két átíras „költsége"/bonyolultsága nyilván nem számottevő a legkisebb prímtényező megkereséséhez képest.

Azért aláhúzunk egy problémát: A $\Sigma = \{1\}$ „minimál ábécé" is egy lehetőség. Ez azonban egy veszélyt rejt.

Az unárisan kódolt szám könnyen felírható például tízes számrendszerben. Fordítva — habár nem látunk nehézséget — mégis problémás a feladat. Az unáris felírás hossza (ennek megfelelően a felíráshoz szükséges idő is) hatalmas. Például az $\{0, 1, 2, \ldots, n-1\}$ elemű halmaz első kódolásában minden szám kódja legfeljebb $\mathcal{O}(\log n)$ 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. Az unáris átírás (algoritmikus probléma nélkül is) exponenciális időt követel. Ha valaki a FAKTORIZÁCIÓ inputját unárisan kódoltnak tételezi fel csal, megtéveszthetí a jóhiszemű olvasót.

$\star$

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 $(v + 2e)(\lceil \log_{10} \rceil v + 1)$-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.

Definíció. A szavak egy részhalmazát *nyelvnek* nevezzük. Azaz egy nyelv: $L \subset \Sigma^*$.

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

2. Az algoritmus matematikai fogalma

Az algoritmust egyelőre nem formalizáltuk, de megmelítünk két különböző szemléletet.

Az algoritmusok véges módon leírható (mondhatjuk azt is hogy kódolhatók). Ilyen kódokat láthatunk algoritmuselméleti könyvekben vagy más tankönyvekben, ahol algoritmusokat ismertetnek.

Emellett 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.

⋆

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 1935. április 19-én tartott előadása az Amerikai Matematikai Társulat egy találkozóján. A felhívást, hogy az algoritmus korrekt definíciójának keresése véget érhet (ott vagyunk a „szent grálnál") *Church-tézisként* hivatkozzák.

Érdekes megjegyezni, hogy a tézis bejelentését megelőzte egy Gödelhez írt levél. Gödel a matematikai logika megkérdőjelezhetetlen óriása volt, aki sziután aktívan fogllakozott a kérdéssel. A Church levelében leírt kiszámíthatóság fogalmat Gödel nem fogadta el. Az 1935-ös változat ennek módosítása volt, Church kötni tudta saját fogalmát Gödel próbálkozásaihoz. De a tézisben megfogalmazott indokokat a matematikai „egyezséghez" Gödel nem tartotta kielégítőnek.

Az igazi áttörést Turing egy 1936-os cikke okozta. Ebben egy új definíciót fogalmazott meg. Ezt már széles körben elfogadták mint a kiszámíthatósag/algoritmus fogalma.

Később a kutatók belátták, hogy Turing, Church, Gödel és mások próbálkozásai mind ekvivalens fogalomhoz vezetnek. A tézist mind a mai napig elfogadják.

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. A fogalaom a Turing-gép, amely konfiguráció egy sorozatán áthaladva hajt végre egy algoritmust. A Turing-gép definíciója több definíció eredője lesz.

Definíció (KONFIGURÁCIÓ). Turing-gép egy konfigurációját írjuk le. Ennek „fizikai" részei: inputszalag, munkaszalag, outputszalag, fej. A szalagok mezők sorozatát tar-

talmazzák. 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 outputszalagra a kiszámolt $f(\omega) \in \Sigma^*$ kerül a számítás során. Ezen időben az output szalag mezőin egy (új) $\smile$ üres/érintetlen karakter is szerepelhet Σ elemei mellett.

A munkaszalaghoz egy Γ ábécé-t használunk, emellett itt is szerepelhet az érintetlen karakter.

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).

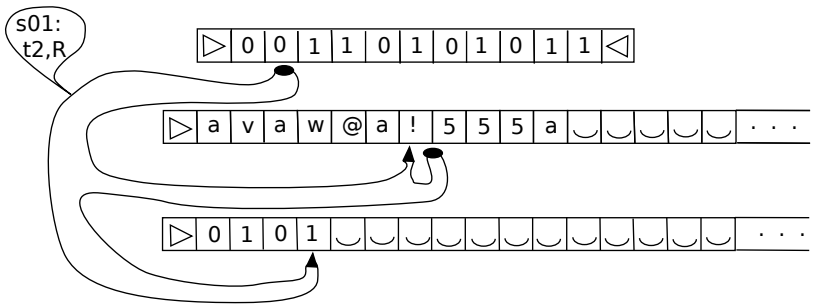
A fejnek van egy bizonyos állapota. A lehetséges állapotok egy S véges halmazt alkotnak (S elemeit állapotoknak hívjuk, S az állapothalmaz)

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 inputszalag funkciója csak az input tárolása. Ezt olvashatjuk az inputszemmel, de nem írhatjuk felül. Azt mondjuk az „inputszalag csak olvasható”.

3) A munkaszalag a „munka helye”. Ide kerülhetnek, „feljegyzések”, „részeredmények”, „részletszámítások”. Ezt a részét a konfigurációnak írhatjuk, „radírozhatjuk” Az algoritmus a munkaszalagot csak lokálisan írja felül. Csak a munkaszalag látható mezőjére (aktuális pozíciójába) lehet írni. A munkaszalag tartalmát a munkaszem látja. Azt mondjuk az „munkaszalag olvasható/írható”.

3) 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.

5) 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 az „outputszalag csak írható”.

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

A fenti megjegyzések alapján egy konfiguráció változtatását egy egyszerű függvény írja le.

Definíció (ÁTMENETIFÜGGVÉNY). Egy Turing-gép *átmenetifüggvénye* egy

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

függvény.

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 6) 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 (KEZDŐKONFIGURÁCIÓ). 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 (FUTÁS). 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.

A fenti definíció egy végtelen konfigurációsorozatot nevez futásnak. Az igazi algoritmus azonban egy ponton leáll, a számítást befejezte, az eredmény kihirdeti.

Definíció (STOP). 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 $\ell = \min\{i : \kappa_i \text{ állapota } STOP\}$. 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.*

Tualjdonképpen most értünk a Turing-gép definiciójának végére. Egy $f : \Sigma^* \to \Sigma^*$ feladathoz tartozó Turing-géphez szükségünk van egy Γ munka ábécére, speciális karakterekre ($\triangleright$, $\triangleleft$, $\smile$), egy S állapothalmazra (speciális állpotokkal: START, STOP) és egy δ átmeneti függvényre. Az is nyilvánvaló mikor oldja meg egy Turing-gép az f feladatot:

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 Σ^*-beli érték, ha a futás véges ω-n.

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

Definíció. Egy $f : \Sigma^* \to \Sigma^*$ függvény akkor *kiszámítható*, ha alkalmas T Turing-gépre $f = f_T$.

3. Eldöntő, felsoroló Turing-gépek

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.)

Definíció. Egy $L \subset \Sigma^*$ nyelv/döntési feladat eldönthető ha van olyan Turing-gép, ami eldönti. Az eldönthető nyelvek halmaza legyen $\mathcal{D}(\Sigma)$.

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.

Definíció. Egy $L \subset \Sigma^*$ nyelv felsorolható ha van olyan Turing-gép, ami felsorolja. Az eldönthető nyelvek halmaza legyen $\mathcal{S}(\Sigma)$.

Adott Σ ábécé esetén nyilván

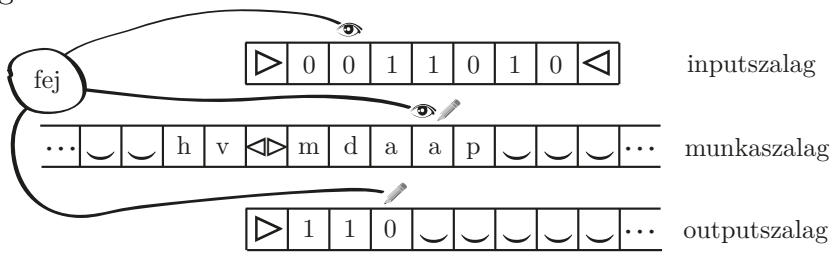
$$\mathcal{D}(\Sigma) \subset \mathcal{S}(\Sigma) \subset \mathcal{P}(\Sigma^*).$$

A Σ ábécé lefixálásának egyetelen szerepe van: így halmazokkal dolgozunk.

4. Az alap Turing-gép fogalmának változatai

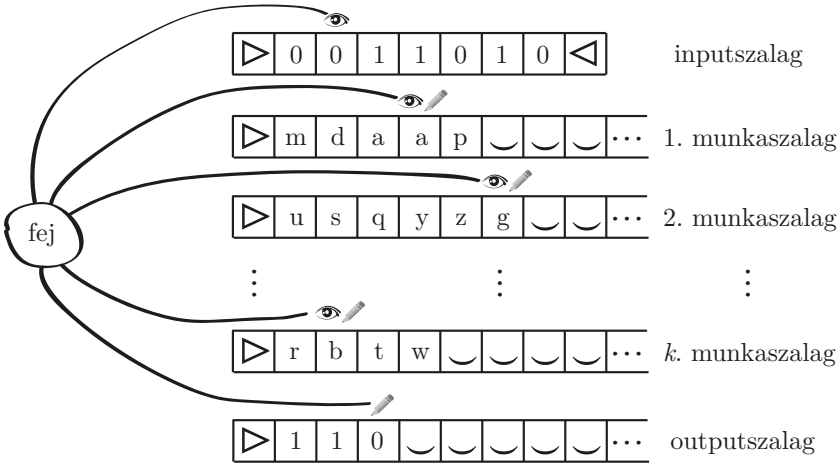
A fent leírt standard modell esetleges, hiszen például az, hogy egy irányban végtelen szalagokat használunk továbbá, hogy éppen három szalagon dolgozunk önkényes választások. Így természetesen léteznek más változatok is, íme néhány példa:

- **Két irányban végtelen munkaszalag:** A munkaszalag bal és jobb irányban is végtelen és nem tartalmaz ⌋idezszalag eleje jelet, csupán egy kezdeti mezőt kell megadni.



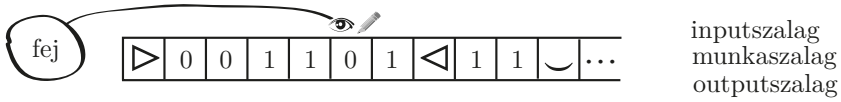
2. ábra. A két irányban végtelen munkaszalag.

- ***k*-szalagos modell:** Rögzített $k \in \mathbb{N}$ számú munkaszalagot használunk, és minden egyes munkaszalaghoz tartozik egy szem-kéz páros. Fontos megjegyezni, hogy k nem függ az input méretétől.



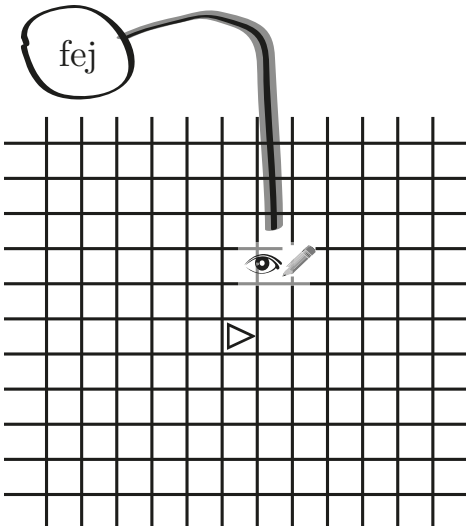
3. ábra. A k-szalagos modell.

- **Egyszalagos modell:** Minden művelet egyetlen szalagon történik, amely így egyszerre input-, munka-, és outputszalag. A szalag jobbra végtelen és a kezdeti mezőket az input foglalja el $\triangleright$,$\triangleleft$ jelek között. Ebben a modellben az input is felülírható és az output a szalag tartalma a $STOP \in S$ állapot elérésekor.



4. ábra. Az egyszalagos modell.

- **Kétdimenziós munkaszalag:** Érdemes a munkaszalagot úgy elképzelni, mint egy végtelen négyzethálós füzetlapot (ez lehet a teljes síkot lefedő háló vagy csupán egy negyedsíknak megfelelő). Ebben az esetben is ki kell jelölnünk egy kezdeti mezőt.



5. ábra. Munkaszalag a síkon.

Nyilvánvaló, hogy a különböző modellekhez más látható részek tartoznak, ebből adódóan pedig eltérőek a modellek átmeneti függvényei. A szükséges módosítások formalizálása nem nehéz, de időigényes; ezt az érdeklődő hallgatókra bízzuk

Fontos kérdés, hogy az aprónak tünő változtatások nem változtatják-e meg a kiszámíthatóságot. Fontos tétel: NEM. Igazából nem is nehéz: bármelyik változatra alapítunk egy algoritmust/Turing-gépet az általa elvégzett feladat megoldható egy másik modellben (szimulálható). Összegezve: A korábban bevezetett függvény-/nyelv-osztályok nem függnek a fenti típusú módosításokon.

A bonyolultságelmélet „feladata" a $\mathcal{D}$ nyelvosztály finomítása, az ide tartozó nyelvek vizsgálata. $\mathcal{D}$-n kívüli nyelvek összehasonlítása és vizsgálata inkább matematikai logika jellegű kérdésekhez vezet.

Számunkra az elkülönített munkaszalag fontos lesz. Ez alapján tudunk majd beszélni „logaritmikus tárban" kiszámítható függvényről. A több (k darab) munkaszalag léte kényelmes azok számára, akik valójában megadnak Turing-gépeket. A k-szalagos Turing-gép modelljét standard modellnek hívom. Ha azt mondjuk, hogy egy L nyelv eldönthető, akkor ez alatt azt értjük, hogy van alkalmas $k \in \mathbb{N}$ és alkalmas k-szalagos Turing-gép, amely eldönti L-et.