Számításelmélet

Dr. Kovásznai Gergely jegyzete

2008. május 18.

Verzió: 1.1

TARTALOMJEGYZÉK

BEVEZETÉS

A Számításelmélet című tantárgy keretein belül *algoritmusok bonyolultságával* fogunk foglalkozni. A számítási bonyolultság a számítástudomány azon területe, mely annak okait kutatja, hogy bizonyos problémákat miért olyan nehéz számítógépen megoldani. Ez a terület, mely kb. 30 évvel ezelőtt még látszólag nem is létezett, igen gyorsan növekedett, és ma a számítástudományi kutatások jelentős része erre a területre irányul.

A digitális számítógépek és az informatika megszületése után töretlen optimizmus uralkodott el a terület művelői körében. Mindenki úgy gondolta, hogy az emberiséget régóta foglalkoztató problémákra belátható rövid időn belül a számítógépekkel megoldást fogunk tudni találni. Az 1960-es évek végén és az 1970-es évek elején azonban ez a lelkesedés látványosan tört le, mikoris olyan megoldandó problémákkal találták szemben magukat az informatikusok, melyekre a korabeli számítógépek nem voltak képesek belátható időn belül megoldást produkálni. Mindenki azt gondolta azonban, hogy ezen nehéz problémák megoldása is csak gyorsabb hardver és nagyobb memória kérdése, azaz csupán a technikai fejlettségen múlik, milyen problémákat vagyunk képesek a gépeinkkel megoldani. Ezt az optimista hozzáállást cáfolta a Steven Cook és Richard Karp által 1971-1972-ben útjára indított **NP**-*teljesség* elmélete. Az idők folyamán sok nevezetes és alapvető problémáról sikerült belátni, hogy **NP**-teljes, azaz *minden bizonnyal kezelhetetlen*. Az azonban hozzátartozik az igazsághoz, hogy a mai napig senkinek sem sikerült bizonyítani, hogy az **NP**-teljes problémák szükségszerűen kezelhetetlenek, de kevés tudós van más véleményen. Ez a nyitott kérdés a számításelméletben a

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

kérdésként ismert. Ezen egyenlőségnek a bizonyítása vagy cáfolása az *algoritmuselmélet központi problémája*. Ez pedig egyáltalán nem véletlen. Bizonyos problémák esetén szeretnénk, ha a fenti egyenlőség teljesülne; ugyanis ez azt jelentené, hogy az adott problémára létezik gyors megoldó algoritmus. Más problémák esetén viszont *nagyon nem szeretnénk*, ha a fenti egyenlőség teljesülne; ugyanis eddig kezelhetetlennek hitt problémákon alapszik sok-sok informatikai megoldás biztonságossága, mint pl. a banki adatok titkosságáért felelős *kriptográfiai protokolloké*.

Az 1. fejezetben az ún. *Turing-gépekkel* ismerkedünk meg, mellyel megalapozzuk a számítási bonyolultság különböző típusú definícióit. A 2. fejezetben már kissé elvonatkoztatunk a Turing-gépektől, és már konkrétan különböző *bonyolultsági osztályokkal* foglalkozunk, köztük a két legfontosabbal: a **P**-vel és az **NP**-vel. A 3. fejezetben fogunk betekintést nyerni a fent említett **NP**-*teljesség* elméletébe.

JELÖLÉSEK

ϵ :

az üres szó.

$\lceil x \rceil$:

az x szám felső egészrésze.

$f(n) = \mathcal{O}(g(n))$:

léteznek olyan $n_0, c > 0$ egész számok, hogy minden $n \geq n_0$-ra $f(n) \leq c \cdot g(n)$.
Szemléletes jelentése: az f függvény legfeljebb olyan gyorsan növekszik, mint a g.

Az $\mathcal{O}(g(n))$ kiejtése: *„ordó”* $g(n)$.

1. TURING-GÉPEK

Mivel algoritmusok bonyolultságával kívánunk foglalkozni, szükséges lesz az algoritmus fogalmát pontosan körülhatárolnunk. Ez nem is olyan könnyű feladat, mint első pillantásra tűnik, hiszen meg kell állapodnunk az algoritmus által használt adattípusok és utasítások milyenségében, illetve abban, hogy ezeket hogyan fűzhetjük egymás után. Mindezt ráadásul úgy kell megtennünk, hogy egy „platformfüggetlen" algoritmusfogalmat kapjunk. A szakirodalomban az „algoritmusnak" különböző definíciói is napvilágot láttak, ezek közül a legfontosabbak:

- Alan M. Turing: Turing-gép (1937)
- S. C. Kleene: rekurzív függvények (1936)
- A. Church: lambda-kalkulus (1941)
- A. Markov: egy szóorientált jelölésrendszer (1961)

A későbbiekben leginkább a *Turing-gépet* használták az algoritmuselméleti kutatásokban, talán azért, mert a definíciója bár matematikailag teljesen egzakt, mégsem annyira elvont, mint az egyéb konstrukciók esetén. Egy Turing-gépet könnyű magunk elé képzelnünk, könnyű párhuzamot találnunk közte és korunk digitális számítógépei között. Azt egyébként bizonyították, hogy a fent felsorolt algoritmusfogalmak *mindegyike ekvivalens* Turing algoritmusfogalmával.

1.1. Definíció. (Turing-gép)
Egy Turing-gép egy $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ ötösként adható meg, ahol:

- Σ a *betűk* (szalagjelek) véges halmaza, az ún. *szalagábécé*. Kötelezően $\triangleright, \sqcup \in \Sigma$.
- Q az *állapotok* véges halmaza.
- $q_0 \in Q$ a *kezdőállapot*.
- $F \subset Q$ az *elfogadó állapotok* halmaza.
- $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{\leftarrow, -, \rightarrow\}$ az *átmenetfüggvény*, ahol megköveteljük:

  ha $\delta(q, \triangleright) = (q', \sigma, m)$, akkor $\sigma = \triangleright$ és $m = \rightarrow$. □

Hogy a Turing-gép működését leírjuk, szükségünk lesz arra, hogy az adott gép pillanatnyi *konfigurációját* és a *konfigurációváltásokat* valamilyen formában kifejezzük.

1.2. Definíció. (Konfiguráció)
Legyen $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ Turing-gép. A T *konfigurációja* egy (q, u, v) hármas, ahol $q \in Q$ és $u, v \in \Sigma^*$. Legyen $\sigma \in \Sigma$. A $(q, u\sigma, v)$ konfigurációt *megállási konfigurációnak* nevezzük, ha $\delta(q, \sigma)$ nincs értelmezve.

A konfigurációk között értelmezzük a $\xrightarrow{T}$ ún. *közvetlen rákövetkezési relációt*. Azt mondjuk, hogy

$$(q, u\sigma, v) \quad \xrightarrow{T} \quad (q', u', v'),$$

ha a következő teljesül: $\delta(q, \sigma) = (q', \sigma', m)$, és

(1) ha $m = -$, akkor $u' = u\sigma'$ és $v' = v$.

(2) ha $m = \leftarrow$, akkor $u' = u$ és $v' = \sigma' v$.

(3) ha $m = \rightarrow$, akkor:

- ha $v = \epsilon$, akkor $u' = u\sigma' \sqcup$ és $v' = \epsilon$;
- egyébként: $u' = u\sigma'\varrho$ és $v' = w$, ahol $\varrho \in \Sigma$ és $v = \varrho w$.

A konfigurációk között értelmezzük a $\xrightarrow{T^t}$ ún. *rákövetkezési relációt* is. Valamely $t \geq 0$ esetén azt mondjuk, hogy

$$(q, u, v) \quad \xrightarrow{T^t} \quad (q', u', v'),$$

ha léteznek olyan (q_i, u_i, v_i) konfigurációk, ahol $i = 1, \ldots, t$, hogy

(1) $(q_1, u_1, v_1) = (q, u, v)$

(2) $(q_t, u_t, v_t) = (q', u', v')$

(3) $(q_i, u_i, v_i) \xrightarrow{T} (q_{i+1}, u_{i+1}, v_{i+1})$, minden $i = 1, \ldots, t-1$-re. □

Most már összpontosíthatunk a Turing-gép megállásának pillanatára: milyen válasszal/kimenettel állhat meg a Turing-gép? És ha már megállt: mennyi idő alatt teszi ezt meg?

1.3. Definíció. (Elfogadás, elutasítás, időkorlát)

A $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ Turing-gép *inputján* egy $x \in \big(\Sigma \backslash \{\triangleright, \sqcup\}\big)^*$ szót értünk.

Ha van olyan $(q, \triangleright u, v)$ megállási konfiguráció, hogy

$$(q_0, \triangleright, x) \quad \xrightarrow{T^t} \quad (q, \triangleright u, v),$$

akkor két esetet különböztetünk meg:

- Ha $q \notin F$, akkor T *elutasítja* x-et.
- Ha $q \in F$, akkor T *elfogadja* x-et; ekkor az előállított *kimenet* uv.

A T *időigénye* az x inputon t.

A T $f(n)$ *időkorlátos*, ha T időigénye minden x inputon legfeljebb $f\big(|x|\big)$.

Az $f(n)$ (időbonyolultsági) függvénnyel kapcsolatban a továbbiakban kikötjük, hogy

$$f(n) \quad \geq \quad n$$

minden $n \in \mathbb{N}$-re. □

A fenti utolsó kikötést a következő miatt tesszük: minden „tisztességes” időkorlát eleget tesz az $f(n) \geq n$ feltételnek, hiszen a Turing-gépnek legalább n lépésre van szüksége ahhoz, hogy egyáltalán *végigolvassa az inputot*.

Most jött el annak a pillanata, hogy a Turing-gépeket összekössük a megoldandó problémákkal. Először is: ezen problémákat hogyan írjuk le? Erre egy szóorientált megközelítést alkalmazunk: a problémákat mint *nyelveket* (azaz szavak halmazait) fogjuk reprezentálni. Ezek után jöhet annak a vizsgálata, hogy egy Turing-gép milyen eredményt produkál a probléma (mint nyelv) egyes példányain (mint szavakon).

1.4. Definíció. (Eldöntött és felismert nyelv)

A Σ szalagábécéjű T Turing-gép *eldönti* az $L \subseteq \big(\Sigma\backslash\{\triangleright, \sqcup\}\big)^*$ nyelvet, ha minden $x \in \big(\Sigma\backslash\{\triangleright, \sqcup\}\big)^*$ inputra

- ha $x \in L$, akkor T elfogadja x-et;
- ha $x \notin L$, akkor T elutasítja x-et.

A T *felismeri* az L-t, ha minden x inputra

- ha $x \in L$, akkor T elfogadja x-et;
- ha $x \notin L$, akkor T nem áll meg. □

1.5. Megjegyzés.

A T Turing-gép által felismert nyelv definíciójában megengedhettük volna azt a lehetőséget is, hogy ha $x \notin L$, akkor a T akár el is utasíthassa x-et. Egyes könyvekben ebben a definícióban valóban megengedik ezt a lehetőséget is; más könyvekben azonban $x \notin L$ esetén mindenképpen végtelen ciklusba kerülést írnak elő. Ennek az az oka, hogy jobban ki szeretnénk hangsúlyozni a *felismerés* fogalmában rejlő aszimmetriát. Vegyük észre, hogy egy olyan Turing-gépből, mely egyes inputjait elutasítja, könnyedén csinálhatunk olyat, mely ezeken az inputon végtelen ciklusba kerül: csupán egy q_c „csapda"-állapotba kell ilyen esetekben a Turing-gépet billentenünk, azaz q_c legyen olyan, hogy minden $\sigma \in \Sigma$-ra $\delta(q_c, \sigma) = (q_c, \sigma, -)$. □

1.6. Példa.

Tekintsük a *palindrómák* (tükörszavak) nyelvét az $\{\mathtt{a}, \mathtt{b}\}$ ábécé felett. Például a `bab` és az `abba` palindrómák, de a `baba` nem az. Az ϵ is tükörszó.

Adjunk meg egy $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ Turing-gépet, mely *eldönti* a palindrómák nyelvét:

- $\Sigma = \{\mathtt{a}, \mathtt{b}, \triangleright, \sqcup\}$
- $Q = \{q_0, q_a, q_b, t_a, t_b, r, f\}$
- $F = \{f\}$, azaz egyetlen elfogadó állapotunk van.
- A δ átmenetfüggvényt a következő táblázattal adjuk meg, ahol a sorokban az állapotok, az oszlopokban pedig a szalagjelek láthatók. Minden cellában a δ által az adott állapothoz és szalagjelhez rendelt hármas van feltüntetve:

	$\triangleright$	`a`	`b`	$\sqcup$
q_0	$(q_0, \triangleright, \rightarrow)$	$(q_\mathtt{a}, \triangleright, \rightarrow)$	$(q_\mathtt{b}, \triangleright, \rightarrow)$	$(f, \sqcup, -)$
$q_\mathtt{a}$		$(q_\mathtt{a}, \mathtt{a}, \rightarrow)$	$(q_\mathtt{a}, \mathtt{b}, \rightarrow)$	$(t_\mathtt{a}, \sqcup, \leftarrow)$
$q_\mathtt{b}$		$(q_\mathtt{b}, \mathtt{a}, \rightarrow)$	$(q_\mathtt{b}, \mathtt{b}, \rightarrow)$	$(t_\mathtt{b}, \sqcup, \leftarrow)$
$t_\mathtt{a}$	$(f, \triangleright, \rightarrow)$	$(r, \sqcup, \leftarrow)$		
$t_\mathtt{b}$	$(f, \triangleright, \rightarrow)$		$(r, \sqcup, \leftarrow)$	
r	$(q_0, \triangleright, \rightarrow)$	$(r, \mathtt{a}, \leftarrow)$	$(r, \mathtt{b}, \leftarrow)$	

Először is figyeljük meg, hogy az 1. oszlopban szereplő hármasok eleget tesznek a Turing-gép definíciójának végén megfogalmazott kikötésnek.

A Turing-gép a q_0 állapotból indulva balról jobbra halad a szalagon az első `a`, illetve `b` betűig, mely olvasásakor a $q_\mathtt{a}$, illetve a $q_\mathtt{b}$ állapotba megy át (az 1. sor 2. és 3. celláiban). Így –úgymond– az állapotban „tároljuk" a szó bal szélén található betűt. A $q_\mathtt{a}$, illetve a $q_\mathtt{b}$ állapotban a további betűk felülírása nélkül elmozgatjuk a fejet a szó jobb széléig, melyhez érve a $t_\mathtt{a}$, illetve a $t_\mathtt{b}$ állapotba megyünk át (a 2. és a 3. sor utolsó celláiban). A $t_\mathtt{a}$ állapotban csupán a szó utolsó betűjére vagyunk kíváncsiak: ha ez `a`, akkor átmegyünk az r állapotba; ha viszont ez `b`, akkor leállunk, hiszen a δ a $(t_\mathtt{a}, \mathtt{b})$ páron nincsen értelmezve (4. sor 3. cella). Hasonlóképp cselekszünk a $t_\mathtt{b}$ állapotban. Az r állapot feladata az, hogy a fejet a szó bal szélére mozgassuk, majd visszaálljunk a q_0 állapotra, azaz a következő ciklus elejére.

Figyeljük meg a következőket:

- A q_0 állapotban a, illetve b betűt olvasva a betű helyére a $\triangleright$ szimbólumot írjuk (az 1. sor 2. és 3. celláiban), azaz a szó elején álló betűt töröljük (és az állapotban tároljuk).
- Hasonlóképpen, a szó végén álló betűt is töröljük (4. sor 2. cella, illetve 5. sor 3. cella), ha az megegyezett a szó elején olvasott betűvel.
- Csak 2 esetben jutunk elfogadó állapotba:
  - Ha a szó már ϵ-ra rövidült (1. sor utolsó cellája); ekkor az input páros számú betűt tartalmazott.
  - Ha a szó végéről való „visszafordulás" után rögtön $\triangleright$ szimbólumot olvasunk (3. és 4. sorok 1. cellája); akkor az input páratlan számú betűből állt.
- Az f állapothoz nem vettünk fel sort a táblázatba, azaz ha ebbe az állapotba kerülünk, rögtön megállunk.

Most nézzük meg a Turing-gép számítási sorozatát (konfigurációinak sorozatát) a bab inputon:

$$(q_0, \triangleright, \mathtt{bab}) \xrightarrow{T} (q_0, \triangleright\mathtt{b}, \mathtt{ab}) \xrightarrow{T} (q_\mathtt{b}, \triangleright \triangleright \mathtt{a}, \mathtt{b}) \xrightarrow{T} (q_\mathtt{b}, \triangleright \triangleright \mathtt{ab}, \epsilon) \xrightarrow{T} (q_\mathtt{b}, \triangleright \triangleright \mathtt{ab}\sqcup, \epsilon) \xrightarrow{T}$$
$$(t_\mathtt{b}, \triangleright \triangleright \mathtt{ab}, \sqcup) \xrightarrow{T} (r, \triangleright \triangleright \mathtt{a}, \sqcup\sqcup) \xrightarrow{T} (r, \triangleright\triangleright, \mathtt{a} \sqcup \sqcup) \xrightarrow{T} (q_0, \triangleright \triangleright \mathtt{a}, \sqcup\sqcup) \xrightarrow{T}$$
$$(q_\mathtt{a}, \triangleright \triangleright \triangleright\sqcup, \sqcup) \xrightarrow{T} (t_\mathtt{a}, \triangleright \triangleright \triangleright, \sqcup\sqcup) \xrightarrow{T} (f, \triangleright \triangleright \triangleright\sqcup, \sqcup)$$

A baba inputon ez a következő:

$$(q_0, \triangleright, \mathtt{baba}) \xrightarrow{T} (q_0, \triangleright\mathtt{b}, \mathtt{aba}) \xrightarrow{T} (q_\mathtt{b}, \triangleright \triangleright \mathtt{a}, \mathtt{ba}) \xrightarrow{T} (q_\mathtt{b}, \triangleright \triangleright \mathtt{ab}, \mathtt{a}) \xrightarrow{T}$$
$$(q_\mathtt{b}, \triangleright \triangleright \mathtt{aba}, \epsilon) \xrightarrow{T} (q_\mathtt{b}, \triangleright \triangleright \mathtt{aba}\sqcup, \epsilon) \xrightarrow{T} (t_\mathtt{b}, \triangleright \triangleright \mathtt{aba}, \sqcup)$$

Ezen az inputon a Turing-gép a t_b állapotban áll meg. Mivel $t_b \notin F$, így a Turing-gép elutasítja a baba inputot.

A T időigénye akkor a legnagyobb, ha az input palindróma. Egy n hosszúságú palindróma esetén a számítási lépések száma a következőképpen számolható ki:

$$(n+1) + n + (n-1) + (n-2) + \cdots + 3 + 2 \quad = \quad \frac{(n+3) \cdot n}{2}$$

Vagyis a T $\frac{(n+3)\cdot n}{2}$ időkorlátos. □

1.7. Példa.

Adjunk meg egy T Turing-gépet, mely az inputként kapott bináris természetes számot megnöveli eggyel! Azaz a T-nek most a szalagján outputot is elő kell állítania. A T összetevői a következők:

- $\Sigma = \{0, 1, \triangleright, \sqcup\}$
- $Q = \{q_0, r, q_1, s, f\}$
- $F = \{f\}$, azaz egyetlen elfogadó állapotunk van.
- A δ átmenetfüggvényt a következő táblázattal adjuk meg:

	$\triangleright$	0	1	$\sqcup$
q_0	$(q_0, \triangleright, \rightarrow)$	$(q_0, 0, \rightarrow)$	$(q_0, 1, \rightarrow)$	$(r, \sqcup, \leftarrow)$
r	$(q_1, \triangleright, \rightarrow)$	$(f, 1, -)$	$(r, 0, \leftarrow)$	
q_1		$(s, 1, \rightarrow)$		
s		$(s, 0, \rightarrow)$		$(f, 0, -)$

Most nézzük meg a Turing-gép számítási sorozatát az 101 inputon:

$$(q_0, \triangleright, 101) \xrightarrow{T} (q_0, \triangleright 1, 01) \xrightarrow{T} (q_0, \triangleright 10, 1) \xrightarrow{T} (q_0, \triangleright 101, \epsilon) \xrightarrow{T} (q_0, \triangleright 101\sqcup, \epsilon) \xrightarrow{T}$$
$$(r, \triangleright 101, \sqcup) \xrightarrow{T} (r, \triangleright 10, 0\sqcup) \xrightarrow{T} (f, \triangleright 11, 0\sqcup)$$

Tehát az előállított output 110.

Most nézzük a gép működését az 11 inputon:

$$(q_0, \triangleright, 11) \xrightarrow{T} (q_0, \triangleright 1, 1) \xrightarrow{T} (q_0, \triangleright 11, \epsilon) \xrightarrow{T} (q_0, \triangleright 11\sqcup, \epsilon) \xrightarrow{T}$$
$$(r, \triangleright 11, \sqcup) \xrightarrow{T} (r, \triangleright 1, 0\sqcup) \xrightarrow{T} (r, \triangleright, 00\sqcup) \xrightarrow{T} (q_1, \triangleright 0, 0\sqcup) \xrightarrow{T}$$
$$(s, \triangleright 10, \sqcup) \xrightarrow{T} (s, \triangleright 10\sqcup, \epsilon) \xrightarrow{T} (s, \triangleright 100, \epsilon)$$ □

1.1. Többszalagos Turing-gépek

Vajon az előző fejezetben definiált (egyszalagos) Turing-gép elegendő eszköz-e az „algoritmus" definiálására? Ha bizonyos pontokon általánosítunk a konstrukción, nem nyerünk-e nagyobb számítási erejű gépet? Be fogjuk látni, hogy a következőkben megismerendő *többszalagos Turing-gépek* a teljesítmény számottevő romlása nélkül *szimulálhatók egyszalagos Turing-géppel.* Ez az eredmény az egyik első lépés annak megmutatására, hogy igazoljuk a Turing-gép meglepő számítási erejét.

1.8. Definíció. (k-szalagos Turing-gép)

Egy k-szalagos Turing-gép egy $\langle k, \Sigma, Q, q_0, F, \delta \rangle$ hatosként adható meg, ahol:

- $k \geq 1$ egész szám a szalagok száma.
- Σ, Q, q_0, F ugyanazok, mint az 1.1. definícióban.
- $\delta : Q \times \Sigma^k \mapsto Q \times \big(\Sigma \times \{\leftarrow, -, \rightarrow\}\big)^k$ az átmenetfüggvény, ahol megköveteljük:

  ha $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k) = (q', \sigma_1', m_1, \sigma_2', m_2, \ldots, \sigma_k', m_k)$, akkor
  ha $\sigma_i = \triangleright$, akkor $\sigma_i' = \triangleright$ és $m_i = \rightarrow$.

□

1.9. Definíció. (Konfiguráció)

Legyen $T = \langle k, \Sigma, Q, q_0, F, \delta \rangle$ k-szalagos Turing-gép. A T *konfigurációja* egy $(q, u_1, v_1, \ldots, u_k, v_k)$ vektor, ahol $q \in Q$ és $u_i, v_i \in \Sigma^*$.

Legyen $\sigma_1, \ldots, \sigma_k \in \Sigma$. A $(q, u_1\sigma_1, v_1, \ldots, u_k\sigma_k, v_k)$ konfigurációt *megállási konfigurációnak* nevezzük, ha $\delta(q, \sigma_1, \ldots, \sigma_k)$ nincs értelmezve.

A konfigurációk között értelmezzük a $\xrightarrow{T}$ ún. *közvetlen rákövetkezési relációt.* Azt mondjuk, hogy

$$(q, u_1\sigma_1, v_1, \ldots, u_k\sigma_k, v_k) \quad \xrightarrow{T} \quad (q', u_1', v_1', \ldots, u_k', v_k'),$$

ha a következő teljesül: $\delta(q, \sigma_1, \ldots, \sigma_k) = (q', \sigma_1', m_1, \ldots, \sigma_k', m_k)$, és

(1) ha $m_i = -$, akkor $u_i' = u_i\sigma_i'$ és $v_i' = v_i$.

(2) ha $m_i = \leftarrow$, akkor $u_i' = u_i$ és $v_i' = \sigma_i' v_i$.

(3) ha $m_i = \rightarrow$, akkor:

- ha $v_i = \epsilon$, akkor $u'_i = u_i\sigma'_i\sqcup$ és $v'_i = \epsilon$;
- egyébként: $u'_i = u_i\sigma'_i\varrho_i$ és $v'_i = w_i$, ahol $\varrho_i \in \Sigma$ és $v_i = \varrho_i w_i$.

A konfigurációk között értelmezzük a $\xrightarrow{T^t}$ ún. *rákövetkezési relációt* is. Valamely $t \geq 0$ esetén azt mondjuk, hogy

$$(q, u_1, v_1, \ldots, u_k, v_k) \quad \xrightarrow{T^t} \quad (q', u'_1, v'_1, \ldots, u'_k, v'_k),$$

ha léteznek olyan $(q_i, u_1^i, v_1^i, \ldots, u_k^i, v_k^i)$ konfigurációk, ahol $i = 1, \ldots, t$, hogy

(1) $(q_1, u_1^1, v_1^1, \ldots, u_k^1, v_k^1) = (q, u_1, v_1, \ldots, u_k, v_k)$

(2) $(q_t, u_1^t, v_1^t, \ldots, u_k^t, v_k^t) = (q', u'_1, v'_1, \ldots, u'_k, v'_k)$

(3) $(q_i, u_1^i, v_1^i \ldots, u_k^i, v_k^i) \xrightarrow{T} (q_{i+1}, u_1^{i+1}, v_1^{i+1}, \ldots, u_k^{i+1}, v_k^{i+1})$, minden $i = 1, \ldots, t-1$-re. □

1.10. Definíció. (Elfogadás, elutasítás, időkorlát)

A $T = \langle k, \Sigma, Q, q_0, F, \delta\rangle$ k-szalagos Turing-gép *inputján* egy $x \in \big(\Sigma\backslash\{\triangleright, \sqcup\}\big)^*$ szót értünk.

Ha van olyan $(q, \triangleright u_1, v_1, \ldots, \triangleright u_k, v_k)$ megállási konfiguráció, hogy

$$(q_0, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon) \quad \xrightarrow{T^t} \quad (q, \triangleright u_1, v_1, \ldots, \triangleright u_k, v_k),$$

akkor két esetet különböztetünk meg:

- Ha $q \notin F$, akkor T *elutasítja* x-et.
- Ha $q \in F$, akkor T *elfogadja* x-et; ekkor az előállított *kimenet* $u_k v_k$.

A T x inputon való *időigényének*, illetve T *időkorlátjának* definíciói ugyanazok, mint egyszalagos esetben. □

1.11. Megjegyzés.

Minden egyszalagos Turing-gép (1.1. definíció) olyan k-szalagos Turing-gép (1.8. definíció), melyre speciálisan $k = 1$. □

1.12. Példa.

A *palindrómák* nyelvének eldöntésére az 1.6. példában megadtunk egy egyszalagos Turing-gépet. Most megadunk egy másik, ezúttal 2-szalagos T Turing-gépet:

- $\Sigma = \{\mathtt{a}, \mathtt{b}, \triangleright, \sqcup\}$
- $Q = \{q_0, r, t, f\}$
- $F = \{f\}$

- Többszalagos Turing-gépek átmenetfüggvényét táblázatos formában leírni kissé körülményes (jelen esetben 3-dimenziós táblázatot kapnánk), ezért a δ átmenetfüggvény most így adjuk meg:

$$\begin{aligned}
\delta(q_0, \triangleright, \triangleright) &= (q_0, \triangleright, \rightarrow, \triangleright, \rightarrow) \\
\delta(q_0, \mathtt{a}, \sqcup) &= (q_0, \mathtt{a}, \rightarrow, \mathtt{a}, \rightarrow) \\
\delta(q_0, \mathtt{b}, \sqcup) &= (q_0, \mathtt{b}, \rightarrow, \mathtt{b}, \rightarrow) \\
\delta(q_0, \sqcup, \sqcup) &= (r, \sqcup, \leftarrow, \sqcup, -) \\
\\
\delta(r, \mathtt{a}, \sqcup) &= (r, \mathtt{a}, \leftarrow, \sqcup, -) \\
\delta(r, \mathtt{b}, \sqcup) &= (r, \mathtt{b}, \leftarrow, \sqcup, -) \\
\delta(r, \triangleright, \sqcup) &= (t, \triangleright, \rightarrow, \sqcup, \leftarrow) \\
\\
\delta(t, \mathtt{a}, \mathtt{a}) &= (t, \mathtt{a}, \rightarrow, \mathtt{a}, \leftarrow) \\
\delta(t, \mathtt{b}, \mathtt{b}) &= (t, \mathtt{b}, \rightarrow, \mathtt{b}, \leftarrow) \\
\delta(t, \sqcup, \triangleright) &= (f, \sqcup, \leftarrow, \triangleright, \rightarrow)
\end{aligned}$$

  A q_0 állapot feladata az 1. szalagon található input átmásolása a 2. szalagra. Ezután a Turing-gép az r állapotba kerül, melyben az 1. szalag fejét visszavisszük az input elejére (a 2. szalag feje marad a másolat-szó végén).

  Ezután a t állapot gondoskodik az input palindróma-voltának a vizsgálatáról: az 1. szalagon balról jobbra, a 2. szalagon jobbról balra mozgatja a fejet, és összeveti a fejek alatt található betűkre. Bármikor eltérést tapasztal, a Turing-gép megáll, mégpedig a t állapotban ($t \notin F$). Csak akkor kerülünk az $f \in F$ állapotba, ha sikeresen elértünk az 1. szalagon található szó végére, illetve –szimultán módon– a 2. szalagon található szó elejére. Ekkor az input palindróma volt.

  A T időigénye palindróma input esetén a legnagyobb. Egy n hosszúságú palindróma input esetén a Turing-gép $3n + 3$ lépést után áll meg. Más szavakkal: a T $3n + 3$ időkorlátos. □

Míg a fenti példában megadott kétszalagos Turing-gép és az 1.6. példában megadott egyszalagos Turing-gép ugyanazt a nyelvet döntik el, addig a kétszalagos gép időkorlátja egy elsőfokú polinom, az egyszalagosé pedig egy másodfokú polinom. A következő tétel pontosan egy ilyesfajta négyzetes összefüggést ír le a többszalagos és az egyszalagos Turing-gépek időkorlátja között.

1.13. Definíció. (Szimuláció)

Egy T' Turing-gép szimulál egy T Turing-gépet, ha minden x input esetén

- ha T nem áll meg, akkor T' sem áll meg;
- ha T elutasítja x-et, akkor T' is elutasítja x-et;
- ha T elfogadja x-et és az y kimenetet állítja elő, akkor T' is elfogadja x-et és az y kimenetet állítja elő. □

1.14. Tétel.

Bármely $f(n)$ időkorlátos (többszalagos) Turing-gép szimulálható $\mathcal{O}\left(f^2(n)\right)$ időkorlátos egyszalagos Turing-géppel.

Bizonyítás.

Legyen $T = \langle k, \Sigma, Q, q_0, F, \delta \rangle$ a szimulálandó k-szalagos Turing-gép. Ekkor a $T' = \langle \Sigma', Q', q_0, F, \delta' \rangle$ egyszalagos Turing-gép a következőképpen működik.

A T k db. szalagját a T' *egyetlen szalagon* fogja kódolni. Ez oly módon lesz megoldható, hogy k db. Σ-beli betű fog a T' egy-egy szalagjelében elhelyezkedni, valamint k db. járulékos információ arról, hogy T k db. feje vajon mely szalagcellákon állna. Ezt az adott cellában levő szimbólum alatt egy _ aláhúzás-jellel fogjuk jelölni; ennek hiányában üres szimbólum fog ott állni. Tehát: $\Sigma' = \Sigma \cup \big(\Sigma \times \{\sqcup, \_\}\big)^k$.

A szimuláció 3 fázisban történik:

(1) Kezdetben a T' az x input minden σ betűjét a $(\sigma, \sqcup, \sqcup, \sqcup, \ldots, \sqcup, \sqcup)$ szimbólummal írja felül, valamint a $\triangleright$ szimbólumot a $(\triangleright, \_, \triangleright, \_, \ldots, \triangleright, \_)$ szimbólummal, majd a fejet erre a szimbólumra pozicionálja. Ez az inicializáló fázis $\mathcal{O}(|x|)$ lépést igényel.

(2) Elkezdődik a tényleges szimuláció. A T egyetlen lépésének szimulálásához a T' végigolvassa a szalagon található szót teljes egészében balról jobbra, majd visszafelé. A balról jobbra való haladás során azon Σ-beli szalagjeleket, melyek alatt közvetlenül a _ szimbólum áll, eltárolja a belső állapotában. Ehhez T'-nek összetett állapotokkal kell rendelkeznie: $\big(Q \times \Sigma^k\big) \subseteq Q'$. Mivel a T' által a szalagon eddig használt szalagcellák száma legfeljebb $f(|x|)$ lehet, így ez a lépés $\mathcal{O}(f(|x|))$ lépésben elvégezhető.

Miután a T' végighaladt a szón, a $(q, \sigma_1, \ldots, \sigma_k)$ alakú belső állapota és δ alapján meg tudja állapítani, milyen változtatásokat kell elvégeznie a szón; ezt természetesen a $\delta(q, \sigma_1, \ldots, \sigma_k)$ írja le. Ezeket a változtatásokat a jobbról balra haladáskor végzi el. Ennek során a _ szimbólum felett álló jeleket helyettesíti a δ által meghatározottakkal, illetve a δ által leírt fejmozgásoknak megfelelően mozgatja el balra vagy jobbra a _ jeleket (vagy helyben hagyja őket). Ez a lépés is $\mathcal{O}(f(|x|))$ lépésben végezhető el.

A T összes, legfeljebb $f(|x|)$ lépését tehát T' $\mathcal{O}\big(f^2(|x|)\big)$ lépésben tudja szimulálni.

(3) Végül el kell végezni a szalagon található szó „visszakódolását" Σ-beli jelekre. A T utolsó szalagján található az output, tehát T' szalagján minden $(\sigma_1, h_1, \sigma_2, h_2, \ldots, \sigma_k, h_k)$ szimbólumot át kell írni σ_k-ra ($\sigma_i \in \Sigma$, $h_i \in \{\sqcup, \_\}$). Ez $\mathcal{O}(|x|)$ lépést igényel.

A T' időigénye az x inputon összesen $\mathcal{O}\big(f^2(|x|)\big)$. □

1.1.1. *Időbonyolultsági osztályok*

A nyelveket időbonyolultsági osztályokba soroljuk. Ehhez általában a többszalagos Turing-gép fogalmát szokták alapul venni.

1.15. Definíció.

Egy L nyelvre azt mondjuk, hogy $L \in \mathbf{TIME}(f(n))$, ha van olyan $\mathcal{O}(f(n))$ időkorlátos többszalagos Turing-gép, mely *eldönti* L-t. □

1.16. Példa.

Jelöljük az ($\{\mathtt{a}, \mathtt{b}\}$ ábécé feletti) palindrómák nyelvét PAL-lal. Két Turing-gépet adtunk meg a PAL eldöntésére. Az 1.6. példában megadott Turing-gép $\frac{(n+3)\cdot n}{2}$ időkorlátos volt, azaz $\text{PAL} \in \mathbf{TIME}\big(n^2\big)$. Az 1.12. példa Turing-gépe $3n+3$ időkorlátos volt, azaz $\text{PAL} \in \mathbf{TIME}(n)$ is teljesül. Nyilván elegendő csak az utóbbi összefüggést ismernünk, ugyanis a $\text{PAL} \in \mathbf{TIME}(n)$-ből következik a $\text{PAL} \in \mathbf{TIME}\big(n^2\big)$ is. □

A következő tétel azt fogalmazza meg, hogy bármelyik Turing-gép tetszőleges, *lineáris* mértékben felgyorsítható:

1.17. Tétel. (Lineáris felgyorsítás)

Legyen L egy $f(n)$ időkorlátos Turing-géppel eldönthető nyelv. Bármely $c > 0$ valós számra belátható, hogy L eldönthető

$$c \cdot f(n) + n + 2$$

időkorlátos Turing-géppel is.

BIZONYÍTÁS.

Legyen $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ az L-t eldöntő $f(n)$ időkorlátos (egyszalagos) Turing-gép. Ekkor a T lépéseit szimuláló T' legyen kétszalagos, azaz $T' = \langle 2, \Sigma', Q', q_0, F, \delta' \rangle$. A részletek kidolgozásához legyen adott egy később meghatározandó $m \geq 1$, csak c-től függő egész szám. T lépéseinek szimulálása 3 fázisban történik:

(1) A T' szalagábécéjébe bevesszük a T szalagjeleiből képzett m-eseket, azaz: $\Sigma' = \Sigma \cup \Sigma^m$. Következik a ($\Sigma$-beli betűkből álló) input kódolása Σ^m-beli betűkre. Ehhez T' végigolvassa az első szalagján található x inputot, és annak minden m db. $\sigma_1, \ldots, \sigma_m \in \Sigma$ betűjének olvasása után a 2. szalagra kiírja az egyetlen $(\sigma_1, \ldots, \sigma_m) \in \Sigma^m$ betűt. Ha az input hossza nem osztható m-mel, akkor az utolsó kiírandó m-es végéhez $\sqcup$ szimbólumokat fűz.

A kódolás végrehajtásához a Q'-nek tartalmaznia kell a Σ^i halmaz elemeit, minden $i = 1, \ldots, m$-re. A kódolás maga $|x| + 2$ lépésben elvégezhető.

Hátra van még a 2. szalagon szereplő kódolt input átmásolása az 1. (input) szalagra, amit az 1. és 2. szalagok fejeinek jobbról balra mozgatásával fogunk kivitelezni, majd az 1. szalagon a kódolt input elé kiírunk egy $\triangleright$ szimbólumot. Ez a másolási szakasz $\left\lceil \frac{|x|}{m} \right\rceil$ lépést igényel.

(2) Ekkor kezdődik T lépéseinek a tényleges szimulációja. A T' legfeljebb 6 lépésből álló szakaszokban szimulálja a T m db. egymást követő lépését. Ehhez a Q'-höz hozzávesszük a $Q \times \{1, \ldots, m\}$ halmaz elemeit, mégpedig azért, hogy a T' belső állapotaiban azt is tároljuk, hogy a T' szalagján a fej alatt található (összetett) betűn belül pontosan hanyadik betű felett állna a T feje.

A T' fejét most eggyel balra, aztán kettővel jobbra, majd eggyel balra mozgatjuk, azaz 4 lépésben bejárjuk a szomszédos cellákat, és visszajutunk az eredeti cellába. A lényeg, hogy eközben az ezen cellákon található betűket ($3m$ db. Σ-beli betűt) letároljuk a T' belső állapotában. Ehhez hozzá kell vennünk a Q'-höz a $Q \times \{1, \ldots, m\} \times \Sigma^{3m}$ halmaz elemeit.

A lényeg, hogy ezek után a belső állapotban tárolt információk alapján T' el tudja dönteni, hogy T mit lépne a következő m db. lépése során. Ez azért van így, mert m lépésben a T feje nem léphet ki az aktuális betű-m-es és a vele közvetlenül szomszédos betű-m-esek által meghatározott területről. Tehát már csak az van hátra, hogy T' elvégezze a T m db. lépésének megfelelő változtatásokat a szalagon és a belső állapotban. Ehhez legfeljebb 2 lépésre van szükség, mivel a T m lépése a T'-nek csak az aktuális celláját, illetve az egyik vele szomszédos cellát érinthetik.

Ily módon a T' tényleg legfeljebb 6 lépésben szimulálja a T m db. lépését.

(3) Ha T nem áll meg az x-en, akkor természetesen T' sem fog megállni. Ha T elutasítja x-et, akkor T' is utasítsa el x-et.

Tehát a T' időigénye a x inputon legfeljebb

$$|x| + 2 + \left\lceil \frac{|x|}{m} \right\rceil + 6 \left\lceil \frac{f(|x|)}{m} \right\rceil \quad \leq \quad |x| + 2 + 7 \left\lceil \frac{f(|x|)}{m} \right\rceil \ ,$$

vagyis az $m = \lceil \frac{7}{c} \rceil$ választás esetén teljesül a bizonyítandó állítás. □

A tétel jelentősége a következő: megerősítést nyert az $\mathcal{O}()$ jelölés használatának helyessége az időbonyolultsági osztályok definíciójában. A tétel tulajdonképpen azt fogalmazza meg, hogy a bonyolultsági függvényekben szereplő *multiplikatív konstansok* (és ezzel együtt tulajdonképpen az additív konstansok is) elhanyagolhatóak.

Ennek folyományaként készen állunk arra, hogy kiterjedtebb, nevezetes időbonyolultsági osztályokba soroljuk a nyelveket. Ilyen például a **P**, mely a *polinom* időkorlátos Turing-géppel eldönthető nyelvek osztályát takarja:

$$\mathbf{P} = \bigcup_{k\geq 1} \mathbf{TIME}\left(n^k\right)$$

Vegyük észre, hogy a **P** definíciójában a polinomokat a legmagasabb kitevőjű tagjuk (n^k) képviseli, melynek a multiplikatív együtthatóját fel sem tüntetjük. Ez az egyszerű definíciós forma tehát mind a lineáris felgyorsítás tételének köszönhető.

A **P**-re egyébként mint a hatékony algoritmussal eldönthető nyelvek osztályára kell tekintenünk. Ha egy nyelvről sikerült megmutatnunk, hogy benne van **P**-ben, akkor ez a számításelméletben annak bizonyítékát jelenti, hogy az adott nyelv eldöntésére létezik hatékony algoritmus.

1.18. Példa.

A PAL, azaz a palindrómák nyelve (mint fentebb megmutattuk) a $\mathbf{TIME}(n)$ osztály eleme. Ezért PAL $\in \mathbf{P}$. □

1.2. Lyukszalagos Turing-gépek

Eddig Turing-gépek időbonyolultságával foglalkoztunk. A lyukszalagos Turing-gépek létrehívásának célja egy másikfajta bonyolultsági fogalom, a *tárbonyolultság* bevezetése.

1.19. Definíció. (Lyukszalagos Turing-gép)

Lyukszalagos Turing-gép alatt egy $\langle k, \Sigma, Q, q_0, F, \delta\rangle$ k-szalagos Turing-gépet értünk ($k > 2$), ahol δ eleget tesz a következő kikötésnek: ha $\delta(q, \sigma_1, \ldots, \sigma_k) = (q', \sigma'_1, m_1, \ldots, \sigma'_k, m_k)$, akkor

(1) $\sigma'_1 = \sigma_1$ és

(2) $m_k \neq \leftarrow$. □

Az (1) azt fejezi ki, hogy az 1. (input) szalag csak olvasható, a (2) pedig azt, hogy az utolsó (output) szalagon nem lehet balra haladni.

1.20. Állítás.

Bármely $f(n)$ időkorlátos k-szalagos Turing-gép szimulálható $\mathcal{O}(f(n))$ időkorlátos $k+2$-szalagos lyukszalagos Turing-géppel.

Bizonyítás.

A bizonyítás triviális: a szimuláló Turing-gép kezdetben átmásolja az inputot a 2. szalagra, majd szimulálja a k-szalagos Turing-gépet a 2-től és a $k+1$-ig terjedő szalagokon. Majd végül a $k+1$. szalagon előállított outputot átmásolja a $k+2$. szalagra. □

Egy Turing-gépek *tárigényén* a Turing-gép által felhasznált szalagcellák számát fogjuk érteni. Ez a meghatározás azonban még csak nagyon elnagyolt, ugyanis ezt használva sok esetben felülbecsülnénk a tényleges tárigényt. A tárigény pontos definíciója a következő:

1.21. Definíció. (Tárkorlát)

Tegyük fel, hogy a T k-szalagos Turing-gépre és annak x inputjára teljesül:

$$(q_0, \rhd, x, \rhd, \epsilon, \ldots, \rhd, \epsilon) \quad \xrightarrow{T^t} \quad (q, u_1, v_1, \ldots, u_k, v_k)$$

valamely $(q, u_1, v_1, \ldots, u_k, v_k)$ megállási konfigurációra.

Ekkor T *tárigénye* az x inputon: $\sum_{i=1}^{k} |u_i v_i|$. Kivételt képez az az eset, mikor a T lyukszalagos. Ekkor a T tárigénye: $\sum_{i=2}^{k-1} |u_i v_i|$.

A T $f(n)$ *tárkorlátos*, ha T tárigénye minden x inputon legfeljebb $f(|x|)$. □

Mint a definícióból látható, lyukszalagos Turing-gépek esetén az input és az output szalagokat nem számítjuk bele a tárigénybe (hiszen ezek ténylegesen csak az input és az output tárolására valók, nem pedig a számításokhoz használandóak).

1.22. Példa.

Az 1.6. példában és az 1.12. példában a PAL eldöntésére adtunk meg Turing-gépeket. Az előbbi legfeljebb $n+2$ szalagcellát használt fel egy n hosszúságú palindróma inputon, az utóbbi pedig $2(n+2)$ cellát.

Ezek a Turing-gépek nem lyukszalagosak voltak, ámbár lyukszalagossá alakítva őket sem tudtuk volna a tárigényüket lecsökkenteni. Most megadunk egy harmadik Turing-gépet a PAL nyelv eldöntésére. Ez a Turing-gép lyukszalagos lesz, és a tárkorlátja $\mathcal{O}(n)$-nél lényegesen kisebb lesz.

Legyen ez a gép *4-szalagos*. Az 1. szalag a (végig változatlan) input szalag, a 4. szalag pedig a végig üres output szalag (mivel nem kell outputot szolgáltatnunk). A 2. szalagon egy i, a 3. szalagon pedig egy j számot fogunk tárolni:

- Az i jelentése: aktuálisan az input elölről i. betűjét akarjuk összevetni a hátulról i. betűjével (az input betűit 0-tól indexeljük). Az i értéke a Turing-gép indításakor 0-ról indul, majd minden sikeres összehasonlítás után 1-gyel növeljük.
- A j jelentése: aktuálisan az input előlről (vagy éppen hátulról) j. betűjén áll az input szalag feje.

A Turing-gép működése a következőképpen képzelhető el: kezdetben az i-t 0-ra állítjuk. Ezek után több ciklusra lehet bontani a gép működését. Minden egyes ciklus kezdetén az input szalag feje az input elején áll, továbbá a j-t 0-ra állítjuk, majd az input szalag fejének jobbra mozgatásával együtt növeljük addig, míg értéke el nem éri az i-t. Ekkor az input szalag feje alatt található betűt a Turing-gép állapotában „megjegyezzük", majd a fejet az input végére mozgatjuk. Ekkor a j-t ismét 0-ra állítjuk, majd az input szalag fejének balra mozgatásával együtt növeljük addig, amíg értéke el nem éri az i-t. Ekkor a fej alatt található betűt összevetjük az állapotban „tárolt" betűvel. Ha a kettő nem egyezik meg, elutasítjuk az inputot. Ha megegyeznek, akkor az i-t megnöveljük, és egy újabb ciklust indítunk. A Turing-gép akkor áll meg elfogadó állapotban, ha az inputnak már nincs i. betűje.

A legcélszerűbb az i és a j számoknak a *bináris* alakját tárolni, így az 1.7. példában megadott Turing-gép szimulációjával az i és a j inkrementálása megoldható.

Az így kapott Turing-gép tárigénye a 2. és a 3. szalagokon felhasznált cellák száma. A kérdés, hogy az i és a j tárolására mennyi cellát kell legfeljebb felhasználnunk? Mivel mind a két szám 0 és n közé esik (ahol n az input hossza), és mivel a bináris alakjukat tároljuk, így egy ilyen szám $\lceil \log_2(n+1) \rceil$ cellán tárolható. Ebből következik, hogy a Turing-gép *tárkorlátja* $\mathcal{O}(\log_2 n)$. □

1.2.1. Tárbonyolultsági osztályok

A nyelveket tárbonyolultsági osztályokba soroljuk. Ehhez általában a lyukszalagos Turing-gép fogalmát szokták alapul venni.

1.23. Definíció.

Egy L nyelvre azt mondjuk, hogy $L \in \mathbf{SPACE}(f(n))$, ha van olyan $\mathcal{O}(f(n))$ tárkorlátos lyukszalagos Turing-gép, mely *eldönti* L-t. □

Az időbonyolultsági osztályokhoz hasonlóan a tárbonyolultsági osztályok között is vannak összetett, nevezetes osztályok. Ilyen például a **PSPACE**, mely a *polinom* tárkorlátos Turing-géppel eldönthető nyelvek osztályát takarja:

$$\mathbf{PSPACE} \quad = \quad \bigcup_{k \geq 1} \mathbf{SPACE}\left(n^k\right)$$

Egy másik nagyon fontos tárbonyolultsági osztály az **L**, mely a *logaritmikus* tárkorlátos Turing-géppel eldönthető nyelvek osztálya:

$$\mathbf{L} \quad = \quad \mathbf{SPACE}\left(log_c n\right)$$

Egy nyelvnek **L**-be tartozni azt jelenti, hogy a nyelvet eldöntő algoritmusnak hatékony a tárfelhasználása.

1.24. Példa.

Az 1.6. és az 1.12. példák Turing-gépei $\mathcal{O}(n)$ tárkorlátosak; ezek alapján PAL $\in$ **PSPACE**. Az 1.22. példában szereplő Turing-gép $\mathcal{O}(\log_2 n)$ tárkorlátos, azaz PAL $\in$ **L**. Egyébként elég csak ez utóbbi összefüggést ismernünk, ugyanis **L** $\subseteq$ **PSPACE**, ahogy azt a 2.2.1. fejezetben be is fogjuk bizonyítani. □

1.3. Nemdeterminisztikus Turing-gépek

Hasonlóan az 1.1. fejezethez, most is megpróbáljuk a hagyományos (determinisztikus) Turing-gépet valamilyen téren továbbfejleszteni, csak hogy megvizsgáljuk, nyerünk-e valamit és mit nyerünk ezzel? A most megismerendő *nemdeterminisztikus Turing-gépek* egy teljesen irreális számítási modellt írnak le, mely Turing-gépekről meg fogjuk mutatni, hogy *exponenciális időveszteséggel szimulálhatók* determinisztikus Turing-géppel. Az teszi izgalmassá a nemdeterminisztikus modellt, hogy nagyon sok híres és alapvető probléma kezelhető könnyen a segítségével.

1.25. Definíció. (Nemdeterminisztikus Turing-gép)

Egy nemdeterminisztikus Turing-gép egy $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ ötösként adható meg[1], ahol:

- Σ, Q, q_0 és F ugyanazok, mint a közönséges (determinisztikus) Turing-gépek esetében voltak (lásd: 1.1. definíció).
- $\delta : Q \times \Sigma \mapsto P\big(Q \times \Sigma \times \{\leftarrow, -, \rightarrow\}\big)$ az átmenetfüggvény, ahol megköveteljük:

  ha $(q', \sigma, m) \in \delta(q, \triangleright)$, akkor $\sigma = \triangleright$ és $m = \rightarrow$. □

Láthatjuk, hogy a δ értékkészlete a $Q \times \Sigma \times \{\leftarrow, -, \rightarrow\}$ *hatványhalmaza*, azaz minden (q, σ) állapot-betű párhoz akár több következő lépés is tartozhat.

1.26. Definíció. (Konfiguráció)

A nemdeterminisztikus Turing-gépek *konfigurációjának* definíciója megegyezik a determinisztikus Turing-gépekével (lásd: 1.2. definíció).

[1] A nemdeterminisztikus Turing-gépeket lehetne *többszalagosként* is definiálni. Nekünk azonban egyszalagos változata is elegendő lesz; ráadásul az 1.14. tétel alapján már tudjuk, hogy minden többszalagos Turing-gép szimulálható egyszalagossal.

Legyen $\sigma \in \Sigma$. A $(q, u\sigma, v)$ konfigurációt *megállási konfigurációnak* nevezzük, ha $\delta(q, \sigma)$ nincs értelmezve vagy $\big|\delta(q, \sigma)\big| = \emptyset$.

A *közvetlen rákövetkezési reláció* fogalma némileg eltér a determinisztikus Turing-gépekétől: minden ugyanaz, mint az 1.2. definícióban, kivéve hogy a $\delta(q, \sigma) = (q', \sigma', m)$ feltétel helyett $(q', \sigma', m) \in \delta(q, \sigma)$-nak kell állnia.

Az ezen alapuló *rákövetkezési reláció* definíciója változatlan. □

Kissé különbözni fog a nemdeterminisztikus Turing-gépek elfogadás- és időkorlát fogalma is a determinisztikus Turing-gépekétől:

1.27. Definíció. (Elfogadás, elutasítás, időkorlát)

A $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ nemdeterminisztikus Turing-gép *inputján* egy $x \in \big(\Sigma \backslash \{\triangleright, \sqcup\}\big)^*$ szót értünk.

A T *elfogadja* x-et, ha *van olyan* (q, u, v) megállási konfiguráció, hogy

$$(q_0, \triangleright, x) \quad \xrightarrow{T^t} \quad (q, u, v)$$

és $q \in F$.

A T *elutasítja* x-et, ha nem fogadja el azt[2].

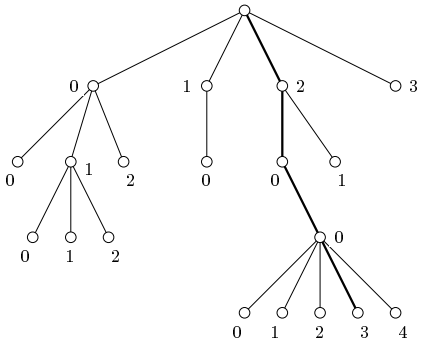
A T $f(n)$ *időkorlátos*, ha minden x input és *minden* (q, u, v) konfiguráció esetén ha $(q_0, \triangleright, x) \xrightarrow{T^t} (q, u, v)$, akkor $t \leq f\big(|x|\big)$. □

Mint látható, egy nemdeterminisztikus Turing-gép csak akkor fogadja el az x inputot, ha az x-en legalább egy számítási sorozata elfogadó állapotban végződik. Azaz x elfogadásához egy elfogadó számítási sorozatnak a *létezését* írjuk elő, azaz nem zárjuk ki, hogy x-en a Turing-gépnek legyenek akár elutasító számítási sorozatai is.

Akkor utasítja el x-et, ha *nincs* ilyen számítási sorozat, azaz a számítási sorozatok vagy nem elfogadó állapotban végződnek, vagy végtelen ciklusba kerülnek. Vegyük észre, hogy az elfogadás és az elutasítás fogalma lényegesen különbözik a determinisztikus Turing-gépekétől: még a végtelen ciklusba kerülés lehetőségét sem zárják ki magukból.

A végtelen ciklusok elkerülésének egyetlen módja valamilyen időkorlát használata lesz; az időkorláttal az *összes* számítási sorozat hosszát lekorlátozzuk, azaz végessé tesszük.

Egy nemdeterminisztikus Turing-gép valamely inputon megadható összes lehetséges számítási sorozatát a legkönnyebb az ún. *számítási fa* segítségével ábrázolni:



A fa csúcsaiban a Turing-gép konfiguráció szerepelnek. A gyökérben az induló konfiguráció található (az adott x input esetén). Egy a fában szereplő konfigurációnak a gyermekei az adott konfiguráció *közvetlen*

[2] Azaz ha *minden* olyan (q, u, v) megállási konfiguráció esetén, melyre $(q_0, \triangleright, x) \xrightarrow{T^t} (q, u, v)$, a következő teljesül: $q \notin F$.

rákövetkezői. Ennek alapján egy-egy számítási sorozatot a fa egy-egy ága reprezentál. A számítási fa segítségével könnyű az x elfogadását szemléltetni: ha van olyan ága a fának, mely elfogadó konfigurációban végződik, akkor a Turing-gép elfogadja az x-et. Ha nincs ilyen ága, akkor elutasítja; ekkor a számítási fa összes véges ágának elutasító konfigurációban kell végződnie. Könnyű a Turing-gép időkorlátosságát is szemléltetni: a gép $f(n)$ időkorlátos, ha minden lehetséges x inputhoz tartozó számítási fa legfeljebb $f(|x|)$ mélységű[3].

Egy nemdeterminisztikus Turing-gép által *eldöntött* nyelvet ugyanúgy definiálunk, mint deterministikus Turing-gépek esetén (lásd: 1.4. definíció). Nemdeterminisztikus Turing-gép által *felismert* nyelvet nincs értelme definiálni.

1.28. Megjegyzés.

Minden deterministikus Turing-gép nemdeterminisztikus is egyben, mégpedig olyan speciális nemdeterminisztikus Turing-gép, melynek δ átmenetfüggvényére a következő teljesül:

$$\left|\delta(q,\sigma)\right| \quad \leq \quad 1$$

minden (q,σ) állapot-betű párra. □

1.29. Tétel.

Bármely $f(n)$ időkorlátos nemdeterminisztikus T Turing-gép szimulálható $\mathcal{O}\left(d^{f(n)}\right)$ időkorlátos deterministikus Turing-géppel, ahol $d > 1$ a T-től függő konstans.

Bizonyítás.

Legyen $T = \langle \Sigma, Q, q_0, F, \delta \rangle$. Tegyük fel, hogy a δ teljesen definiált függvény (ezt az általánosság megszorítása nélkül feltehetjük). Vezessük be a d konstanst:

$$d \quad = \quad \max_{\substack{q \in Q \\ \sigma \in \Sigma}} \left|\delta(q,\sigma)\right|$$

Azaz a d az átmenetfüggvény értékkészletébe tartozó halmazok számosságának a maximuma. A számítási fával szemléltetve: a fában a d a maximuma az egy csúcsból induló élek számának. Pl. a fent ábrázolt fán ez az érték 5.

Állítsuk elő a T' deterministikus Turing-gépet, mely T-t fogja szimulálni. A T' működésének az alapötlete a következő. A T egy számítási sorozata nemdeterminisztikus választások egy sorozata. Egy t lépésből álló ilyen választási sorozat reprezentálható a $0, 1, \ldots, d-1$ intervallumba eső egész számok egy t hosszú sorozatával. Pl. a fent ábrázolt fán (melyben egyébként a számokat csak ennek illusztrálására tüntettük fel) a vastagon szedett számítási sorozat leírható az ágon szereplő számokkal: 2,0,0,3. A szimulációt végző T' Turing-gép elő fogja állítani az összes ilyen számsort, és mindegyikre elvégzi a T szimulációját.

A T' legyen *3-szalagos*. Az 1. szalag tartalma nem fog változni, ezen végig megőrizzük az inputot (hiszen a T több számítási sorozatát is le kell majd futtatnunk rajta). A 2. szalagon fogjuk tárolni az aktuális számítási sorozatot leíró $c_1, c_2, \ldots, c_t$ számsort, ahol $0 \leq c_i \leq d-1$. A 3. szalag fog megfelelni a T egyetlen szalagjának.

A T' a következőképpen működik: kezdetben a 2. szalagra kiír egy 0-t, és elvégzi a T szimulálását az ez által leírt nemdeterminisztikus választás mellett. Ha a T elfogadó állapotba kerül, akkor T' is elfogadja az inputot. Ha nem, akkor törli a 3. szalagját, illetve *megnöveli* a 2. szalagon található számot; ehhez vegyük észre, hogy a 2. szalagon található számsor felfogható egy *d-alapú számrendszerben* felírt számként, amit tehát minden szimuláció után 1-gyel megnövelünk. Ezután újra lefuttatunk egy szimulációs fázist. Ha ez elfogadó állapotban végződik, akkor T' is elfogadja az inputot. Ha nem, akkor törli a 3. szalagot, 1-gyel növeli a 2. szalagon található számot, és újra lefuttat egy szimulációs fázist. És így tovább.

[3] Egy fa mélysége alatt a fa leghosszabb ágának hosszát értjük. Ha a fában van végtelen ág, akkor a fa mélysége is végtelen.

A T' tehát akkor fogadja el az inputját, ha valamelyik szimulációs fázis elfogadó állapotot eredményezett. Mikor fogja elutasítani az inputját? Egyáltalán nincs-e meg annak a veszélye, hogy végtelen ciklusba kerülünk? Vegyük észre, hogy a T-ről azt kötöttük ki, hogy van *időkorlátja*; azaz minden számítási sorozata véges, egész pontosan legfeljebb $f(n)$ hosszúságú. Ily módon biztosak lehetünk abban, hogy a szimuláció is véges sok lépésen belül véget ért, egész pontosan $\mathcal{O}\left(d^{f(n)}\right)$ lépésben. A T' akkor fogja elutasítani az inputját, ha semelyik szimulációs fázis sem ért véget elfogadó állapotban. □

1.3.1. Nemdeterminisztikus bonyolultsági osztályok

A nemdeterminisztikus Turing-géppel eldönthető nyelveket is idő- és tárbonyolultsági osztályokba soroljuk. Az időbonyolultsági osztályokhoz (a determinisztikus Turing-gépekhez hasonlóan) a *többszalagos* nemdeterminisztikus Turing-gépeket érdemes alapul venni, a tárbonyolultsági osztályok definiálásához pedig a *lyukszalagos* nemdeterminisztikus Turing-gépeket használjuk fel. A többszalagos, illetve a lyukszalagos nemdeterminisztikus Turing-gépek, valamint a nemdeterminisztikus Turing-gépek tárkorlátja könnyen definiálhatók az előző fejezetek alapján.

1.30. Definíció.

Egy L nyelvre azt mondjuk, hogy $L \in \mathbf{NTIME}\left(f(n)\right)$, ha van olyan $\mathcal{O}\left(f(n)\right)$ időkorlátos többszalagos nemdeterminisztikus Turing-gép, mely *eldönti* L-t. □

1.31. Definíció.

Egy L nyelvre azt mondjuk, hogy $L \in \mathbf{NSPACE}\left(f(n)\right)$, ha van olyan $\mathcal{O}\left(f(n)\right)$ tárkorlátos lyukszalagos nemdeterminisztikus Turing-gép, mely *eldönti* L-t. □

Nemdeterminisztikus Turing-gépek esetén is definiálunk összetett, nevezetes osztályokat. Ezek között a legismertebb az **NP**, mely a *polinom* időkorlátos nemdeterminisztikus Turing-géppel eldönthető nyelvek osztálya:

$$\mathbf{NP} \quad = \quad \bigcup_{k \geq 1} \mathbf{NTIME}\left(n^k\right)$$

Sok nevezetes nyelv eleme az **NP**-nek, például sok gráfokkal kapcsolatos nyelv, számelméleti problémák, egyenletek megoldásai, utazóügynök-útvonalak stb. Nagyon sok olyan nevezetes **NP**-beli nyelv ismert, melyről azonban nem tudjuk, hogy benne van-e **P**-ben.

1.3.2. **NP***-beli nyelvek*

Hogy egy nyelvről megmutassuk, hogy benne van **NP**-ben, az eldöntésére egy nemdeterminisztikus Turing-gépet kéne konstruálni. Ez a legtöbb esetben nehéz feladat, főleg amiatt, hogy a nemdeterminisztikus algoritmusok világa kicsit távol áll az emberi gondolkodástól. Emiatt a következőt tesszük: a nemdeterminisztikus eldöntési problémát megpróbáljuk visszavezetni egy determinisztikus eldöntési problémára. Erről szól a következő fontos tétel:

1.32. Tétel. (Tanú-tétel)

Egy $L \subseteq I^*$ nyelv esetén

$$L \in \mathbf{NP}$$

$$\Updownarrow$$

$\exists L' \in \mathbf{P}$ *nyelv és* $\exists c > 0$ *konstans, hogy* $\forall x \in I^*$ *szó esetén*

$$x \in L \quad \Leftrightarrow \quad \exists y \in I^* \text{ szó, hogy } |y| \leq |x|^c \text{ és } (x, y) \in L'$$

BIZONYÍTÁS.

A tételből leszűrhető, hogy az L' nyelv I ábécé feletti szópárokból áll, azaz $L' \subseteq (I^*)^2$. A döntés fókuszában levő x szóhoz előállítunk egy ún. *tanút*, ez az y szó, melynek segítségével az $x \in L$ eldöntése determinisztikus algoritmussal megválaszolható (hiszen $L' \in \mathbf{P}$). A tétel bizonyításához tegyük fel, hogy az $I = \{0, 1\}$ bináris ábécé felett dolgozunk.

A tételnek *csak az egyik irányát* bizonyítjuk: feltesszük, hogy $L \in \mathbf{NP}$, és bebizonyítjuk a $\Updownarrow$ jel alatti állítás helyességét.

Mivel $L \in \mathbf{NP}$, így létezik egy olyan T polinom időkorlátos nemdeterminisztikus Turing-gép, mely eldönti L-t. Tekintsük a T-nek az x inputon megadott számítási fáját. Az 1.29. tétel bizonyításához hasonlóan fogunk most is eljárni: a T ágait a rajtuk szereplő számok $c_1, c_2, \ldots, c_t$ sorozatával reprezentáljuk. Mivel a T valamilyen $\mathcal{O}\left(n^k\right)$ időkorlátos, így a számítási fa $\mathcal{O}\left(n^k\right)$ mélységű, azaz $t = \mathcal{O}\left(n^k\right)$. Minden c_i-re igaz, hogy $0 \leq c_i \leq d - 1$, ahol d az 1.29. tételben definiált szám. Most a c_i-ket bináris számrendszerben írjuk fel, azaz egy-egy ilyen szám $\lceil \log_2 d \rceil$ darab bináris számjeggyel írható fel. Vagyis egy-egy $c_1, \ldots, c_t$ számsor

$$\mathcal{O}\left(n^k\right) \cdot \lceil \log_2 d \rceil \quad \leq \quad n^c$$

darab bináris számjeggyel írható fel, ahol c egy alkalmas konstans. Az ilyen számsorokat fogjuk *tanúként* használni, vagyis $y = c_1, \ldots, c_t$. Vegyük észre, hogy $y \in I^*$, és $|y| \leq n^c$ (ahol $n = |x|$), szóval az y ténylegesen megfelel az x tanújának.

Most konstruálunk egy T' *determinisztikus* Turing-gépet, mely a tételben szereplő L' nyelvet dönti el. A T' (x, y) alakú inputokat kap, ahol x a T-nek adott input, az y pedig az előbb leírt módszerrel x-hez előállított tanú. A T' csinálja pontosan azt, amit az 1.29. tételben egyetlen szimulációs fázisban végeztünk el: az y-ban kódolt nemdeterminisztikus választásokat meglépve szimulálja a T-t.

Most vizsgáljuk meg a két lehetséges esetet:

$x \in L$: Ekkor biztos hogy *létezik* olyan y tanú, melyre a T' elfogadó állapotban áll meg (azaz elfogadja az (x, y) inputot). Nem azt mondjuk, hogy minden tanú ilyen, de legalább egy ilyennek léteznie kell közöttük.

$x \notin L$: Ekkor nem tudunk olyan y tanút találni, melyre a T' elfogadó állapotban állna meg. Ilyenkor a T' bármilyen y tanúra el fogja utasítani az (x, y) inputot. □

Az alábbiakban néhány nevezetes **NP**-be tartozó problémával ismerkedünk meg. Mint azt látni fogjuk, a tanú-tétel nagyon hatékony eszköz annak bizonyítására, hogy egy nyelv ténylegesen az **NP**-be tartozik.

3 színnel színezhető gráfok

1.33. DEFINÍCIÓ. (k SZÍNNEL SZÍNEZHETŐ GRÁF)

Egy $\langle V, E \rangle$ gráf $k \geq 1$ színnel színezhető, ha van olyan $f : V \mapsto \{0, 1, \ldots, k - 1\}$ hozzárendelés, hogy minden $(v_1, v_2) \in E$ élre $f(v_1) \neq f(v_2)$.

A k színnel színezhető gráfok nyelvének jelölése: k-SZIN. □

Most bebizonyítjuk, hogy a 3 színnel színezhető gráfok nyelve az **NP**-ben van:

1.34. ÁLLÍTÁS.

3-SZIN $\in \mathbf{NP}$

BIZONYÍTÁS.

Az állítás könnyen bizonyítható a tanú-tétel segítségével. Elegendő megmutatni, hogy egy gráf 3 színnel színezhetőségének van (polinom hosszú) tanúja.

Egy n csúcsú gráfhoz tartozó tanú a gráf egy jó színezése lesz. Egy ilyen színezés leírható $2n$ darab bináris számjeggyel, mivel minden szín reprezentálható kétjegyű bináris számmal (hiszen csak 3 szín van).

Tehát a tanú-tételbeli L' nyelv (x, y) párjaiban az x maga a gráf, az y pedig az x egy 3 színnel színezése. Magától értetődő, hogy annak vizsgálata az y tanújelölt helyes színezése-e az x-nek, polinom időkorlátos *determinisztikus* Turing-géppel megoldható: csupán végig kell ellenőrizni az x éleit, hogy azok mindegyike különböző színű csúcsokat köt-e össze.

Figyeljük meg, hogy a tanú-tétel

$$x \in L \quad \Leftrightarrow \quad (x, y) \in L'$$

feltétele tényleg teljesül. Ha $x \in$ 3-SZIN, akkor van helyes y színezés, azaz: $\exists y$, hogy $(x, y) \in L'$. Ha $x \notin$ 3-SZIN, akkor nem lehet találni helyes y színezést, azaz: $\forall y$-ra $(x, y) \notin L'$. □

A bizonyítással kapcsolatban megjegyezzük még, hogy az x-ben tulajdonképpen a gráf egy bináris reprezentációját tároljuk (hiszen a tanú-tételben $x \in I^*$). Léteznek széles körben ismert ilyen reprezentációk (pl. a gráf adjacencia-mátrixának bináris kódolása).

Hamilton-körrel rendelkező gráfok

1.35. DEFINÍCIÓ. (HAMILTON-KÖR)

Egy gráf egy köre Hamilton-kör, ha abban a gráf minden csúcsa pontosan egyszer szerepel.

A Hamilton-körrel rendelkező gráfok nyelvének jelölése: HAM. □

1.36. ÁLLÍTÁS.

HAM $\in$ **NP**

BIZONYÍTÁS.

Ismét a tanú-tétel segítségével bizonyítjuk az állítást. Az x gráf egy lehetséges y tanújelöltje az x csúcsainak egy $v_1, v_2, \ldots, v_n$ permutációja. Egy ilyen permutáció leírható $n \cdot \lceil \log_2 n \rceil$ darab bináris számjeggyel.

Annak vizsgálata, hogy az $y = v_1, \ldots, v_n$ köre-e az x-nek, megoldható polinom időkorlátos *determinisztikus* Turing-géppel: ellenőrizni kell, hogy léteznek-e $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)$ élek az x-ben.

Ha $x \in$ HAM, akkor meg tudunk adni megfelelő y permutációt, ha viszont $x \notin$ HAM, akkor egyik tanújelölt sem bizonyul körnek. □

Prímszámok

Jelöljük a prímszámok nyelvét PRÍM-mel.

1.37. ÁLLÍTÁS.

PRÍM $\in$ **NP**

Az állítás bizonyítását mellőzzük (egy számelméleti lemmát kell hozzá felhasználni). Inkább vizsgáljuk meg a következő kérdést: annak eldöntése, hogy egy szám *nem prím*, azaz *összetett szám*, milyen időbonyolultságú algoritmussal oldható meg?

Ehhez előbb bevezetjük a komplementer nyelvosztály fogalmát.

1.38. Definíció. (Komplementer nyelv)

Legyen $L \subseteq I^*$ egy nyelv. Az L komplementere alatt a következőt értjük:

$$\mathbf{co}L \quad = \quad I^* \backslash L$$ □

1.39. Definíció. (Komplementer nyelvosztály)

Legyen $\mathcal{C} \subseteq P(I^*)$ egy nyelvosztály. A $\mathcal{C}$ komplementere alatt a következőt értjük:

$$\mathbf{co}\mathcal{C} \quad = \quad \left\{ L \subseteq I^* \mid \mathbf{co}L \in \mathcal{C} \right\}$$ □

Könnyen igazolhatóak a következő összefüggések a komplementerképzéssel kapcsolatosan:

1.40. Állítás.

(1) $\mathbf{co}(\mathbf{co}\mathcal{C}) = \mathcal{C}$

(2) *ha* $\mathcal{C} \subseteq \mathcal{D}$*, akkor* $\mathbf{co}\mathcal{C} \subseteq \mathbf{co}\mathcal{D}$

Most lássuk a prím nyelvre vonatkozó bizonyítandó állítást:

1.41. Állítás.

prím $\in \mathbf{coNP}$

Bizonyítás.

Azt kell belátni, hogy **co**prím $\in \mathbf{NP}$. A **co**prím nyelv az *összetett számok nyelve*. A tanú-tétel alapján minden x számhoz megadunk valamilyen polinom hosszú y tanújelöltet. Az y legyen egy szám, mégpedig $2 \leq y \leq x-1$. Nyilvánvaló, hogy az y bináris reprezentációjának hossza nem haladhatja meg az x bináris reprezentációjának hosszát; jelölésben: $|y| \leq |x|$.

Nyilvánvaló, hogy egy y tanújelöltre annak vizsgálata, hogy y *osztója-e* x-nek, polinom időkorlátos *determinisztikus* Turing-géppel megoldható[4].

Ha $x \in$ **co**prím, vagyis x összetett szám, akkor biztosan van legalább egy y osztója a $[2, x-1]$ intervallumban. Ha $x \notin$ **co**prím, vagyis x prím, akkor természetesen nincs ilyen osztó. □

Tehát azt kaptuk, hogy prím $\in \mathbf{NP} \cap \mathbf{coNP}$, vagyis ez egy ún. *jól karakterizált* nyelv.

1.42. Definíció. (Jól karakterizált nyelv)

Egy L nyelv *jól karakterizált*, ha $L \in \mathbf{NP} \cap \mathbf{coNP}$. □

[4] Például a célnak megfelel az általános iskolában elsajátított eljárás, mellyel két számot lehet egymással osztani; ennek az időkorlátja egyébként $\mathcal{O}(n^2)$, vagyis polinom.

2. NYELVOSZTÁLYOK KÖZÖTTI ÖSSZEFÜGGÉSEK

Az első kérdés, amit megvizsgálunk, a következő: vannak-e algoritmussal (Turing-géppel) *nem kezelhető* (*eldönthetetlen*) problémák (nyelvek)? Ez egy nagyon lényeges és egyáltalán nem triviális kérdés. Az ilyen problémák létezése teljes meglepetésnek számított, amikor az 1930-as években felfedezték őket (Church és Turing). Ezt és más hasonló kérdéseket fogunk mi is vizsgálni a 2.1. fejezetben.

A 2.2. fejezetben már csak a Turing-gépekkel *eldönthető* nyelvekkel foglalkozunk. Ezeket a nyelveket különböző osztályokba soroljuk (melyek közül párat már korábban is megismertünk), és vizsgáljuk az ezen osztályok közötti összefüggéseket,

2.1. Rekurzív nyelvek és rekurzíve felsorolható nyelvek

Célunk ebben a fejezetben az, hogy a kiszámíthatóság határait feszegessük. Hogy erről értelmesen tudjuk beszélni, pontosan meg kell határoznunk, mely nyelveket tekintjük „kiszámíthatónak". A *rekurzív nyelvek* azon nyelvek lesznek, melyek algoritmussal teljes mértékben kezelhetők („kiszámíthatók"). A *rekurzíve felsorolhatóak* pedig azok, melyek kezelésére csak egy „fél algoritmus" algoritmus adható.

2.1. Definíció. (Rekurzív nyelv)

Egy L nyelv rekurzív, ha van olyan Turing-gép, mely *eldönti* L-t.

A rekurzív nyelvek osztályát **R**-rel jelöljük. □

2.2. Definíció. (Rekurzíve felsorolható nyelv)

Egy L nyelv rekurzíve felsorolható, ha van olyan Turing-gép, mely *felismeri* L-t.

A rekurzíve felsorolható nyelvek osztályát **RE**-vel jelöljük. □

2.3. Állítás.

Ha egy nyelv rekurzív, akkor rekurzívan felsorolható is. Azaz: $\mathbf{R} \subseteq \mathbf{RE}$.

Bizonyítás.

Legyen L egy rekurzív nyelv, melyet a T Turing-gép dönt el. Megadunk egy olyan T' Turing-gépet, mely felismeri az L-t.

T' működjön ugyanúgy mint T, kivéve azt az esetet, mikor a T éppen elutasítaná az inputját: ekkor T' egy „csapda" állapotba menjen át (lásd: 1.5. megjegyzés). □

2.1.1. Univerzális Turing-gép

Az univerzális Turing-gép egy olyan Turing-gép, mely bármely Turing-gép szimulálására képes. Pontosabban: konstruálni fogunk egy olyan U Turing-gépet, mely bármely T Turing-gép és x input esetén az $\omega\#x$ inputon ugyanazt a számítási eredményt adja, mint a T Turing-gép az x inputon, ahol az ω a T-nek az ún. *programja*. Az ugyanazon számítási eredmény alatt azt értjük, hogy az U és a T ugyanazon x-ekre kerül végtelen ciklusba, ugyanazokat utasítja el, és ugyanazokat fogadja el ugyanazon outputot szolgáltatva.

Először leírjuk, hogy hogyan lehet előállítani egy T Turing-gép *programját*. Feltesszük T-ről, hogy *egyszalagos* (ezt az 1.14. tétel alapján feltehetjük) és *determinisztikus* (az 1.29. tétel alapján ezt is feltehetjük). Legyen $T = \langle \Sigma, Q, q_0, F, \delta \rangle$. Mivel a T szalagjeleinek és állapotainak elnevezései nem relevánsak, feltesszük, hogy a gép megadásában szereplő szimbólumok (szalagjelek, állapotok és mozgási irányok) *egész számok*. Legyen tehát

- $\Sigma = \{0, 1, \ldots, |\Sigma| - 1\}$, $\triangleright = 0$, $\sqcup = 1$
- $Q = \{0, 1, \ldots, |Q| - 1\}$, $q_0 = 0$, $F = \{1, 2, \ldots, |F|\}$
- $\leftarrow = 0$, $- = 1$, $\rightarrow = 2$

Ekkor a Turing-gép programja a következő:

$$|F|\#q_1\#\sigma_1\#q_1'\#\sigma_1'\#m_1\#\ldots\#q_r\#\sigma_r\#q_r'\#\sigma_r'\#m_r\#$$

ahol a δ által megadott $\delta(q_i, \sigma_i) = (q_i', \sigma_i', m_i)$ hozzárendeléseket (mindazokon a helyeken, ahol δ értelmezett) $q_i\#\sigma_i\#q_i'\#\sigma_i'\#m_i$ alakban soroltuk fel. A Turing-gép programja az $I = \{0, 1\}$ bináris ábécé feletti szó legyen, azaz a programban az adott számoknak a *bináris* reprezentációja szerepel. Így 3 különböző jelet kapunk: 0, 1 és a # elválasztó jel. Ezeknek is vegyük valamilyen bináris átírását, például: $0 \mapsto 00$, $1 \mapsto 01$, $\# \mapsto 11$. Így a kapott program egy I^*-beli szó.

Egy Turing-gép ismeretében annak $\omega \in I^*$ programja algoritmussal kiszámítható. Ha az ω adott, akkor algoritmussal megválaszolható az a kérdés, hogy az ω érvényes program-e. Az ω által megadott Turing-gépet T_ω-val fogjuk jelölni (amennyiben az létezik).

2.4. Tétel. (Univerzális Turing-gép)

Van olyan U Turing-gép, melyre minden $\omega, x \in I^*$ esetén –feltéve, hogy T_ω létezik– teljesül a következő: az U az $\omega\#x$ inputon ugyanazon számítási eredményt adja, mint T_ω az x inputon.

Bizonyítás.

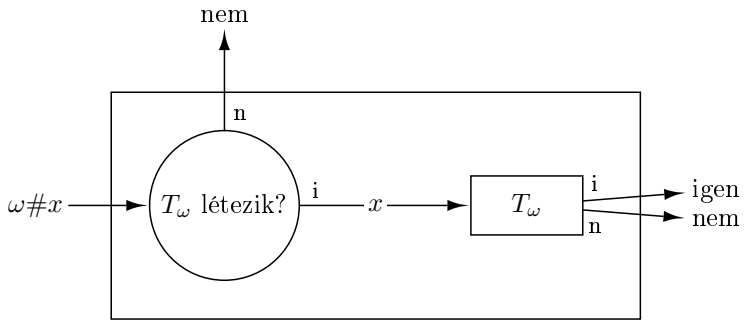
Legyen U *3-szalagos* Turing-gép. Az U 1. szalagja tartalmazza az $\omega\#x$ inputot. Ennek tartalma az U működése alatt nem fog változni, azaz U ezt a szalagot csak olvasni fogja.

Az U 2. és 3. szalagján a szimuláció alatt a következő információkat tároljuk: ha a T_ω aktuális konfigurációja (q, u, v), akkor a 2. szalagon a q kódja, a 3. szalagon az uv szó kódja található (persze mindkét szalagon a $\triangleright$ szimbólum után).

Kezdetben az U ellenőrzi az 1. szalagján található $\omega\#x$ alapján, hogy az ω ténylegesen egy Turing-gép programja-e, azaz T_ω létezik-e egyáltalán. Ha nem, akkor az U elutasítja az $\omega\#x$ inputot. Egyébként az U a 2. szalagjára felírja a q_0 kódját (vagyis 0-t), a 3. szalagjára pedig az $\triangleright x$ kódját.

Ezek után az U szimulálja a T_ω egy-egy lépését. A 2. szalagról leolvassuk a T_ω aktuális q állapotát, a 3. szalagról pedig a fej alatt található σ betűt. Az U ezek után az 1. szalagon megkeresi a $\delta(q, \sigma)$ leírását, és ennek megfelelően módosítja a 2. és a 3. szalagok tartalmát, illetve lépteti a 3. szalag fejét. Könnyen látható, hogy egy ilyen szimulációs lépés könnyen megvalósítható Turing-géppel.

Az U akkor áll meg, ha T_ω megáll. Ez esetben pontosan akkor fogadja el az $\omega\#x$ inputot, ha a megállás pillanatában a T_ω valamely elfogadó állapotának a kódja található a 2. szalagon. □



2.1.2. Egy nem rekurzíve felsorolható nyelv

Az univerzális Turing-gép létezése már önmagában is meglepő, ráadásul ez a tény teljesen egybevág a Neumann által javasolt „tárolt program" elvével. Mi most az univerzális Turing-gépet arra fogjuk felhasználni, hogy egy „kiszámíthatatlansági" eredményt igazoljunk: létezik olyan probléma, mely algoritmussal nem kezelhető.

2.5. Definíció. (Diagonális nyelv)

Az L_d diagonális nyelv a következő:

$$L_d \quad = \quad \{\omega \in I^* \mid T_\omega \text{ létezik és } T_\omega \text{ nem fogadja el } \omega\text{-t}\} \qquad \square$$

2.6. Állítás.

$L_d \notin \mathbf{RE}$

Bizonyítás.

A bizonyítás indirekt. Azaz feltesszük, hogy L_d rekurzíve felsorolható. Ez annyit tesz, hogy van olyan T Turing-gép, mely *felismeri* L_d-t. Most vizsgáljuk meg a T ω programjának és az L_d-nek a kapcsolatát. Két eset lehetséges, és megmutatjuk, hogy mindkét eset ellentmondáshoz vezet:

(1) Ha $\omega \in L_d$, akkor L_d definíciója szerint T_ω (mely maga T) nem fogadja el ω-t. Mivel a T_ω által felismert nyelv L_d, ez azt jelentené, hogy $\omega \notin L_d$. Ellentmondásra jutottunk.

(2) Ha $\omega \notin L_d$, akkor L_d definíciója szerint – mivel T_ω az indirekt feltevés szerint létezik – T_ω elfogadja ω-t. Ez azt jelentené, hogy ω benne van a T_ω által felismert nyelvben, vagyis $\omega \in L_d$. Ismét ellentmondásba ütköztünk. $\square$

2.1.3. Egy rekurzíve felsorolható, de nem rekurzív nyelv

Az előző fejezetben kicsit túl is lőttünk a célon: a diagonális nyelv személyében olyan problémára akadtunk, melynek kezelésére még egy „fél algoritmus" sem található. Most egy olyan problémát keresünk, melynek kezelésére csak és kizárólag egy „fél algoritmus" létezik.

2.7. Definíció. (Univerzális nyelv)

Az L_u univerzális nyelv a következő:

$$L_u \quad = \quad \{\omega\#x \mid \omega, x \in I^*,\ T_\omega \text{ létezik és } T_\omega \text{ elfogadja } x\text{-et}\} \qquad \square$$

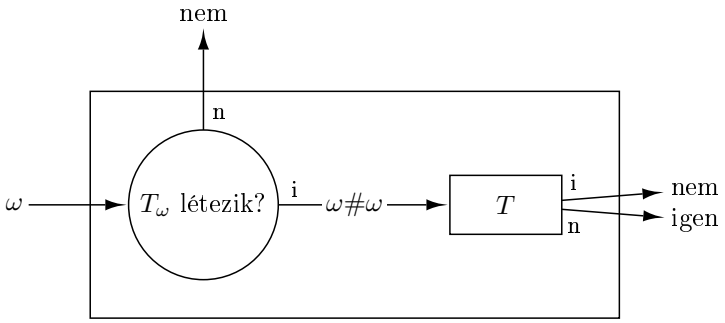
2.8. Állítás.

$L_u \in \mathbf{RE} \backslash \mathbf{R}$

Bizonyítás.

Könnyen észrevehető, hogy az L_u pontosan azon szavakból áll, melyeket az univerzális Turing-gép elfogad. Vagyis L_u az *univerzális Turing-gép által felismert nyelv.* Éppen ezért (mivel létezik egy őt felismerő Turing-gép) rekurzíve felsorolható.

Annak megmutatásához, hogy L_u nem rekurzív, tegyük fel az ellenkezőjét: L_u rekurzív. Ez azt jelentené, hogy létezik olyan T Turing-gép, mely *eldönti* az L_u-t. Itt fontos, hogy a feltevés alapján T minden inputon megáll. Legyen T' a következőképpen megkonstruált Turing-gép:



Tehát a T' Turing-gép az ω inputtal a következő lépéseket végzi:

(1) Ellenőrzi, hogy T_ω létezik-e. Ha nem, akkor elutasítja az inputot.

(2) Ha viszont T_ω létezik, akkor szimulálja T-t az $\omega\#\omega$ inputtal. Ha T elfogadja azt, akkor T' elutasítja, ha pedig T elutasítja azt, akkor T' elfogadja.

Vegyük észre, hogy T' mindig megáll, hiszen a T_ω létezésének a tesztje algoritmussal megoldható, valamint az indirekt feltevés szerint T mindig megáll. Világos továbbá, hogy T' pontosan akkor fogadja el az ω inputot, ha T_ω létezik és $\omega \notin L_u$, vagyis: ha T_ω létezik és T_ω nem fogadja el ω-t.

Ez viszont azt jelenti, hogy T' éppen az L_d diagonális nyelvet ismeri fel. Ez azonban képtelenség, mivel a 2.6. tételben bebizonyítottuk, hogy L_d nem rekurzíve felsorolható. □

2.1.4. További összefüggések **R** *és* **RE** *között*

A 2.6. állítás megmutatta, hogy $\mathbf{RE} \subset I^*$; tehát itt *valódi tartalmazásról* beszélhetünk.

A 2.3. állítás alapján tudjuk, hogy $\mathbf{R} \subseteq \mathbf{RE}$. A 2.8. tétel megmutatta, hogy $\mathbf{R} \subset \mathbf{RE}$; tehát itt is valódi a tartalmazás.

A kép még cizelláltabb, mint az alábbi eredmények mutatják, melyek kapcsán megvizsgáljuk az **R** és az **RE** osztályok komplementereit is (lásd: 1.39. definíció).

2.9. Állítás.

$\mathbf{R} = co\mathbf{R}$

Bizonyítás.

Legyen $L \in \mathbf{R}$ egy nyelv. Ennek megfelelően van olyan $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ Turing-gép, mely az L nyelvet dönti el. A $T' = \langle \Sigma, Q, q_0, Q \backslash F, \delta \rangle$ Turing-gép pont a $\mathbf{co}L$ nyelvet dönti el. Vagyis $L \in \mathbf{coR}$. Ezzel beláttuk, hogy $\mathbf{R} \subseteq \mathbf{coR}$.

Innen már adódik a fordított irányú tartalmazás is: $\mathbf{coR} \subseteq \mathbf{R}$. Ennek belátására alkalmazzuk a **co-**operátorra ismert összefüggéseket (lásd: 1.40. állítás):

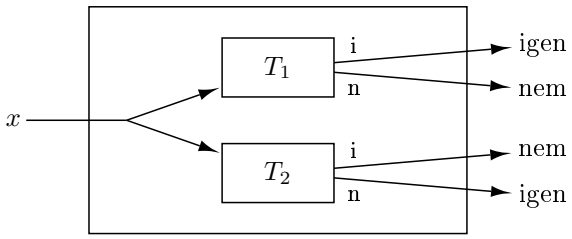
$$\mathbf{coR} \quad \subseteq \quad \mathbf{co}(\mathbf{coR}) = \mathbf{R} \qquad \square$$

2.10. Állítás.
$\mathbf{R} = \mathbf{RE} \cap \mathbf{coRE}$

Bizonyítás.

Az $\mathbf{R} \subseteq \mathbf{RE}$ összefüggésből komplementerképzéssel nyerjük a $\mathbf{coR} \subseteq \mathbf{coRE}$ összefüggést (lásd: 1.40. állítás). Mivel az előző tételből tudjuk, hogy $\mathbf{R} = \mathbf{coR}$, így ebből következik, hogy $\mathbf{R} \subseteq \mathbf{RE} \cap \mathbf{coRE}$.

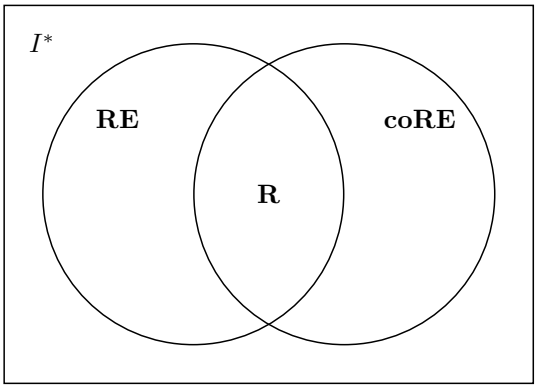
A fordított irányú tartalmazás bizonyításához tegyük fel, hogy $L \in \mathbf{RE} \cap \mathbf{coRE}$. Ennek megfelelően létezik olyan T_1 Turing-gép, mely az L nyelvet ismeri fel, és létezik olyan T_2 Turing-gép, mely a $\mathbf{co}L$ nyelvet ismeri fel. Ezen Turing-gépek ötvözésével szerkesztünk majd egy olyan T Turing-gépet, mely *eldönti* az L-t. Az $x \in I^*$ inputon elindítva T-nek úgy kell a működését elképzelnünk, hogy egymástól független párhuzamosan szimulálja a T_1 és T_2 lépéseit az x inputon:



A T *felváltva szimulálja* a T_1 és T_2 gépeket: minden $(2i-1)$. lépésben a T_1 i. lépését, minden $2i$. lépésben a T_2 i. lépését végzi el. A két Turing-gép párhuzamos szimulálása megoldható például több szalag használatával.

Világos, hogy T minden x inputra megáll, ugyanis az x-re T_1 és T_2 közül legalább az egyik meg fog állni. Az is világos, hogy T pontosan az L nyelvet dönti fel. Ezzel beláttuk, hogy $\mathbf{RE} \cap \mathbf{coRE} \subseteq \mathbf{R}$. $\square$

A rekurzív és a rekurzíve felsorolható nyelvek osztályai közötti tartalmazási viszonyokat a következő ábra szemlélteti:



2.2. Bonyolultsági osztályok

A Turing-géppel eldönthető nyelveket, azaz az **R**-be tartozó nyelveket bonyolultsági osztályokba soroljuk. Ezen osztályokat definiáljuk $\mathbf{TIME}(f(n))$, $\mathbf{SPACE}(f(n))$, $\mathbf{NTIME}(f(n))$ és $\mathbf{NSPACE}(f(n))$

alakban, illetve a **co** operátor használatával[1]. Továbbá ezen osztályok egyesítéseivel nagyobb osztályokat alkotunk, és vizsgáljuk azok egymáshoz való viszonyát.

2.11. Állítás.

(1) $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$

(2) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$

Bizonyítás.

A bizonyítás triviális, hiszen minden determinisztikus Turing-gép nemdeterminisztikus is egyben. □

2.12. Állítás.

(1) $\mathbf{TIME}(f(n)) = \mathbf{coTIME}(f(n))$

(2) $\mathbf{SPACE}(f(n)) = \mathbf{coSPACE}(f(n))$

Bizonyítás.

Legyen adott az L nyelvet eldöntő $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ (determinisztikus!) Turing-gép.

Legyen $T' = \langle \Sigma, Q, q_0, Q \backslash F, \delta \rangle$. Triviális, hogy T' a $\mathbf{co}L$ nyelvet dönti el, és a T' időkorlátja és tárkorlátja a T időkorlátjával és tárkorlátjával egyezik meg. □

2.13. Állítás.

(1) $\mathbf{TIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$

(2) $\mathbf{NTIME}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$

(3) $\mathbf{NTIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$

Bizonyítás.

Az (1) és a (2) triviális, hiszen egy Turing-gép k számítási lépésben legfeljebb k szalagcellát használ.

A (3) bizonyításához tekintsünk egy L nyelvet, melyet eldönt az $\mathcal{O}(f(n))$ *időkorlátos nemdeterminisztikus* T Turing-gép. Most megadunk egy $\mathcal{O}(f(n))$ *tárkorlátos determinisztikus* T' Turing-gépet, mely eldönti L-t.

Az ötlet kísértetiesen hasonlít az 1.29. tétel bizonyításában használt ötletre: T' állítsa elő a T lehetséges nemdeterminisztikus választásainak egy

$$c_1, c_2, \ldots, c_t$$

sorozatát, ahol $0 \leq c_i \leq d-1$ és $t = \mathcal{O}(f(n))$. A d az 1.29. tételben definiált szám. Ezután T' szimulálja T működését az adott választások mellett. Ez a (determinisztikus) szimuláció $\mathcal{O}(f(n))$ lépést foglal magában, tehát $\mathcal{O}(f(n))$ cellát használ (a mostani tétel (1) pontja miatt). Ha a szimuláció végén nem kerülünk elfogadó állapotba, akkor a sorrendben következő $c_1, \ldots, c_t$ sorozatot állítjuk elő, és újra szimuláljuk T működését erre a sorozatra vonatkozóan.

Az 1.29. tétel alapján tudjuk, hogy legrosszabb esetben exponenciálisan sok ilyen szimulációt kell végrehajtanunk annak ellenőrzésére, hogy létezik-e elfogadást eredményező választási sorozat. A lényeg, hogy ezeket egyenként végrehajtjuk, de minden egyes alkalommal *ugyanazt a tárterületet használva.* Ez a tárterület –az előbbiek alapján– $\mathcal{O}(f(n))$ cellát foglal magában. □

[1] Lásd az 1.15., az 1.23., az 1.30., az 1.31. és az 1.39. definíciókat.

A következő tételt nem bizonyítjuk:

2.14. Tétel. (Tár-idő tétel)
$\mathbf{NSPACE}\left(f(n)\right) \subseteq \mathbf{TIME}\left(c^{f(n)}\right)$

2.2.1. A $\mathbf{P}, \mathbf{NP}, \mathbf{EXP}, \mathbf{PSPACE}, \mathbf{L}$ *és* $\mathbf{NL}$ *osztályok*

Definiálunk kiterjedésre nagy, fontos bonyolultsági osztályokat:

2.15. Definíció. (Fontos bonyolultsági osztályok)

(1) Polinomiális időkorlátos (determinisztikus, illetve nemdeterminisztikus) Turing-géppel eldönthető nyelvek osztályai:

$$\begin{aligned} \mathbf{P} &= \bigcup_{k\geq 1} \mathbf{TIME}\left(n^k\right) \\ \mathbf{NP} &= \bigcup_{k\geq 1} \mathbf{NTIME}\left(n^k\right) \end{aligned}$$

(2) Exponenciális időkorlátos Turing-géppel eldönthető nyelvek osztálya:

$$\mathbf{EXP} = \bigcup_{k\geq 1} \mathbf{TIME}\left(c^{n^k}\right)$$

(3) Polinomiális tárkorlátos Turing-géppel eldönthető nyelvek osztálya:

$$\mathbf{PSPACE} = \bigcup_{k\geq 1} \mathbf{SPACE}\left(n^k\right)$$

(4) Logaritmikus tárkorlátos (determinisztikus, illetve nemdeterminisztikus) Turing-géppel eldönthető nyelvek osztályai:

$$\begin{aligned} \mathbf{L} &= \mathbf{SPACE}\left(\log_c n\right) \\ \mathbf{NL} &= \mathbf{NSPACE}\left(\log_c n\right) \end{aligned}$$

□

A következő kérdések merülnek fel a fenti osztályokkal kapcsolatosan:

- A (2)-ben nemdeterminisztikus Turing-gépekre miért nem adunk egy hasonló **NEXP** időbonyolultsági osztályt? A válasz egyszerű: az 1.29. tétel miatt minden nemdeterminisztikus Turing-gép az időkorlát *exponenciális* mértékben való növekedése mellett szimulálható determinisztikus Turing-géppel. Nyilvánvaló, hogy ha az eredeti Turing-gép exponenciális időkorlátos volt, akkor a szimuláló Turing-gép is az lesz. Vagyis: $\mathbf{EXP} = \mathbf{NEXP}$.
- A (4)-ben miért nem $\mathbf{SPACE}\left(\log_c n^k\right)$ osztályoknak az uniójaként adjuk meg az **L**-t? Nos, a k exponens konstans szorzóként kiemelhető, azaz a következő összefüggés teljesül:

  $$\log_c n^k = k \log_c n = \mathcal{O}\left(\log_c n\right)$$

  Vagyis:

  $$\bigcup_{k\geq 1} \mathbf{SPACE}\left(\log_c n^k\right) = \mathbf{SPACE}\left(\log_c n\right)$$

- A (3)-ban nemdeterminisztikus Turing-gépekre miért nem adunk egy hasonló **NPSPACE** tárbonyolultsági osztályt? A válasz a következő összefüggésben rejlik (nem bizonyítjuk):

2.16. Állítás.

$$\mathbf{NSPACE}\left(f(n)\right) \subseteq \mathbf{SPACE}\left(f^2(n)\right)$$

Ez azt fejezi ki, hogy minden nemdeterminisztikus Turing-gép a tárkorlát *négyzetes* mértékben való növekedése mellett szimulálható determinisztikus Turing-géppel. Nyilvánvaló, hogy ha az eredeti Turing-gép polinomiális tárkorlátos volt, akkor a szimuláló Turing-gép is az lesz. Vagyis: **PSPACE** = **NPSPACE**.

2.17. Állítás.

(1) $\mathbf{L} \subseteq \mathbf{NL}$

(2) $\mathbf{NL} \subseteq \mathbf{P}$

Bizonyítás.

(1) A 2.11.(2)-ből következik.

(2) A tár-idő (2.14.) tételből következik, hiszen

$$\mathbf{NSPACE}\left(\log_c n\right) \quad \subseteq \quad \mathbf{TIME}\left(d^{\log_c n}\right)$$

ahol c és d konstansok. További átalakításokat végezve:

$$d^{\log_c n} = d^{\log_d n \cdot \log_c d} = \left(d^{\log_d n}\right)^{\log_c d} = n^{\log_c d} = n^k$$

valamilyen alkalmas k konstansra. Tehát:

$$\mathbf{NSPACE}\left(\log_c n\right) \quad \subseteq \quad \mathbf{TIME}\left(n^k\right)$$

Innen következik az állítás helyessége. □

2.18. Állítás.

$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$

Bizonyítás.

A 2.11.(1)-ből következik, hogy

$$\mathbf{P} \quad \subseteq \quad \mathbf{NP}$$

Ebből következik, hogy

$$\mathbf{coP} \quad \subseteq \quad \mathbf{coNP}$$

A 2.12.(1) alapján

$$\mathbf{P} \quad = \quad \mathbf{coP}$$

és ezért

$$\mathbf{P} \subseteq \mathbf{coNP}$$

Mivel $\mathbf{P} \subseteq \mathbf{NP}$ és $\mathbf{P} \subseteq \mathbf{coNP}$, így

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$$ □

2.19. Állítás.

(1) $\mathbf{NP} \subseteq \mathbf{PSPACE}$

(2) $\mathbf{coNP} \subseteq \mathbf{PSPACE}$

Bizonyítás.

(1) A 2.13.(3)-ból következik.

(2) Az előző pont alapján $\mathbf{coNP} \subseteq \mathbf{coPSPACE}$. A 2.12.(2)-ből azonban az is következik, hogy $\mathbf{PSPACE} = \mathbf{coPSPACE}$. □

2.20. Állítás.

$\mathbf{PSPACE} \subseteq \mathbf{EXP}$

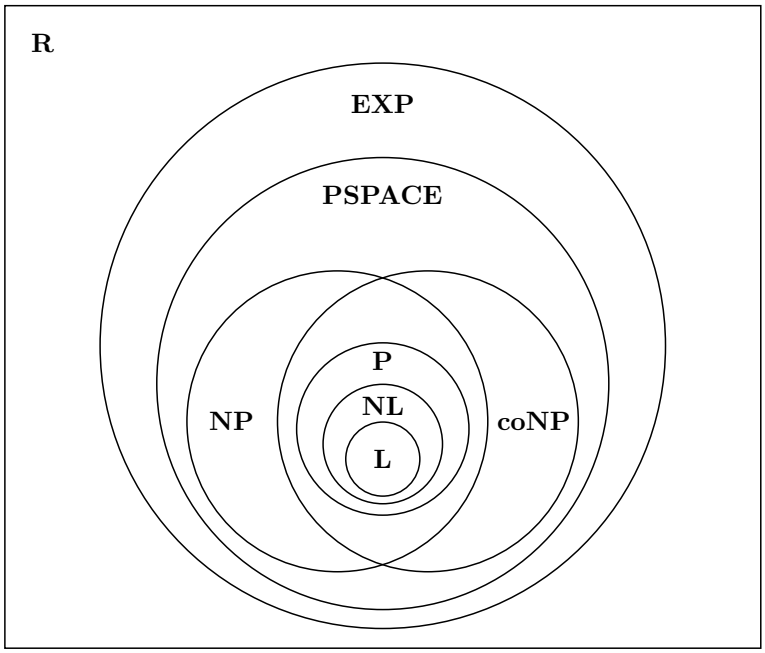
Bizonyítás.

Tudjuk, hogy $\mathbf{PSPACE} = \mathbf{NPSPACE}$. A tár-idő tétel (2.14.) alapján azt is tudjuk, hogy

$$\mathbf{NPSPACE} \subseteq \mathbf{EXP}$$

Ezekből következik az állítás. □

Alább látható a bonyolultsági osztályok fenti eredmények alapján összeállított „térképe”:



A fenti tételekben az egyes bonyolultsági osztályokra vonatkozóan *nem szigorú* tartalmazási viszonyokat bizonyítottunk. Például: $\mathbf{P} \subseteq \mathbf{NP}$, de azt nem tudjuk, hogy a $\mathbf{P}$ *valódi része-e* az $\mathbf{NP}$-nek. Ez a kérdés

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

formában híresült el, és ez az *algoritmuselmélet központi kérdése.*

További ehhez hasonló és ezzel összefüggő nyitott kérdések léteznek az algoritmuselméletben. Például:

$$\mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$$

Ha ez utóbbira sikerülne bebizonyítani, hogy *igen* a válasz, akkor azzal a $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ kérdésre is választ kapnánk:

$$\mathbf{P} = \mathbf{PSPACE} \quad \Rightarrow \quad \mathbf{P} = \mathbf{NP}$$

Sőt, ekkor megválaszolhatnánk az

$$\mathbf{NP} \stackrel{?}{=} \mathbf{coNP}$$

nyitott kérdést is, vagyis a következő következtetési lánc írható fel:

$$\mathbf{P} = \mathbf{PSPACE} \quad \Rightarrow \quad \mathbf{P} = \mathbf{NP} \quad \Rightarrow \quad \mathbf{NP} = \mathbf{coNP}$$

Természetesen ez a következtetési lánc visszafelé is alkalmazható, azaz:

$$\mathbf{NP} \neq \mathbf{coNP} \quad \Rightarrow \quad \mathbf{P} \neq \mathbf{NP} \quad \Rightarrow \quad \mathbf{P} \neq \mathbf{PSPACE}$$

Azaz ha pl. sikerülne bebizonyítani, hogy van olyan nyelv, amely benne van $\mathbf{NP}$-ben, de $\mathbf{coNP}$-ben nincs benne, akkor azzal a $\mathbf{P} \neq \mathbf{NP}$-t is belátnánk. Ez utóbbi összefüggés tehát a mai napig nincs bizonyítva, csupán erős sejtésként él a algoritmuselméletben.

3. TELJES NYELVEK

A valamely $\mathcal{C}$ bonyolultsági osztályra nézve *teljes* problémák (nyelvek) speciálisak a következő értelemben: valamiképpen magukban hordozzák a $\mathcal{C}$-be tartozó összes probléma nehézségét. Ezen kívül még egy specialitással bírnak: segítségükkel könnyű bebizonyítani két osztályról, hogy a két osztály egybeesik.

3.1. Visszavezetések

Hogy fejezzük ki azt, hogy egy nyelv „magában hordozza" egy másik nyelv „nehézségét"? Erre fogjuk használni az ún. *visszavezetéseket*. Egy nyelv visszavezetése egy másik nyelvre tulajdonképpen egy *ekvivalens összerendelés* lesz a két nyelv szavai között.

3.1. Definíció. (Visszavezetés)
Az $L_1 \subseteq I^*$ *visszavezethető* az $L_2 \subseteq I^*$ nyelvre, ha van olyan $f : I^* \mapsto I^*$ polinomiális időkorlátos determinisztikus Turing-géppel kiszámítható függvény, hogy minden $x \in I^*$ szóra teljesül a következő:

$$x \in L_1 \quad \Leftrightarrow \quad f(x) \in L_2.$$

Jelölése:

$$L_1 \prec L_2$$

Az f-et az L_1 L_2-re való visszavezetésének (*Karp-redukciójának*) nevezzük. □

A következő állítás triviálisan teljesül:

3.2. Állítás. (Tranzitivitás)
Ha $L_1 \prec L_2$ *és* $L_2 \prec L_3$, *akkor* $L_1 \prec L_3$.

3.3. Definíció. (Visszavezetésre való zártság)
Egy $\mathcal{C}$ osztály zárt a visszavezetésre nézve, ha bármely L_1, L_2 nyelvre igaz a következő:

$$\text{ha } L_1 \prec L_2 \text{ és } L_2 \in \mathcal{C}, \text{ akkor } L_1 \in \mathcal{C}.$$ □

3.4. Állítás.
A **P**, **NP**, **coNP**, **NP** $\cap$ **coNP**, **PSPACE** *és* **EXP** *osztályok zártak a visszavezetésre nézve.*

Bizonyítás.

A bizonyítás meglehetősen triviális, de demonstráljuk azt a **P** esetében!

Legyen tehát $L_2 \in \mathbf{P}$. A feltétel szerint van L_1-nek L_2-re való f Karp-redukciója, melyet kiszámít egy $\mathcal{O}\left(n^k\right)$ időkorlátos determinisztikus T_1 Turing-gép. Továbbá van L_2-t eldöntő $\mathcal{O}\left(n^l\right)$ időkorlátos determinisztikus T_2 Turing-gép. Ekkor az a Turing-gép, mely először az x inputra szimulálja T_1-et, majd az így kiszámolt $f(x)$ szóra szimulálja T_2-t, eldönti L_1-et, mégpedig $\mathcal{O}\left(n^{kl}\right)$ időkorláttal. Tehát $L_1 \in \mathbf{P}$.□

3.5. Definíció. (Teljesség)

Legyen $\mathcal{C}$ egy nyelvosztály. Egy L nyelvről akkor mondjuk, hogy *$\mathcal{C}$-teljes*, ha

(1) $L \in \mathcal{C}$, és

(2) minden $L' \in \mathcal{C}$ nyelv esetén $L' \prec L$. □

Ha már sikerült találtunk egy $\mathcal{C}$-teljes nyelvet, akkor más $\mathcal{C}$-teljes nyelveket találni könnyebb dolog. Erről szól a következő, triviálisan igaz állítás:

3.6. Állítás.

Ha az L_1 nyelv $\mathcal{C}$-teljes, $L_2 \in \mathcal{C}$ és $L_1 \prec L_2$, akkor L_2 is $\mathcal{C}$-teljes.

A teljes nyelveket jól használhatjuk például annak igazolására, hogy két bonyolultsági osztály megegyezik. A következő állítás szól erről:

3.7. Állítás.

Ha a $\mathcal{C}$ és $\mathcal{C}'$ nyelvosztályok zártak a visszavezetésre nézve, és van olyan L nyelv, mely mindkét osztályban teljes, akkor $\mathcal{C} = \mathcal{C}'$.

Bizonyítás.

Mivel L $\mathcal{C}$-teljes, így $L \in \mathcal{C}$. Továbbá mivel L $\mathcal{C}'$-teljes, így minden $L' \in \mathcal{C}'$ esetén $L' \prec L$, melyekre $L' \in \mathcal{C}$ is teljesül, hiszen $\mathcal{C}$ zárt a visszavezetésre nézve. Innen következik, hogy $\mathcal{C}' \subseteq \mathcal{C}$.

A fordított irányú tartalmazás, azaz hogy $\mathcal{C} \subseteq \mathcal{C}'$, teljesen hasonlóan látható be. □

A továbbiakban **NP**-teljes problémákkal fogunk foglalkozni. Ha egy ilyen problémáról sikerülne belátni, hogy benne van **P**-ben, azzal a fenti tétel értelmében belátnánk, hogy $\mathbf{P} = \mathbf{NP}$.

3.2. A SAT nyelv NP-teljessége

Nagy kérdés, hogy léteznek-e egyáltalán teljes nyelvek? Létezik-e akár egy **NP**-teljes nyelv is? Bármennyire is valószínűtlen, a kérdésre igen a válasz. Sőt, számos nyelvről sikerült bebizonyítani, hogy **NP**-teljes. Ami igazán izgalmassá teszi ezt a témát, az az, hogy ezen nyelvek között nagyon sok nevezetes és alapvető nyelv található.

A 3.6. állítás alapján tudjuk, hogy ha már van egy **NP**-teljes nyelvünk, onnantól viszonylag már könnyű más **NP**-teljes nyelveket is találnunk. De hogyan találjuk meg az első **NP**-teljes nyelvünket? Nincs más lehetőségünk, mint a kérdéses nyelvre valamilyen módon visszavezetni az *összes* **NP**-beli nyelvet; és ez nehéz munka.

Történelmileg úgy alakult, hogy elsőként egy logikai nyelvről, a SAT-ról látták be az **NP**-teljességet. Ennek bizonyítását nézzük meg lentebb, előbb a szükséges (*nulladrendű logikából* ismerős) fogalmakat átismételve. Természetes módon felmerülhet bennünk a kérdés: miért pont egy ilyen nyelv bizonyult ilyen alapvető fontosságúnak? Talán azért, mert a logika kifejezőereje képes akár a Turing-gépek működését is leírni.

3.8. Definíció. (Boole-formula)

Legyen adott megszámlálható sok $x_1, x_2, \ldots$ ún. *ítéletváltozó*. A *Boole-formulák* induktíve definiálhatók a következőképpen:

(1) x_i és $\overline{x}_i$ Boole-formulák (ún. *literálok*), ha x_i ítéletváltozó.

(2) $(\phi_1 \wedge \phi_2)$ és $(\phi_1 \vee \phi_2)$ Boole-formulák, ha ϕ_1 és ϕ_2 Boole-formulák. □

A Boole-formulákban szereplő zárójelpárok közül egyeseket esetenként elhagyunk, a konjunkció ($\wedge$), illetve a diszjunkció ($\vee$) műveletek asszociativitását kihasználva.

3.9. Definíció. (Kiértékelés)

Legyenek adott megszámlálható sok $x_1, x_2, \ldots$ ítéletváltozó. Ezen ítéletváltozók egy *kiértékelése* alatt egy $T : \{x_1, x_2, \ldots\} \mapsto \{0, 1\}$ függvényt értünk. Most induktíve definiáljuk azt a tényt, hogy a T kiértékelés *kielégíti* a ϕ Boole-formulát, röviden $T \models \phi$:

(1) $T \models x_i$ akkor és csak akkor, ha $T(x_i) = 1$.

(2) $T \models \overline{x}_i$ akkor és csak akkor, ha $T(x_i) = 0$.

(3) $T \models (\phi_1 \wedge \phi_2)$ akkor és csak akkor, ha $T \models \phi_1$ és $T \models \phi_2$.

(4) $T \models (\phi_1 \vee \phi_2)$ akkor és csak akkor, ha $T \models \phi_1$ vagy $T \models \phi_2$. □

Szokás a negáción, konjunkción és diszjunkción kívül más logikai műveleteket is használni. Ezek azonban kifejezhetők az előző három művelet segítségével[1]:

- Implikáció:

$$\phi_1 \Rightarrow \phi_2 \quad \sim \quad \overline{\phi}_1 \vee \phi_2$$

- Ekvivalencia:

$$\phi_1 \Leftrightarrow \phi_2 \quad \sim \quad (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$$

3.10. Definíció. (Kielégíthető formula)

Egy ϕ Boole-formula *kielégíthető*, ha a benne szereplő ítéletváltozóknak van olyan T kiértékelése, hogy $T \models \phi$. □

3.11. Definíció. (SAT)

A sat a kielégíthető Boole-formulák nyelve. □

A sat **NP**-teljessége bizonyításának az alapötlete a következő: egy Turing-gép működését fejezzük ki egy Boole-formula alakjában. Ez a formula pontosan akkor legyen kielégíthető, ha a Turing-gép elfogadó állapotban áll meg.

3.12. Tétel.

A sat *egy* **NP***-teljes nyelv.*

Bizonyítás.

[1] Az átíráshoz szükséges még a $\overline{\overline{\phi}} \sim \phi$ szabály, illetve a de Morgan azonosságok:

$$\overline{\phi_1 \wedge \phi_2} \quad \sim \quad (\overline{\phi}_1 \vee \overline{\phi}_2)$$
$$\overline{\phi_1 \vee \phi_2} \quad \sim \quad (\overline{\phi}_1 \wedge \overline{\phi}_2)$$

Először is azt kell belátni, hogy SAT $\in$ **NP**. Ez a *tanú-tétel* segítségével könnyen belátható: egy Boole-formula kielégíthetőségének a tanúja a formulában szereplő ítéletváltozóknak egy olyan kiértékelése, mely kielégíti a formulát. Nyilván egy ilyen tanú hossza *polinomiális* a formula hosszának függvényében, továbbá az adott kiértékelés mellett a formula logikai értékének kiszámítása megoldható *polinom időkorlátos determinisztikus* Turing-géppel.

A tétel érdemi részéhez azt kell igazolni, hogy tetszőleges $L \in \mathbf{NP}$ nyelvre $L \prec$ SAT. Azaz tetszőleges $x \in I^*$ inputra meg kell adnunk egy ϕ Boole-formulát, hogy

$$\phi \text{ kielégíthető} \quad \Leftrightarrow \quad x \in L$$

Annyi még a megkötés, hogy ϕ-nek az x alapján polinom időben és determinisztikusan kiszámíthatónak kell lennie (a Karp-redukció definíciója miatt).

Mivel $L \in \mathbf{NP}$, a tanú-tétel (1.32. tétel) alapján van olyan $L' \in \mathbf{P}$ nyelv és olyan $c > 0$ konstans, hogy

$$x \in L \quad \Leftrightarrow \quad \exists y \in I^*, \text{melyre } |y| \leq |x|^c \text{ és } x\#y \in L'$$

Mivel $L' \in \mathbf{P}$, így létezik olyan egyszalagos, polinom időkorlátos determinisztikus $T = \langle \Sigma, Q, q_0, F, \delta \rangle$ Turing-gép, mely L'-t dönti el. A T tehát az $x\#y$ inputot kapja[2], az ezen végzett (determinisztikus) számítási sorozatát kell valami módon egy Boole-formula alakjára leképeznünk.

Ha x hosszát n-nel jelöljük, akkor az $|x\#y|$ polinomiális függvénye n-nek. Továbbá a T által az $x\#y$ inputon megtett lépések száma is valamilyen polinomja n-nek; tegyük fel, hogy e lépések száma legfeljebb n^d (valamilyen alkalmas d konstansra).

A lényeg, hogy a továbbiakban csak legfeljebb n^d számítási lépést fogunk belekódolni a végeredményül kapott ϕ Boole-formulába, illetve a szalagnak csupán n^d celláját fogjuk figyelembe venni (hiszen T legfeljebb ennyit tud manipulálni n^d számítási lépés során). A ϕ megkonstruálásához szükségünk lesz a következő ítéletváltozókra:

$$\left.\begin{array}{rcl} C_{i,j}^{\sigma} & = & \begin{cases} 1 & \text{, ha az } i. \text{ lépés után a } j. \text{ cella tartalma } \sigma \\ 0 & \text{, különben} \end{cases} \\ F_{i,j} & = & \begin{cases} 1 & \text{, ha az } i. \text{ lépés után a fej a } j. \text{ cellán áll} \\ 0 & \text{, különben} \end{cases} \\ Q_i^q & = & \begin{cases} 1 & \text{, ha az } i. \text{ lépés után a Turing-gép a } q \text{ állapotban van} \\ 0 & \text{, különben} \end{cases} \end{array}\right\} \begin{array}{l} \text{minden } \sigma \in \Sigma \text{ -ra,} \\ \text{minden } q \in Q \text{ -ra,} \\ \text{minden } i,j = 0 \ldots n^d \text{ -re} \end{array}$$

Tehát ez összesen $(n^d+1)^2 \cdot (|\Sigma|+1) + (n^d+1) \cdot |Q|$ db. ítéletváltozó. A lényeg, hogy ezen ítéletváltozók száma is polinomiális függvénye n-nek.

Ezután következhet a T adott inputon elvégzett számítási sorozatát leíró ϕ Boole-formula megkonstruálása. A ϕ a következőképpen képzett formulák *konjunkciója* lesz:

(1) Először technikai jellegű megkötéseket fogalmazunk meg, ezek a következők:

(a) A szalag minden celláján egy adott időpontban éppen egy betű van:

$$\left(\bigvee_{\sigma \in \Sigma} C_{i,j}^{\sigma}\right) \wedge \bigwedge_{\substack{\sigma_1, \sigma_2 \in \Sigma \\ \sigma_1 \neq \sigma_2}} \left(\overline{C}_{i,j}^{\sigma_1} \vee \overline{C}_{i,j}^{\sigma_2}\right)$$

Például ha $\Sigma = \{0, 1, \triangleright, \sqcup\}$, akkor ez a formula így néz ki:

$$\left(C_{i,j}^{0} \vee C_{i,j}^{1} \vee C_{i,j}^{\triangleright} \vee C_{i,j}^{\sqcup}\right) \wedge$$
$$\wedge \left(\overline{C}_{i,j}^{0} \vee \overline{C}_{i,j}^{1}\right) \wedge \left(\overline{C}_{i,j}^{0} \vee \overline{C}_{i,j}^{\triangleright}\right) \wedge \left(\overline{C}_{i,j}^{0} \vee \overline{C}_{i,j}^{\sqcup}\right) \wedge \left(\overline{C}_{i,j}^{1} \vee \overline{C}_{i,j}^{\triangleright}\right) \wedge \left(\overline{C}_{i,j}^{1} \vee \overline{C}_{i,j}^{\sqcup}\right) \wedge \left(\overline{C}_{i,j}^{\triangleright} \vee \overline{C}_{i,j}^{\sqcup}\right)$$

[2] Most az $x \in L$ és az y tanú közé elválasztójelként a # karaktert tesszük; persze ez csak egy nem lényegbeli változtatás a tanú-tételben használt elválasztójelhez képest.

Ezeket a formulákat minden $i, j = 0 \dots n^d$ értékre előállítjuk.
Összesen tehát $(n^d + 1)^2$ ilyen formulánk lesz.

(b) A fej minden időpontban pontosan egy cellán áll:

$$\left(\bigvee_{j=0 \dots n^d} F_{i,j} \right) \wedge \bigwedge_{\substack{j_1, j_2 = 0 \dots n^d \\ j_1 \neq j_2}} \left(\overline{F}_{i,j_1} \vee \overline{F}_{i,j_2} \right)$$

Ezt a formulát minden $i = 0 \dots n^d$ értékre előállítjuk.
Összesen tehát $n^d + 1$ ilyen formulánk lesz.

(c) A Turing-gép minden időpontban pontosan egy állapotban van:

$$\left(\bigvee_{q \in Q} Q_i^q \right) \wedge \bigwedge_{\substack{q_1, q_2 \in Q \\ q_1 \neq q_2}} \left(\overline{Q}_i^{q_1} \vee \overline{Q}_i^{q_2} \right)$$

Ezt a formulát minden $i = 0 \dots n^d$ értékre előállítjuk.
Összesen tehát $n^d + 1$ ilyen formulánk lesz.

(2) Ezután le kell írnunk, hogy a Turing-gép egy időpillanatból hogyan mehet át egy másikba:

(a) Le kell írnunk a δ átmenetfüggvényt: ha $\delta(q, \sigma) = (q', \sigma', m)$, akkor

$$\left(Q_i^q \wedge C_{i,j}^\sigma \wedge F_{i,j} \right) \quad \Rightarrow \quad \left(Q_{i+1}^{q'} \wedge C_{i+1,j}^{\sigma'} \wedge F_{i+1,j'} \right)$$

ahol

$$j' \quad = \quad \begin{cases} j-1 & , \text{ha } m = \leftarrow \\ j & , \text{ha } m = - \\ j+1 & , \text{ha } m = \rightarrow \end{cases}$$

Például: ha $\delta(q_5, 1) = (q_7, 0, \leftarrow)$, akkor:

$$\left(Q_i^{q_5} \wedge C_{i,j}^1 \wedge F_{i,j} \right) \quad \Rightarrow \quad \left(Q_{i+1}^{q_7} \wedge C_{i+1,j}^0 \wedge F_{i+1,j-1} \right)$$

Ezeket a formulákat minden $i, j = 0 \dots n^d - 1$ értékre előállítjuk.
Összesen $\mathcal{O}\left((n^d)^2\right)$ ilyen formulánk lesz, mivel a δ által megadott hozzárendelések száma független n-től (azaz konstans).

(b) Az előbbi formulák leírják, hogy mi változik egyik időpillanatról a másikra. Azt is meg kell mondanunk, hogy más változás nincs: a szalag azon cellái, melyek felett nincs a fej, megtartják értéküket.

$$\overline{F}_{i,j} \quad \Rightarrow \quad \left(C_{i,j}^\sigma \Leftrightarrow C_{i+1,j}^\sigma \right)$$

Ezt a formulát minden $i, j = 0 \dots n^d - 1$ és minden $\sigma \in \Sigma$ értékre fel kell írni.
Összesen $\mathcal{O}\left((n^d)^2\right)$ ilyen formulánk lesz, mivel a $|\Sigma|$ független n-től.

(c) A Turing-gép elfogadó állapotban való megállására vonatkozó kritériumot is be kell vennünk a formulák közé, csak hogy szavatoljuk, hogy a teljes formulánk akkor és csak akkor legyen kielégíthető, ha a Turing-gép elfogadja az inputját:

$$\bigvee_{\substack{f \in F \\ i = 0 \dots n^d}} Q_i^f$$

(3) Végül a Turing-gép kezdő konfigurációját kell még megadnunk.

(a) A gép kezdetben (a 0. időpillanatban) a q_0 állapotban van:

$$Q_0^{q_0}$$

(b) A fej kezdetben a 0. pozíción áll:

$$F_{0,0}$$

(c) A szalag kezdeti tartalmát is meg kell adni. Álljon az input a $\sigma_i \in \Sigma$ betűkből, azaz $x\#y = \sigma_1, \sigma_2, \ldots, \sigma_k$. Ekkor állítsuk elő a következő formulát:

$$C_{0,0}^{\triangleright} \wedge C_{0,1}^{\sigma_1} \wedge C_{0,2}^{\sigma_2} \wedge \ldots \wedge C_{0,k}^{\sigma_k} \wedge C_{0,k+1}^{\sqcup} \wedge \ldots \wedge C_{0,n^d}^{\sqcup}$$

Például ha az input 010#0, akkor az előző formula így fest:

$$C_{0,0}^{\triangleright} \wedge C_{0,1}^{0} \wedge C_{0,2}^{1} \wedge C_{0,3}^{0} \wedge C_{0,4}^{\#} \wedge C_{0,5}^{0} \wedge C_{0,6}^{\sqcup} \wedge \ldots \wedge C_{0,n^d}^{\sqcup}$$

Látható, hogy a ϕ formula n-nek függvényében polinom időben megadható, és ϕ konstrukciója miatt ϕ akkor és csak akkor elégíthető ki, ha $x\#y \in L'$, azaz ha $x \in L$. Ezzel a tételt bebizonyítottuk. □

3.3. Példák NP-teljes nyelvekre

Mivel a SAT nyelv személyében találtunk egy **NP**-teljes nyelvet, a 3.6. állítás alapján innentől már könnyebb dolgunk van akkor, mikor egy L nyelv **NP**-teljességét be akarjuk bizonyítani. Csupán két vizsgálatot kell elvégeznünk L-lel kapcsolatosan:

(1) Be kell látnunk, hogy $L \in \mathbf{NP}$: a tanú-tétel alkalmazásával általában könnyen belátható.

(2) A SAT-nak találnunk kell egy visszavezetését az L-re.

Ebben a fejezetben a 3 színnel színezhető gráfok (lásd: 1.33. definíció) nyelvéről fogjuk belátni, hogy **NP**-teljes. A 3-SZIN **NP**-teljességét nem közvetlenül a SAT **NP**-teljességéből fogjuk levezetni, hanem az ún. 3-SAT nyelv **NP**-teljességéből; ennek leírása következik az alábbiakban.

3.13. Definíció. (Konjunktív normálforma)

Egy Boole-formula *konjunktív normálformában* van, ha $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ alakú, ahol minden ϕ_i elemi diszjunkció.
Elemi diszjunkció alatt egy $(l_1 \vee l_2 \vee \ldots \vee l_m)$ alakú formulát értünk, ahol minden l_j *literál*. □

3.14. Definíció. (k-konjunktív normálforma)

Legyen $k \geq 1$ természetes szám. Egy Boole-formula *k-konjunktív normálformában* van, ha

- konjunktív normálformában van, és
- benne minden elemi diszjunkció legfeljebb k db. literált tartalmaz. □

Példa egy 4-konjunktív normálformájú Boole-formulára:

$$(\overline{x}_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_4 \vee \overline{x}_7 \vee \overline{x}_{18}) \wedge (x_9 \vee \overline{x}_{10} \vee x_{13})$$

3.15. Definíció. (k-SAT)

A k-SAT a kielégíthető k-konjunktív normálformájú Boole-formulák nyelve. □

Ismert a következő összefüggés: bármely Boole-formula átírható logikailag ekvivalens 3-konjunktív normálformába. Azaz van a SAT-nak *visszavezetése* a 3-SAT-ra. Ennek következménye a következő állítás:

3.16. Állítás.

A 3-SAT *egy* **NP***-teljes nyelv.*

Az állítást nem bizonyítjuk, csupán felhasználjuk a következő tétel bizonyításához:

3.17. Tétel.

A 3-SZIN *egy* **NP***-teljes nyelv.*

Bizonyítás.

Először azt kell belátni, hogy 3-SZIN $\in$ **NP**. Ezt már bizonyítottuk az 1.34. állítás kapcsán.

Jöhet a bizonyítás érdemi része: meg kell adnunk a 3-SAT-nak egy visszavezetését a 3-SZIN-re. Azaz tetszőleges 3-konjunktív normálformájú ϕ formulához polinom időben meg kell adnunk egy G gráfot úgy, hogy

$$\phi \in \text{3-SAT} \quad \Leftrightarrow \quad G \in \text{3-SZIN}$$

Szavakkal leírva: ϕ pontosan akkor kielégíthető, ha G 3 színnel színezhető.
Vezessük be a következő jelöléseket:

- $x_1, x_2, \ldots, x_n$: a ϕ-ben szereplő ítéletváltozók.
- $\phi_1, \phi_2, \ldots, \phi_m$: a ϕ-ben szereplő elemi diszjunkciók.

Tegyük fel, hogy minden ϕ_i pontosan 3 literált tartalmaz. Ezt nyugodtan feltehetjük, ugyanis ha egy elemi diszjunkció kevesebb mint 3 literált tartalmazna, nyugodtan kiegészíthetjük 3-tagúra. Például $x_1 \vee \overline{x}_2$ helyett vehetjük az $x_1 \vee \overline{x}_2 \vee \overline{x}_2$ formulát.
A G csúcsai a következők lesznek:

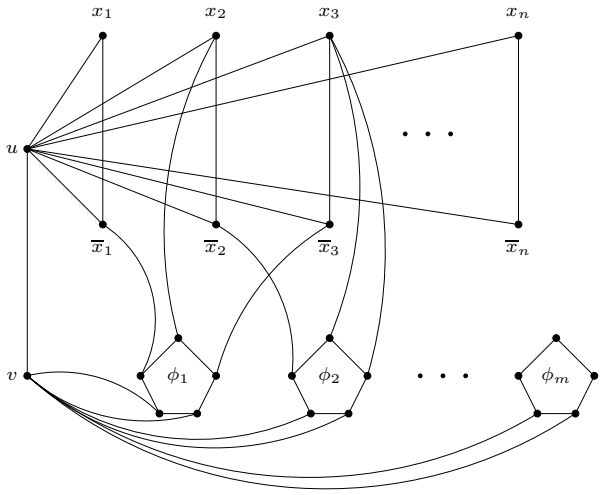
- Lesz két „belépő” csúcs: u és v.
- Minden x_i ítéletváltozónak 2-2 csúcsot feleltetünk meg: az egyiket x_i-vel, a másikat $\overline{x}_i$-vel jelöljük.
- Minden ϕ_i elemi diszjunkciónak 5-5 csúcs fog megfelelni, ezek jele: ϕ_i^1, ϕ_i^2, ϕ_i^3, ϕ_i^4, ϕ_i^5.

A G élei a következők:

- (u, v)
- minden $i = 1 \ldots n$ -re: $(u, x_i), (u, \overline{x}_i)$
- minden $i = 1 \ldots n$ -re: $(x_i, \overline{x}_i)$
- minden $i = 1 \ldots m$ -re: $(v, \phi_i^4), (v, \phi_i^5)$
- minden $i = 1 \ldots m$ -re: $(\phi_i^1, \phi_i^2), (\phi_i^2, \phi_i^3), (\phi_i^3, \phi_i^4), (\phi_i^4, \phi_i^5), (\phi_i^5, \phi_i^1)$
- minden $i = 1, \ldots, m$ -re és $j = 1, 2, 3$ -ra: ha ϕ_i-nek a j. literálja
  - az x_k ítéletváltozó, akkor $(\phi_i^j, \overline{x}_k)$
  - az $\overline{x}_k$ literál, akkor (ϕ_i^j, x_k)

Tehát minden ítéletváltozót összekötünk a negáltjával, illetve mindkettőjüket az u csúccsal. Minden ϕ_i elemi diszjunkciónak egy-egy ötszög felel meg, melyek alsó 2-2 csúcsát összekötjük a v csúccsal. A ϕ_i-nek megfelelő ötszög felső 3 csúcsa felel meg a ϕ_i 3 literáljának. Ezt a három csúcsot a negáltjaiknak megfelelő csúcsokkal kötjük össze.

A következő ábrán látható módon alakítjuk ki a G-t. Az ábra azt az esetet szemlélteti, mikor $\phi_1 = x_1 \vee \overline{x}_2 \vee x_3$ és $\phi_2 = x_2 \vee \overline{x}_3 \vee \overline{x}_3$.



Az rögtön látható, hogy G könnyen (polinom időben) felépíthető ϕ ismeretében. Érdekességképpen megjegyezzük, hogy G mérete a ϕ méretének függvényében nemcsak hogy polinomiális, de csupán *lineáris*. Igazolnunk kell még, hogy

$$\phi \in \text{3-SAT} \quad \Leftrightarrow \quad G \in \text{3-SZIN}$$

Most csak visszafelé bizonyítjuk ezt az összefüggést (a másik irányú bizonyítás is viszonylag könnyen megadható). Azaz feltesszük most, hogy G-nek van 3 színnel színezése; ez a 3 szín (a szemléletesség kedvéért) legyen a piros, a zöld és a sárga. Be fogjuk látni, hogy ilyenkor a ϕ kielégíthető: megadjuk az $x_1, \ldots, x_n$ ítéletváltozóknak egy olyan T kiértékelését, hogy $T \models \phi$.

Színezzük be az u-t sárgára, a v-t pedig zöldre. Mivel u sárga, így minden x_i és $\overline{x}_i$ csak a piros és zöld színek egyikét veheti fel (a sárgát nem, mivel össze vannak kötve u-val). Ráadásul mivel x_i és $\overline{x}_i$ egymással is össze vannak kötve, csak ellentétes színeket vehetnek fel; azaz ha x_i piros, akkor $\overline{x}_i$ zöld, illetve ha x_i zöld, akkor $\overline{x}_i$ piros. A piros szín jelölje a hamis logikai értéket, a zöld az igazat; azaz

$$T(x_i) \quad = \quad \begin{cases} 0 & \text{, ha } x_i \text{ piros} \\ 1 & \text{, ha } x_i \text{ zöld} \end{cases}$$

A sárga színre tekinthetünk úgy, mint egy határozatlan logikai értéket képviselő színre.

Hogy belássuk, hogy $T \models \phi$, csak annyit kell bizonyítanunk, hogy minden ϕ_i ötszögben van zöld csúcs. Nyilván az ötszögek alsó 2-2 csúcsa közül egyik sem lehet zöld, mivel v zöld. Azaz ezek az alsó csúcsok csak pirosak vagy sárgák lehetnek. Könnyen látható, hogy az ötszögek felső 3 csúcsa között muszáj legalább egy zöldnek lennie, mivel egy ötszög csúcsai nem festhetők ki felváltva két színnel (pirossal és sárgával). □

A 3-SZIN nyelv csak egy példa **NP**-teljes nyelvekre. Az irodalomban több ezer nyelvről bizonyították hasonlóképpen az **NP**-teljességet. Ezek közül pár nevezetes nyelv (azaz probléma).

- Hamilton-kört tartalmazó gráfok nyelve (HAM, lásd: 1.3.2. fejezet)
- utazóügynök-probléma

- maximális független csúcshalmaz keresése gráfokban
- minimális/maximális feszítőfa keresése gráfokban
- lineáris programozási problémák
- sávszélesség minimalizálásának problémája
- prímszámok nyelve (prímtesztelés problémája)
- számok faktorizációja (szorzótényezőkre bontás problémája) – nyilvános kulcsú kriptográfiai protokollok kapcsán

IRODALOMJEGYZÉK

[1] C. H. Papadimitriou: *Számítási bonyolultság.*
Novadat, 1999.

[2] Rónyai L., Ivanyos G., Szabó R.: *Algoritmusok.*
Typotex, 1998.

[3] Cormen, Leiserson, Rivest: *Algoritmusok.*
Műszaki Kiadó, 1999.