Az előző részben egy bolha üldözése során pár dolgot megtudtunk az algoritmusokról, speciálisan az algoritmusokkal megadható és az általános, „véletlen" sorozatok közti különbségről. E különbségről sajnos csak annyi derült ki, hogy általános sorozat – Cantor átlós elve szerint – több van, mint algoritmikus, vannak programmal meg nem adható sorozatok. Láttunk is erre példát a megállási függvény „átlója", $H(n,n)$ például ilyen volt.

E sorozat, amely speciálisan 0-kból és 1-ekből áll, azonban aligha tekinthető egy pénzdobás (0=fej,1=írás) tipikus eredményének. Igen egyszerű például sok helyen meghatározni: hajtsuk végre az összes programot 10^{10} lépésig, nagy részük végeredményt fog szolgáltatni.

Nem tartanánk azt sem véletlennek, ha a bolha mindig éppen eggyel az előtt a mező előtt járna, amelyre mi, pl. az V. eset stratégiája szerint, csapnánk. Különösképpen pedig nem tudjuk a választ a legelőször feltett kérdésre: ha csak néhány, véges sok lépését ismerjük a bolhának (vagy egy fej-írás sorozatnak), honnan ered az az érzetünk, hogy ez szabályos, illetve véletlenszerű. Hiszen bármely sorozat véges sok eleme előállítható programmal (készítsük el e programot), a $\texttt{KONSTANS}_0(n)$ utasítás segítségével.

A Kolmogorov-komplexitás

Módosítsuk a bolhás feladatot úgy, hogy miután (mellé)csaptunk leendő áldozatunknak, meglátjuk, hol is volt valójában. Így a következő lépésben elkaphatjuk a bolhát, ha sikerül kifigyelnünk, milyen szabályt követ az ugrálása. Így a feladat első 4 változata 3 időegység alatt biztosan véget ér.

Mi a helyzet az ötödikkel? Tudjuk, hogy egy rögzített szabályt követ a bolha, ezt a szabályt induktívan kell kikövetkeztetnünk, hasonlóan a TV műsorokban, intelligenciatesztekben előforduló feladatokhoz. Célunk itt a legegyszerűbb szabály megtalálása. Például a 3, 5, 7 sorozatot biztosan 9-cel folytatnánk (páratlan számok), és bosszúsak lennénk, ha a bolha tovább ugrálna, mert a páratlan prímekre gondolt és 11 a folytatás.

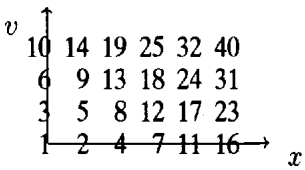
Mit is jelent az, hogy egy szabály egyszerűbb a másiknál? Ezt megfoghatjuk matematikailag úgy, ha a szabályt leíró program hosszát vizsgáljuk. Vezessük be tehát egy $n \in \mathbb{N}$ számra (amelynek persze a már látott módon megfeleltethető egy x sorozat) a $K(n) = \{$a legrövidebb program hossza, amely n-et előállítja$\}$ mennyiséget. Ezt a sorozat Kolmogorov-komplexitásának nevezzük, a híres, 1987-ben elhunyt szovjet matematikus után.

$K(n)$ természetesen függ attól, hogy milyen „jó" programnyelvet használunk. Azonban bármely gépen (ha nincs memóriakorlát) elkészíthető bármely programnyelv fordítója, amelynek hosszát a P nyelvben megírva jelölje C_{PQ}, és a Q nyelvet fordítsa a P-re. Így a P és Q nyelvekhez tartozó K_P és K_Q komplexitásokra igaz lesz $K_P(n) \leq K_Q(n)+C_{PQ}$, minden $n \in \mathbb{N}$ esetén; C_{PQ} n-től nem függ, hiszen a Q-beli programhoz még a C_{PQ} hosszú fordítóprogramot kell hozzáilleszteni. Ehhez persze még az is kell, hogy a P és Q nyelvek ugyanazokat a jeleket – pl. 0–9, vessző stb. – használják. Így, intuitíven

$$K_P(x) \approx K_Q(x), \quad \text{ha} \quad K_P(X) \gg C_{PQ}.$$

$K(x)$ nem kiszámítható

Ha ismerjük $K(x)$-et, az x-et előállító legegyszerűbb szabályt meg is kereshetjük: a $K(x)$ hosszú programokat kell lépésenként(!) végrehajtanunk (előfordulhatnak köztük olyanok is, amelyek végtelen ciklusba esnek), azaz az összes $K(x)$ hosszú programot az összes $t \in \mathbb{N}$ lépésszámig ki kell próbálnunk, az egyik biztosan szolgáltatni fogja x-et. A próbálgatás módja pedig azonos a 2-dimenziós bolhaüldözéshez: számpárokat kell kipróbálnunk, az első paraméter a program sorszáma, a második pedig a t lépésszám.



Célul most a $K(x)$-et meghatározó algoritmus keresését tűzhetnénk ki. Most már azonban tudjuk, hogy nem minden függvény kiszámítható. $K(x)$ meghatározását a $H(k,n)$ megállási függvény ismeretében elvégezhetjük: kipróbáljuk hossz szerinti növekvő sorrendben az összes programot, amely nem esik végtelen ciklusba. Mivel azonban $H(k,n)$ nem kiszámítható, kénytelenek vagyunk ismét csak a lépésenkénti kipróbálás ötletét használni. Például az *ábrán* látható sorrendben próbáljuk ki a programokat a megfelelő lépésszámig. Mivel minden véges(!) sorozatot elő tudunk állítani programmal, előbb-utóbb találunk is egy ilyent, pl. az n_0-adikat a t_0 végrehajtási idővel. Ha minden, az n_0-adik programnál rövidebb program vagy véget ért ekkorra, vagy valamilyen módon bizonyítani tudjuk, hogy végtelen ciklusba esik (pl. $0\,\text{UGRIK}_0(0,0)$), készen vagyunk. Általában azonban lesznek programok, amelyeket még nem próbáltunk ki elegendő sokáig, ezek között lehet, hogy valamelyik egyszerűbb szabályt szolgáltat x-re.

3. Tétel *$K(x)$nem kiszámítható.*

Bizonyítás. A „legkisebb, magyar nyelven ezer betűvel le nem írható szám" tartalmi paradoxon ötletét vegyük át (ez a szám azért nem létezhet, mivel éppen most írtuk le, ezernél jóval kevesebb betűvel). Tekintsük tehát a legkisebb, C-nél nagyobb komplexitású számot, ahol C-t majd később választjuk meg. Ha $K(x)$ kiszámítható volna, az összes

számra kiszámítanánk, amíg az első C-nél nagyobbat el nem érjük. Tételezzük fel, hogy ez a program létezik, és C-t, mint adatot tartalmazza.

Legyen C_0 a program hossza az adat nélkül. C-t, mint adatot legfeljebb $\log_{10}C + 1$ decimális jeggyel leírhatjuk. Így egy x-et előállító programot kapunk, amelynek hossza

$$C_0 + \log_{10}C + 1 \geq K(x) \geq C,$$

azaz 10-et erre a hatványra emelve

$$C \cdot 10^{C_0+1} \geq 10^C, \quad 10^{C_0+1} \geq 10^C \ /C,$$

ahol $10^C/C$ tetszőlegesen nagy lehet, tehát 10^{C_0+1}-nél is nagyobb. Ekkora C választásával tehát ellentmondásra jutunk. Itt persze szó sincs paradoxonról: indirekten azt igazoltuk, hogy $K(x)$ mégsem kiszámítható. □

A véletlenszerűség és a komplexitás

Most már módunkban áll jobb jellemzését adni a véletlenszerűségnek. Egy algoritmikusan előállított végtelen sorozatot egy véges hosszúságú program szolgáltat: a kezdőszeletek komplexitása tehát egy megfelelő hossz után már nagyon kicsiny lesz. A véletlenszerűségérzet ezzel ellentétben a magas komplexitáshoz társul. De mekkora lehet egy x véges sorozat komplexitása? Biztosan nem lehet nagyobb mint a PRINT "x" utasítás megfelelő programnyelvre való fordítása, azaz $l(x) + C$, ahol $l(x)$ a sorozatban szereplő számjegyek és vesszők összhossza, C pedig x-től nem, csak a programnyelvtől függ.

Másrészt tekintsük a programokat a már látott módon, a 0–9 számjegyekkel és a vessző segítségével kódolva. Ekkor l-nél rövidebb program

$$1 + 11 + 11^2 \ldots + 11^{l-1} = \frac{11^l - 1}{10} < 11^l$$

darab van, míg olyan sorozat, amelynek a hossza l (az elválasztó vesszőkkel együtt, két egymást követő vessző közé ismét 0-t képzelve) 11^l; így van olyan x sorozat, amelyre $l(x) \leq K(x)$.

Ráadásul az $l+C$ hosszú sorozatoknak, amelyek száma 11^{l+C}, már csak kis része, kevesebb mint $1/11^C$ része lehet l-nél „egyszerűbb".

Egy x véges sorozat véletlenszerűségét tehát úgy foghatjuk meg, hogy x-re $l(x) \approx K(x)$, azaz „nagy" a komplexitása. A véletlen-érzetünk is megmagyarázható: a sorozatok nagy részére nincs igazán rövid leírás, az egyszerűen előállíthatók aránya kicsi. Így valóban ritkábban fordulnak elő a szabályos, mint a szabálytalan sorozatok. A legtöbb rögzített hosszúságú sorozatot csak a `PRINT "x"`-nek megfelelő eljárás állítja elő, ez pedig a bolha üldözése szempontjából csekély segítség, hiszen nem ad jóslatot a bolha következő lépésére.

Egy gyenge prímszámtétel

Végezetül lássunk a Kolmogorov-komplexitásra két, különböző területről származó alkalmazást.

Legyen $p(n)$ az n-nél nem nagyobb prímek száma.[1]

Tétel. $p(n) \geq \dfrac{\log_2 n}{\log_2(\log_2 n)} - 2$, *ha* n *elég nagy.*

Bizonyításvázlat. Tekintsünk egy olyan számítógépet, amely kettes számrendszerbeli be- és kimenetekkel dolgozik. Legyen n egy véletlenszerű szám, azaz amelyre $K(n) \geq l(n) - C + 1 \geq \log_2 n - C$ ($[\log_2 n]$ darab bináris jegyből áll n). Tudjuk, hogy $n = \prod_{p<n \text{ prím}} P_i^{\alpha_p}$, valamely α_p természetes számokra. Mivel $n \geq p^{\alpha_p} \geq 2^{\alpha_p}$, ezért $\alpha_p \leq \log_2 n$, így az α_p-t leíró jegyek száma, $l(\alpha_p) \leq \log_2 \log_2 n$.

Állítsuk elő azt a jelsorozatot, amely $(\log_2 \log_2 n - 1)$ egyesből, egy 0-ból (ezek α_p hosszát határozzák meg) majd sorra $p = 2, 3, 5, \ldots$ esetén α_p-kből áll:

$$\underbrace{11\ldots10}_{\log_2\log_2 n}\underbrace{}_{\alpha_2\text{ jegyei}}\underbrace{}_{\alpha_3\text{ jegyei}}\cdots\underbrace{}_{\alpha_p\text{ jegyei}}$$
$$\underbrace{\phantom{11\ldots10\,\alpha_2\text{ jegyei}\,\alpha_3\text{ jegyei}\cdots\alpha_p\text{ jegyei}}}_{p(n)\text{ darab }n\text{-nél kisebb prím}}$$

Ennek összhossza tehát (legfeljebb) $(p(n) + 1)\log_2 \log_2 n$.

Tekintsünk egy programot, amely tartalmaz egy adatot. Megszámolja először az adat elején lévő egyeseket, az első 0-ig, majd ilyen hosszú „szeletekre" vágja a további részt, így α_p-ket előállítva. Végül pedig az

$$n = \prod_{p \text{ prím}} p^{\alpha_p}$$

kimenettel áll le.

[1] A prímszámtétel azt mondja ki, hogy $\lim_{n\to\infty} \dfrac{p(n)}{n/\log n} = 1$.

Legyen e program hossza adat nélkül C_1, adattal együtt tehát

$$C_1 + (p(n)+1)\log_2\log_2 n.$$

E program előállítja n-et, tehát hossza $K(n)$-nél nem kisebb:

$$C_1 + (p(n)+1)\log_2\log_2 n \geq K(n) \geq \log_2 n - C_2,$$

azaz

$$\left(\frac{C}{\log_2\log_2 n}+1\right) + p(n) \geq \frac{\log_2 n}{\log_2\log_2 n}.$$

Mivel $\log_2\log_2 n$ tetszőleges nagy lehet, van olyan n_0, amelytől kezdve $\left(\frac{C}{\log_2\log_2 n}+1\right) \leq 2$. Ha n-t ennél nagyobbnak választjuk, a bizonyítandót kapjuk.□

Gödel nemteljességi tétele

Gödel tételének egy változata

Az is Hilbert egy kérdése volt akkor, hogy az aritmetika axiómái helyesen írják-e le az egész számok világát. Ez 1931-ben kiábrándító választ nyert: Kurt Gödel (1906–1978) jött arra rá, hogy amennyiben az aritmetikának tetszőleges, algoritmussal felsorolható és ellentmondásmentes axiómarendszerét tekintjük, mindig léteznek olyan állítások, amelyek nem bizonyíthatók ebben az axiómarendszerben.

Meg fogok ilyen állításokat adni, előbb azonban ismét csak az aritmetikai állításokról és bizonyításokról alkotott intuitív képünket kell javítanunk. Hogy ez mennyire fontos, a következő példa mutatja. Mit mondjunk a következő állításról:

„Én nem vagyok bizonyítható".

Ami bizonyítható, az igaz, így az állítás nem lehet hamis, hiszen akkor ő maga mondja ki, hogy bizonyítható. Tehát *bebizonyítottuk*, hogy az állítás igaz?

Hogyan oldható fel az ellentmondás? Úgy, hogy ezt az állítást egy formális rendszerben csak a következőképpen fogalmazhatjuk meg:

„Én nem vagyok bizonyítható az adott formális rendszerben",

és ekkor bizonyításunk kívül esik a rendszeren, de az állítás valóban igaz. (A Gödel-tétel bizonyításai általában ilyen típusú állítást adnak meg, formálisan.)

E példán látható, hogy mindig beszélhetünk egy objektumon pl. ($\mathbb{N}$, összeadás, szorzás), igaz állításokról. Gödel tétele nem azt jelenti, hogy az egész számokra vonatkozó bizonyos állítások „kétféleképpen viselkedhetnek", „mindössze" annyit, hogy nem tudjuk őket az adott alapfogalmakban adott axiómákból levezetni. Ez azzal egyenértékű (ez is bizonyításra szoruló állítás!), hogy van két olyan objektum, amelyek nagyon hasonlítanak: érvényes rajtuk az axiómarendszer, azonban bizonyos állítások különböznek rajtuk. Az aritmetika esetén az egyik objektum az $(\mathbb{N},+,*)$, ahhoz, hogy ezt a másik objektumtól elkülöníthessük, újabb axiómákra van szükség. De mi lenne, ha n összes igaz állítását axiómának tekintenénk? Igaz, hogy ez egy végtelen axiómarendszer lenne, de a végességet nem is követeltük meg. Viszont ahhoz, hogy legalábbis elő tudjunk tetszőleges axiómát állítani, ha szükségünk van rá, kell egy algoritmus, amely ezeket felsorolja. A tétel szerint ilyen nem fog létezni.

Fogjunk most hozzá az aritmetika nyelvének vizsgálatához. Alapfogalmak lehetnek a $\underline{0}$ szám, a rákövetkezés függvénye: $S(n) = n+1$, és az összeadás és szorzás függvények. $\mathbb{N}$ elemei előállíthatók, mint a $\underline{0}$ rákövetkezői. Mivel az axiómarendszernek más modellje is lehet, az újonnan definiált számokat jelöljük aláhúzással. Így $\underline{1} = S(\underline{0})$, $\underline{n+1} = S(\underline{n}) = \underbrace{S\ldots S}_{n+1}(\underline{0})$.

Végül axiómák legyenek az összeadás, szorzás és a $\leq$ reláció legegyszerűbb ismert tulajdonságai:

(1) $\underline{m}+\underline{n} = \underline{p}$, ha $m+n=p$ (mi tudjuk, hogy ez mely $(m,\ n,\ p)$ hármasra teljesül, és erre kell a rendszert „megtanítanunk"),

(2) $\underline{m}*\underline{n} = \underline{p}$, ha $m\cdot n = p$ (hasonlóan),

(3) $x \leq \underline{n}$ pontosan akkor, ha $x = \underline{0},\ \underline{1},\ldots,\underline{n-1}$ (x most nem aláhúzott, hiszen nem tudjuk, hogy előáll-e, mint $S\ldots S(\underline{0})$).

Nem triviális, de igaz, hogy a programokkal, azaz kiszámítható függvényekkel kapcsolatos állítások megfogalmazhatók e nyelven, így pl. a $K(x)$ komplexitás is.

A másik tény, amelyre szükségünk lesz, az, hogy az axiómákból következő bizonyításokat fel tudjuk sorolni egy programmal, feltéve, hogy maguk az axiómák felsorolhatók. (Persze így várni a Fermat–sejtés bizonyítására olyan, mintha ABC sorrendben minden szöveget felírva akarnánk Shakespeare műveit megkapni.)

Most már következhet a

Tétel. *Rögzítsünk egy, az aritmetikát $(\underline{0}, S, +, *, \leq)$ tartalmazó rendszert és axióma rendszerét, amely a fenti axiómákat tartalmazza, ellentmondásmentes és programmal felsorolható. Ekkor van olyan C, hogy a „$C \leq K(x)$" igaz állítások ezen axiómából nem bizonyíthatók.*(Tudjuk, hogy van olyan x, amelyre $C \leq K(x)$, így „$C \leq K(x)$"-nek igaz állításnak kell lennie.)

Bizonyítás. Hasonlóan történik annak igazolásához, hogy $K(x)$ nem kiszámítható. Ismét válasszunk egy nagy C-t és keressük meg a helyes bizonyítások felsorolásában az elsőt, amelynek konklúziója az, hogy valamely x-re $K(x) \geq C$.

Így x előállítható annak a programnak a segítségével, amely felsorolja a helyes bizonyításokat. Ha C jóval nagyobb, mint e program hossza (emlékezzünk arra, hogy C-t, mint adatot kellett még a programban tárolni), ismét ellentmondásra jutunk, egy C-nél nagyobb komplexitású számot egyszerűbben állítunk elő. Itt azonban az ellentmondás csak úgy oldható fel, hogy „$K(x) \geq C$" állítást soha nem is találunk. Ezzel a tételt beláttuk. □

A C konstans elsősorban az axiómarendszer komplexitásától függ, hiszen a formálisan helyes bizonyításokat felsoroló program az axiómáktól függetlenül elkészíthető (az axiómák adatok lehetnek). E megfigyelés alapján Chaitin (1947 –) amerikai matematikus, akitől e bizonyítás származik, a tételt így foglalta össze:

„Ha csak egy 10 font súlyú axiómarendszerünk van, nem tudunk egy 20 font súlyú tételt levezetni belőle."