FOGALMAK ÉS MÓDSZEREK

GÁCS PÉTER

Próbálgatással megoldható feladatokról

Bevezetés

Van néhány feladat az operációkutatásban (kombinatorikában, algebrában, topológiában és a logikában), melynek már régen keresik megoldási algoritmusát, de egyik eddig javasolt eljárás se működik nagyságrendekkel gyorsabban, mint az összes lehetőségek végigpróbálgatása. Operációkutatók számára legismertebb ezek közül az általános egészértékű programozási feladat. Kisebb jelentőségű, de népszerű az utazó ügynök problémája, a „hátizsák" probléma, és még néhány rokon feladat. Egyes kutatókban felmerülhetett már a *gyanú*, hogy a feladatot, mellyel foglalkoznak talán nem is lehet lényegesen kevesebb lépésben megoldani. *Ilyen jellegű állítások bizonyításához — látszólag — még teljesen hiányoznak az eszközeink.* Az utóbbi egy-két év igen érdekes fejleménye, hogy kiderült: egyszerű kombinatorikus eszközökkel sok híres feladatról olyasmit állapíthatunk meg, ami valamennyire *helyettesíti* és ugynakkor erősen valószínűsíti azt a kijelentést, hogy nincs rövid megoldó algoritmus.

Az összes szóban forgó feladat ugyanis egy jól definiálható viszonylag tág feladatcsaládba tartozik. Mármost

sok, egészen egyszerű feladatról kiderült, hogy ezen a családon belül univerzális, a család minden feladata őrá visszavezethető, és így ő ezek között a lehető legnehezebben oldható meg.

Ismertető cikkünk első két részében meghatározzuk a szóban forgó feladatcsaládot. A 3. részben a visszavezethetőség és az univerzalitás fogalmait definiáljuk, a 4. részben megmutatjuk, hogy *van* univerzális feladat, végül az 5. részben példákon mutatjuk be, hogyan lehet egy feladatról bebizonyítani, hogy ő univerzális. A Függelék a technikai részletek iránt érdeklődő olvasóknak kiegészítő információkat ad.

A közölt eredményeket az USA-ban Cook [2] és Karp [1], a Szovjetunióban tőlük függetlenül L. A. Levin [3] érte el.

1. Egy példa

Egy példa elemzésén keresztül jutunk el az általános definícióhoz.

1. *Példa*: Az utazó ügynök problémája.

Adva van

(i) egy n pontból álló halmaz, V, (a városok), és

(ii) a V-beli p_1, p_2 pontpárok közül bizonyosakat összekötő irányítatlan $\{p_1, p_2\}$ élek (útvonalak) A halmaza. V és A meghatároz egy G gráfot.

(iii) minden $a \in A$ útvonalnak van egy $t(a)$ költsége, ahol $t(a)$ nemnegatív valós szám.

(iv) Adott még egy c valós szám, az utazásra szánható összeg felső határa.

Az ügynöknek be kell járnia minden várost úgy, hogy az egész utazás költsége minél kisebb legyen. Ezzel kapcsolatban négy lehetséges matematikai feladat vetődik fel.

I.a) Megvalósítható-e az utazás *c-nél kisebb* költséggel?
I.b) Találjunk egy ilyen útitervet.
II.a) Mennyi az utazás *minimális* költsége?
II.b) Találjunk egy legolcsóbb útitervet.

1. Első észrevételünk, hogy a feladat még nincs pontosan definiálva. Miután mi igazi számolási eljárásokat készülünk vizsgálni, mégpedig elsősorban a végzendő elemi (aritmetikai) műveletek száma szempontjából, adataink nem lehetnek tetszőleges valós számok, csak olyanok, amelyek korlátozott terjedelmű adatmennyiséggel vannak megadva. Megelégedhetünk most a következő definícióval.

1. *Definíció*: k hosszúságú valós számnak nevezzük a kettes számrendszerben $\pm\, a_1 a_2 \ldots a_i, a_{i+1} \ldots a_k$ „kettedestört" alakban felírható számokat ($a_j = 0$ vagy 1). A feladatot úgy kell pontosítani, hogy $t(a)$-ról és c-ről feltesszük, hogy k hosszúságú valós számok, ahol k egy adott természetes szám.

2. Második észrevételünk is a valós számok hosszúságára vonatkozik. Ha azt akarjuk vizsgálni, hogy a szükséges megoldó algoritmus hosszúságán hogyan tükröződik e speciális feladat természete, akkor meg kell akadályoznunk, hogy a szükséges műveletszám nagyságrendekkel növekedhessen egy triviális ok, az adatokban szereplő valós számok irtózatos hosszúsága miatt. Tegyük ezért, például, még a következő kikötést

(v) $t(a)$, c legfeljebb n hosszúságú valós számok.

3. Egy G gráf sincs nyilván azonnal „gépre vihető" formában megadva, de itt a kódolás lehetősége triviális. Mindenesetre, hogy a feladatot formálisan kezelni tudjuk, feltesszük, hogy adva van

(vi) egy kódolás, melynek segítségével feladatunk összes adata egyetlen, 0-kból és 1-esekből álló sorozattal (bináris sorozattal) írható fel.

2. *Definíció*: Egy feladat adatai *terjedelmének* az őket kódoló bináris sorozat hosszát nevezzük.

Könnyű látni, hogy az utazó ügynök problémájában G-t egy n^2 hosszúságú, $t(a)$-t egy $n^2 \cdot n = n^3$, c-t egy n hosszúságú, és így az összes adatot egy legfeljebb $2n^3$ hosszúságú bináris sorozattal kódolhatjuk, $2n^3$ tehát az adatok terjedelme.

4. Ennek a feladatnak van egy kézenfekvő megoldási eljárása — a próbálgatás. A négy feltett kérdés közül bármelyikre válaszolni tudunk akkor, ha végignézzük az összes szóban forgó útvonalakat és kiválasztjuk a legkisebb költségűt.

Nyilvánvaló, hogy elég olyan útvonalakra szorítkozni, melyeknek hosszúsága legfeljebb n^2, de az is megmutatható, hogyha van egy útvonal, mely minden pontot bejár, és költsége c-nél kisebb, akkor van legfeljebb $2n$ hosszúságú ilyen útvonal is.

Próbálgatási eljárásunk tehát a következő lehet. Valamilyen rendszer szerint sorra vesszük az összes $2n$ hosszúságú útvonalat, és kiválasztjuk közülük azt, amely (A): minden ponton keresztülmegy és (B): minimális költségű.

Legyen az összes $2n$ hosszúságú vonalak halmaza U. Akkor U elemszáma, $|U| = n^{2n} = 2^{2n \log n}$. Az egyes útvonalak sorravétele és költségük kiszámítása egyenként legfeljebb konst. n^2 lépést vesz igénybe. Ezért az egész eljárás

munkaigényének nagyságrendje $n^2 \cdot 2^{2n \log n}$ lépés. Ez már egészen kis n-ek esetén is csillagászati szám. A próbálgatáson persze még sokat lehet racionalizálni, de a lényegen, az *exponenciális* nagyságrendű lépésszámon így nem változtathatunk. Exponenciálisnál sokkal kevesebb lépésben dolgozó eljárást erre a feladatra, és még sok távoli rokonára, nem ismer a tudomány. Az összes feladatnál, amiről a továbbiakban szó lesz, ez a tét: exponenciális vagy jóval kisebb lépésszám. Ilyen lépték mellett érthető a következő definíció.

3. *Definíció*: Egy feladat *könnyen megoldható*, ha van egy k konstans és egy eljárás, amely, ha az adatok terjedelme n, a megoldást n^k lépés alatt szolgáltatja. Az ilyen algoritmust néha *jónak*, ill. *hatékonynak* is nevezzük. (Exponenciálishoz képest az n-hatvány mennyiségű lépésszám valóban gyerekjáték!)

Ha $f(n)$ és $g(n)$ nemnegatív függvények, akkor írjuk:

$$f \lesssim g$$

(g *majorálja* f-et), ha van egy olyan k, hogy minden elég nagy n-re

$$f(n) \leq (g(n) + 1)^k$$

(Rendezésünk ugyanazt a hozzáállást tükrözi: Hatvány nagyságrendű különbségektől eltekintünk.) $f \approx g$ (f *ekvivalens* g-vel), ha $f \lesssim g$ és $g \lesssim f$.

5. Ha $n-k$ hatvány nagyságrendű különbségektől eltekintünk, akkor az I.a) és II.a) feladatok ekvivalensek egymással. Ugyanez vonatkozik a b) jelű feladatokra is.

Nyilvánvaló ugyanis, hogy a II. jelű feladatok többet kívánnak. Viszont az I. feladatokat n-szer egymás után megoldva, mindig feleakkora intervallumba szoríthatjuk a keresett minimum értékét. Így a II. feladatok követelte lépésszám csak n-szerese az I. feladatokhoz szükségesnek, ami jelenlegi álláspontunkon elhanyagolható különbség.

A feladatok I. formája alkalmasabb az általánosításra, ezentúl csak erről lesz szó.

Megjegyzés: Eddig arra építettünk, hogy az olvasónak van intuitív elképzelése arról, mi egy számítási eljárás, és egy-egy konkrét eljárásnál mi lehet az adatok terjedelme, a lépésszám és a memóriaigény. A következőkben még pontosítani fogjuk ugyan ezeket a fogalmakat, de formális definíciók csak a Függelékben találhatók.

Most arra hívjuk fel a figyelmet, hogy ha korábban nem is lett volna jogos egy eljárás megvalósításához szükséges $f(n)$ „lépésszámról" beszélni (annyira függ ez attól, milyen gép áll rendelkezésünkre, és mi számít egy lépésnek), a 3. Definícióban bevezetett ekvivalencia erejéig már minden józanul definiált lépésszám megegyezik.

Lépésszámról mindig csak egy *eljárás* (algoritmus) kapcsán lehet szó. Ha néha egy feladat megoldásához szükséges lépésszámról beszélünk, ez sohase érthető szó szerint; az összefüggésből kell kiderülnie, mit mondunk a lehetséges megoldó eljárások lépésszámáról.

2. Kereső feladatok

Az Ia—b) feladatok lényegét így foglalhatjuk össze:

(i) Minden adott n-re egy n^3 terjedelmű x adatösszesség adja a feladat feltételeit. (x egy bináris sorozat, mely G-t $t(a)$-t és c-t kódolja.)

(ii) Minden x feltétel meghatároz egy U véges halmazt (a $2n$ hosszú útvonalak halmazát), melynek minden elemét egy n^2 hosszúságú y bináris sorozattal kódolhatjuk. U neve: *a szóba jövő megoldások halmaza.*

(iii) Az U halmaznak adva van egy A_x részhalmaza, azon útvonalak, melyek minden várost érintenek, és összköltségük legfeljebb c.

Számunkra az A_x halmaz leglényegesebb tulajdonsága a következő: minden $u \in U$-ra *könnyen eldönthető* (egy n^2 lépésben dolgozó eljárással), benne van-e A_x-ben vagy sem? Ha kezünkben van egy útvonal, akkor könnyű eldönteni, megfelel-e vagy nem? A probléma csak az, hogyan lehetne *találni* egy megfelelő útvonalat, viszonylag kevés számolás árán? A_x neve: *a megoldások halmaza.*

Az U halmazra formálisan nincs szükség, (ii)′ és (iii)′ helyett egyszerűbben így fogalmazhatunk.

(ii) Minden x feltételhez tartozik az n^2 hosszúságú bináris sorozatoknak egy A_x részhalmaza:

$A_x = \{y \mid y$ egy u utat kódol, mely minden V-beli ponton keresztülmegy és összköltsége legfeljebb $c\}$,

és van egy *eljárás*, mely minden x-re és y-ra *könnyen eldönti* (n^2 lépés alatt), igaz-e $y \in A_x$?

Az I.a) feladat mármost annak *eldöntését* követeli, hogy az A_x halmaz *üres-e?* Az I.b) feladatot akkor oldjuk meg, ha találunk egy $y \in A_x$-et.

Az utazó ügynök problémájának számunkra fontos vonása tehát a következő.

Egy véges halmazban bizonyos tulajdonsággal rendelkező elemet keresünk. A halmaz minden eleméről könnyű eldönteni, rendelkezik-e az adott tulajdonsággal, de a halmaz elemszáma túl nagy ahhoz, hogy a végigpróbálgatás eljárásával megelégedhetnénk.

Ennyi előkészítés után áttérhetünk a formális definíciókra.

4. *Definíció*: Legyen $\mathfrak{D} = \{0, 1\}$, $\mathfrak{D}^n$ az n hosszúságú bináris sorozatok halmaza, $\mathfrak{D}^* = \bigcup_{n=1}^{\infty} \mathfrak{D}^n$ az összes véges bináris sorozatok halmaza.

Egy $\mathcal{A}$ *kereső feladat* a következőkkel van adva.

(i) Egy k szám.

(ii) Minden n-re, $x \in \mathfrak{D}^n$-re egy $A_x \subset \mathfrak{D}^{n^k}$ halmaz, az x feltétel melletti megoldások halmaza.

(iii) Egy *eljárás*, mely minden n-re, $x \in \mathfrak{D}^n$-re és $y \in \mathfrak{D}^{n^k}$-ra legfeljebb n^k lépésben eldönti az $y \in A_x$? kérdést.

Egy eljárás az $\mathcal{A}$ feladat *kvázimegoldását* adja, ha minden x-re eldönti, A_x üres vagy nem.

Egy eljárás az $\mathcal{A}$ feladat *megoldását* adja, ha kiszámol egy $f(x)$ függvényt, melyre

$$f(x) = \begin{cases} 0 \text{ ha } A_x = \emptyset \\ A_x \text{ egy eleme, ha } A_x \neq \emptyset \end{cases}$$

(Azaz minden x-hez *talál* egy x feltétel melletti megoldást, illetve közli, ha ilyen nincs.)

Rengeteg példa győz meg minket arról, milyen széles feladatosztályt definiáltunk.

Példák:

2.a) A lineáris programozás feladata lényegében a valós lineáris egyenlőtlenségrendszerek megoldásának feladatát jelenti.

Adott egy $n \times n$-es A mátrix és egy n hosszú b vektor.

Keresendő olyan x n hosszú vektor, melyre

$$Ax \leq b.$$

Ha A, b n hosszú valós számokból áll, akkor, ha van megoldás, van n^2 hosszú megoldás is. Úgy pontosíthatjuk tehát a feladatot, hogy keresendő egy x, n^2 hosszú valós számokból álló vektor, mely az egyenlőtlenséget kielégíti.

A lineáris programozás feladatára sok olyan algoritmus van, ami a gyakorlatban jól működik (pl. a szimplex módszer), de egyes patologikus ellenpéldákon exponenciális lépésszámot követel (Klee mutatott ilyet). Mégis, az az általános vélemény, hogy előbb-utóbb lesz minden esetre kiterjedő „jó" (a 3. Definíció értelmében) algoritmus.

b) Az *integer programozás* feladata csak annyiban tér el az előbbitől, hogy A, b, x-től megköveteljük: egész számok legyenek. Erre már „gyakorlatilag jó" algoritmust sem ismer az irodalom.

Az integer programozás feladatának vannak nagyon speciális esetei is, amelyek nehezeknek bizonyultak, de olyanok is, amik fontosak, és jó algoritmus is van rájuk. Nehéz speciális eset a

3. *Hátizsák probléma*

Adott a_i $(i \leq n)$, és b n hosszú egész számok. Keresendő a_i-knek egy olyan $\{a_i \mid i \in I\}$ részhalmaza, amelyre

$$\sum_{i \in I} a_i = b$$

Könnyű, (de nem triviális) algoritmus ismeretes a következő, ugyancsak kereső típusú feladat megoldására.

4. *Hozzárendelési probléma* (A „szállítási probléma" speciális esete)

Adottak az A és B n elemű halmazok. A bizonyos pontjaiból élek vezetnek B bizonyos pontjaiba. Feleltessük meg A minden a pontjának B egy *vele összekötött*, $\varphi(a)$ pontját úgy, hogy ha $a_1 \neq a_2$, akkor $\varphi(a_1) \neq \varphi(a_2)$.

5. *Prímszámprobléma*

A számelméletben régóta nehéznek ismert feladat: eldönteni egy számról, prímszám-e? Itt a következő kereső feladatról van szó:

Adott egy n hosszú egész szám. Keressük ennek egy nemtriviális osztóját.

További példákat még a kifejtés során említünk, mások [1]-ben találhatók.

3. A visszavezethetőség fogalma

Gyakran úgy sikerül egy feladatra hatékony megoldási eljárást találni, hogy visszavezetjük egy másik, ismert feladatra, amelyre már van algoritmusunk. Az ilyen visszavezetések mindennaposak, hosszúságuk különösebb magyarázat nélkül nyilvánvaló.

De a visszavezetéseknek más funkciója is lehet. Ha egy gráfelméleti problémáról kiderül, hogy megoldása egyben a híres „négyszínsejtés" megoldását is jelentené, (tehát ha őrá sikerül a négyszínsejtést „visszavezetni"), akkor ez fontos információ a probléma nehézségéről. A kétféle visszavezetés között csak abban van a különbség, *mire használjuk őket?* Bemutatunk egy egyszerű példát a visszavezetések utóbbi típusára, mely az átlag alkalmazó matematikus számára szokatlanabb lehet.

6. *Példa*

A pontos fedés problémája és visszavezetése a hátizsák problémára.

Adott egy n elemű halmaz, H, és részhalmazainak egy $S_i (i \leq k)$ rendszere $(k \leq n)$.

Válasszunk ki ebből egy olyan $\{S_i \mid i \in I\}$ részrendszert, mely H-nak egy *partícióját* adja (azaz $\bigcup_{i \in I} S_i = H$, és $i, j \in I$, $i \neq j$ esetén $S_i \cap S_j = \emptyset$), illetve állapítsuk meg, ilyen rendszer kiválasztható-e?.

Ez egy elég általános, kereső típusú kombinatorikai feladat, melynek a „hozzárendelési probléma" pl. egyszerű speciális esete. Erős alapunk van feltételezni, hogy ezt a feladatot „nehéz" megoldani. (L. az 5. részt.) Ezért nem haszontalan a következő gondolatmenet, amellyel megmutatjuk, hogy a feladat visszavezethető a hátizsák problémára; hiszen így kiderül, hogy utóbbi is legalább annyira nehéz, mint a pontos fedés problémája.

Ha adott a H halmaz és az $S_i (i \leq k)$ halmazrendszer, akkor meg fogunk feleltetni ezeknek $a_i (i \leq k)$, b számokat úgy, hogy $\{S_i\}$ partíciót adó részrendszereinek $\{a_i\}$ olyan részrendszerei feleljenek meg, melyek elemeinek összege b. Ezt pl. a következőképpen tehetjük.

Legyen $d = k + 1$, és térjünk át a számok d alapú számrendszerbeli felírására. (Azaz $x = \sum_{j=1}^{n} x_j d^j$, ahol $x_j < d$.)

Legyen $H = \{h_1, \ldots, h_n\}$. Az S_i halmaznak feleltessük meg az

$$a_i = \sum_{j=1}^{n} \varepsilon_{ij} d^j \quad \text{számot, ahol} \quad \varepsilon_{ij} = \begin{cases} 1 \text{ ha } h_j \in S_i \\ 0 \text{ különben} \end{cases}$$

$\{S_i \mid i \in I\}$ nyilván épp akkor lesz pontosan fedő részrendszer, amikor

$$\sum_{i \in I} a_i = \sum_{j=1}^{n} d^j$$

Ezért $b = \sum_{j=1}^{n} d^j$ választással sikerül tetszőleges teljes-fedés feladatot alkalmas hátizsákfeladatra visszavezetni. Minden algoritmus, ami megoldja az általános hátizsákfeladatot, e fogás segítségével jelentéktelen többletmunkával már arra is használható lenne, hogy megoldja az általános teljes-fedés feladatot.

Figyeljük meg az ilyen típusú visszavezetés jellegzetességét: (látszólag) bonyolultabb feladatot üggyel-bajjal „belekódolunk" egy egyszerűbbe.

Ha $\mathcal{A}$ feladatot visszavezetjük a $\mathcal{B}$-re, akkor vagy azt akarjuk megmutatni, hogy $\mathcal{A}$-t könnyű, vagy azt, hogy $\mathcal{B}$-t nehéz megoldani.

5. *Definíció*: Legyen adott egy $\mathcal{A}$ és $\mathcal{B}$ kereső feladat, a 4. Definíció (ii) részében szereplő A_x és B_x halmazokkal.

Akkor mondjuk, hogy $\mathcal{A}$ *visszavezethető* $\mathcal{B}$-re (írjuk $\mathcal{A} \prec \mathcal{B}$), ha adva vannak $p(x)$, $q(x, y)$, $r(x, y)$ *könnyen kiszámítható* (a 3. Definíció értelmében) függvények, melyekkel

$$y \in A_x \Rightarrow q(x, y) \in B_{p(x)} \tag{1}$$

és

$$y \in B_{p(x)} \Rightarrow r\big(p(x), y\big) \in A_x \tag{2}$$

Mit fejez ki ez a definíció? Azt, hogy bármilyen x bemenő adatokkal kell megoldani az $\mathcal{A}$ feladatot, ezeket átalakíthatjuk a $\mathcal{B}$ feladat számára olyan $p(x)$ bemenő adatokká, hogy azokra a $\mathcal{B}$ feladatnak épp akkor van megoldása, amikor A-nak az x-re. Sőt, (1) azt fejezi ki, hogy $\mathcal{A}$-nak minden x feltétel melletti y megoldását átalakíthatjuk B-nek $p(x)$ feltétel melletti $q(x, y)$ megoldásává, míg (2) azt, hogy $\mathcal{B}$-nek minden $p(x)$ feltétel melletti y megoldását átalakíthatjuk $\mathcal{A}$-nak x feltétel melletti $r(p(x), y)$ megoldásává, és ezek az átalakítások kis lépésszámban valósíthatók meg (polinomiális idő alatt, a 3. Definíció értelmében). Ezért, ha $\mathcal{B}$ feladatra van jó eljárásunk, akkor a visszavezetés révén $\mathcal{A}$-t is jól meg tudjuk oldani.

Hogy az előbbi példában mi $p(x)$, $q(x, y)$ és $r(x, y)$, annak kitalálását az olvasóra bízzuk. Megjegyezzük, hogy ez esetben $q(x, y)$ és $r(p(x), y)$ inverzei egymásnak a következő értelemben:

$$q\big(x, r(p(x), y)\big) = y, \quad r\big(p(x), q(x, y)\big) = y \tag{3}$$

azaz az y-ok között olyan kölcsönösen egyértelmű megfeleltetést lehet létesíteni, hogy y akkor és csak akkor megoldása x feltétel mellett az $\mathcal{A}$ feladatnak, amikor a neki megfelelő $q(x, y)$ megoldása $p(x)$ feltétel mellett a $\mathcal{B}$ feladatnak. Az ismert visszavezetések legtöbbjénél (3) fennáll, nekünk azonban erre a tulajdonságra nem lesz szükségünk.

Konkrét visszavezetések végigvitelénél nem szoktuk használni a visszavezethetőség formális definícióját. Helyességük vagy helytelenségük formalizálás nélkül is általában olyan evidens, mint a matematikai bizonyításoké.

1. *Lemma.* $\mathcal{A} \prec \mathcal{B}$ és $\mathcal{B} \prec \mathcal{C}$-ből következik $\mathcal{A} \prec \mathcal{C}$.

Bizonyítás. Az olvasóra hagyjuk, egyszerűsége miatt.

Az 1. Lemmából következik, hogy $\prec$ egy parciális rendezést létesít a kereső feladatok között. Írjuk: $\mathcal{A} \sim \mathcal{B}$, ha $\mathcal{A} \prec \mathcal{B}$ és $\mathcal{B} \prec \mathcal{A}$. ($\mathcal{A}$ *ekvivalens* $\mathcal{B}$-vel.) A következő fejezetből kiderül, hogy e rendezésben van *maximális* elem, azaz olyan $\mathcal{A}_0$ feladat, melyre minden más feladat visszavezethető. Az ilyen feladatokat *univerzális kereső feladatnak* nevezzük. (Angolul „complete combinatorial problem", oroszul „unyiverszálnaja perebórnaja zadácsa" a terminus.) Nyilván bármely két univerzális feladat ekvivalens egymással, azaz, ha $\mathcal{A}_0$, $\mathcal{B}_0$ univerzálisak, akkor $\mathcal{A}_0 \sim \mathcal{B}_0$.

A 2. Lemmában azt az állítást fogalmazzuk meg pontosan, hogy az univerzális feladatokat a *lehető legnehezebb megoldani.*

6. *Definíció*: Tekintsünk egy $f(x)$ függvényt. A $t(x)$ függvényre azt mondjuk, hogy f *idő-bonyolultságának alsó becslése*, ha bármely olyan algoritmus, mely f-et kiszámolja, végtelen sok x szám esetén legalább $t(x)$ lépést végez.

2. *Lemma.* a.) Legyenek $\mathcal{A}$, $\mathcal{B}$ kereső feladatok; $s(x)$ egy függvény, $\mathcal{A} \prec \mathcal{B}$, és tegyük fel, hogy van egy eljárás, mely $\mathcal{B}$-nek minden x-re legfeljebb $s(x)$ lépésben (kvázi) megoldását adja.

Ekkor van egy $\tilde{s}(x) \approx s(x)$ függvény (ekvivalencia a 3. Definíció értelmében), és egy olyan eljárás, melyz az $\mathcal{A}$-nak legfeljebb V lépésben (kvázi) megoldását adja.

b.) Legyen $t(x)$ egy függvény, $\mathcal{A}_0$ egy univerzális kereső feladat. Tegyük fel, hogy *van egy olyan* $\mathcal{B}$ kereső feladat, melynél $t(x)$ a (kvázi) megoldás idő-bonyolultságának alsó becslése.

Akkor van egy olyan $\tilde{t}(x) \approx t(x)$ függvény, mely az $\mathcal{A}_0$ feladat (kvázi) megoldása idő-bonyolultságának alsó becslése.

Bizonyítás. A definícióból nyilvánvaló.

Ha egy univerzális feladatra van $s(x)$ lépésben dolgozó megoldási algoritmusunk, akkor ennek segítségével rögtön minden más kereső feladatot is meg tudunk közel $s(x)$ lépésben oldani. Nagyon meglepő volna épp ezért, ha az univerzális feladatokat az exponenciális lépésszámnál sokkal gyorsabban is meg lehetne oldani. *Ez mindenesetre nyitott kérdés.*

4. Univerzális feladatok

A Függelékben megmutatjuk, hogy az általános kereső feladat gondos megfogalmazása szinte automatikusan kezünkbe ad egy univerzális kereső feladatot. Most azonban egy, az előzőkhöz hasonlóan egyszerű, kombinatorikai jellegű univerzális kereső feladatot adunk meg. Ezzel a feladattal mindenki találkozott már, aki a matematikai logika elemeivel ismerkedett.

Legyenek $x_1, x_2, \ldots, y$ stb. *kijelentésváltozók*, melyek 0 (hamis) vagy 1 (igaz) értékeket vehetnek fel. Ezek között műveleteket (Boole-operációkat) definiálunk a következőképpen

$\neg x$ (tagadás, „nem x"):

$$\neg x = \begin{cases} 0 \text{ ha } x = 1 \\ 1 \text{ ha } x = 0 \end{cases}$$

$x \vee y$ (diszjunkció, „x vagy y")

$$x \vee y = \begin{cases} 0 \text{ ha } x \text{ és } y = 0 \\ 1 \text{ különben} \end{cases}$$

$x \,\&\, y$ (konjunkció, „x és y")

$$x \,\&\, y = \begin{cases} 1 \text{ ha } x \text{ és } y = 1 \\ 0 \text{ különben} \end{cases}$$

azaz

$$x \,\&\, y = \neg(\neg x \vee \neg y)$$

$x \to y$ (implikáció, „ha x, akkor y")

$$x \to y = \begin{cases} 0 \text{ ha } x = 1 \text{ és } y = 0 \\ 1 \text{ különben} \end{cases}$$

azaz

$$x \to y = \neg x \vee y$$

$x \leftrightarrow y$ (ekvivalencia, „x akkor és csak akkor, ha y")

$$x \leftrightarrow y = \begin{cases} 1 \text{ ha } x \text{ és } y \text{ ugyanazt az értéket veszik fel} \\ 0 \text{ különben} \end{cases}$$

azaz $$x \leftrightarrow y = (x \to y) \,\&\, (y \to x)$$

Ezek a műveletek tehát tulajdonképpen igazságfüggvények.

Egy $F(x_1, \ldots, x_n)$ kifejezést, melyet ezekből a függvényekből építettünk fel szuperpozícióval, [pl. $(x_1 \to x_2) \,\&\, (\neg x_1 \vee x_3)$] *logikai formulának* nevezzük. A formula meghatároz egy függvényt, mely 0-t vagy 1-et vesz fel aszerint, milyen igazságértékeket adtunk a változóknak.

[Pl. ha $F(x_1, x_2, x_3) = (x_1 \to x_2) \,\&\, (\neg x_1 \vee x_3)$, akkor $F(0, 1, 0) = (0 \to 1) \,\&\, \neg(0 \vee 0) = 1$.] Azt mondjuk, hogy az $\varepsilon_1, \ldots, \varepsilon_n$ ($\varepsilon_i = 0$ vagy 1) n-es *kielégíti* az $F(x_1, \ldots, x_n)$ formulát, ha $F(\varepsilon_1, \ldots, \varepsilon_n) = 1$. F *kielégíthető*, ha van olyan $\varepsilon_1, \ldots, \varepsilon_n$ sorozat, amit őt kielégíti. Egy F formula *hosszának* a benne szereplő logikai jelek számát nevezzük. A kereső feladat a következő.

7. *Példa*: *A kielégíthetőség problémája.*

Adott egy $F(x_1, \ldots, x_n)$ legfeljebb n hosszúságú logikai formula. Keresendő olyan $\varepsilon_1, \ldots, \varepsilon_n$ értékrendszer, mely F-et kielégíti.

Az előbbiekből nyilvánvaló, hogy adott $\varepsilon_1, \ldots, \varepsilon_n$-re *könnyű eldönteni*, kielégíti-e F-et? Tehát valóban kereső feladattal van dolgunk.

1. *Tétel.* A kielégíthetőség problémája univerzális feladat.

Bizonyítás. A tétel bizonyítása nem szükséges a továbbiak megértéséhez. A Függelék elolvasása után látható, hogyan pontosítható az alábbi vázlatos gondolatmenet.

A kereső feladat definíciójában az $A_x \subset \mathfrak{D}^{n^k}$ feltevést helyettesíthetjük $A_x \subset \mathfrak{D}^n$-nel is. Ugyanis semmi akadálya, hogy bemenő adatainkat formálisan n^k hosszúságúra hosszabbítsuk. (És n-nek nevezzük, ami eddig n^k volt.) Így a lépésszámra tett követelményben is írhatunk n-et n^k helyett. Mostantól feltesszük, hogy kereső feladataink ilyen formában vannak adva.

Megmutatjuk, hogy minden $\mathcal{A}$ kereső feladathoz van olyan l szám, valamint minden n-re egy $F_{\mathcal{A},n}(x, y, z)$ formula ($x, y \in \mathfrak{D}^n$, $z \in \mathfrak{D}^{n^l}$), melynek hosszúsága legfeljebb n^l, hogy minden x, y, z 0—1 sorozatra

(i) $F(x, y, z) = 1 \Rightarrow y \in A_x$

(ii) Van egy olyan $q(x, y)$ egyszerűen kiszámolható függvény, hogy

$$y \in A_x \Rightarrow F\big(x, y, q(x, y)\big) = 1$$

(iii) Maga az F formula is n-ből *könnyen* kiszámolható.

Ez nyilván azt jelenti, hogy $\mathrm{A} \prec \mathbb{S}$.

A formula megadásához alaposan szemügyre kell vennünk azt az eljárást, mely adott x-re y-ról n lépésben eldönti, megfelelő-e?

Egy n hosszúságú eljárásról a következőket tételezzük fel, arra támaszkodva, hogy minden eddig ismert, és racionálisan definiált lépésszámú eljárás e feltételeket kielégíti.

1. Az eljárás t-edik lépésében az összes adat egy n^r nagyságú (r konstans) „memóriában", tehát z_{it} ($t \leq n$, $i \leq n^r$, $z_{it} = 0$ vagy 1) változókkal írható le.

2. Az eljárás egy lépése egy *elemi* művelet végzését jelenti a $\{z_{it}\}$ adattömegen, így kapjuk a következő lépésben rendelkezésre álló $\{z_{i,t+1}\}$ adatokat.

3. Egy elemi műveletet *erősen korlátozott* mennyiségű változón lehet csak végrehajtani. Tegyük fel, hogy a felhasználható változók száma legfeljebb

olyan nagyságrendű, hogy egy tetszőleges z_{it} adat „helyére", i-re való hivatkozás beférjen, tehát $s \cdot \log_2 n$ (s konstans).

Elemi művelet tehát egy $\Phi : \mathfrak{D}^{s\log n} \to \mathfrak{D}$ függvény. Ehhez pedig található olyan G, $c \cdot n^s$ hosszú formula, hogy

$$G(z, z_1, \ldots, z_{s\log n}) = 1 \Leftrightarrow \Phi(z_1, \ldots, z_{s\log n}) = z$$

További részletezés nélkül, ennyiből is sejthető, hogy sok ilyen típusú formula összekapcsolásával kaphatunk egy olyan

$$F_{\mathfrak{A},n}(x, y, \{z_{it}\})$$

formulát, mely akkor és csak akkor „igaz", ha eljárásunk n lépésben az $y \in A_x$? kérdésre pozitív választ ad, és a részeredmények azaz a számolás „jegyzőkönyve" éppen $\{z_{it}\}$.

Ez a formula eleget tesz (i)-nek és (ii)-nek, és meghatározása is „könnyen" ment.

Ezzel vázlatos bizonyításunkat befejeztük.

A kielégíthetőség problémájának nem domborodik ki eléggé egyszerű kombinatorikai jellege. Megmutatjuk, hogy $\mathfrak{S}$ visszavezethető egy jól áttekinthető speciális esetére.

8. *Példa: Kielégíthetőség legfeljebb* 3 *tagú diszjunkciókkal.*

$\mathfrak{S}_3$ a következő kereső feladat. Legyen

$$F(x_1, \ldots, x_n) = D_1 \,\&\, \ldots \,\&\, D_k \quad (k \leq n)$$

ahol

$$D_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3} \quad (\tilde{x}_j = x_j \text{ vagy } \neg x_j).$$

Az ilyen F-et *speciális formulának* nevezzük.

A feladat: keresni olyan $\varepsilon_1, \ldots, \varepsilon_n$ értékrendszert, mely F-et kielégíti.

2. *Tétel.* S_3 univerzális kereső feladat.

Bizonyítás. Megmutatjuk, hogy tetszőleges $G(x_1, \ldots, x_n)$ legfeljebb n hosszú formulához található olyan $F(x_1, \ldots, x_n, y_0, y_1, \ldots, y_n)$ speciális formula, melynek tagszáma legfeljebb $4n$, hogy

(i) $F(x_1, \ldots, x_n, y_0, \ldots, y_n) = 1 \Rightarrow G(x_1, \ldots, x_n) = 1$

(ii) Van egy $q(G, x)$ könnyen kiszámolható függvény, hogy

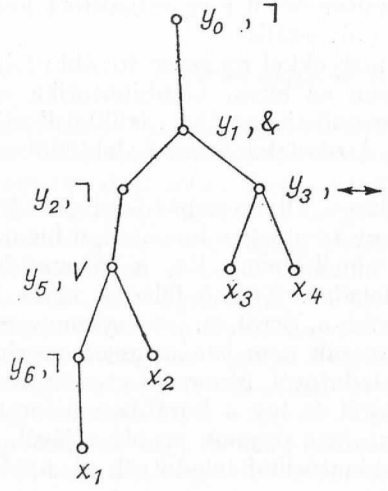
$$G(x_1, \ldots, x_n) = 1 \Rightarrow F\big(x_1, \ldots, x_n, q(G, x_1, \ldots, x_n)\big) = 1$$

Ebből állításunk nyilván következik.

G-ben a logikai jelek száma $k \leq n$. G-hez felrajzolhatunk egy *fát* a következőképpen (l. 1. ábra arra az esetre, ha $G = \neg(\neg(\neg x_1 \vee x_2) \,\&\, (x_3 \leftrightarrow x_4))$.

A fa minden pontjában a legalsókat kivéve, egy logikai jel van. A legalsó pontok $x_1, \ldots, x_n$-nel, a többiek különböző $y_0, \ldots, y_k$ szimbólumokkal vannak megjelölve. Az ábráról látható, hogyan felel meg a fa a formula konstrukciójának. Ha a fa y_i pontja mellett $\vee$ jel van, alatta y_j, y_l, akkor legyen

$$C_i = y_i \leftrightarrow (y_j \vee y_l) = (\neg y_i \vee y_j \vee y_l) \,\&\, (\neg y_j \vee y_i) \,\&\, (\neg y_l \vee y_i)$$



1. Ábra

Hasonlóan definiálható C_i más logikai jelekhez is. Könnyű belátni, hogy ha $F = C_0 \,\&\, \ldots \,\&\, C_k$, akkor F a követelményeknek megfelel.

5. Egyszerű univerzális feladatok

S_3 a következőképpen fogalmazható át:
Adott $F = D_1 \,\&\, \ldots \,\&\, D_k$, ahol

$$D_i = \tilde{x}_{s_{i1}} \vee \tilde{x}_{s_{i2}} \vee \tilde{x}_{s_{i3}}.$$

A feladat: találni egy olyan

$$\tilde{x}_{r_1}, \ldots, \tilde{x}_{r_k}$$

sorozatot, hogy $\tilde{x}_{r_i}$ benne van D_i-ben, és egyik $\tilde{x}_{r_i}$ se tagadása egy másiknak. (Ha ilyen $\tilde{x}_{r_1}, \ldots, \tilde{x}_{r_k}$ van, akkor ezeket 1-nek választva elégíthetjük ki F-et.)
Ilyen $\tilde{x}_{r_1}, \ldots, \tilde{x}_{r_k}$ sorozatra is azt mondjuk, hogy *kielégíti* F-et.
Következő példánk egy igazán jól ismert kombinatorikai probléma, melyről belátjuk, hogy univerzális.

9. *Példa*: *Maximális független pontrendszer.*

Adott egy n pontú G gráf és egy m szám. Az M feladat: keresendő G-ben m össze nem kötött pontból álló rendszer.

3. *Tétel.* $S_3 \prec M$.

Bizonyítás. Adott $F = D_1 \,\&\, \ldots \,\&\, D_k$-hez $(k \leq n)$ megkonstruálunk egy G gráfot. G pontjai $\{a_i^1, a_i^2, a_i^3 \mid i = 1, \ldots, k\}$ a_i^j össze van kötve a_p^q-val, ha $\tilde{x}_{s_{ij}}$ és $\tilde{x}_{s_{pq}}$ egymás tagadásai vagy $i = p$.
Világos, hogy G-ben m elemű független pontrendszer épp akkor lesz, amikor F-et kielégítő $\tilde{x}_{r_1}, \ldots, \tilde{x}_{r_k}$ sorozat.

Ez az első, már mindenféle logikai szólásmódtól megszabadított kombinatorikai feladat, melyről beláttuk, hogy univerzális.

Ahogy sejthető is, azok az eljárások, melyekkel rengeteg további feladatról megmutatható, hogy univerzális, teljesen az elemi kombinatorika szintjén maradnak. Ezek az elvileg egyszerű, technikailag néha „trükkös" eljárások nem tartoznak elvi mondanivalónkhoz. A részletek iránt érdeklődőket [1]-re és [4]-re utaljuk.

A kereső feladat fogalma nagyon általános. Olyan nehéz feladatok is kereső típusúak, mint annak eldöntése, van-e egy tételnek n hosszúságú bizonyítása. (Valamilyen axiomatikus, formalizált elméletben.) Ez, a *bizonyításkeresés problémája*, természetesen univerzális feladat. Kereső feladat az is, ha két adott gráfról el akarjuk dönteni, izomorfak-e. Erről, az *izomorfizmusproblémáról*, nem ismeretes, univerzális-e? Ugyancsak nem ismeretes ez a prímszámproblémáról és a lineáris programozás feladatáról, ismeretes viszont az integer programozásról, a pontos fedés feladatáról és így a korábban bizonyítottak értelmében a hátizsákproblémáról is, az utazó ügynök problémájáról, és még sok igen egyszerűen fogalmazható kombinatorikai feladatról, pl. arról, hogy adott G gráf kiszínezhető-e 3 színnel?

Ha egy kereső feladatról kiderül, hogy univerzális, akkor nyugodtan levonhatjuk azt a következtetést, hogy a feladat általános, jól használható megoldó algoritmusát nem érdemes tovább keresni. Nem állítható viszont az, hogy egy-egy feladat speciális természetét kihasználva, ne találhatnánk olyan eljárásokat, melyek a feladatot a gyakorlat számára kielégítő pontossággal megoldják.

Vannak olyan fontos feladatok az operációkutatásban, melyekről kiderült, hogy univerzálisak, ugyanakkor azt is megmutatták, hogy a legegyszerűbb stratégiák is elvisznek az optimum konstansszorosáig. Optimista befejezésül ilyen szituációt mutatunk be egy triviális példán.

A „hátizsákprobléma" következő módon is fogalmazható.

Adott egy b alapterületű raktárunk és $a_1, \ldots, a_n$ alapterületet elfoglaló n darab áru. Az a feladatunk, hogy olyan árukat helyezzünk el a raktárban, melyek az alapterületet a lehető legjobban kitöltik.

Ennek a feladatnak pontos megoldása — az előbbiek értelmében — univerzális kereső feladat megoldását jelentené. Miyen stratégiát alkalmaznánk, ha nem ér meg sok fejtörést a dolog? Talán a következőt.

Előbb elhelyezzük a lehető legnagyobb árut, ami elfér, aztán a maradékból megint a legnagyobbat, ami elfér stb.

Könnyű belátni, hogy így is legalább feleannyi területet sikerül kitölteni, mint az optimális elhelyezésnél.

Egyik mentési lehetőség tehát ez: elégedjünk meg nem pontos optimummal. Másik, ami a lineáris programozás jelenlegi helyzetére jellemző: elégedjünk meg olyan eljárásokkal (illetve olyanokat keressünk), melyek a feladatot az esetek *túlnyomó többségében* (ez a szimplexmódszer esetén még csak empirikusan, nem matematikailag bizonyított) rövid idő alatt oldják meg.

FÜGGELÉK

A) Algoritmus és annak lépésszáma

Algoritmusok, azaz effektív, mechanikus (számolási) eljárások mindig is fontosak voltak a matematika és alkalmazásai számára. De e század 20-as, 30-as éveig nem vetődött fel az igény az algoritmus fogalmának általános definíciója iránt. Ha valaki egy probléma megoldására algoritmust mutatott, akkor soha sem volt vitás, vajon ez tényleg algoritmus-e? Megváltozott a helyzet, amikor egyes feladatoknál az volt valószínű, hogy *nem létezik* megoldó algoritmus. (Történelmileg első ilyen feladat a bizonyításkeresés feladata volt. Egy formalizált, axiomatikus elmélettel kapcsolatban felmerülhet a kérdés, hogy tetszőleges állításról hogyan dönthető el, van-e formális bizonyítása?)

Ilyen kijelentések nyilván nem értelmesek, amíg nem mondjuk meg pontosan, mit nevezünk algoritmusnak. Gödel, Church, Turing, Post, Kleene munkája nyomán az algoritmus fogalmának sok alternatív definíciója született melyekről kiderült, hogy ekvivalensek (minden alkalommal ugyanazok a függvények bizonyultak algoritmikusan kiszámolhatónak). Minden azóta született eredmény is azt támasztja alá, hogy kezünkben tartjuk az algoritmus általános fogalmának egyetlen lehetséges definícióját.

Amióta az elektronikus számítógépek elterjedtek, a mechanikus eljárások iránti érdeklődés megnőtt. Ugyanakkor, az algoritmus általános fogalmának megértése sokkal egyszerűbbé vált, ugyanis az algoritmus nem más, mint számítógépen megvalósítható eljárás.

Azt kell tehát csak matematikailag definiálni, *mi egy számítógép?* Szerencsére ehhez nem kell valamilyen nagy számítógépet minden részletében leírnunk, és attól se kell tartani, hogy ha jobb, korszerűbb számítógépek épülnek akkor tágulni fog az algoritmus fogalma. Utóbbit teljesen meghatározzák a számítógépek működésének legáltalánosabb elvei. Az egyes számítógépek nem abban különböznek egymástól, *mit* lehet rajtuk kiszámítani, csak abban, *milyen gyorsan*: milyen utasítások azok, amiket „megértenek", milyen műveleteket végeznek el tanítás nélkül, stb. Ezért megelégedhetnénk egy nagyon primitív felépítésű számítógéppel is az algoritmus általános definíciójához, mi azonban inkább olyant ismertetünk, melynek lehetőségei rendkívül gazdagok. Ezzel megtakarítjuk az érvelést definíciónk általánossága mellett.

Aki az alább leírt számítógépet irreálisan hatékonynak, korlátozás nélkülinek tartja, nem értette meg annak rendeltetését. Olyan gépet akarunk definiálni, ami mindent tud, ami valaha is számolásnak lesz tekinthető. Ebbe a gépbe természetesen minden korlátozás beépíthető (pl. a programba).

Az ismertetett gépmodell a Turing-féle gép ([5]) egy Kolmogorov—Uszpenszkij-féle változata ([6]), Ja. M. Barzgyiny által általánosítva.

Számítógépünk *diszkrét időben* dolgozik, tehát a számítási eljárás elkülöníthető lépésekből áll. Az eljárás teljesen *determinisztikus*. Minden időpontban a számítógép akkori állapota teljesen meghatározza a következő időpontbeli állapotot.

A gép sok egységből áll, ezek két különböző típusba tartoznak: aktívak és passzívak. Az aktív egységek szerepe a vezérlés, lebonyolítás, utasításvégrehajtás. A passzív egységek képezik a memóriát.

Minden egyes egység csak *korlátos* mennyiségű információt képes hordozni. Ezt matematikailag úgy fejezzük ki, hogy minden egységnek csak korlátos

sok *állapota* van. Legyen a t-edik időpontban a passzív egységek (véges) halmaza M_t, az aktívaké V_t.

Minden aktív egység minden időpillanatban „rajta áll" egy passzív egységen, (megfigyeli azt), azon t-ről $t+1$-re áttérve egy műveletet hajt végre, majd átlép egy másik passzív egységre.

Az M_t és V_t halmazok nem állandóak, *növekedhetnek*. Az aktív egységek elemi műveletei között szerepelni fog ugyanis újabb aktív vagy passzív egység létrejozása.

Amint mondottuk, minden egység véges sok állapotban lehet. Legyen a lehetséges állapotok halmaza Q. Q végig ugyanaz, tulajdonképpen még azt is feltehetnénk, hogy csak 5—10 elemből áll. Ha $m \in M_t$ ill. $v \in V_t$ egy egység, akkor jelölje $s_t(m) \in Q$ ill. $s_t(v) \in Q$ az állapotukat a t időpontban.

A gép szervezésének legérdekesebb kérdése az, hogy ha egy v aktív egység a t időpontban egy m passzív egységen áll — jelöljük ezt a szituációt így:

$$\varphi_t(v) = m$$

— akkor hogyan választja ki azt a passzív egységet, melyre áttér a következő lépésben?

A probléma a következő. Az egységek halmaza egyre növekszik. Ha v az összes passzív egység közül választhatná ki, melyikre lép át, ellentétbe kerülnénk azzal a követelménnyel, hogy a v által végrehajtott művelet csak *egészen elemi* lehet. v minden időpontban csak korlátos mennyiségű egységet tekinthet át.

A következő megoldást választjuk. Rögzítünk egy k pozitív egész számot. Minden passzív m egységhez tartozik, minden t időpontban a passzív egységeknek egy

$$\mu_t^1(m), \ldots, \mu_t^k(m)$$

sorozata, azok, amelyek a t időpontban m-ből *elérhetőek*. Minden v aktív egységhez is tartozik egy

$$\{\mu_t^i(v)\}_{i \leq k}$$

sorozat, ugyancsak passzív egységekből. Ezek a v-ből elérhető egységek.

Ezek után megadhatjuk, mit nevezünk egy elemi műveletnek, (melyet v hajt végre t-ről $t+1$-re áttérbe azon az m-en, amit megfigyel a t időpontban).

Az elemi művelet csak attól függ, mi m és v *állapota* a t időpontban: azaz $s_t(m)$-től és $s_t(v)$-től.

Az elemi művelet a következő négyféle operáció *együttes* végrehajtását jelenti.

1. m és v új állapotának (azaz $s_{t+1}(m)$-nek és $s_{t+1}(v)$-nek) meghatározása.
2. *v új helyének* (azaz v által a $t+1$-edik időpontban megfigyelt passzív egységnek, $\varphi_{t+1}(v)$-nek) meghatározása.

Kikötjük, hogy *v új helye csak olyan passzív egység lehet, mely m-ből elérhető.* Tehát ez a művelet a $\mu_t^i(m)$ egységek közül választja ki v új helyét.

3. Azon egységek meghatározása, melyek m-ből és v-ből a $t+1$-edik időpontban elérhetők lesznek (tehát $\mu_{t+1}^i(m)$ és $\mu_{t+1}^i(v)$ új értékei).

Itt kikötjük, hogy *mindkét sorozat csak olyan egységeket tartalmazhat, melyek vagy m-ből, vagy v-ből elérhetők voltak.* Azaz

$$\mu_{t+1}^i(m),\ \mu_{t+1}^i(v) \in \{\mu_t^j(m) \mid j \leq k\} \cup \{\mu_t^j(v) \mid j \leq k\}$$

Tehát egy elemi művelet hatására az m-ből (ill. v-ből) elérhető egységek összessége csak úgy változhat, hogy a v-ből (ill. m-ből) elérhető egységek közül újak kerülnek be, és régiek kimaradnak.

4. Egy új aktív vagy passzív w egység esetleges létrehozása, állapotának meghatározása. Az új egységből elérhető passzív egységek is csak az m-ből és v-ből elérhetők közül kerülhetnek ki.

További megállapodások a gép működésével kapcsolatban

Ha egy passzív egységen éppen nem áll aktív, akkor állapota is, a belőle elérhető egységek sorozata is t-ről $t+1$-re áttérve változatlan marad.

Ha egy passzív egységre 1-nél több aktív egység „érkezik", (azaz a t-edik időpontban még nem volt ilyen kollízió, de a $t+1$-edik időpontra lenne), akkor a gép leáll.

Az egyik Q-beli állapot, q_f a végállapot. Ha valamelyik egység ilyen állapotba kerül, a gép leáll.

Ezekután a gép működési módját teljesen meghatározza, ha megadjuk az 1—4-ben értelmezett elemi műveleteket.

Az 1. művelet egy

$$\sigma_M : Q \times Q \to Q \quad \text{és} \quad \sigma_V : Q \times Q \to Q$$

véges függvénnyel írható le, a következő módon:

$$s_{t+1}(m) = \sigma_M\big(s_t(m), s_t(v)\big), \quad s_{t+1}(v) = \sigma_V\big(s_t(m), s_t(v)\big).$$

A 2. müvelet egy

$$\gamma = Q \times Q \to \{1, \ldots, k\}$$

véges függvénnyel adható meg, a következő képlet szerint:

$$\varphi_{t+1}(v) = \mu_t^{\gamma(s_t(m), s_t(v))}(m)$$

azaz γ megadja, v az m-ből elérhető egységek közül sorrendben *hányadikra* lép át?

A 3. művelet két darab,

$$\alpha_M, \alpha_V : Q \times Q \times \{1, \ldots, k\} \to \{1, \ldots, 2k\}$$

függvénnyel adható meg a következő képletek szerint.

$$\mu_{t+1}^i(m) = \begin{cases} \mu_t^j(m) & \text{ha} \quad \alpha_M^i\big(s_t(m), s_t(v)\big) = 2j-1 \\ \mu_t^j(v) & \text{ha} \quad \alpha_M^i\big(s_t(m), s_t(v)\big) = 2j \end{cases}$$

$$\mu_{t+1}^i(v) = \begin{cases} \mu_t^j(m) & \text{ha} \quad \alpha_V^i\big(s_t(m), s_t(v)\big) = 2j-1 \\ \mu_t^j(v) & \text{ha} \quad \alpha_V^i\big(s_t(m), s_t(v)\big) = 2j. \end{cases}$$

A 4. művelet egy

$$\delta : Q \times Q \to \{0, 1, 2\}, \quad \varrho : Q \times Q \to Q$$

és

$$\beta : Q \times Q \times \{1, \ldots, k\} \to \{1, \ldots, 2k\}$$

véges függvénnyel adható meg így:

Nincs új egység, ha $\delta\big(s_t(m), s_t(v)\big) = 0$.

Az új egység w, passzív, ha $\delta(s_t(m), s_t(v)) = 1$, aktív, ha 2. Ez esetben

$$s_{t+1}(w) = \varrho(s_t(m), s_t(v))$$

és

$$\mu^i_{t+1}(w) = \begin{cases} \mu^j(m) & \text{ha } \beta^i(s_t(m), s_t(v)) = 2j-1 \\ \mu^j(v) & \text{ha } \beta^i(s_t(m), s_t(v)) = 2j. \end{cases}$$

A

$$\sigma_M, \sigma_V, \gamma, \alpha_M, \alpha_V, \delta, \varrho, \beta$$

véges függvények gépünk működését teljesen meghatározzák. Ezek képezik, ha úgy tetszik, a gép „hardware"-jét. Aszerint kapunk más és más gépet, ahogy ezeket másképp választjuk. Ezért azt is mondhatjuk, hogy egy gép *nem más*, mint egy $T = (Q, k, \sigma_M, \sigma_V, \gamma, \alpha_M, \alpha_V, \delta, \varrho, \beta)$ összesség.

A számolás t-edik időpontjában a *pillanatnyi szituációt* teljesen meghatározzák

$$M_t, V_t, \varphi_t : V_t \to M_t, s_t : M_t \cup V_t \to Q \quad \text{és}$$

$$\mu_t : (M_t \cup V_t) \times \{1, \ldots, k\} \to (M_t)^k.$$

Ezért a t időpontbeli pillanatnyi szituációt azonosíthatjuk a

$$D_t = (M_t, V_t, \varphi_t, s_t, \mu_t)$$

összességgel.

Az, hogy a gép működése teljesen determinisztikus, azt jelenti, hogy D_{t+1} a D_t-ből egyértelműen kiszámolható.

Hogyan használhatjuk a gépet számolásra? Mindenekelőtt megadjuk D_0-at, az induló szituációt. Választunk egy alkalmas nagyságú M_0 induló memóriát, azaz megfelelő számú passzív egységet. A memóriában elhelyezzük kiinduló adatainkat, és a szükséges programokat. Ez a passzív egységek állapotainak meghatározását jelenti. (pl. ha kettes számrendszerben kell dolgoznunk, akkor $s_0(m) = q_0$ vagy q_1, Q-beli elemek. Q-t általában valamilyen ábécének képzelhetjük el).

Ezután „megszervezzük" a memóriát a kívánt módon, azaz megadjuk, hogy a számolás kezdetén melyik passzív egységből melyik legyen elérhető. Ez $\mu^i_0(m)$ megadását jelenti.

Végül meghatározzuk az induláskor szükséges végrehajtó egységeket, azok induló helyét, állapotát és az induláskor belőlük elérhető passzív egységeket. Ez V_0, $s_0(v)$ és $\mu^0_i(v)$ megadását jelenti.

Beindítjuk a gépet. Az számolni kezd, sorban létrehozza a $D_1, D_2, \ldots$ szituációkat a fent megadott szabályoknak megfelelően. Ha megáll, akkor a memóriából, azaz a passzív egységek állapotaiból előre megállapított módon kiolvashatjuk a számolás végeredményét.

A számolás *lépésszáma* a megállásig eltelt időegységek száma. Előfordulhat, hogy a gép nem áll meg, a gép által megvalósított eljárás nem ér véget véges sok lépésben, eljárásunk nem alkalmazható a betáplált kiinduló adatokra, tehát *nem általánosan alkalmazható*. Mi csak általánosan alkalmazható eljárásokkal fogunk foglalkozni.

Miután meghatároztuk, mit értünk számítógép alatt, rátérhetünk az algoritmus (eljárás) fogalmának pontosítására.

Mindenekelőtt meg kell állapítani, hogy *eljárásról* sohasem egyetlen feladatnak csak egy *feladatcsaládnak* a megoldásánál értelmes beszélni. (Amit a cikk főrészében „kereső feladatnak" neveztünk, az se egyetlen feladat, hanem

minden n-re és $x \in \mathfrak{D}^n$-re egy külön feladatból álló feladatcsalád). Vagy azt is mondhatjuk, hogy *egyetlen feladatnak változó bemenő adatokkal* való megoldásánál.

Ismét megállapodhatunk, hogy a bemenő és kimenő adatokat bináris sorozatokkal kódoljuk. Korábban az összes bináris sorozatok halmazát $\mathfrak{D}^*$-al jelöltük. Olyan feladatokról lesz szó, ahol *ki kell számolni* valamit.

Egy feladat tehát, ahogy mi értjük, mindig egy

$$f(x) : \mathfrak{D}^* \to \mathfrak{D}^*$$

függvénnyel adható meg. Egy eljárás *megadja* az f függvénnyel adott feladatot, ha minden lehetséges x bemenő adathoz kiszámolja $f(x)$-et, ezaz a megoldást.

Egy eljárást akkor nevezünk mechanikus eljárásnak, *algoritmusnak*, ha egy az előbbiekben ismertetett T gépen elvégezhető. Mit jelent ez? Ami most következik, az egy eléggé esetleges szabványosítási megállapodás arról, mit tekintsünk a gép kezdő- és végszituációjának.

Az egységek állapotainak Q halmazából a q_f végállapoton kívül tüntessünk ki még egy q_b állapotot, amit kezdő állapotnak nevezünk, valamint két további, q_0 és q_1 állapotot.

A gép *kezdő szituációban* van, ha

(i) egyetlen m_0 passzív egység van a q_b állapotban.

(ii) m_0-ból egyetlen m_1 egység érhető el, m_1-ből egyetlen m_2, m_2-ből egyetlen m_3, stb. valamilyen m_n-ig, $m_1, \ldots, m_n$ mind különbözők.

(iii) $m_1, \ldots, m_n$ mind q_0 vagy q_1 állapotban vannak.

(iv) m_{i+1} nem érhető el más egységből, mint m_i-ből.

A gép a *végszituációban* akkor van, ha ugyancsak a fenti feltételek teljesülnek q_b helyett q_f állapotot írva.

Egy gép kezdő szituációját tehát két egymástól független adatösszességgel írhatjuk le. Egyik, hogy milyen állapotban vannak az $m_1, \ldots, m_n$ egységek, (és hány van belőlük), azaz mi van az adatszalagon? Ezt az információt egy $x = x_1, \ldots, x_n$ sorozattal adjuk meg, ahol $x_i = 0$ vagy 1 ha $s_0(m_i) = q_0$ vagy q_1.

A gép összes többi aktív és passzív egységei, ezek egymáshoz és m_0-hoz való viszonya (m_{i+1} nem érhető el belőlük a 0. időpillanatban) lesz a másik adatösszesség. Ez képezi a *számolás programját*, P-vel jelöljük. Az adatszalag kezdő szituációbeli állapota adja a számolás *bemenő adatait*, a végszituációban ugyanez a *megoldás*.

Egy kezdő szituáció tehát a

$$D_0 = (P, x)$$

összességgel jellemezhető. A gép akkor valósít meg valamilyen *számolási eljárást*, ha míg x végigfut az összes lehetséges bemenő adatokon, *minden számoláshoz ugyanazt a P-t használjuk.*

7. *Definíció*: Azt mondjuk, hogy a T gépen az $f(x)$ függvény kiszámolható, ha van olyan P program, mellyel a gép minden (P, x) kezdő szituációból olyan végszituációhoz jut el, ahol az adatszalagon $f(x)$ van. A T gép és a P program együtt megad egy *algoritmust* f kiszámolására. *f algoritmikusan kiszámolható*, ha *van olyan* T gép, melyen kiszámolható. A kiszámolható függvényeket *rekurzív* függvényeknek is szokták nevezni.

Rögtön megjegyezzük: vannak olyan gépek melyeken minden függvény kiszámolható, ami egyáltalán valahol kiszámolható. Az ilyen gépek nem is feltétlenül bonyolult felépítésűek. Univerzálisnak nevezzük őket. A gyakorlatban használt nagyméretű számítógépek mind univerzálisak. Nem elvi nehézségek, csak a bizonyítás körülményessége miatt tekintünk el a következő tétel bizonyításától.

4. *Tétel.* Vannak univerzális gépek.

A bizonyítás azon alapszik, hogy egy egészen egyszerű géppel is képesek lehetünk minden más gép működését utánozni.

A leírt géptípus sok olyan lehetőséggel rendelkezik, amivel a mai számítógépek nem. Legfontosabb ezek közül az, hogy az aktív egységek száma korlátlanul növekedhet. Nem nehéz belátni, hogy ilyen géppel az utazó ügynök problémája is nagyon gyorsan megoldható volna. Ugyanis a gép aktív részei n lépés alatt 2^n-felé osztódhatnak, és minden lehetőséget egyszerre kipróbálhatnak. Ezért aki úgy véli, hogy a próbálgató megoldás exponenciális mennyiségű időt vesz igénybe, nyilván olyan gépre gondol, melynek korlátos számú aktív egysége van. A cikk főrészében mi is mindig ilyen gépet tartottunk szem előtt.

Egy $f(x)$ függvényről tehát akkor mondjuk, hogy $t(x)$ *lépésben kiszámolható*, ha van olyan T gép, és P program hogy ezek $t(x)$ lépésben kiszámolják $f(x)$-et, és *számolás közben új aktív egység nem keletkezik.*

Az olyan gépet, melyen számolás közben nem keletkezik aktív egység, *végesfejűnek* hívjuk. (Az aktív egységeket szokták „fej"-nek is nevezni).
Megjegyzés: Van végesfejű univerzális gép.

Kifogásolható az is, hogy a memória struktúrája időben változhat, tehát változhat egy-egy passzív egységből elérhető más passzív egységek halmaza.

Itt sokféle megkötést fogadhatunk el, de még a legszigorúbb értelmes megkötés se korlátozza a kiszámolható függvények körét. Sőt, a főrész 3. Definíciójában adott ekvivalencia (azaz hatványozás) erejéig még a számolás lépésszámát sem.

Egyfejű Turing-gép az olyan, melyben a passzív egységek egy egyenesen helyezkednek el egymás mellett. Egyetlen aktív egység van, ebből csak az az egység érhető el, melyet megfigyel. A passzív egységekből csak szomszédjaik érhetők el.

Az egyfejű Turing-gép már lényegesen kevesebbet „tud" mint a mai számítógépek. De van univerzális egyfejű Turing-gép!

Még mindig megmaradhatott az a kifogás, hogy a gép memóriája korlátlanul növelhető.

8. *Definíció*: A (P, x) indulószituációjú eljárás *memóriaigénye* nem más, mint a számolás befejezéséig keletkezett egységek száma (az induló egységekkel együtt).

Erre az ellenvetésre azt válaszoljuk: semmi akadálya minden alkalommal csak olyan eljárásokat venni figyelembe, melyeknek lépésszáma és memóriaigénye előre adott $t(x)$ és $l(x)$ függvényekkel van korlátozva. De ilyen korlátozást az algoritmus általános fogalmába beépíteni nem célszerű.

Az ebben a részben bevezetett fogalmak minden, cikkünk 1—3. pontjában szereplő definíciót pontos matematikai definícióvá változtattak.

B) Univerzális kereső feladatok

Az univerzális gépekről tulajdonképpen többet tudunk, mint hogy rajtuk minden kiszámolható függvény előállítható.

Egy T gép leírása nyilván kódolható egyetlen $\ulcorner T \urcorner$ bináris sorozatba. Ugyanez vonatkozik T-nek egy P programjára is, melynek kódját $\ulcorner P \urcorner$-vel jelöljük.

Ha $x = x_1, \ldots x_n$ egy 0—1 sorozat, akkor legyen $\bar{x} = x_1 x_1 x_2 x_2 \ldots x_n x_n 01$. Ezekután az x, y sorozatpárt, az x, y, z sorozathármast kódolhatjuk egyetlen $\bar{x}y$, $\bar{x}\bar{y}z$, stb. sorozattal, melyből az eredeti összetevők egyértelműen kiolvashatók.

Egy x bináris sorozat hosszát $|x|$-el jelöljük.

5. *Tétel.* Van egy olyan végesfejű T_0 gép, és ennek egy P_0 programja, mely bármely (T, P) gép-programpárra és x bináris sorozatra a következőket hajtja végre.

1. A

$$(P_0, \overline{\ulcorner T \urcorner}\, \overline{\ulcorner P \urcorner}\, x)$$

kezdő szituációból legfeljebb akkora lépésszám alatt, mint $(|\ulcorner T \urcorner| + |\ulcorner P \urcorner|)^2$, előállít egy

$$(\Pi_{T,P}, x)$$

kezdőszituációt.

2. A $(\Pi_{T,P}, x)$ kezdőszituációból kiszámolja ugyanazt, amit a T gép a (P, x) kezdőszituációból, és ha T végesfejű, akkor ehhez nem követel több lépést, mint a T gép lépésszámának $|\ulcorner T \urcorner|^2$-szerese.

Ezt a tételt nem bizonyítjuk, de aki jól megérti az állítást, az belátja, hogy a bizonyítás csak türelem kérdése.

A T_0 géppel kapcsolatban definiálunk egy $\mathfrak{U}$ kereső feladatot, melyről aztán bebizonyítjuk, hogy univerzális. A kereső feladat a következő.

Adott x n-hosszúságú 0—1 sorozat. Keresendő y legfeljebb n hosszúságú olyan 0—1 sorozat, hogy a T_0 gép a

$$(P_0, xy)$$

kezdőszituációból legfeljebb n lépés alatt 0-t számoljon ki.

Hogy ez kereső feladat, az a definícióból világos. (Adott x-re a megoldások U_x halmaza azon y-ok halmaza, melyekre T_0 legfeljebb n lépésben 0-t számol ki).

6. *Tétel.* Az $\mathfrak{U}$ feladat univerzális.

Bizonyítás. Tekintsünk egy tetszőleges $\mathfrak{R}$ kereső feladatot. Ehhez van egy eljárás, mely minden n hosszú x-re és y-ra legfeljebb n lépésben eldönteni az $y \in A_x$? kérdést. Azaz van egy T gép és P program, hogy T minden n hosszú x, y sorozatpárra a $(P, \bar{x}y)$ kezdőszituációból legfeljebb n lépés alatt 0-t számol ki, ha $y \in A_x$, 1-et, ha $y \notin A_x$.

Nézzük mit csinál ekkor T_0 a

$$(P_0, \overline{\ulcorner T \urcorner}\, \overline{\ulcorner P \urcorner}\, \bar{x}y)$$

kezdőszituációban? A tétel szerint legfeljebb

$$(|\ulcorner T \urcorner| + |\ulcorner P \urcorner|)^2 + |\ulcorner T \urcorner|^2 \cdot n$$

lépésben 0-t számol ki, ha $y \in A_x$, 1-et, ha $y \notin A_x$. Tehát az A feladatnak x bemenő adat melletti megoldását visszavezettük az U feladat

$$\ulcorner T \urcorner \ulcorner P \urcorner \bar{x}$$

bemenő adat melletti megoldására.

Ha az olvasó most mégegyszer megnézi a bizonyításvázlatot, amit a 4. pont 1. Tételére adtunk, láthatja, hogy könnyen szabatos bizonyítássá tehető, például úgy, hogy belátjuk:

$$\mathfrak{U} \prec \mathfrak{S}$$

Ehhez T_0 működésének „jegyzőkönyvét" kell olyan módon kódolni, hogy a kód egyértelműen adódjék, mint egy bizonyos logikai formulát kielégítő sorozat. Mi inkább közvetlenül bizonyítjuk $\mathfrak{S}$ univerzalitását.

Ha $\mathfrak{A}$ egy kereső feladat, akkor nyilván minden n-re lesz egy T_n gép, mely n lépés alatt eldönti az n hosszú x, y sorozatokra az $y \in A_x$? kérdést, és a következő formájú.

(i) T_n-nek $c_1 n$ passzív egysége van, melyek q_0-t vagy q_1-et vehetnek fel, és 1 aktív egysége, mely $c_2 n$ különböző állapotot vehet fel (c_1, c_2 konstansok).

(ii) Új egység a számolás során nem keletkezik.

(iii) Minden egységből minden másik egység elérhető.

A T_n gép nyilván elég általános ahhoz, hogy minden n lépéses eljárás ilyen gépen elvégezhető legyen.

Legyenek T_n passzív egységei $m_1, \ldots, m_{c_1 n}$, az aktív egység v, log mindig 2-es alapú logaritmust jelent.

A D_t t-időpontbeli szituációt nyilván egyértelműen meghatározzák a következő adatok.

$$z_{t_1}, \ldots, z_{t,c_1 n}$$

ahol

$$z_{ti} = \begin{cases} 0 & \text{ha} \quad s_t(m_i) = q_0 \\ 1 & \text{ha} \quad s_t(m_i) = q_1 \end{cases}$$

továbbá

$$\lambda_{1t}, \ldots, \lambda_{t \log c_2 n}$$

melyek együtt egy bináris számot adnak, ami v állapotát kódolja, és a

$$\chi_{1t}, \ldots, \chi_{t \log c_1 n}$$

bináris szám, ami v helyét kódolja. A 3. Tétel bizonyításában szereplő F formulát most már tényleg fel lehet írni a

$$z_{ti}, \lambda_{tj}, \chi_{tk}$$

változókra.

Tudjuk, hogy ha v a t-edik időpontban m_i-n van, akkor új helyét egy

$$\gamma : \{0, 1\} \times \{1, \ldots, c_2 n\} \to \{1, \ldots, c_1 n\}$$

függvény adja meg.

Ha x egy logikai változó, akkor legyen

$$x^0 = \neg x, \quad x^1 = x$$

Ha j egy természetes szám, akkor legyen

$$[j]_n$$

a j 2-es számrendszerbeli felírásának n-edik számjegye.

Minden t-re; minden i-re, j-re, és ε-ra ($\varepsilon = 0$ vagy 1) írjuk fel a következő formulát.

$$G_{t,i,j,\varepsilon}(z_{ti}, \{\lambda_{tk}\}\ \{\chi_{tk}\}, \{\chi_{t+1,k}\}) = z_{ti}^{\varepsilon} \,\&\, \chi_1^{[i]_1} \,\&\, \ldots \,\&\, \chi_{t\log c_1 n}^{[i]\log c_1 n} \,\&\, \lambda_1^{[j]_1} \,\&\, \ldots \,\&\, \lambda_{t\log c_2 n}^{[j]\log c_2 n} \to$$

$$\to \chi_{t+1,1}^{[\gamma(\varepsilon,j)]_1} \,\&\, \ldots \,\&\, \chi_{t+1,\log c_1 n}^{[\gamma(\varepsilon,j)]\log c_1 n}.$$

Ha minden $G_{t,i,j,\varepsilon}$ formulát & jellel összekapcsolunk, akkor még mindig csak egy n^c hosszúságú formulánk lesz (c konstans). Ez a G_γ formula fejezi azt ki, mit csinál a γ függvény.

Ugyanígy kell venni

$$G_{\sigma_M} \quad \text{és} \quad G_{\sigma_V}$$

formulákat. Végül a 3. Tétel bizonyításában szereplő formula a következő.

$$F = G_{\sigma_M} \& G_{\sigma_V} \& G_\gamma \& (z_{01} \leftrightarrow x_1) \& \ldots \& (z_{0n} \leftrightarrow x_n) \&.$$

$$\&(z_{0n+1} \leftrightarrow y_1) \;\&\; (z_{02n} \leftrightarrow y_n) \;\&\; z_{n0}.$$

Az utolsó tag azt fejezi ki, hogy a számolás végén a memória első pozíciójában 1 van, amit az $y \in A_x$? kérdésre adott pozitív válaszként értékelhetünk.

IRODALOM

1. R. Karp: Reducibility among combinatorial problems. Complexity of computer computations, ed. R. E. Miller and J. W. Thatcher, N. Y., 1972. Plenum Press. 85—104. pp.
2. Cook, S.: The complexity of theorem-proving procedures, Conf. Record of Third ACM Symp. on the Theory of Computing (1970) 151—158. pp.
3. Л. А. Левин: Универсальные переборные задачи. Доклады Академии Наук.
4. A. R. Meyer—L. J. Stockmayer: Word problems requiring exponential time. Proc. of Fifth Annual ACM Symp. on Th. of Computing Austin, Texas, April 3—May 2, 1973. 1—9. pp.
5. Péter Rózsa: Rekursive Funktionen. Budapest, 1951. Akadémiai Kiadó.
6. А. Н. Колмогоров—В. А. Успенский: К определению алгоритма. Успехи М. Н. 82 3—28. стр.

ON PROBLEMS SOLVABLE BY SEARCH

Hopeless as it seems to find an algorithm with an acceptable number of steps for some problems (as e. g. the problem of the travelling salesman), still no theoremes exist that could give some non-trivial lower bounds for the necessary time, number of steps, memory etc. for the solution of such problems. Of course, for stating such a theorem the exact definition of an algorithm is essential. This notion has been applied in mathematical logics in so far.

A research trend „computer science" deals with these difficult questions. This research verges on mathematical logics and practical operations research. In the last couple of years new results have been emerging creating a widespread echo. A very broad scale of problems can be determined, and it can be proved about quite a few problems that

their solution is — within their group — the most difficult. We are considering problems that are evidently, absolutely solvable but every solution contains trials with steps of exponential number (as e. g. the problem of the travelling salesman).

The main part of the article makes an attempt to clarify the intuitive contents of the notions and statements. The appendix makes it theoretically possible to formalize the whole main part.

О ЗАДАЧАХ, РАЗРЕШИМЫХ ПЕРЕБОРОМ

Каким бы безнадежным ни казалось найти альгоритм с приемлемым количеством шагов к некоторым задачам (например, к проблеме коммивояжера), не имеем еще таких теорем, которые дают нетривиальную нижнюю оценку для времени,количества шагов, памяти и т. п. необходимого к решению данной задачи. Конечно, для установления такой теоремы надо дать точное определение понятия альгоритма. До сих пор, в первую очередь математическая логика использовала это понятие. Возникающими трудными вопросами занимается так называемый «Computer Science», стоящий на границе математической логики и практического исследования операций. В последних годах родились интересные результаты. Можно определить широкую категорию задач и можно доказать о много известных задачах, что они являются внутри категории самыми трудными. Здесь речь идет о таких задачах, о которых знаем, что разрешимые, но каждое известное решение содержит экспоненциальное количество шагов (например, проблема коммивояжера).

Основная часть статьи освещает интуитивное содержание понятий и определений. Приложение в принципе допускает формализацию основной части.