Tudod-e?

Algoritmusok

I. Alapfogalmak

Ma már köznapivá vált az algoritmus fogalma. Mindenki használja, annak ellenére, hogy sokan nem tudják, hogy ők éppen egy algoritmust hajtanak végre. Íme egy egyszerű példa: két természetes szám összeadása. Ha két természetes számot össze szeretnénk adni, akkor „egyszerűen összeadjuk őket", de ezt is egy bizonyos, jól meghatározott, algoritmus alapján tesszük, habár ez már automatizmussá vált számunkra, és már a lépéseket is összeolvasztjuk vagy kihagyjuk. Kissé jobban megvizsgálva ezt az összeadást, mint algoritmust, az alábbi „momentumokat" (lépéseket) különíthetjük el:

1) egymás alá írjuk a számokat,
2) ha nem azonos nagyságrendűek, akkor gondolatban kiegészítjük a rövidebb számot zérusokkal (balról), míg akkora nem lesz, mint a másik,
3) jobbról-balra haladva összeadjuk az egymás alatt levő számjegyeket, hozzáadjuk az átvitelt, ha van ilyen, leírjuk az így kapott összeg osztási maradékát az illető számrendszer alapjával, az átvitel pedig a hányados lesz,
4) ha már nincs több számjegy, és az átvitel nem nulla, akkor ezt egy „újabb" helyértékre írjuk, vége az összeadásnak, különben folytatjuk a 3) lépést.

Körülbelül így néz ki nagyvonalakban az összeadás, lépésekre bontva.

Mivel a számítástechnika egyik bűvszava az algoritmus, lássuk hogyan is alakult ki ez a szó. D. E. Knuth így vélekedik a szó eredetéről: „Az *algoritmus* szó már önmagában is nagyon érdekes. Úgy tűnhet egy pillanatig, hogy a logaritmus szót akarta valaki leírni, de nem sikerült neki, mert összezagyválta az első négy betűt. 1957-ig az algoritmus szó nem is fordul elő a Webster-féle értelmező szótárban. Csak a megfelelő angol szó (algorithm) archaikus változatát találjuk meg (algorism).", mely aritmetikai műveletek arab számokkal való végzését jelentették. Végül is a matematikatörténészek találták meg pontos eredetét. Azt állítják, hogy *Abu-Ja far Mohammed ibn Mura al-Kvarizmi* (780? - 850?) arab matematikus nevének, *al-Kvarizmi* latinos elferdítéséből származik. Említésre méltó, hogy ezen arab matematikus egyik híres, *Kitáb al-dzsabr val-mukábala* (A rövidítés és törlés tudománya) c. munkájának második szava szolgáltatta a matematika másik műszavát, az *algebrát.* Mivel ma már az algoritmus fogalma nagyon széles körben alkalmazott, igencsak nehéz egy pontos, egzakt definícióval szolgálni. (lásd [Kn], [Sai])

Megvizsgálva a szakirodalmat, az alábbi definíciókkal találkozhatunk:

„Ez a megnevezés a négy aritmetikai művelet fogalmát takarja, nevezetesen az összeadását, a szorzását, a kivonását és az osztását" (*Vollständiger Mathematischer Lexikon, Lipcse 1747*)

„egy feladatcsoport megoldására szolgáló, adott módon végzendő számolási eljárás. Az algoritmikus eljárásoknál a kitűzött feladat megoldására egy bizonyos művelet (vagy műveletcsoport) ismételt alkalmazásával jutunk el, a művelet *n*-edik végrehajtásánál felhasználjuk az előzőleg kapott eredményeket." (*Új magyar lexikon, 1961*)

„...az algoritmus fogalmát tárgyaljuk. Ez a fogalom témánk szempontjából alapvető, mégsem definiáljuk. Inkább olyan intuitív fogalomnak tekintjük melynek formalizálására (és ezzel matematikai szempontból való vizsgálatára) különféle lehetőségek vannak. Az algoritmus olyan matematikai eljárást jelent, mely valamely számítás vagy konstrukció elvégzésére – valamely függvény kiszámítása – szolgál, és

melyet gondolkodás nélkül, gépiesen lehet végrehajtani. Ezért az algoritmus fogalma helyett a matematikai gép különböző fogalmait vezetjük be." (*Lovász László: Algoritmusok bonyolultsága, ELTE, Budapest, egyetemi jegyzet, 1992*)

„Algoritmusnak nevezünk adott alakú speciális feladat megoldására kidolgozott olyan eljárást, amely utasításszerűen előre megadott számolási lépések sorozatából áll" (*Számítástechnikai kislexikon, Műszaki Könyvkiadó, Budapest, 1973*)

Mint láttuk, minden értelmezésben szerepel valami közös, azonban egyik sem fedi teljes mértékben az algoritmus szó jelentését. Fogadjuk el az algoritmust, mint intuitív fogalmat, és vizsgáljuk meg az alábbiakban, az algoritmusok tulajdonságait, hogyan lehet őket leírni, és nézzünk meg néhány érdekes algoritmust!

Egy algoritmusról azt mondjuk, hogy strukturáltan helyes, ha eleget tesz az alábbi követelményeknek.

1) ***Bemeneti adatok:*** egy algoritmus igényelhet vagy nem bizonyos kezdéti értékeket, amelyből majd „származtatja" az eredményt. Ha vannak ilyen adatok, akkor ezek értelmezési tartománya jól meghatározott. Lásd az összeadási algoritmus esetén a két összeadandót. Ha nem két természetes szám a bemeneti adat, akkor az algoritmus már nem működik helyesen. Ezért igen nagy hangsúlyt kell fektetni a bemeneti adatok tesztelésére.

2) ***Meghatározottság:*** azt jelenti, hogy bármelyik pillanatban, tudjuk, hogy éppen mi fog történni, vagyis az algoritmusnak minden lépése jól meghatározott, és egyértelműen eldönthető egy adott feladat esetén. Az összeadás algoritmusa esetén, mindig tudjuk, hogy éppen melyik számjegypárt kell összeadnunk, és ezzel mit kell tennünk.

3) ***Kimeneti adatok:*** az algoritmusok valamilyen eredményt kell szolgáltassanak, mert csak így van értelmük, és csak így dönthető el, hogy helyes adatokat szolgáltat vagy sem. Lásd az összeadási algoritmus esetén az összeget.

4) ***Elvégezhetőség:*** alatt azt értjük, hogy az ember papírral és ceruzával a kezében is képes legyen eljutni az eredményhez, ha szigorúan betartja az algoritmus lépéseit.

5) ***Végesség:*** azt jelenti, hogy egy adott algoritmus véges sok lépésen belül véget ér vagyis befejeződik. Tehát a lépések száma, illetve a végrehajtási idő véges. Jelen esetben is az algoritmusunk csak véges nagyságrendű számok összeadása esetén fog valós időben befejeződni. Ha leszűkítjük az intervallumot (pl. [0,100000]) ahonnan vehetik az értékeiket az összeadandók, akkor már minden tökéletesen működik. Képzeljük el, most, hogy azt a feladatot kaptuk, hogy írjunk egy olyan algoritmust, mely meghatározza az összes prímszámot. Ez az algoritmus sosem fog leállni mivel a prímszámok száma végtelen. (lásd Eukleidész bizonyítását).

Ha ezen feltételek mellé még beszámítjuk a **helyességet**, vagyis matematikailag is igazolható, hogy az illető algoritmus minden bemeneti adatra a kívánt eredményt szolgáltatja, akkor helyes algoritmusról beszélhetünk. Ha a meghatározottságot ilyen értelemben használjuk, akkor *determinisztikus algoritmusokról* beszélhetünk, de ezzel ellentétben, vannak olyan algoritmusok, melyek esetén egyértelműen nem dönthető el, hogy éppen melyik lépést hajtják végre az algoritmusban, vannak olyan részek melyek egy időben hajtódnak végre, úgymond „párhuzamosan". Ezeket az algoritmusokat *nem determinisztikus algoritmusoknak* nevezzük. Az ilyen algoritmusok, főleg több processzoros architektúrák esetén alkalmazhatók. (lásd [CoLeRi])

II. Algoritmusok leírására szolgáló eszközök

Az algoritmusok könnyebb leírása és megértése érdekében bevezették a folyamatábrákat (amelyek grafikusan szemléltetik a lépések soronkövethetőségét) és a pszeudokódot. A folyamatábrák esetén a „műveleteket" bizonyos konvenciós jelekkel írjuk le, majd ezeket irányított nyíllal kötjük össze, hogy tudjuk, hogy milyen sorrendben követik egymást a műveletek. Nagy előnye ennek az ábrázolásnak, hogy szemléletes, de „nagyobb" algoritmusok esetén már nagyon bonyolult ábra keletkezik, amit szinte lehetetlen átlátni. (lásd [Ká])

A továbbiakban a pszeudokódos leírást használjuk, mivel így elkerüljük annak a lehetőségét, hogy bizonyos programozási nyelv (Basic, Pascal, C/C++, Delphi)

nyújtotta lehetőség kihasználásával már nem is algoritmust, hanem a számítógép számára érthető programot írunk. Nem szabad szem elől téveszteni, hogy a legfontosabb része a programozásnak nem a programírás, hanem az algoritmus, annak minél általánosabb leírása, hogy mikor az illető algoritmust programozni kell, akkor már a bizonyos nyelv által nyújtotta lehetőségek csak könnyíthetik a programozást. („Egyszer gondolkozzunk, és utána programozzunk!" tipológia kialakulását kell szorgalmaznunk). Ha az algoritmust egyből bizonyos programnyelven írjuk le, más programnyelvbe való átültetése bizony gondokat okozhat.

III. Algoritmusok vizsgálata

Az algoritmusok megjelenése maga után vonta egy új diszciplína kialakulását, amelynek tárgya az algoritmusok helyességének, illetve optimalitásának vizsgálata. A helyességet ma már matematikai módszerekkel igazolják, formalizálva az algoritmust. Az optimalitást két megvilágításban kell tárgyalnunk. Egy algoritmus lehet optimális a háttértároló szempontjából, illetve az illető algoritmus időbeni végrehajtása szempontjából. Így beszélhetünk *tárbonyolúltságról* illetve *időbonyolúltságról.* A továbbiakban mi a bonyolultság alatt időbonyolultságot értünk.

Lássunk egy egyszerű példát. Ha adott az $(x_n)_n \in N$ számsorozat, keressük meg a sorozat maximumát!

```
Adottak n, x_i (i=1,n)
Max:=x_1
Minden i:=2,3, ...., n végezd el,
   Ha Max<x_i akkor
        Max:=x_I
   (Ha) vége
(Minden) vége
Eredmény Max;
```

Ha megvizsgáljuk ezt az algoritmust, alapműveletnek tekintve két szám összehasonlítását, vagyis feltételezzük, hogy ezt a műveletet egy időegység alatt képes elvégezni az algoritmus, azt láthatjuk, hogy ezt az algoritmus *n-1* időegységet igényel (ennyi összehasonlítást végez összesen). Azt mondjuk, hogy az illető algoritmusunk bonyolultsága *n-1.*

Vannak esetek mikor nem dönthető el, hogy pontosan hány műveletet fog végezni az algoritmusunk, ekkor a bonyolultság legrosszabb értékét *W(n)*-nel (*worst case*) jelöljük, míg az átlagértékét *A(n)*-nel (*average case*). A fenti algoritmus esetében: *W(n)=A(n)=n-1.*

Jól ismert a Hanoi-tornyai probléma: Helyezzünk át egy rúdról *n* darab korongot (ezek lentről felfele csökkenő sorban vannak) egy másik rúdra, egy harmadik segítségével úgy, hogy kisebb méretű korongra soha nem kerülhet nagyobb korong.

```
Hanoi(n,rúd1,rúd2,rúd3):
  Ha n>0 akkor
        Hanoi(n-1,rúd1,rúd3,rúd2);
        Átrak(rúd1,rúd3);
        Hanoi(n-1,rúd2,rúd1,rúd3);
   (Ha) vége
(Hanoi) vége
```

Ha alapműveletnek tekintjük egy korong áthelyezését egyik rúdról a másikra, és H(n)-nel jelöljük az áthelyezések számát *n* db. korong esetén, felírhatjuk, hogy:

$$\begin{cases} H(1) = 1 \\ H(k) = 2H(k-1) + 1, \quad k \geq 2 \end{cases}$$, amiből következik, hogy $H(n)=2^n-1$.

Látható, hogy az illető algoritmus exponenciális idejű, vagyis ha *n* értéke nagy a korongok száma nagyon megnövekszik.

Az algoritmusok esetén fontos, hogy az illető algoritmus optimális legyen. Tudjuk, hogy egy feladatot sokféleképpen meg lehet oldani. Példának okáért bármely rekurzív algoritmusra adható egy vele ekvivalens iteratív algoritmus. (Ez nem jelenti feltétlenül azt, hogy az illető algoritmusok időbonyolultsága azonos).

Tekintsünk egy F feladatot, illetve az ezt megoldó $A_1, A_2, \ldots A_n$ algoritmusokat. Ekkor azt mondjuk az A_k algoritmusról, hogy *optimális*, ha nincs olyan A_i ($i \neq k$) algoritmus, amely kevesebb művelettel képes megoldani az F feladatot, mint A_k. (Tartózkodjunk a legoptimálisabb algoritmus fogalmától, mert nem helyes. Mivel az optimális szó már magában foglalja a legjobb, legrövidebb időbonyolultságot, a legoptimálisabb értelmét veszti.)

Egy adott feladatról azt mondjuk, hogy *P osztálybeli feladat* vagy *polinomiális feladat,* ha az algoritmus – amely megoldja – lépéseinek száma megbecsülhető egy polinommal, vagyis létezik legalább egy olyan algoritmus, amely megoldja a feladatot polinomiális időben. Az algoritmust, amely megoldja az ilyen típusú feladatokat, *polinomiális idejű algoritmusnak* nevezzük.

Ezzel ellentétben vannak olyan feladatok, amelyek nem oldhatók meg polinomiális időben, ezeket nevezzük *nemdeterminisztikusan polinomiális feladatosztálynak*, vagy egyszerűen csak *NP-feladatoknak*. (lásd. leghosszabb Hamilton-kör megkeresése, hátizsák-probléma, logikai formula kielégíthetősége stb.). Belátható, hogy a *P feladatosztály* valódi részhalmaza az *NP feladatosztálynak*. ($P \subseteq NP$) Mindmáig eldöntetlen kérdés, hogy $P=NP$ vagy $P \neq NP$. A kutatások során meghatároztak egy *NP-nehéz* feladat alosztályt amelybe besorolták a legnehezebb NP feladatokat. Azt tartják, hogy ha találnánk egy NP-nehéz feladatra egy polinomiális idejű algoritmust, akkor N=NP. (Máig ez még nem sikerült senkinek !)

Mivel a kutatók nem nyugodtak bele az NP-feladatok polinomiális időben való megoldhatatlanságába, rájöttek, hogy adhatók olyan algoritmusok (mindenik feladatra specifikusan), amelyek polinomiális időben közelítő megoldást adnak, azaz úgy oldják meg a feladatot, hogy nem bíztos, hogy a legjobb megoldást szolgáltatják. Ezeket az algoritmusokat nevezzük *szuboptimális* vagy *közelítő algoritmusoknak*. Nézzük meg például a ládapakolási feladatot! A feladat abból áll, hogy egységnél kisebb tömegű tárgyakat helyezzünk el minél kevesebb számú ládába, tudva azt, hogy mindegyik láda csak egységnyi tömeget bír el. Ez NP-nehéz feladat.

A ládapakolási probléma esetén (adott tárgyakat helyezzünk lehetőleg minél kevesebb számú adott kapacitású ládákba) ilyen közelítő algoritmusok a következők:

1) **First Fit (*FF – first fit = első egyezés*)**: Veszem az első tárgyat, és beteszem az első olyan ládába amelybe belefér. Veszem a második tárgyat, és kezdem elölről. Próbálom berakni az első ládába. Ha nem fér bele, megpróbálom betenni a másodikba, és így tovább míg mindeniket berakom.

2) **Best Fit (*BF – best fit = legjobb egyezés*)**: Itt az a fontos, hogy a soron következő tárgyat mindig abba a ládába tegyem, ahol a lehető legkevesebb hely marad. Látható, hogy mindkét módszerrel polinomiális idejű algoritmus konstruálható, de általában nem az optimális megoldást szolgáltatják. Helyzettől függően, van amikor jobb az FF algoritmus, és van amikor jobb a BF.

Szakirodalom:

[Ba] Babai László, *Transparent Proofs and Limits to Approximation*, Proc. First European Congress of Mathematics, Birkhäuser (1994)

[CoLeRi] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Algoritmusok*, Műszaki Könyvkiadó, Budapest (1997)

[Ká] Kása Zoltán, *Algoritmusok tervezése*, Stúdium Könyvkiadó, Kolozsvár (1994)

[Kn] Donald E. Knuth, *A számítógép-programozás művészete I*, Műszaki Könyvkiadó, Budapest (1994)

[LoGá] Lovász László, Gács Péter, *Algoritmusok*, Műszaki Könyvkiadó, Budapest (1978)

[BaaS] S. Baase, *Computer Algorithms, Introduction to Design and Analysis*, Addison-Wesley (1983)

Vajda Szilárd, egyetemi hallgató