$$I=\int^{R} L/(4\,\pi\,r^2)\; n\; (4\,\pi\,r^2)\; dr = L\; n \int^{R} dr = L\; n\; R$$

Ez az integrál végtelen, ha az Univerzum R sugara végtelen. Az égbolt éjszaka nem lehet sötét, sőt éjjel-nappal végtelen fényes kell, hogy legyen. Ezt a következtetést az sem változtatja meg, hogy a távoli csillagok fénye abszorbeálódhat menetközben, ugyanis egyensúlyban ugyanennyi a reemisszió. Az Univerzum tehát nem lehet végtelen, ha egyensúlyban van. Ha viszont nincs egyensúlyban, akkor időben nem lehet állandó. A táguló Világegyetem modelljének a fényében az Olbers-paradoxon tehát egyáltalán nem paradoxon, hanem egy meggyőző érv azzal a feltevéssel szemben, hogy az Univerzum végtelen.

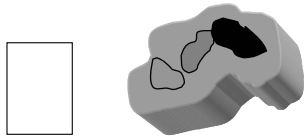
A zsidó, a keresztény, a mohamedán és sok más vallás azt tanítja, hogy a világot Isten teremtette. Ezt a felfogást igen sokan elfogadjuk. A felvilágosodás korának racionalizmusa megkísérelte Istent száműzni az emberi gondolkodásból, ehhez a teremtés hitét valami mással kellett helyettesíteni. A legegyszerűbbnek az tűnt, hogy a világot térben és időben végtelennek deklarálták és így elvben ki lehetett iktatni az emberi gondolkodásból mind a teremtés, mind pedig a végítélet ideáját. Kant szerint a tér és az idő csak a tudatunkban létező kategóriák, amik meghatározzák gondolkodásunkat, ezért a világot el sem tudjuk képzelni másnak, mint végtelennek. A Világ végtelenségének dogmája annyira eluralkodott a filozófiában, hogy Einstein saját egyenleteinek helyességében kételkedett, amikor időben változó megoldást kapott. Nem hitte el az eredményt, mert az ellentétben állt az általánosan elfogadott dogmával! Észrevette azonban, hogy ha az egyenleteibe beír egy konstans tagot, akkor időtől független megoldást is lehet kapni. Egy ideig azt hitte, hogy ilyen módon a dogmával való ellentmondást megszüntette. Kitűnt azonban, hogy az így kibővített, az ún. kozmológikus konstanst tartalmazó egyenlet időtől független megoldása instabil, azaz a legkisebb perturbáció hatására időfüggővé válik, ezért kénytelen volt ezt a konstans tagot elhagyni. Élete végéig a legnagyobb tévedésének tartotta ezt a „kisiklást", annál is inkább, mert a Friedmann-féle megoldás, ami teljes összhangban áll a Hubble-törvénnyel, meggyőzte arról, hogy a Világegyetem időben tágul.

Lovas István
a Magyar Tudományos Akadémia tagja

Programozási technikák felülnézetből

I. rész

Végezzük el a következő kísérletet: mutassunk fel egy ívlapot és kérjük meg a tanulókat, hogy nevezzék meg minél több tulajdonságát. Ezután – egy másik osztályban – ismételjük meg a kísérletet, de úgy, hogy az ívlappal együtt egy másik alakzatot is felmutatunk, mondjuk ami fából készült és körülbelül úgy néz ki mint az alábbi.



Mit fogunk tapasztalni? Azt, hogy a második osztályban az ívlapnak számottevően több tulajdonsága fog megfogalmazódni a tanulókban. Például nem valószínű, hogy az első osztályban felfigyelnek arra, hogy az ívlap egyszínű, síkidom, összegyűrhető, stb.

Ez az egyszerű kísérlet egy régismert igazságot emel ki: *Az ellentétek felhívják a figyelmet, mind magukra, mind a hasonlóságokra.*

Hogyan lehetne alkalmazni ezt az alapelvet az informatika oktatásában?

A legtöbben megtesszük ezt – még ha nem is tudatosan – amikor a rendezéseket tanítjuk. A fejezet végén veszünk egy konkrét számsorozatot amelyen elmímeljük, vagy elmímeltetjük, az összes megtanított rendezési algoritmust, felhívva a figyelmet a hasonlóságokra és a különbségekre.

Mi a helyzet azonban akkor, ha a programozási technikákkal (Greedy, Back-track, Divide et impera, Dinamikus programozás) foglalkozó anyagrészt fejeztük be? Lehetne-e ugyanezt a módszert alkalmazni? Sokan talán idegenkednének ettől úgy érvelve, hogy amíg a rendezési algoritmusok ugyanazt a feladatot oldják meg, addig minden egyes technikának megvan – többé kevésbé – a saját felségterülete. De vajon ez azt jelenti-e, hogy egyáltalán nem lehet találni olyan feladatokat, amelyekhez úgymond mindenik technika „hozzá tudjon szólni" – még ha nem is tartozik kifejezetten az ő hatáskörébe – lehetővé téve azáltal a párhuzamos tanulmányozásukat?

Szolgáljon válaszul a következő feladat az optimizálási feladatok kategóriájából, hiszen ez egy olyan terület, amellyel mind a négy technika foglalkozik valamilyen szinten.

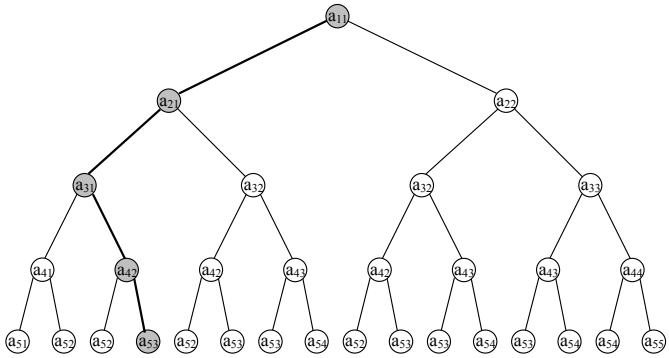
Egy n soros négyzetes mátrix főátlóján és főátló alatti háromszögében természetes számok találhatók. Feltételezzük, hogy a mátrix egy *a* nevű kétdimenziós tömbben van eltárolva. Határozzuk meg a „leghosszabb" csúcsból (*a[1][1]* elem) alapra (*n*-ik sor) vezető utat, figyelembe véve a következőket:

- egy úton az *a[i][j]* elemet vagy az *a[i+1][j]* (függőlegesen le), vagy az *a[i+1][j+1]* elem (átlósan jobbra) követheti, ahol *1 <= i < n* és *1 <= j < n*.
- egy út „hossza" alatt az út mentén található elemek összegét értjük.

7				
5	9			
10	1	4		
2	7	3	1	
2	5	8	3	1

Például, ha *n=5* esetén a mátrix az alábbi, akkor a „leghosszabb" csúcsból alapra vezető út a besatírozott, hossza pedig *37*:

Ez egy olyan optimizálási feladat, amelyben az optimális megoldáshoz *n-1* döntés nyomán juthatunk el és mindenik döntésnél *2* választásunk van (melyik irányba lépjek tovább, függőlegesen le vagy átlósan jobbra). Az alábbi ún. megoldásfa, jól szemlélteti mindezt a példaként megadott mátrix esetében.



Az optimális megoldás meghatározása az optimális döntéssorozat megtalálását jelenti. Úgy is mondhatnánk, hogy meg kell találjuk a megoldásfa 2^{n-1} darab gyökértől levélhez vezető útja közül a „legjobbat". Más szóval, meg kell keressük a megoldásfának a „legjobb levelét", azt amelyikhez a „legjobb út" vezet.

A megoldásfának egy alaposabb vizsgálata további észrevételekhez vezethet el:

1. A megoldásfa csomópontjainak száma $1+2+2^2+...+2^{n-1}=2^n-1$. Ez azt jelenti, hogy bármely algoritmus amely bejárja a teljes fát ahhoz, hogy megtalálja az optimális utat, exponenciális komplexitású lesz.
2. Amíg a fa a teljes feladatot képviseli, addig a részfái azokat a hasonló, de egyszerűbb részfeladatokat, amelyre ez lebontható. Konkrétan: az a_{ij} gyökerű részfa, az *a[i][j]* elemtől az alapra vezető „leghosszabb út" meghatározásának feladatát ábrázolja.
3. A fenti ábra azt is kiemeli, hogy különböző döntéssorozatok azonos részfeladatokhoz vezethetnek, ami azt jelenti, hogy a megoldásfának vannak identikus részfái. Nem nehéz átlátni, hogy a különböző részfeladatok száma azonos a mátrix elemeinek számával, azaz $n(n+1)/2$. Tehát az az algoritmus, amelynek sikerül elkerülni az azonos részfeladatok többszöri megoldását, négyzetes komplexitású lesz.

És most lássuk, milyen sajátos stratégiákkal keresi az említett négy technika, a megoldásfa „optimális útját".

Greedy

Indulok a csúcsból. Két út áll előttem, melyiket válasszam? Természetesen, mindig a nagyobbik elemet. Mi a következménye mohóságának? Jelen esetben az, hogy egyáltalán nem biztos, hogy megtalálja a legjobb utat. Azt feltételezte, hogy a lokális optimum globális optimumhoz vezet, ami persze erre a feladatra nem igaz. A példa-mátrix esetében a Greedy-út *31* hosszú lesz, és ez a következő: *a[1][1], a[2][2], a[3][3], a[4][3], a[5][3]*.

Back-track

Kigenerálom az összes csúcsból alapra vezető utat, és kiválasztom közülük a „legjobbat". Ennek érdekében mélységében fogom bejárni a megoldásfát, keresve a „legjobb levelet".

Divide et impera

Észrevehető, hogy az a_{ij} elemtől induló „legjobb út" meghatározása visszavezethető az a_{i+1j}, illetve a_{i+1j+1} elemektől induló „legjobb utak" meghatározására, ugyanis amennyiben ezek rendelkezésre állnak, nem marad más hátra minthogy „bevágódjunk" az a_{ij} elemmel a hosszabbik elé. Mindez matematikailag a következő képlettel írható le, ahol c_{ij}-vel az a_{ij} elemtől induló „leghosszabb út" hosszát jelöltem:

$$c_{ij} = \begin{cases} a_{ij}, ha\ i = n \\ a_{ij} + \max(c_{i+1j}, c_{i+1j+1}), ha\ i < n, j <= i \end{cases}$$

Átültetve a fenti képletet egy rekurzív függvényre, egy olyan algoritmust kapunk, amely a rekurzivitás mechanizmusa által – miközben mélységében bejárja a fát – először lebontja a feladatot egyszerűbb és egyszerűbb részfeladatokra, majd pedig a „visszaúton" meghatározza az optimális út hosszát.

Megjegyezendő, hogy a Divide et impera ebben a változatban csak a legjobb út hosszát határozza meg és nem magát az utat is.

Dinamikus programozás

Az én alapötletem az, hogy kiindulva a banális részfeladatok kézenfekvő megoldásaiból, felépítsem az egyre bonyolultabb részfeladatok megoldásait, míg végül el nem jutok a fő feladat megoldásához. Mivel el szeretném kerülni az identikus részfeladatok többszöri megoldását, ezért az optimális részmegoldásokat eltárolom egy *c* kétdimenziós tömbben. Azért elég a részfeladatoknak csak az optimális megoldását eltárolni, mert a feladatra érvényes az optimalitás alapelve, miszerint, az optimális megoldás optimális

részmegoldásokból épül fel. A *c* tömböt a fenti képlet alapján töltöm fel – lentről felfelé, soronként – hiszen leírja miként építhető fel az optimális megoldás, lépésről-lépésre, az optimális részmegoldásokból. Végül a *c[1][1]*-ben lesz az optimális út hossza. Ahhoz, hogy meglegyen maga az út is, annyi szükséges még, hogy végigmenjek a *c* tömbön Greedy módra.

A Dinamikus programozásnak van egy rekurzív változata is, amikor a fenti képletet rekurzívan használom ugyan de – a részeredmények eltárolása által – ügyelek arra, hogy ne oldjak meg többször azonos részfeladatokat.

A következő részben arról olvashatsz, hogy milyen érdekfeszítő következtetésekhez vezethetnek el a technikák ezen párhuzamos bemutatása.

37				
30	25			
25	16	15		
7	15	11	4	
2	5	8	3	1

Kátai Zoltán

A programozási nyelvek osztályozása

Programozási nyelveket több szempont szerint is osztályozhatunk, különféle metszeteket készíthetünk, különböző nyelvosztályokat állíthatunk fel, de az egyes jellemzők közé éles határ nem húzható. Hibrid nyelvekről akkor beszélünk, ha az adott nyelv egy osztályozási szempont szerint több osztályba tartozik. Napjaink programozási nyelveinek többsége hibrid.

Amatőr és professzionális nyelvek

Az *amatőr programozási nyelvekre* az interaktivitás, a sok nyelvi elem, a gyors nyelvi fejlődés jellemző. Ezekben a nyelvekben a programok szerkezete egyszerű, és ezek speciális gépi tulajdonságokra épülnek rá.

A *professzionális nyelvekre* a modularitás, a magas fokú stabilitás és a kevés nyelvi elem a jellemző. Ezen nyelvek igen hatékonyak, sok lehetőséggel bírnak és gépfüggetlen kódot generálnak, vagy a kód átvihető más architektúrájú gépekre is.

Emberközeliség

A *gépi nyelvek* használatával minden hardver lehetőség kihasználható, azonban a memóriacímeket, a memória-kiosztást és a programkódot önerőből kell megvalósítani (a memóriában lévő utasításkódokat közvetlenül a programozó adja meg). A megírt programok gépközeliek, közvetlenül a processzor utasításkészletére épülnek.

Az *alacsony szintű nyelvek* géporientált nyelvek ugyan, de megjelennek a szimbolikus utasítások, azonosítók és címkenevek, megjelenik a feltételes vezérlésátadás fogalma, az eljárások és a visszatérések. Az adatokat deklarálni, definiálni lehet és a tárhely is ennek függvényében foglalódik le. Megjelennek a makrók és a direktívák. A közvetlen kódok helyett rövid, könnyen megjegyezhető szavakat alkalmazunk (mnemonikok) a könnyebb megjegyzés, a jobb átláthatóság kedvéért. Jobban áttekinthetők a címzési módok, a programokba megjegyzéseket szúrhatunk be, külön fordítható egységekkel dolgozhatunk.