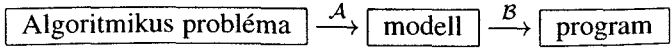
1.

Algoritmikus problémák megoldása

A tudományok nem próbálnak megmagyarázni semmit, alig-alig próbálkoznak értelmezéssel; főleg modelleket készítenek... Egy ilyen matematikai konstrukciónak csak és kizárólag az adja a létjogosultságát, hogy használható. NEUMANN JÁNOS

Az egyik legfontosabb célunk, hogy a hatékony algoritmusok tervezésébe bevezessük az olvasót. Mint a problémamegoldás általában, ez is sokszínű, többféle képességet igénylő kreatív folyamat. A problémával kapcsolatos ismeretek mellett komoly szerep jut az ötleteknek. Ezek olykor egyszerűek, mintegy maguktól adódnak, máskor komolyan meg kell dolgoznunk értük. Néha a megoldás szinte kiolvasható magából a feladatból, de az is előfordul, hogy csak komoly szellemi erőfeszítés árán jutunk előbbre. Nem várhatunk ezért biztos recepteket, amelyekkel minden esetben célhoz érünk. Nem vagyunk ugyanakkor teljesen támasz híján sem. Mára az algoritmika a számítógépes tudományok eszközökben és módszerekben gazdag területévé nőtte ki magát. Kialakultak olyan fogalmak, megközelítési módok, melyek iránytűként segíthetnek bennünket a megoldások felé vezető úton. Ezen a területen is fontos szerepet játszanak a *példák*, az értelmes és sikeres megoldások tanulságai, amelyek sokszor túlmutatnak felmerülésük esetleges keretein.

Ebben a rövid, bevezető jellegű fejezetben mi is példák segítségével szeretnénk első képet adni tárgyunkról, az algoritmusok világáról. Az algoritmikus problémák megoldását hasznos kétlépcsős folyamatként elképzelni, amit a következő egyszerű rajz szemléltet:



Az $\mathcal{A}$ leképezés azt a gyakran járt utat jelenti, hogy a problémát áttesszük, átfogalmazzuk egy kellően absztrakt, többé-kevésbé formalizált *modellbe*. Ez a lépés általában pontosítást és egyszerűsítést jelent. Megszabadulunk a probléma megfogalmazásában levő olyan elemektől, amelyek a megoldás szempontjából lényegtelenek. Legtöbbször arról van szó, hogy a feladatot kihámozzuk abból a tarkabarka, természetes nyelvből szőtt köntösből, amiben elénk került.

Az $\mathcal{A}$ lépés eredményének keretéül szolgáló modell sokféle lehet. Elég tágnak kell lennie ahhoz, hogy a probléma megoldásához szükséges tényezőket hűen kifejezze. Másfelől elég egyszerűnek is kell lennie azért, hogy kezelni tudjuk. A modell – úgy képzeljük – valahol a félúton van a probléma és a megoldását adó program között. A modell formalizáltsága, pontossága közbülső lépcsőt jelent a programsorok szigorú egyértelműsége felé. Az igazán jó modellek fogalmai könnyen és standard módon képezhetők le számítógépes adatszerkezetekre. Olykor már magában a modellben is helyet kapnak a megcélzott gépi környezet jellemzői. A másik oldalról nézve ezt a közbülső helyzetet azt is elvárjuk, hogy a modell mentes legyen a programnyelvi utasítások földhözragadtságától. Hogy tényleg lehetővé tegye azt, hogy a problémát valamiféle értelmes általánosság szintjén szemlélhessük. Divatos fordulattal élve a modellt tekinthetjük sokadik (legalább negyedik) generációs nyelvi eszköznek.

A $\mathcal{B}$ leképezés jelenti a *hatékony algoritmusok tervezését*, kidolgozását. Ez a folyamat többnyire a modell által adott keretekből indul. Ezen a szinten már egy *pontos algoritmikus problémával* van dolgunk. Ennek összetevői a *bemenő adatok* (más szóval *bemenet* vagy *input*) megadása; valamint az *eredménnyel*, az *outputtal* kapcsolatos követelmények.

A megoldó módszer első változatát a modell fogalmaival fejezzük ki. Ezt a nagyvonalú megoldást lehet azután már többé-kevésbé mechanizálható lépésekkel tényleges programmá finomítani. A későbbiek során legtöbbször ezzel a $\mathcal{B}$ lépéssel, annak is az első részével, a nagyvonalú megoldás kialakításával foglalkozunk. A munkának ebben a fázisában érdemes foglalkozni a kapott algoritmus *elemzésével, értékelésével*, megvizsgálva, hogy a módszer mennyire hatékony, mennyit lehetne még javítani rajta.

Az itt vázolt képben sok egyszerűsítés van. Ezek közül talán a legsúlyosabb, hogy a problémamegoldás folyamatát magabiztos, mindig csak előre való masírozásként tünteti fel. Ez a valóságban legtöbbször nem így van. Gyakran kényszerülünk arra, hogy visszaforduljunk egy zsákutcának bizonyuló irányból, és egy korábbi pontból új utat keressünk.

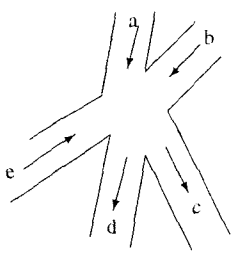
Ennyi általánosság után néhány egyszerű példával szeretnénk érzékeltetni az eddig elmondottakat. A példák bevezető jellegűek; a felmerülő fogalmakat később részletesebben fogjuk tárgyalni.

1.1. A feladattól a modellig

Itt az $\mathcal{A}$ leképezésre szeretnénk néhány példát mutatni. A modell, amibe átírjuk a problémákat, tulajdonképpen egyetlen egyszerű fogalomra, a *gráfra* épül. Ennek az egyszerű fogalomnak az alkalmazásával az eredeti problémák tiszta, pontos megfogalmazását kapjuk.

1.1.1. Közlekedési lámpák ütemezése

Képzeljük el, hogy egy útkereszteződés közlekedési lámpáinak a működését kell megterveznünk. Tegyük fel, hogy a kereszteződésben (északról indulva, órajárás szerint) az a, b, c, d, e egysávos utak találkoznak. Az a, b és e utak a kereszteződés felé, a többi pedig a kereszteződéssel ellentétes irányban egyirányú.



Tegyük fel, hogy minden értelmes, a nyilakat követő áthaladási irányhoz van egy közlekedési lámpánk. Ezek az irányok esetünkben ac, ad, bc, bd, ec és ed. Összesen tehát hat lámpánk van. Például az ac irány lámpája csak az ebben az irányban való átjutást szabályozza (tiltja, illetve engedélyezi). Ennyi a probléma leírása.

Ezután elkezdhetünk gondolkodni. Mi is a feladat tulajdonképpen? Némi tűnődés után arra jutunk, hogy a lámpákból álló rendszer számunkra érdekes állapotait egy-egy *lámpák* $\rightarrow$ {piros, zöld } függvényként foghatjuk fel. Ezzel máris egyszerűsítettük a képet, felismerve, hogy a sárga színnek nincs komoly szerepe a feladat szempontjából. Például a $jujj$ nevű állapot (függvény) lehet a következő:

$$jujj(ac) = jujj(ad) = jujj(bd) = \text{zöld},$$
$$jujj(bc) = jujj(ed) = jujj(ec) = \text{piros}.$$

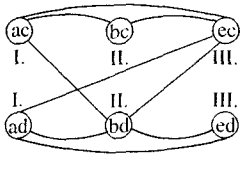
Ezt az állapotot nem szabad megengednünk, hiszen például az ac és bd irányok találkoznak. E két iránynál nem lehet egyszerre mindkét lámpa zöld. Kis további gondolkodás után oda jutunk, hogy az ilyen konfliktuslehetőségek megfoghatók

mint irányokból álló *párok*. Például (ac, bd) egy tiltott pár, (ac, ad) pedig nem. A kapcsolat szimmetrikus, nincs különbség mondjuk az (ac, bd) és a (bd, ac) párok megítélése között.

Mindezek azt sugallják, hogy próbáljuk meg gráffal ábrázolni a helyzetet. A G gráfunk csúcsai az áthaladási irányok: ac, ad, bc, bd, ec és ed. Két csúcsot akkor kössön össze él, ha az irányok konfliktusban vannak. Más szóval, ha a megfelelő két lámpa nem lehet egyszerre zöld. Egy állapot akkor engedhető meg, ha a benne levő zöld csúcsok között nem fut él. További egyszerűsítést jelent az a felismerés, hogy egy állapotot jellemezhetünk pusztán a benne levő zöld csúcsok halmazával.

Ezután megpróbálkozhatunk a feladat pontos megfogalmazásával. Az első változat így hangzik: adjunk meg minél kevesebb állapotot úgy, hogy egyfelől a megadott állapotok mind biztonságosak legyenek; másfelől ne legyen olyan áthaladási irány, amelyhez az összes megadott állapotban a piros szín tartozik. Az utóbbi feltétel azt hivatott szavatolni, hogy nem okozunk örök dugót.

A G gráf segítségével ez még egyszerűbben kifejezhető. Úgy kell a G csúcshalmazát minél kevesebb osztályba (részhalmazba) sorolni, hogy az egy osztályban levő csúcsok között ne menjen él. Esetünkben ilyen osztályozás az $\{ac, ad\}$, $\{bc, bd\}$, $\{ec, ed\}$. A rajzon római számmal tüntettük fel a csúcsok osztályának sorszámát.



Feladat: Mutassuk meg, hogy két osztállyal nem oldható meg a probléma.

Mi történt itt valójában? Az eredeti feladatot átfogalmaztuk a *gráfok* nyelvére, eltüntetve belőle olyan, a vizsgált kérdés szempontjából érdektelen elemeket, mint „közlekedés", „út", stb. Ebben a modellben lényegében csak csúcsok és őket összekötő élek szerepelnek. Valamivel pontosabban egy $G = (V, E)$ gráf két összetevőből áll. Az egyik a csúcsok (pontok) V halmaza, a másik pedig E, az élek összessége. Az E halmaz bizonyos V-beli párokból áll. Rajzon szemléltetve a csúcsokat pontokkal ábrázoljuk, az éleknek megfelelő párok tagjait pedig vonallal összekötjük. Az átfogalmazás után az eredeti feladat egy klasszikus gráfelméleti probléma algoritmikus változatához vezet.

Definíció (színezés, kromatikus szám):
A G gráf k-színezésén egy $f : V \to \{1, \ldots, k\}$ leképezést értünk, melyre ha $(v, w) \in E$, akkor $f(v) \neq f(w)$. A G kromatikus száma a legkisebb k érték, melyre van G-nek k-színezése. A G kromatikus számának a jele $\chi(G)$.

A lámpák ütemezésének problémája tehát egy gráf kromatikus számának, illetve ennek megfelelő színezésnek a meghatározásához vezetett. Ezzel az úgynevezett *színezési feladattal* később még találkozni fogunk.

Az $\mathcal{A}$ leképezéshez tartozó modelltől azt is elvárjuk, hogy elemei jól ábrázolhatók legyenek számítógépes adatszerkezetekkel. A gráfokra ez teljesül. Egyik természetes ábrázolási módjuk az adjacencia mátrixszal való megadás. Ha a gráf V csúcshalmazának n eleme van, akkor az A adjacencia mátrixa egy n-szer n-es mátrix. Legyen az egyszerűség kedvéért $V = \{1, \ldots, n\}$. Ekkor

$$A[i, j] = \begin{cases} 0 & \text{ha } (i, j) \notin E \\ 1 & \text{ha } (i, j) \in E. \end{cases}$$

Az A mátrixot tekinthetjük kétdimenziós $A[1 : n, 1 : n]$ egész, vagy Boole típusú tömbnek. Modellünk fogalmai ezen az úton könnyen és hajlékonyan ábrázolhatók számítógépes adattípusokkal.

1.1.2. Arthur király civilizációs törekvései

Arthur király fényes udvarában 150 lovag és 150 udvarhölgy él. A király, aki közismert civilizációs erőfeszítéseiről, elhatározza, hogy megházasítja jó lovagjait és szép udvarhölgyeit. Mindezt persze emberségesen szeretné tenni. Csak olyan párok egybekelését akarja, amelyek tagjai kölcsönösen vonzalmat éreznek egymás iránt. Hogyan fogjon hozzá? Természetesen pártfogójához, a nagyhatalmú varázslóhoz, Merlinhez fordul. Merlin rögvest felismeri, hogy itt is bináris szimmetrikus viszonyok ábrázolásáról van szó. Nagy darab pergament vesz elő, és nekilát gráfot rajzolni. A pergamen egyik felén levő U ponthalmaz az udvarhölgyeket jelenti. A másik oldalra kerül a 150 pontú L halmaz; ez pedig a lovagokat ábrázolja. Az l lovagnak megfelelő pont és az u udvarhölgy pontja közé élet rajzol, ha l és u kölcsönösen kedvelik egymást. A modell, amit a mágus használ, a *páros gráf*. Ebben a pontok V halmaza két egymást nem metsző rész egyesítése; él csak a két rész között futhat.

Definíció: *A $G = (V; E)$ gráf egy páros (kétrészes) gráf, ha a V csúcshalmaz felbontható két nem üres L és U részre úgy, hogy $L \cap U = \emptyset$, $L \cup U = V$, és ha $(v_1, v_2) \in E$, akkor a v_1 és v_2 csúcsok egyike U-ban, a másika pedig L-ben van.*

A királyi parancs teljesítéséhez Merlinnek élek egy olyan rendszerét kell kiválasztania a gráf éleiből, hogy a kiválasztott élek közül a gráf minden pontjához pontosan egy csatlakozzon. A kiválasztott élek felelnek meg a tervezett házasságoknak. A gráfelmélet nyelvén *teljes párosítást* kell keresnie.

Definíció: *A* $G = (L, U; E)$, $\mid L \mid = \mid U \mid = n$ *kétrészes gráf éleinek egy* $S \subseteq E$ *részhalmaza teljes párosítás, ha az* $(L, U; S)$ *gráfban minden pontból pontosan 1 él indul ki.*

Merlinnek tehát a pergamenre rajzolt gráf egy teljes párosítását kellene megadnia. Ha a legenda fővátozatát fogadjuk el hitelesnek (a Sir Thomas Malory által írt *La Morte d'Arthur*), akkor arra jutunk, hogy a feladatnak nincs megoldása. Ismeretes ugyanis, hogy Arthur és Sir Lancelot is csak és kizárólag Guinevere királyné iránt érez vonzalmat. Hogyan járuljon mármost Merlin a nagyúr elé, aki nem örül túlságosan, ha parancsát nem teljesítik? A király az Excalibur után kapkodó lobbanékonyságán kívül éles elméjéről is nevezetes. Merlin így megmentheti fejét. Annyit kell tennie, hogy megmutatja a rajz megfelelő részletét (amiből kitűnik, hogy Arthur és Lancelot is csak Guinevere-rel van összekötve) a királynak, aki a bizonyíték hatására békésen eláll tervétől.

Általánosabban fogalmazva ugyanez lenne a helyzet, ha tetszőleges k esetén van k lovag úgy, hogy a kedvenceik kevesebb, mint k udvarhölgy közül kerülnek ki. A megfelelő fogalom a gráfok világából a König-akadály. Legyen $G = (L, U; E)$ egy kétrészes gráf, $\mid L \mid = \mid U \mid$. A nem üres $X \subseteq L$ csúcshalmaz egy *König-akadály*, ha van olyan $Y \subseteq U$, $\mid Y \mid < \mid X \mid$, hogy X-ből minden él Y-ba megy. Igazolható, hogy pontosan akkor *nincs* G-ben teljes párosítás, ha van benne König-akadály. Ezt tudva tovább pontosíthatjuk a feladatot: keressünk G-ben teljes párosítást, vagy egy König-akadályt, ha nem létezik teljes párosítás. Az utóbbi esetben világos bizonyítékot kapunk arra, hogy az eredeti feladatnak nincs megoldása. Az $\mathcal{A}$ leképezés ezúttal is segített tisztábbá tenni az eredeti feladatot. A modellel – jelen esetben a gráfokkal – kapcsolatos ismeretek komoly segítséget jelentettek a feladat értelmes és pontos megfogalmazásában.

Feladat: Arthur király kifogyhatatlan az ötletekből, ha a kulturált viselkedést kell előmozdítani Camelot várában. Legutóbb például feltalálta a késsel-villával való étkezés misztériumát. Azt szeretné, ha lovagjai ezentúl nem puszta kézzel nyúlnának a falatokhoz a Kerek Asztal melletti nagy lakomákon. Félő azonban, hogy a nyers modorú bajnokok eme szúró-vágó eszközök birtokában megpróbálják kiegyenlíteni számláikat az asztalszomszédjaikkal. Az ilyen csetepaték elkerüléséhez úgy kellene leültetni a lovagokat, hogy az asztalszomszédok között ne feszüljön ellenséges érzelem. Merlin feladata ezúttal egy olyan, a Kerek Asztal körüli ülésrend készítése, mely szerint senki sem kerül haragosa mellé. Ismert, hogy kik

nem állnak hadilábon egymással (ismét egy bináris szimmetrikus relációval leírható kapcsolat). Próbáljuk gráfok segítségével megfogalmazni a problémát. Milyen objektum keresésére vezet a feladat?

1.2. Algoritmusok

Két példát mutatunk ezután a $\mathcal{B}$ lépésre. Itt már kellően absztrakt, ugyanakkor gépközeli modellben megfogalmazott algoritmikus problémák megoldásáról lesz szó. Először egy irányított gráfokról szóló feladatot veszünk górcső alá.

1.2.1. Szuperforrás keresése

Az előző feladatok irányítatlan gráfokhoz vezettek. Nem tettünk különbséget az (u, v) és a (v, u) pár között. Ha az egyik E-be tartozott, akkor a másik is. Az irányított gráfok abban különböznek ettől, hogy az (u, v) él megléte nem feltétlenül jelenti a (v, u) él meglétét a gráfban. Ennek megfelelően az A adjacencia mátrix nem feltétlenül szimmetrikus, mint az eddigi példákban. Ha egy ilyen gráfot rajzolunk le, akkor az $(u, v) \in E$ élet ábrázoló vonalra v felé mutató nyilat teszünk. Most egy olyan feladattal fogunk foglalkozni, ami már kellően pontos megfogalmazásban áll rendelkezésünkre.

Definíció: *A G irányított gráf $s \in V$ csúcsa szuperforrás, ha minden s-től különböző $y \in V$ csúcs esetén teljesül, hogy $(s, y) \in E$ és $(y, s) \notin E$.*

A szuperforrás olyan s csúcs, amiből a gráf minden más csúcsába él vezet; az s-be pedig egyetlen más csúcsból sem megy él. Nyilvánvaló, hogy egy gráfban legfeljebb egy szuperforrás lehet.

A feladat a következő: *tegyük fel, hogy az A adjacencia mátrixával adott a $G = (V, E)$ irányított gráf, aminek a csúcshalmaza $V = \{1, \ldots, n\}$. Döntsük el, hogy van-e G-ben szuperforrás. Ha igen, találjuk meg.*

Egy megoldás rögtön kínálkozik. Sorra vesszük az $i \in V$ csúcsokat, mindegyikről megnézve, hogy szuperforrás-e. Az i csúcs vizsgálata során az i-be menő és az i-ből kiinduló élek hiányát illetve meglétét kell ellenőrizni. Ez lényegében az A mátrix i-edik sorának és i-edik oszlopának a végignézését jelenti.

Mindjárt felmerül a kérdés, hogy mennyire jó a kapott megoldás. Nem lehetséges-e ennél hatékonyabb, gyorsabb módszer? Hogy erre válaszolni tudjunk, mérnünk kell valahogy a módszereink hatékonyságát, sebességét. Meghatározhatnánk például a programunkban szereplő elemi utasítások végrehajtási idejét, amiből aztán becsülhető lenne a futási idő a különböző n értékekre. Az így kapott

mérőszám egyrészt kissé nehézkes: többféle elemi utasítás idejét kellene kimérni. Másfelől túlságosan gépfüggő is: más környezetben ezek az elemi idők egészen máshogyan alakulhatnak. Ezek a problémák azt sugallják, hogy keressünk valami egyszerű, gép- és környezetfüggetlen mérőszámot. Az algoritmus sebességét mérő szám legyen az *A mátrix megvizsgált elemeinek a száma.*

Első megoldásunkról kiderül, hogy nem valami fényes. Az A mátrix minden főátlón kívüli elemét megnézzük, mégpedig kétszer. Az összköltség $2(n^2 - n)$ ilyen vizsgálat.

A következő módszer azon az egyszerű észrevételen alapul, hogy ha $i \neq j$ esetén $(i, j) \in E$, akkor j nem lehet szuperforrás, $(i, j) \notin E$ pedig i-t zárja ki. Úgy is fogalmazhatunk, hogy $A[i, j]$ értékének ismeretében vagy i, vagy j biztosan kizárható mint lehetséges szuperforrás. Hogy a kettő közül melyik lesz a kizárt csúcs, az függ $A[i, j]$ értékétől, de egyiktől mindenképpen megszabadulhatunk. Ezzel az ötlettel gyorsan tudjuk szűkíteni a tudomásunk szerint még lehetséges szuperforrások körét.

$i := 1, j := n;$
while $i \neq j$ **do**
    if $A[i, j] = 1$ **then** $j := j - 1$

    else $i := i + 1;$
(Amikor ideérünk, már csak i lehet szuperforrás,*
*ezt ellenőrizzük a továbbiakban. *)*
for $k = 1$ **to** n **do**
    if $k \neq i$ és $(A[i, k] \neq 1$ vagy $A[k, i] \neq 0)$ **then return**(nincs szuperforrás)
return(i szuperforrás).

Az eljárás magától értetődő. Az első ciklus leszűkíti a keresést egyetlen jelöltre. A második ciklus ezt az egy szem jelöltet ellenőrzi. Nézzük most a módszerünk hatékonyságát! Evégből először felső becslést adunk a költségére. A **while**-ciklus végrehajtása során az A tömb $n - 1$ elemét vizsgáljuk meg. Utána már csak az i-edik sor és oszlop elemeit kell néznünk; ez további $2n - 2$ hely. Összesen tehát legfeljebb $3n - 3$ mátrix-elem megtekintését igényli az algoritmus.

Jelölje $T(n)$ a legjobb (leggyorsabb) algoritmus által megvizsgált mátrix-elemek számának maximumát az összes n pontú gráfra. Eddigi fejtegetésünk szerint $T(n) \leq 3n - 3$, hiszen *van* egy módszerünk, ami megoldja a feladatot ennél nem több lépésben. Hogy a módszerünket értékelhessük, alsó becslést kell adnunk $T(n)$-re. Ennek segítségével képet kaphatunk arról, hogy eljárásunk teljesítménye mennyire tér el a lehető legjobb módszerétől. Algoritmikus problémák elemzésekor általában igen nehéz érdemi alsó becsléseket nyerni; ezt valamennyire érthe-

tővé teszi az, hogy egy ilyen korlát az összes algoritmusra érvényes megállapítás. A szuperforrás-feladat ilyen szempontból szerencsésnek mondható. Nyilvánvaló, hogy $2n - 2 \leq T(n)$, hiszen ennyi elemet meg kell néznünk pusztán ahhoz, hogy egy adott csúcsról ellenőrizzük, hogy szuperforrás-e.

Ennél élesebb korlátot kaphatunk a következő észrevétellel: 1 él megkérdezése legfeljebb 1 csúcsot zár ki mint lehetséges szuperforrást. Ha ugyanis az $A[i,j]$ vizsgálata előtti tudásunk szerint i és j is lehet még szuperforrás, akkor $A[i,j] = 1$ esetén i, $A[i,j] = 0$ esetén pedig j továbbra is a jelöltek között marad. Az i-től és j-től különböző csúcsok megítélésén a válasz nem változtat.

Arra juthatunk mindebből, hogy a legjobb algoritmus első $n - 2$ kérdése után még legalább két csúcs – mondjuk i és j – lehet szuperforrás. Ez azt jelenti, hogy az A táblázat még nem nézett részének lehet olyan kitöltése, amely szerint i lesz szuperforrás, és olyan is, amelynél j a szuperforrás. Eddig összesen az A mátrix $n - 2$ elemét néztük meg. A megvizsgált elemek közül tehát valamelyik pontunk (mondjuk az i) sorába és oszlopába legfeljebb $(n-2)/2$ esik. Képzeljük el, hogy az „ellenség" úgy tölti ki az A még nem nézett elemeit, hogy i legyen a szuperforrás. Ez is egy lehetséges bemenete (inputja) a feladatnak; a legjobb algoritmusnak erre is működnie kell. Hogy az i szuperforrás-tulajdonságát igazolja, $A[i,i]$ kivételével ismernie kell az i-edik sor és oszlop értékeit. Ez még legalább $2n - 2 - (n-2)/2$ kérdést jelent. Innen kapjuk, hogy

$$T(n) \geq 2n - 2 - (n-2)/2 + n - 2 = 3n - 4 - (n-2)/2.$$

Az érvelés tanúsága szerint legfeljebb $n/2$ kérdés erejéig megközelítettük az optimumot. Elmondhatjuk, hogy gyors és egyszerű algoritmusunk van a feladat megoldására. Az első módszerhez képest a javulás drámai; például ha $n = 500$, akkor az A mátrix 250 000 eleméből legfeljebb csak 1497-et vizsgálunk meg. Megjegyezzük, hogy ennél jobb alsó korlát is adható; ebből kiderül, hogy a módszer nagyon közel van az optimálishoz.

Feladat: Mutassuk meg, hogy $T(n) \geq 3n - 3 - \log_2 n$. (Az ellenség-módszer alkalmazható. Az A mátrix bizonyos elemeinek megkérdezése utáni helyzet *potenciálja* legyen $\sum_i 2^{k_i}$; az összegben azok az i csúcsok szerepelnek, amelyek az adott helyzetbeli ismeretek szerint még lehetnek szuperforrások, k_i pedig az A i-edik sorából és oszlopából már ismert értékek száma. Az ellenség stratégiája: ha lehet, úgy válaszoljon az algoritmus kérdésére, hogy a szuperforrás-jelöltek ne fogyjanak; ha ezt nem teheti, akkor úgy válaszoljon, hogy a potenciál ne nőjön.)

A feladat érdekessége, hogy megoldható volt úgy is, hogy az A adjacencia mátrix n^2 elemének csak egy töredékét néztük meg. Ez a jelenség elég ritka a természetesen felmerülő gráftulajdonságok körében. Legtöbbször az egész mátrixot végig kell néznünk a megoldás során.

Érdemes megfigyelni az alsó becslésnél alkalmazott *ellenség-módszert.* A feltételezett legjobb algoritmus bizonyos kérdéseire adaptívan válaszoltunk: az algoritmus korábbi lépéseitől függően határoztunk meg néhány $A[i,j]$ értéket. Így találtunk egy olyan bemenetet, amely a módszert (viszonylag) sok munkára kényszeríti.

1.2.2. Hosszú egészek párhuzamos összeadása

> *Ha [a kivonásnál] semmi sem marad, írj egy köröcskét, hogy a hely ne maradjon üresen. A köröcskének kell elfoglalnia ezt a helyet, mert másképp kevesebb hely lenne, és például a másodikat hinnénk elsőnek.*
> MOHAMED IBN MÚSZA (AL KHVARIZMI) a nulláról.

Itt ismét egy már pontosan megfogalmazott feladatról lesz szó, mégpedig olyanról, *amelyet mindenki régóta ismer. Tegyük fel, hogy adottak a decimálisan írt* $a_1a_2\ldots a_n$ és $b_1b_2\ldots b_n$ n-jegyű természetes számok. Számítsuk ki az összegük $e_0e_1e_2\ldots e_n$ decimális rendszerbeli alakját. Ez a legelső valamire való algoritmikus probléma, amivel az iskolában találkoztunk. Bizony, rengeteget gyakoroltuk a hindu–arab-algoritmust, amivel jobbról balra haladva kitölthetjük az alábbi táblázat alsó sorát.

$$\begin{array}{cccccc} & a_1 & a_2 & \ldots & a_{n-1} & a_n \\ + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\ \hline e_0 & e_1 & e_2 & \ldots & e_{n-1} & e_n \end{array}$$

Összesen $2n-1$ egyjegyű összeadás árán megkapjuk az e_i számjegyeket: mindegyik i-re kiszámoljuk a helyi a_i+b_i összeget (n elemi összeadás), és ehhez még hozzáadjuk az átvitel értékét ($n-1$ egyjegyű összeadás). A módszer gyors és praktikus. Tulajdonképpen a bemenet hosszával arányos lépésszámban, *lineáris időben* megkapjuk az eredményt. Hasonlót mondhatunk arra az esetre, amikor a műveletet számítógép segítségével végezzük. Ha n viszonylag kicsi, akkor a számaink egy-egy gépi szóban ábrázolhatók, és a feladat egyetlen gépi összeadással megoldható. Ha a számok nagyon hosszúak, akkor a hindu–arab-algoritmus mintájára járhatunk el. A különbség annyi, hogy az elemi lépéseket a gépünk szóhosszától függő méretű többjegyű számokkal végezzük. A műveletek száma így is arányos lesz a bemenet hosszával; az arányossági tényező a szóhossz függvénye. Az ősi módszer tehát hatékony megoldást nyújt mind a kézi, mind pedig a szokásos értelemben vett gépi számoláshoz.

Mit tehetünk akkor, ha a feladat megoldására több processzorunk (számítógépünk) van, amelyek egyidejűen, *párhuzamosan* dolgozhatnak? Tudjuk-e így lényegesen csökkenteni a számolás idejét? Némi kísérletezgetés után rájövünk, hogy a

fő gondot a teendők értelmes szétosztása jelenti. Úgy kellene megszervezni a munkát, hogy a processzorok dolgozhassanak anélkül, hogy túl sokat kelljen várniuk egymásra. Példaképpen nézzük a 843712 és 156288 számok összegét. Megtehetnénk, hogy minden helyiértékhez egy processzort rendelünk, amely elvégzi az ott levő jegyek összeadását. Az utolsót kivéve mindegyik 9-et kap eredményül. Utána várniuk kell a jobb felől érkező átvitelre. A bal szélső processzor csak az összes többi után fejezheti be a munkát; meg kell várnia, amíg a jobb szélről az 1-es átvitel végigballag a processzorokon. Az idő így ismét arányos lesz az összeadandók hosszával, ami azt jelenti, hogy nem értünk el jelentős gyorsulást.

A párhuzamos algoritmusok tervezésében tipikus az itt vázolt helyzet. A megoldáshoz vezető teendőket kell minél inkább párhuzamosítani, függetlenül végezhető részfeladatokra bontani. Vannak olyan feladatok, ahol ez szinte magától értetődő, mint mondjuk a kukoricatörésnél, ahol a munkások külön-külön sorban dolgozva, egymástól függetlenül tevékenykedhetnek. Az ilyen esetekben ideális gyorsulás érhető el; ha t munkás van, a szükséges idő a t-edrészére csökken. Más feladatoknál a párhuzamosítás sokkal nehezebb; úgy találjuk, hogy a processzorok között jelentős egymásra utaltság van. Az országúti kerékpáros csapatversenyt említhetjük példaként. Finoman összehangolt munkával négyen együtt gyorsabban legyűrik a távot, mint azt egy ember tenné. De szó sincs arról, hogy negyedére csökkenne a szükséges idő. Az utóbbi egy nehezebben párhuzamosítható feladat.

Itt most egy erőteljesen párhuzamos megoldást adunk hosszú egészek összeadására. Mint látni fogjuk, lesz egy kis huncutság a dologban. A szélvészgyors megoldás ára az, hogy az eredményt nem a megszokott formában kapjuk.

Definíció: *A* $\sum_{i=1}^{n} a_i 10^{n-i}$, $0 \leq a_i \leq 9$ *alakú számábrázolást* standard *ábrázolásnak nevezzük. A* $\sum_{i=1}^{n} a_i 10^{n-i}$, $-9 \leq a_i \leq 9$ *alakú számábrázolást pedig* kiegyensúlyozottnak *nevezzük.*

A standard ábrázolás tehát a szokásos decimális felírást jelenti, ahol a megengedett számjegyek a $0, 1, 2, \ldots, 9$. A kiegyensúlyozott ábrázolásban tizenkilenc *jegyet* használhatunk. Ezek: $-9, -8, \ldots, -1, 0, 1, \ldots, 8, 9$. Tudjuk, hogy a standard alakban való felírás egyértelmű. Az így ábrázolt számokról könnyű megállapítani, hogy melyik a nagyobb. A kiegyensúlyozott ábrázolás nem egyértelmű, az ilyen számok összehasonlítása elég körülményes. A többértelmű felírásnak van viszont egy olyan előnye, ami segíteni fog abban, hogy megállítsuk az átvitel hosszú futását az összeadásnál.

Állítás: *Tetszőleges* $-18 \leq a \leq 18$ *egésznek van olyan kiegyensúlyozott ábrázolása, melyben nincs* $+9$*-es és* -9*-es jegy, továbbá a 10-es helyiértékű jegy -1,0 vagy 1.*

Bizonyítás: Ha $a \neq 9$ és $a \neq -9$, akkor a standard alak megfelelő. A kimaradó két számnál a $9 = 10 - 1$ és $-9 = -10 + 1$ egyenlőségeket használhatjuk. A kiegyensúlyozott felírások ekkor $1 - 1$, illetve -11. □

Ezután ismertetjük *Avizienis algoritmusát* két n-jegyű kiegyensúlyozott felírású szám összeadására. Az $a_1 \dots a_n$ és $b_1 \dots b_n$ n-jegyű egészek $e_0 e_1 \dots e_n$ kiegyensúlyozott alakú összegét fogjuk megkapni. Az algoritmus n processzort használ, ezek közül az i-edik processzor az i helyiértékű jegyeken dolgozik. Az első lépésben párhuzamosan kiszámítják az $a_i + b_i$ összeg egy olyan (kétjegyű) $c_i d_i$ kiegyensúlyozott reprezentációját, amilyet az állítás szavatol: $a_i + b_i = c_i 10 + d_i$, ahol $c_i \in \{-1, 0, 1\}$ és $-8 \leq d_i \leq 8$. A második lépésben csak az 1. és n. processzorok dolgoznak, megadva a végeredmény első és utolsó jegyét: $e_0 = c_1$, $e_n = d_n$. A harmadik, egyben utolsó lépésben az i-edik processzor $1 \leq i \leq n - 1$ egyetlen összeadással megkapja e_i-t: $e_i = d_i + c_{i+1}$. A módszer helyessége abból adódik, hogy az $i+2, \dots, n$ helyiértékeknél kapott eredmények nem befolyásolják az $i + 1$. helyről az i-edikre menő átvitel értékét. A d_{i+1} jegy ugyanis -8 és 8 közé esik, amit a jobbról jövő -1, 0, vagy 1 átvitel nem tud 9 fölé, vagy -9 alá vinni. Az első lépésben számított c_{i+1} érték tényleg a helyes átvitel, amit az i. helyen figyelembe kell vennünk. A számítás menete így szemléltethető:

	a_1	...	a_{n-1}	a_n
	b_1	...	b_{n-1}	b_n
1. lépés:	$c_1 d_1 := a_1 + b_1$	...	$c_{n-1} d_{n-1} := a_{n-1} + b_{n-1}$	$c_n d_n := a_n + b_n$
2. lépés:	$e_0 := c_1$			$e_n := d_n$
3. lépés:	$e_1 := d_1 + c_2$	...	$e_{n-1} := d_{n-1} + c_n$	

Nézzük meg, hogy fest ez egy konkrét példán. A korábban problémásnak minősült 843712 és 156288 számokon követjük a módszer lépéseit. Csak a kapott részeredményeket tüntetjük fel.

		8	4	3	7	1	2
		1	5	6	2	8	8
1. lépés:		$1-1$	$1-1$	$1-1$	$1-1$	$1-1$	10
2. lépés:	1						0
3. lépés:		0	0	0	0	0	
Összeg:	1	0	0	0	0	0	0

Avizienis algoritmusával tehát konstans párhuzamos időben megkaptuk az eredmény egy kiegyensúlyozott reprezentációját. Ezzel a kukoricatörésnél tapasztalthoz hasonló ideális felgyorsítást sikerült elérni.

Egy n-jegyű kiegyensúlyozott szám standard alakra hozására csak viszonylag lassúnak mondható, $c \log n$ párhuzamos lépést igénylő módszer ismert. Ezt

használva $O(\log n)$ párhuzamos idejű módszer adódik *két* n-jegyű egész standard felírású összegének a meghatározására. Avizienis módszere különösen akkor hasznos, ha *sok* összeadást kell elvégeznünk egyszerre. Erre a helyzetre értelmes példát kínál a szorzás.

Feladat: Mutassuk meg, hogy két n-jegyű egész szorzása elvégezhető $O(\log n)$ párhuzamos időben. (Az iskolában tanult szorzóalgoritmus összeadásait Avizienis módszerével végezhetjük el; csak a végeredményt írjuk át standard formába.)

A módszer kulcsa a 9-es nélküli felírhatóságról szóló állítás. Ez kézenfekvően általánosítható tetszőleges $k > 2$ alapú számrendszerre; ekkor a 9-es szerepét $k-1$ játssza. A gyakorlatban a négyes számrendszert használják, mert a gépeken szokásos kettes számrendszer és a négyes számrendszer közötti átalakítások gyorsak.

Feladat: Adjunk konstans párhuzamos idejű módszert kettes számrendszerben adott számok négyes számrendszerbeli átírására, és a fordított irányú átalakításra.

A hosszú egészek összeadásának feladatával azt is szerettük volna érzékeltetni, hogy a megoldás hatékonyságáról való képünk függ attól is, hogy milyen számítógépes eszközök állnak a rendelkezésünkre. A megoldás keresése során ezért figyelemmel kell lennünk erre a háttérre. Úgy is fogalmazhatunk, hogy az eszköz-háttér alapvető jellemzői (mint pl. a processzorok száma) részét képezik a modellnek, amivel dolgozunk.