34. NP-teljesség

Az eddig vizsgált algoritmusok csaknem valamennyien ***polinomiális idejűek,*** azaz n méretű bemeneten futási idejük a legrosszabb esetben is $O(n^k)$, valamely k konstanssal. Természetes a kérdés: vajon *minden* probléma megoldható-e polinomiális időben. A válasz természetesen: nem. Vannak problémák – mint például Turing híres megállási problémája –, melyek egyáltalán nem oldhatók meg számítógéppel, bármennyi idő áll is rendelkezésre. Vannak olyanok is, melyek megoldhatók ugyan, de nem polinomiális időben. A polinomiális idejű algoritmussal megoldható problémákat általában könnyűnek tekintjük, a szuperpolinomiális időt igénylőket pedig nehéznek.

E fejezet témája egy rendkívül érdekes problémaosztály – az úgynevezett NP-teljes problémák osztálya. NP-teljes problémára mind ez idáig senki sem adott polinomiális algoritmust, mint ahogy szuperpolinomiális alsó becslést sem. A P $\neq$ NP sejtés felvetése, azaz 1971 óta az elméleti számítógéptudomány egyik legmélyebb, legnehezebb kutatási területe.

Az NP-teljes problémák egyik különösen kellemetlen tulajdonsága, hogy számosan közülük ránézésre igen hasonlóak olyan problémákhoz, melyekre létezik polinomiális algoritmus. A következő problémapárok mindegyikében az egyik probléma polinomiális időben megoldható, míg a másik NP-teljes, annak ellenére, hogy a köztük lévő különbség csekélynek tűnik.

Legrövidebb és leghosszabb utak: A 24. fejezetben láttuk, hogy még negatív élsúlyok mellett is meg tudjuk találni az egy adott pontból a többi pontba menő *legrövidebb* utakat egy irányított $G = (V, E)$ gráfban $O(VE)$ idő alatt. A két adott pont közötti egyik *leghosszabb* út megtalálása viszont nehéz. Már annak eldöntése is NP-teljes, hogy létezik-e egy gráfban egy adott k számú élnél többet tartalmazó egyszerű út.

Euler- és Hamilton-körök: egy összefüggő, irányított $G = (V, E)$ gráf ***Euler-körének*** nevezünk egy körsétát, ha G minden élén pontosan egyszer megy végig, egy csúcson azonban többször is átmehet. A 22-3. feladatnál láttuk, hogy annak eldöntése, hogy egy adott irányított gráf tartalmaz-e Euler-kört, elvégezhető $O(E)$ időben, sőt $O(E)$ idő alatt meg is találhatunk egy Euler-kört (ha van). Egy irányított $G = (V, E)$ gráf ***Hamilton-körének*** nevezünk egy egyszerű kört, ha a V-ben szereplő valamennyi csúcsot tartalmazza. Annak eldöntése, hogy egy adott gráf tartalmaz-e Hamilton-kört, NP-teljes. (Később bebizonyítjuk, hogy annak eldöntése, hogy egy *irányítatlan* gráf tartalmaz-e Hamilton-kört, NP-teljes.)

2-CNF kielégíthetőség és 3-CNF kielégíthetőség: egy Boole-formula 0 vagy 1 értékű változókból, egy- vagy kétváltozós Boole-függvényekből, mint például $\wedge$ (ÉS), $\vee$ (VAGY), $\neg$ (NEM) és zárójelekből áll. Egy Boole-formula ***kielégíthető,*** ha létezik a változóknak olyan behelyettesítése, amelyre a formula értéke 1. A most következő fogalmakat később pontosan is definiáljuk, egyelőre érjük be annyival, hogy egy Boole-formula ***k-konjunktív normálformájú*** vagy k-CNF (conjunctive normal form), ha olyan zárójeles tagok ÉS-ekkel való összekapcsolásából áll, melyek pontosan k változót vagy negált változót tartalmaznak. Például az $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$ Boole-formula 2-konjunktív normálformájú (és kielégíthető: az $x_1 = 1, x_2 = 0, x_3 = 1$ behelyettesítésre az értéke 1). Annak eldöntésére, hogy egy 2-CNF kielégíthető-e, létezik polinomiális idejű algoritmus, a 3-CNF formulák kielégíthetőségének kérdése viszont – amint azt látni is fogjuk – NP-teljes.

A P és NP osztályok, NP-teljesség

E fejezetben három problémaosztállyal fogunk foglalkozni: a P, NP és NPC osztályokkal. Informálisan leírjuk őket most is, a pontos definíciót azonban csak később adjuk meg.

A P osztály a polinom időben megoldható problémákból áll, azaz olyan problémákból, melyekhez létezik olyan k konstans, hogy a probléma n hosszú bemenet esetén $O(n^k)$ idő alatt megoldható. Az eddigi fejezetekben vizsgált problémák túlnyomó többsége P-be tartozik.

Az NP osztály olyan problémákból áll, amelyek polinom időben ellenőrizhetők. Ez valami olyasmit jelent, hogy ha valaki egy „bizonyítékot” adna nekünk a megoldásról, akkor annak helyességét polinomiális időben le tudnánk ellenőrizni. Például a Hamilton-kör probléma esetén egy $G = (V, E)$ gráfhoz tartozó bizonyítéknak megfelelne egy $\langle v_1, v_2, v_3, \ldots, v_{|V|}\rangle$ különböző V-beli csúcsokból álló sorozat. Könnyen ellenőrizhető polinomiális időben, hogy a csúcsok valóban különbözők-e, továbbá az is, hogy $(v_i, v_{i+1}) \in E$ minden $i = 1, 2, 3, \ldots, |V| - 1$-re és $(v_{|V|}, v_1) \in E$ teljesül-e. Másik példaként említhetjük a 3-CNF kielégíthetőséget, itt a változók egy behelyettesítése a bizonyíték, melyről valóban ellenőrizhető polinom időben, hogy csakugyan kielégíti-e az adott Boole-formulát.

Bármely P-beli probléma az NP osztályba is beletartozik, hiszen a P-beli problémákat meg tudjuk oldani polinomiális időben, még bizonyíték nélkül is. Ezt az elképzelést később pontosan leírjuk, addig kénytelenek vagyunk elhinni, hogy $P \subseteq NP$. Nyitott kérdés, hogy P valódi részhalmaza-e NP-nek.

Az ***NP-teljes*** problémák osztályát NPC-vel jelöljük. Ide azok a problémák tartoznak, melyek „legalább olyan nehezek”, mint bármely NP-beli probléma. Hogy a „legalább olyan nehéz” mit is jelent, azt később pontosan definiálni fogjuk. Addig is kimondjuk bizonyítás nélkül, hogy ha az NP-teljes problémák közül *akár csak egy is* megoldható polinomiális időben, akkor *minden* NP-beli probléma megoldható polinomiális időben. A matematikusok nagy része úgy gondolja, hogy az NP-teljes problémák nem polinomiálisak. Napjainkig sokféle NP-teljes problémát vizsgáltak, a polinomiális idejű algoritmus irányába történő legcsekélyebb előrehaladás nélkül, így igencsak meglepő lenne, ha kiderülne, hogy ezen problémák mindegyikére létezik polinomiális megoldás. Tekintetbe véve mindazonáltal azt is, hogy a $P \neq NP$ állítás bizonyításának szentelt jelentős erőfeszítések sem hoztak eredményt, nem zárható ki annak lehetősége, hogy az NP-teljes problémák mégiscsak megoldhatók polinom időben.

Aki jó algoritmusokat szeretne tervezni, annak feltétlenül értenie kell az NP-teljesség elméletének alapjait. Ha például valaki mérnöki munkája során NP-teljes problémával találkozik, többnyire az a legjobb, ha alkalmas közelítő algoritmust (lásd 35. fejezet) próbál találni, és nem vesződik a gyors, pontos megoldás kitalálásával. Ezen túlmenően igen sok érdekes és természetesen felmerülő probléma, amely első ránézésre nem tűnik nehezebbnek, mint a keresés, a rendezés vagy a hálózati folyamok, NP-teljesnek bizonyul. Ezért fontos, hogy közelebbről megismerkedjünk ezzel a figyelemre méltó problémaosztállyal.

Hogyan mutatjuk meg, hogy egy probléma NP-teljes?

Azok a módszerek, melyeket egy adott probléma NP-teljességének bizonyítására használunk, lényegesen különböznek az e könyvben algoritmusok tervezésére és elemzésére használatos technikák legnagyobb részétől. Ennek alapvetően az az oka, hogy ilyenkor azt mutatjuk meg, hogy egy probléma mennyire nehéz (vagy legalábbis mennyire nehéznek gondoljuk), nem pedig azt, hogy mennyire könnyű. Nem azt próbáljuk meg bebizonyítani, hogy létezik hatékony algoritmus, hanem éppen ellenkezőleg: azt, hogy valószínűleg egyáltalán nem létezik hatékony algoritmus. Ilyeténképpen az NP-teljességi bizonyítások leginkább arra a bizonyításra hasonlítanak, melyet a 8.1. alfejezetben láttunk az összehasonlító rendezések lépésszámának $\Omega(n \lg n)$-es alsó becslésére, bár az ott használt, döntési fákon alapuló módszer itt nem fordul elő.

Az NP-teljességi bizonyításoknak három fő építőköve van.

Döntési és optimalizálási problémák

Az érdeklődésünk középpontjában álló problémák jelentős része ***optimalizálási probléma,*** ahol minden megengedett megoldáshoz egy érték tartozik, feladatunk pedig olyan megengedett megoldás megtalálása, amelyhez a legjobb érték van rendelve. Például a LEGRÖVIDEBB-ÚT-nak nevezett probléma esetében adott egy G irányítatlan gráf és az u, v csúcsok, feladatunk pedig olyan u és v közötti út megtalálása, melynek a lehető legkevesebb éle van. (Más szavakkal LEGRÖVIDEBB-ÚT a két adott pont közti legrövidebb út probléma egy súlyozatlan, irányítatlan gráfban.) Az NP-teljesség fogalmát azonban nem optimalizálási problémákra alkalmazzuk közvetlenül, hanem úgynevezett ***döntési problémákra,*** ahol a válasz egyszerűen „igen” vagy „nem” (formálisabban „1” vagy „0”).

Bár problémák NP-teljességének bizonyításakor tevékenységünket a döntési problémák birodalmára kell korlátoznunk, valójában nyilatkozhatunk optimalizálási problémákról is. Létezik ugyanis egy rendkívül hasznos összefüggés az optimalizálási és a döntési problémák között. Egy adott optimalizálási problémának megfeleltethetünk egy döntési problémát oly módon, hogy az optimalizálandó értékre egy korlátot állapítunk meg. A LEGRÖVIDEBB-ÚT esetében például a megfelelő döntési probléma, melyet ÚT-nak fogunk nevezni, a következő: adott egy G irányítatlan gráf és az u, v csúcsok, továbbá egy k egész szám, döntsük el, hogy létezik-e u és v között olyan út, amely legfeljebb k élből áll.

Az optimalizálási problémák és a hozzájuk rendelt döntési problémák között fennálló kapcsolat nagy segítségünkre van abban, hogy megmutassuk, az adott optimalizálási probléma valóban „nehéz”. Ennek az az oka, hogy a döntési probléma bizonyos értelemben „könnyebb”, vagy legalábbis „nem nehezebb”, mint az optimalizálási probléma, amihez tartozik. Például az ÚT problémát könnyűszerrel meg tudjuk oldani, ha a LEGRÖVIDEBB-ÚT probléma megoldását már ismerjük: egyszerűen össze kell hasonlítanunk a legrövidebb út hosszát az ÚT döntési problémában szereplő k paraméterrel. Más szóval, ha egy optimalizálási probléma könnyű, akkor a hozzá tartozó döntési probléma is könnyű. Az NP-teljesség elméletéhez jobban illően megfogalmazva: ha bizonyítékunk van rá, hogy egy döntési probléma nehéz, akkor arra is van bizonyítékunk, hogy az az optimalizálási probléma, amelyhez tartozik, szintén nehéz. Ily módon, bár az NP-teljesség elmélete csak döntési problémákkal foglalkozik, gyakran alkalmazható optimalizálási problémákra is.

Visszavezetések

Az imént látott módszer annak megmutatására, hogy egy probléma nem nehezebb egy másiknál, alkalmazható akkor is, ha mindkét probléma döntési. Ezt az ötletet csaknem minden NP-teljességi bizonyításban használni fogjuk, a következőképpen. Tekintsünk egy A döntési problémát, amelyet szeretnénk polinom időben megoldani. Egy adott probléma bemenetét a probléma egy ***esetének*** nevezzük; az ÚT probléma egy esete például egy adott G gráf, az adott u és v csúcsai G-nek és az adott k egész szám. Tegyük fel, hogy van egy másik döntési problémánk, nevezzük B-nek, melyről már tudjuk, hogyan lehet polinomiális időben megoldani. Végül tegyük fel, hogy rendelkezésünkre áll egy eljárás, melynek segítségével az A probléma egy α esetét átalakíthatjuk a B probléma egy β esetévé oly módon, hogy az alábbiak teljesüljenek:

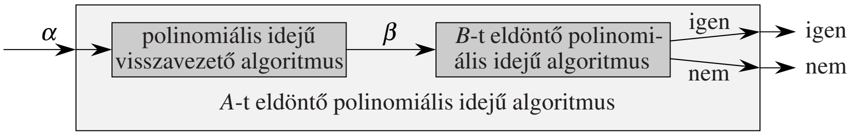
1. Az átalakítás polinomiális ideig tart.
2. A válaszok azonosak, azaz a válasz az α esetre akkor és csak akkor „igen”, ha a válasz a β esetre „igen”.

Az ilyen eljárások, melyeket polinomiális idejű ***visszavezető algoritmusnak*** hívunk, a 34.1. ábrán látható módon szolgáltatnak polinomiális idejű megoldást az A problémára.

1. Az A probléma adott α esetét a polinomiális idejű visszavezető algoritmus segítségével a B probléma egy β esetévé alakítjuk át.
2. Lefuttatjuk a B döntési problémát polinomiális időben megoldó algoritmust a β esetre.
3. A β esetre kapott választ adjuk meg az A probléma α esetéhez tartozó válaszként.

Amennyiben e három lépés mindegyike polinomiális időben megvalósítható, akkor a három lépés együttesen is polinomiális ideig tart, így az A problémát valóban el tudjuk dönteni polinomiális idő alatt. Más szóval, az A probléma megoldását „visszavezetve” a B probléma megoldására, B „könnyűségét” használjuk arra, hogy A „könnyűségét” bizonyítsuk.

Az NP-teljesség elmélete azonban nem arról szól, hogy hogyan láthatjuk be, hogy egy probléma könnyű, hanem éppen az ellenkezőjéről, ezért a polinomiális visszavezetés módszerét a fordított irányban fogjuk alkalmazni, problémák NP-teljességének bizonyítására. Gondoljuk tovább az ötletet és nézzük meg, hogyan láthatnánk be, hogy egy adott B problémára nem létezik polinomiális idejű algoritmus. Tegyük fel, hogy rendelkezésünkre áll egy A döntési probléma, melyről tudjuk, hogy nem oldható meg polinomiális időben. (Egyelőre ne foglalkozzunk azzal, hogyan is találhatnánk ilyen A problémát.) Tegyük fel továbbá, hogy ugyancsak rendelkezésünkre áll egy polinomiális idejű visszavezető algoritmus, amely A eseteit B eseteivé alakítja át. Ekkor egyszerű indirekt bizonyítást adhatunk arra, hogy B nem oldható meg polinom időben. Tegyük fel ugyanis, hogy B megoldására létezik polinomiális algoritmus. A 34.1. ábrán látható módszert alkalmazva A egy polinomiális idejű megoldását kapjuk, ami ellentmond annak a feltevésünknek, hogy A nem oldható meg polinom időben.



34.1. ábra. Polinomiális idejű visszavezető algoritmus használata az A döntési probléma megoldására, a B problémára ismert polinomiális algoritmus ismeretében. A egy α esetét átalakítjuk B egy β esetévé, megoldjuk B-t a β esetre, végül a β-ra kapott választ adjuk meg az A α esetéhez tartozó válaszként.

Az NP-teljesség esetében persze nem tehetjük fel, hogy egyáltalán nem létezik polinomiális megoldás az A problémára. A bizonyítási módszer azonban annyiban hasonló lesz, hogy B NP-teljességét úgy fogjuk belátni, hogy feltesszük, hogy A NP-teljes.

Kezdeti NP-teljes probléma

Minthogy a visszavezetési technika használata során nélkülözhetetlen egy NP-teljes probléma ahhoz, hogy egy másik probléma NP-teljességét bizonyítsuk, szükségünk van (legalább) egy „kezdeti” NP-teljes problémára. Erre a célra a Boole-hálózatok kielégíthetőségének problémáját fogjuk használni, melyben adott egy ÉS, VAGY és NEM kapukból álló Boole-hálózat, melyről el kell döntenünk, hogy léteznek-e olyan bemenetei, melyekre a hálózat kimenete 1. E probléma NP-teljességét a 34.3. alfejezetben bizonyítjuk.

A fejezet tartalma

E helyütt az NP-teljesség azon megközelítéseit tanulmányozzuk, melyek a legszorosabb kapcsolatban állnak az algoritmusok elemzésével. A 34.1. alfejezetben formalizáljuk a „probléma” fogalmát, és definiáljuk a polinomiális időben megoldható problémák P osztályát. Megvizsgáljuk továbbá, hogyan illeszkednek ezen fogalmak a formális nyelvek elméletének szerkezetébe. A 34.2. alfejezetben definiáljuk a polinomiális időben ellenőrizhető megoldású eldöntési problémák NP osztályát. Itt foglalkozunk először a P $\neq$ NP kérdéssel is.

A 34.3. alfejezetben megmutatjuk, hogyan vizsgálhatók a problémák közti összefüggések a ***polinomiális visszavezetés*** segítségével. Definiáljuk az NP-teljességet, és vázoljuk egy bizonyítását annak, hogy egy konkrét probléma – a Boole-hálózatok kielégíthetősége – NP-teljes. Miután találtunk egy NP-teljes problémát, a 34.4. alfejezetben azt mutatjuk meg, miképp bizonyíthatjuk újabb problémák NP-teljességét már jóval egyszerűbben, a polinomiális visszavezetés módszerével. A módszer illusztrációjaként megmutatjuk két formulakielégíthetőségi probléma NP-teljességét. NP-teljességi bizonyítások széles választéka található a 34.5. alfejezetben.

34.1. Polinomiális idő

Az NP-teljesség tanulmányozását a polinomiális időben megoldható probléma fogalmának definiálásával kezdjük. E problémákat – mint már említettük – többnyire jól kezelhetőknek tartjuk. Ennek alátámasztására három érvünk is van, ezek persze sokkal inkább filozófiai, mint matematikai jellegűek.

Először is, noha indokolt egy $\Theta(n^{100})$ lépést igénylő problémát nehezen kezelhetőnek tekinteni, valójában igen kevés gyakorlati probléma létezik, amelyhez ilyen magas fokú polinomiális lépésszám szükséges. A gyakorlatban előforduló polinomiális problémák ennél többnyire lényegesen gyorsabban megoldhatók. Emellett a tapasztalat azt mutatja, hogy egy probléma első polinomiális idejű megoldását gyakran követik újabb, hatékonyabb algoritmusok. Még ha igaz is, hogy egy problémára a leggyorsabb ismert algoritmus $\Theta(n^{100})$ lépést igényel, valószínű, hogy hamarosan felfedeznek majd egy lényegesen gyorsabb algoritmust.

Másodszor, számos ésszerű számítási modellre igaz, hogy egy probléma, amely polinom időben megoldható az egyikben, az a másikban is polinomiális. Például a – könyvünkben általában használatos – véletlen elérésű számítógéppel (RAM) polinom időben megoldható problémák osztálya azonos a Turing-géppel[1] polinom időben megoldható problémák osztályával. Sőt, ugyanezt az osztályt kapjuk akkor is, ha olyan párhuzamos számítógépet használunk, ahol a processzorok száma a bemenet méretével polinomiálisan nő.

Harmadszor, a polinom időben megoldható problémák osztályának szép zártsági tulajdonságai vannak, mivel a polinomok zártak az összeadásra, a szorzásra és a kompozícióra. Például, ha egy polinomiális algoritmus eredményét betápláljuk egy másik polinomiális algoritmusba, a kapott összetett eljárás is polinomiális lesz, csakúgy, mint az az algoritmus, amely polinomiális számú lépésén kívül konstansszor hív meg egy polinomiális szubrutint.

Absztrakt problémák

Ahhoz, hogy a polinom időben megoldható problémák osztályát (vagy egyáltalán bármiféle problémaosztályt) átlássunk, először is meg kell mondanunk, mit is értünk „problémán". A Q ***absztrakt problémát*** kétváltozós relációként definiáljuk a probléma ***eseteinek*** (bemeneteinek) I és a probléma ***megoldásainak*** S halmazán. A LEGRÖVIDEBB-ÚT probléma egy esete például egy gráf és annak két csúcsa által alkotott hármas, egy megoldás pedig csúcsok valamely sorozata, beleértve az üres sorozatot is, arra az esetre, ha két pont közt nincs út. Maga a LEGRÖVIDEBB-ÚT probléma az a reláció, mely a gráf és a két adott csúcs által alkotott hármashoz egy olyan csúcssorozatot rendel, melynek hossza a legrövidebb a két csúcsot összekötő sorozatok között. Minthogy a legrövidebb út nem egyértelmű, egy esethez több megoldás is tartozhat.

Az absztrakt probléma fenti definíciója jóval általánosabb, mint amire szükségünk van. Amint láttuk, az NP-teljesség elmélete csak ***döntési problémákkal*** foglalkozik, azaz olyanokkal, melyekre a megoldás „igen", vagy „nem". Egy absztrakt döntési problémát tekinthetünk olyan függvénynek, mely az I esethalmazt a $\{0, 1\}$ megoldáshalmazra képezi. Ilyen például a már látott ÚT probléma, melyet LEGRÖVIDEBB-ÚT módosításával kaptunk. Legyen $i = \langle G, u, v, k\rangle$ az ÚT döntési probléma egy esete. Ekkor $\text{ÚT}(i) = 1$ („igen"), ha az u és v közötti (egyik) legrövidebb út hossza legfeljebb k, és $\text{ÚT}(i) = 0$ („nem"), különben. Sok absztrakt probléma nem döntési, hanem ***optimalizálási probléma,*** amelyben valamely értéket minimalizálni vagy maximalizálni szeretnénk. Amint azt már láttuk, az optimalizálási problémák rendszerint könnyen átalakíthatók olyan döntési problémákká, melyek nem nehezebbek az eredetinél.

---

[1] A Turing-gép-modell alapos áttekintése megtalálható pl. Hopcroft és Ullmann [156] vagy Lewis és Papadimitriou [204] műveiben.

Kódolások

Ha számítógépes programmal szeretnénk absztrakt problémákat megoldani, az eseteket úgy kell megadnunk, hogy azt a program megértse. Egy absztrakt objektumokból álló S halmaz ***kódolása*** egy e leképezés S-ről a bináris sorozatokba.[2] Valamennyien jól ismerjük a természetes számok ($\mathbf{N} = \{0, 1, 2, 3, 4, \ldots\}$) bináris kódolását: $\{0, 1, 10, 11, 100, \ldots\}$. E kódolást használva például $e(17) = 10001$. Bárki, aki foglalkozott a számítógép karaktereinek ábrázolásával, otthonos az ASCII vagy az EBCDIC kódok valamelyikében. ASCII kódban például e (A)=1000001. Összetett objektumok is kódolhatók binárisan, az összetevők kódjainak kombinációjaként. Sokszögek, gráfok, függvények, rendezett párok, programok – valamennyien kódolhatók bináris sorozatként.

Egy számítógépes algoritmus, amely megold valamilyen absztrakt döntési problémát, ezek szerint a probléma eseteinek egy kódolását kapja bemenetként. Az olyan problémát, melynek esetei a bináris sorozatok, ***konkrét problémának*** hívjuk. Azt mondjuk, hogy egy algoritmus egy konkrét problémát $O(T(n))$ idő alatt ***megold,*** ha bármely n hosszúságú i esetre a megoldás $O(T(n))$ lépést igényel.[3] Egy konkrét probléma ***polinomiális időben megoldható*** (röviden: polinom időben megoldható), ha létezik algoritmus, ami $O(n^k)$ idő alatt megoldja, valamely k szám mellett.

Most már definiálhatjuk a **P** ***bonyolultsági osztályt***: P a polinom időben megoldható konkrét döntési problémák halmaza.

Kódolás segítségével absztrakt problémákat konkrét problémákká tudunk átalakítani. A Q absztrakt döntési problémát, amely az I esethalmazt $\{0, 1\}$-re képezi, az $e : I \to \{0, 1\}^*$ kódolással konkrét döntési problémává alakíthatjuk, amit $e(Q)$-val jelölünk.[4] Ha az absztrakt probléma egy i esetére $Q(i) \in \{0, 1\}$ a megoldás, akkor a kapott konkrét probléma $e(i)$ esetére is $Q(i)$ a megoldás. Természetesen előfordulhat, hogy bizonyos bináris sorozatok nem állnak elő egyetlen absztrakt eset képeként sem. A kényelem kedvéért megállapodunk abban, hogy az ilyen sorozatok képe a 0. Ily módon a konkrét probléma ugyanazon megoldásokat adja, mint az absztrakt, azokon a bináris sorozatokon, melyek az absztrakt probléma eseteinek felelnek meg.

Szeretnénk a polinomiális idejű megoldhatóság definícióját kiterjeszteni az absztrakt problémákra is, a kódolások felhasználásával, de azt is szeretnénk, hogy a definíció független legyen minden konkrét kódolástól. Ez azt jelentené, hogy a probléma megoldásának hatékonysága nem függene attól, hogy miképp kódoljuk. Sajnálatos módon ez egyáltalán nincs így. Például képzeljük el, hogy egy algoritmusnak a k természetes számot szeretnénk beadni, és tegyük fel, hogy az algoritmus futási ideje $\Theta(k)$. Ha k-t ***unárisan*** kódoljuk, azaz k darab egyesből álló sorozatként, akkor n hosszúságú bemenetre a futási idő $\Theta(n)$, ami persze polinomiális. Ha a lényegesen természetesebb bináris kódolást használjuk, akkor a bemenet hossza csak $n = [\log(k)] + 1$, így a futási idő: $\Theta(k) = \Theta(2^n)$, tehát a bemenet méretében exponenciális. Ez jól mutatja, hogy ugyanaz az algoritmus a kódolástól függően lehet polinomiális és szuperpolinomiális is.

Az absztrakt problémák kódolása tehát igen lényeges a polinomiális idejűség tanulmányozásában. Absztrakt problémák megoldásáról még csak nem is beszélhetünk, amíg nem rögzítünk egy kódolást. Mindamellett a gyakorlatban, ha kizárjuk az olyan „költséges”

---

[2]Bináris sorozatok helyett bármilyen véges, legalább 2 elemű halmaz elemeiből képzett sorozatok is megfelelnek.

[3]Feltesszük, hogy az algoritmus kimenete elkülönül a bemenettől. Minthogy a kimenet minden bitjének kiírása egy időegységet igényel, a kimenet mérete is $O(T(n))$.

[4]Amint azt hamarosan látni fogjuk, $\{0, 1\}^*$ a 0 és 1 szimbólumokból álló összes lehetséges sztring halmazát jelöli.

kódolásokat, mint például az unáris, a probléma konkrét kódolása nemigen befolyásolja a polinomialitás kérdését. Például az egész számok kettes helyett hármas számrendszerben való megadása nincs hatással a polinomiális megoldhatóságra, hiszen ezek a reprezentációk polinomiális időben átalakíthatók egymásba.

Azt mondjuk, hogy az $f : \{0,1\}^* \to \{0,1\}^*$ függvény ***polinom időben kiszámítható,*** ha létezik olyan A polinomiális algoritmus, amely tetszőleges $x \in \{0,1\}^*$ bemenetre $f(x)$-et adja eredményeként. Legyen I egy probléma eseteinek halmaza. Azt mondjuk, hogy az e_1 és e_2 kódolások ***polinomiálisan kapcsoltak,*** ha léteznek olyan f_{12} és f_{21} polinom időben kiszámítható függvények, hogy bármely $i \in I$ esetén $f_{12}(e_1(i)) = e_2(i)$ és $f_{21}(e_2(i)) = e_1(i)$,[5] vagyis az $e_2(i)$ kódolás polinomiális idejű algoritmussal kiszámítható $e_1(i)$-ből és fordítva. A következő lemma szerint ha egy absztrakt probléma e_1 és e_2 kódolásai polinomiálisan kapcsoltak, akkor mindegy, hogy melyiküket használjuk annak eldöntésére, hogy a probléma polinomiális időben megoldható-e.

34.1. lemma. *Legyen Q absztrakt döntési probléma az I esethalmazzal, legyenek továbbá e_1 és e_2 I-nek polinomiálisan kapcsolt kódolásai. Ekkor*

$$e_1(Q) \in P \iff e_2(Q) \in P.$$

Bizonyítás. Szimmetriaokokból nyilván elég azt bizonyítanunk, hogy ha $e_1(Q) \in P$, akkor $e_2(Q) \in P$. Tegyük fel tehát, hogy $e_1(Q)$ $O(n^k)$ időben megoldható, és hogy minden $i \in I$-re az $e_1(i)$ kódolás $O(n^c)$ időben kiszámítható az $e_2(i)$ kódolásból, ahol $n = |e_2(i)|$, k és c konstansok. $e_2(Q)$ megoldásához az $e_2(i)$ bemeneten először ki kell számítanunk $e_1(i)$-t, majd le kell futtatnunk az $e_1(Q)$-t megoldó algoritmust az $e_1(i)$ bemeneten. Mennyi ideig tart ez? Az átkódolás $O(n^c)$ lépés, így $|e_1(i)| = O(n^c)$, hiszen egy (soros) számítógép kimenete nem lehet hosszabb, mint a futási idő. Eszerint a teljes megoldás $O(|e_1(i)|^k) = O(n^{ck})$ lépést igényel, ami polinomiális, hiszen c és k konstans. ■

Láttuk, hogy míg a problémák kettő, illetve három elemű ábécé feletti kódolása nincs hatással „bonyolultságukra”, azaz polinomiális idejű megoldhatóságukra, addig pl. az unáris kódolás megváltoztathatja azt. Annak érdekében, hogy a tárgyalásmódot kódolásfüggetlenné tehessük, általában fel fogjuk tenni, hogy a problémák esetei valamely ésszerű, tömör formában vannak kódolva, hacsak mást nem mondunk. Pontosabban megfogalmazva: feltesszük, hogy az egész számok kódolása polinomiálisan kapcsolt bináris reprezentációjukkal, a véges halmazok kódolása pedig polinomiálisan kapcsolt azon kódolásukkal, melyet elemeik zárójelbe tett, vesszőkkel elválasztott felsorolásával kapunk. (Ilyen kódolás például az ASCII.) Ezen „szabványos” kódolás segítségével más matematikai objektumok, mint például vektorok, gráfok, formulák ésszerű kódolását is nyerhetjük. Objektumok szabványos kódolásának jelölésére a hegyes zárójelet használjuk, például $\langle G \rangle$-vel jelöljük a G gráf szabványos kódolását.

Amíg csak a szabványossal polinomiálisan kapcsolt kódokat használunk, beszélhetünk közvetlenül az absztrakt problémák „bonyolultságáról”, anélkül, hogy egy konkrét kódolást kiemelnénk. Éppen ezért általában feltesszük, hogy a problémák esetei bináris sorozatok formájában vannak kódolva, a szabványos kódolás szerint, mindaddig, amíg mást nem

[5]Technikai okokból azt is megköveteljük, hogy az f_{12} és f_{21} függvények „nem eseteket” „nem esetekbe” képezzenek. A ***nem eset*** az $e \in \{0,1\}^*$ sztring olyan dekódolása, amelyre nem létezik olyan i sztring, amelyre $f(i) = e$. Megköveteljük, hogy $f_{12}(x) = y$ teljesüljön az e_1 dekódolás minden x nem esetére, ahol y az e_2 valamelyik nem esete, továbbá hogy $f_{21}(x') = y'$ fennálljon e_2 minden x' nem esetére, ahol y' az e_1 valamely nem esete.

mondunk. Az absztrakt és a konkrét problémákat többnyire nem különböztetjük meg. Az olvasónak hasznos lesz időnként átgondolni az olyan problémák reprezentációját, melyek szabványos kódolása nem nyilvánvaló, és a kódolás befolyásolhatja a bonyolultságot.

Formális nyelvek

E ponton érdemes áttekinteni a formális nyelvek elméletének néhány alapvető definícióját. ***Ábécén*** szimbólumok egy véges halmazát értjük. ***Szavak*** az ábécé elemeiből képzett véges sorozatok, a ***nyelv*** pedig nem más, mint szavak egy tetszőleges halmaza. Például a $\Sigma = \{0, 1\}$ ábécé feletti $L = \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}$ nyelv a prímszámok bináris reprezentációjának nyelve. Az ***üres szót*** ε-nal jelöljük, az ***üres nyelvet*** $\emptyset$-zal, a Σ feletti összes szót tartalmazó nyelvet pedig Σ^*-gal. Ha például $\Sigma = \{0, 1\}$, akkor $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Bármely Σ feletti nyelv Σ^* részhalmaza.

A nyelveken igen sok művelet értelmezhető. Halmazelméleti műveletek, mint például az ***unió*** és a ***metszet*** azonnal következnek abból, hogy a nyelveket halmazokként definiáltuk. Az L nyelv ***komplementere*** $\overline{L} = \Sigma^* - L$. Az L_1 és L_2 nyelvek ***konkatenációja*** (egymás mellé írása, összefűzése) az

$$L = \{x_1x_2 : x_1 \in L_1,\ x_2 \in L_2\}$$

nyelv. Az L nyelv ***lezártja,*** vagy Kleene-féle csillaga az

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cdots \text{ nyelv, ahol } L^k = \underbrace{LL\ldots L}_{k}.$$

A formális nyelvek szempontjából tetszőleges Q döntési probléma esethalmaza egyszerűen Σ^*, ahol $\Sigma = \{0, 1\}$. Minthogy Q-t egyértelműen jellemzi az esetek azon részhalmaza, amire az 1 („igen”) választ kapjuk, Q-t tekinthetjük a következő, $\Sigma = \{0, 1\}$ feletti nyelvnek:

$$L = \{x \in \Sigma^* : Q(x) = 1\}.$$

Például az ÚT döntési problémához tartozó nyelv:

$$\begin{aligned} \text{ÚT} = \{\langle G, u, v, k\rangle :\ & G = (V, E) \text{ irányítatlan gráf,} \\ & u, v \in V, \\ & k \text{ nemnegatív egész,} \\ & \text{létezik út } G\text{-ben } u \text{ és } v \text{ között, ami legfeljebb } k \text{ hosszú}\}. \end{aligned}$$

(Ahol nem okozhat félreértést, ott a döntési problémákra és a hozzájuk tartozó nyelvekre ugyanazon a néven hivatkozunk.)

A formális nyelvek elmélete lehetővé teszi, hogy a döntési problémák és az őket megoldó algoritmusok közti összefüggéseket tömören kifejezzük. Azt mondjuk, hogy az A algoritmus ***elfogadja*** az $x \in \{0, 1\}^*$ szót, ha az x bemeneten A 1-et ad, röviden, ha $A(x) = 1$; ha $A(x) = 0$, akkor A ***elutasítja*** x-et. Az A algoritmus elfogadja az L nyelvet, ha L minden szavát elfogadja.

Elképzelhető, hogy az L nyelvet elfogadó A algoritmus nem utasít vissza egy x szót, annak ellenére, hogy $x \notin L$; például, ha az algoritmus végtelen ciklusba kerül. Azt mondjuk, hogy az A algoritmus ***eldönti*** az L nyelvet, ha az L-beli szavakat elfogadja, a többi szót pedig elutasítja. (Azaz, ha $L = \{x \in \{0, 1\}^* : A(x) = 1\}$ és $\overline{L} = \{x \in \{0, 1\}^* : A(x) = 0\}$.) Az A algoritmus ***polinom időben elfogadja*** az L nyelvet, ha bármely n hosszúságú $x \in L$ szót

$O(n^k)$ időben elfogad, valamely k szám mellett. Az A algoritmus ***polinom időben eldönti*** az L nyelvet, ha bármely n hosszúságú $x \in \{0, 1\}^*$ szót $O(n^k)$ időben elfogad, vagy elutasít, aszerint, hogy a szó L-beli, vagy sem (k most is konstans). Az L nyelv (polinom időben) elfogadható/eldönthető, ha létezik algoritmus, amely (polinom időben) elfogadja/eldönti.

Nézzük meg példaként az ÚT nyelvet, ez polinom időben elfogadható: először egy – szélességi keresésen alapuló – polinomiális algoritmussal kiszámítjuk az u és v közti (egyik) legrövidebb út hosszát, majd ezt összehasonlítjuk k-val. Ha a kapott szám legfeljebb k, az algoritmus kiadja az 1-et eredményként, ha nem, akkor fut tovább a végtelenségig. Ez az algoritmus nem dönti el az ÚT nyelvet, hiszen nem ad 0-t azon esetekben, amikor egy legrövidebb út hossza nagyobb, mint k. Az algoritmus utolsó lépésének csekély módosításával ez persze elérhető lenne. Léteznek azonban nyelvek – mint például a Turing-féle megállási problémához (halting problem) tartozó nyelv – melyek elfogadhatók ugyan, de nem eldönthetők.

A ***bonyolultsági osztályok*** informális definíciója: bonyolultsági osztálynak nevezzük nyelvek egy halmazát, ahol a halmazhoz tartozást az adott nyelvet eldöntő algoritmusok valamilyen ***bonyolultsági mértéke*** – mint például futási idő – határozza meg. A bonyolultsági osztályok tényleges definíciója némiképp technikaibb jellegű – az érdeklődő Olvasó megtalálhatja pl. Hartmanis és Stearns cikkében [140].

A megismert nyelvelméleti fogalmak segítségével megadhatjuk a P bonyolultsági osztály alternatív definícióját: P a $\{0, 1\}$ feletti, polinom időben eldönthető nyelvek halmaza. Sőt, P megegyezik a polinom időben elfogadható nyelvek halmazával is.

34.2. tétel. P = $\{L$: *L polinom időben elfogadható*$\}$.

Bizonyítás. Mint az a definícióból azonnal látható, a polinomiálisan eldönthető problémák a polinomiálisan elfogadható problémák halmazának részhalmazát képezik, így csak azt kell megmutatnunk, hogy ha létezik L-et elfogadó polinomiális algoritmus, akkor létezik olyan is, ami polinom időben eldönti L-et. Legyen L tetszőleges nyelv, amit az A polinomiális algoritmus elfogad. Egy klasszikus szimulációs trükk segítségével konstruálunk egy olyan A' algoritmust, ami polinom időben el is dönti L-et. Mivel A az n hosszúságú $x \in L$ bemenetet $O(n^k)$ (k konstans) időben elfogadja, létezik c konstans, hogy A x-et legfeljebb $T = cn^k$ lépésben fogadja el. Tetszőleges x bemenetre A' az első T lépésben szimulálja A működését. Ezután A' megvizsgálja, hogy A miképpen viselkedett. Ha A elfogadta x-et, akkor persze A' is elfogadja (azaz kiadja az 1-est végeredményként), ha viszont A nem fogadta el, akkor 0-t ad. Nyilvánvaló, hogy A' polinomiális, és az is, hogy eldönti L-et. ■

Meg kell jegyeznünk, hogy a fenti bizonyítás nem konstruktív. Egy adott $L \in$ P nyelvhez nem feltétlen ismerjük egy őt elfogadó algoritmus futási idejének korlátját. Mindazonáltal tudjuk, hogy ez a korlát létezik, és így persze az az A' algoritmus is, ami ezt a korlátot ellenőrzi. (Jóllehet ennek megadása sokszor cseppet sem egyszerű.)

Gyakorlatok

34.1-1. Definiáljuk a LEGHOSSZABB-ÚT optimalizálási problémát olyan relációként, ami egy gráfnak és két kijelölt pontjának a két pont közti (egyik) leghosszabb egyszerű út hosszát felelteti meg. Definiáljuk az ÚT′ döntési problémát a következőképp: ÚT′ = $\{\langle G, u, v, k\rangle : G = (V, E)$ irányítatlan gráf, $u, v \in V, \quad k \in \mathbf{N},$ létezik egyszerű út G-ben

u és v között, ami legalább k hosszú}. Mutassuk meg, hogy LEGHOSSZABB-ÚT $\in$ P pontosan akkor, ha ÚT$'$ $\in$ P.

34.1-2. Adjunk formális definíciót a következő problémára: találjuk meg egy irányítatlan gráf leghosszabb körét. Fogalmazzuk meg a megfelelő optimalizálási és az ahhoz tartozó döntési problémát, végül adjuk meg a döntési problémához tartozó nyelvet.

34.1-3. Adjuk meg az irányított gráfok bináris szavakként való kódolásainak egyikét, a szomszédsági mátrixok felhasználásával. Adjunk meg ilyen kódolást a szomszédsági listák segítségével is. Igazoljuk, hogy a két kódolás polinomiálisan kapcsolt.

34.1-4. Igaz-e, hogy a 0-1 hátizsák-problémára adható dinamikus programozási algoritmus (lásd 16.2-2. gyakorlat) polinomiális?

34.1-5. Mutassuk meg, hogy egy olyan (egyéb lépéseit tekintve polinomiális) algoritmus, mely konstansszor hív meg polinomiális szubrutinokat, polinomiális, míg ha polinomiális számú hívást engedünk meg, akkor az algoritmus exponenciális időt is igényelhet.

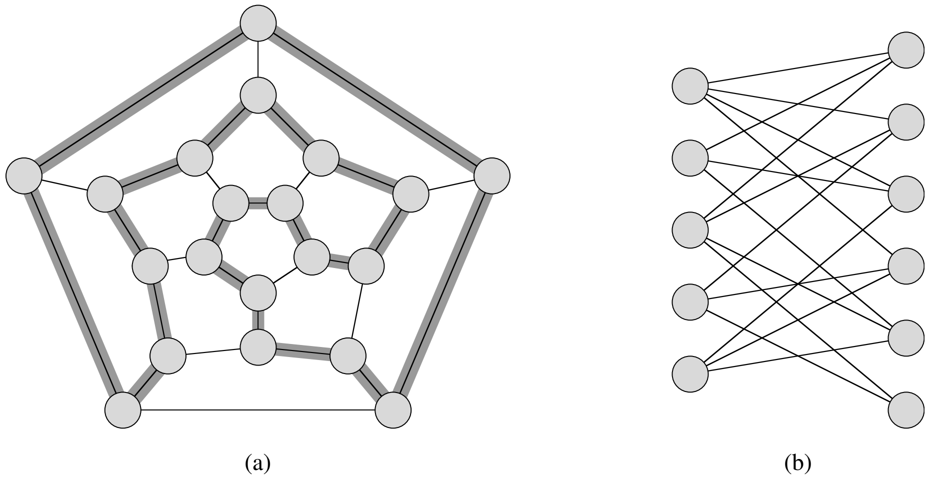
34.1-6. Igazoljuk, hogy a P osztály mint nyelvek egy halmaza, zárt az unióképzés, metszetképzés, konkatenáció, komplementálás és lezárás műveletekre.

34.2. Polinomiális idejű ellenőrzés

Ebben az alfejezetben olyan algoritmusokkal fogunk foglalkozni, amelyek azt „ellenőrzik”, hogy egy adott szó benne van-e egy adott nyelvben. Például tegyük fel, hogy az ÚT döntési probléma egy bizonyos (G, u, v, k) esetéhez ismerünk egy p utat is u-ból v-be. Könnyűszerrel ellenőrizhetjük, hogy p hossza legfeljebb k-e, és ha igen, akkor p-t egyfajta „tanúsítványnak” tekinthetjük, ami bizonyítja, hogy ÚT $((G, u, v, k)) = 1$. Az ÚT döntési probléma esetében ez a tanúsítvány nem tűnik igazán hasznosnak, hiszen ÚT P-ben van, sőt, lineáris időben megoldható, így a tanúsítvány ellenőrzése körülbelül ugyanannyi időt igényel, mint magának a problémának a megoldása. Az alábbiakban egy olyan problémával foglalkozunk, amelyre nem ismeretes polinomiális algoritmus, de egy adott tanúsítvány ellenőrzése könnyű.

Hamilton-körök

A Hamilton-kör keresésének problémáját irányítatlan gráfban már több, mint száz éve tanulmányozzák. Egy irányítatlan $G = (V, E)$ gráf ***Hamilton-körén*** olyan kört értünk, amely G minden pontján pontosan egyszer megy át. A Hamilton-kört tartalmazó gráfokat szokás ***hamiltoni,*** vagy Hamilton-gráfoknak nevezni, a Hamilton-kört nem tartalmazókat pedig – értelemszerűen – ***nem hamiltoni*** gráfoknak. Bondy és Murty [45] idézi W. R. Hamilton egy levelét, mely egy, a dodekaéderen (34.2(a) ábra) játszható matematikai játék leírását tartalmazza. Az egyik játékos kijelöl egy tetszőleges 5 hosszúságú utat, s a másiknak ezt kell az összes csúcsot tartalmazó körré kiegészítenie. Ebből is sejthető, hogy a dodekaédernek van Hamilton-köre, egy ilyen látható is a 34.2(a) ábrán. Természetesen nem minden gráf hamiltoni. Például, ha egy gráfban nincsen kör, akkor persze Hamilton-kör sincs. A 34.2(b) ábrán egy – köröket is tartalmazó – páros gráf látható, melynek páratlan sok csúcsa van. (A 34.2-2. gyakorlatban az Olvasónak kell belátnia, hogy az ilyen gráfok nem tartalmaznak Hamilton-kört.)



34.2. ábra. **(a)** Dodekaéder gráfreprezentációja; a vastagított élek Hamilton-kört alkotnak. **(b)** Páros gráf, páratlan számú csúccsal. Az ilyen gráfoknak soha nincs Hamilton-körük.

A ***Hamilton-kör probléma*** (HAM): „Van-e a G gráfnak Hamilton-köre?”. Formális nyelvként definiálva:

$$\text{HAM} = \{\langle G \rangle : G \text{ Hamilton-gráf}\}.$$

Hogyan lehetne algoritmussal eldönteni a HAM nyelvet? Egy adott $\langle G \rangle$ esetre egy lehetséges algoritmus az, ha felsoroljuk G összes csúcsának minden lehetséges permutációját, és mindegyikről megnézzük, hogy Hamilton-kör-e. Mennyi időt igényel ez az algoritmus? Ha a gráf „ésszerű”, szomszédsági mátrixos kódolását használjuk, a csúcsok számára $m = \Omega(\sqrt{n})$ adódik, ahol $n = |\langle G \rangle|$ a kódolás hossza. A csúcsoknak $m!$ lehetséges permutációja van, így a futási idő $\Omega(m!) = \Omega((\sqrt{n})!) = \Omega(2^{\sqrt{n}})$, ami nem $O(n^k)$ egyetlen k konstanssal sem. Ez a „naiv” algoritmus tehát nem polinomiális, sőt a 34.5. alfejezetben látni fogjuk, hogy a Hamilton-kör probléma NP-teljes.

Ellenőrző algoritmusok

Foglalkozzunk most az előbbinél kicsivel egyszerűbb problémával. Tegyük fel, hogy egy jó barátunk megsúgja nekünk, hogy egy adott G gráfban van Hamilton-kör, és felajánlja, hogy bizonyítékul megad egyet. Ezt a bizonyítékot meglehetősen könnyű ellenőrizni: egyszerűen megvizsgáljuk, hogy a megadott pontsorozat permutációja-e G csúcsainak, és hogy az egymás utáni pontok (meg persze az első és az utolsó) szomszédosak-e. Ez az eljárás kényelmesen végrehajtható $O(n^2)$ lépésben, ahol n a G gráf kódolásának hossza. Ezek szerint a Hamilton-kör létezésének effajta bizonyítéka polinom időben ellenőrizhető.

Ellenőrző algoritmusnak olyan két bemenetű algoritmust nevezünk, ahol az egyik bemenet megegyezik a döntési algoritmusoknál megszokottal, azaz a probléma egy esetének (bináris) kódolása, a másik egy bináris szó, amit ***tanúnak*** nevezünk. Azt mondjuk, hogy az A ellenőrző algoritmus ***bizonyítja*** az x szót, ha létezik olyan y tanú, hogy $A(x, y) = 1$. A bizonyítja az L nyelvet, ha minden szavát bizonyítja, és minden szó, amit bizonyít, L-ben van. Röviden:

$$L = \{x \in \{0,1\}^* : \text{létezik } y \in \{0,1\}^*, \text{ amelyre } A(x,y) = 1\}.$$

Például HAM esetén a tanúk pontsorozatok (kódolásai) voltak, az ismertetett ellenőrző eljárásról pedig jól látható, hogy megfelel a kritériumoknak. Ha egy gráf nem tartalmaz Hamilton-kört, akkor nincs olyan csúcssorozat, amelynek alapján az ellenőrző algoritmus hibásan következtetne, mivel az algoritmus gondosan ellenőrzi a a javasolt „kört."

Az NP bonyolultsági osztály

Az **NP *bonyolultsági osztály*** azon nyelvek halmaza, melyek polinomiális algoritmussal bizonyíthatók.[6] Pontosabban: az L nyelv pontosan akkor van NP-ben, ha létezik két bemenetű polinomiális A algoritmus, és c konstans, hogy

$$L = \{x \in \{0,1\}^* : \text{létezik } y \in \{0,1\}^* \text{ tanú, melyre } |y| = O(|x|^c),\ A(x,y) = 1\}.$$

Ekkor azt mondjuk, hogy A ***polinom időben bizonyítja*** L-et.

A Hamilton-kör probléma eddigi vizsgálatából látható, hogy HAM $\in$ NP. (Mindig kellemes érzés megtudni, hogy egy fontos halmaz nem üres.) Ezen túlmenően, ha $L \in$ P, akkor $L \in$ NP is, hiszen ha L az A polinomiális algoritmussal eldönthető, akkor ebből az algoritmusból könnyen kaphatunk megfelelő ellenőrző algoritmust, ami a második bemenetet egyszerűen figyelmen kívül hagyja, és akkor ad az (x, y) párra 1-et, ha $A(x) = 1$. Ezzel beláttuk, hogy P $\subseteq$ NP.

Nem tudjuk azonban, hogy itt valódi tartalmazás áll-e fenn, azaz P $\subset$ NP, vagy P = NP; a kutatók többsége azonban úgy gondolja, hogy P és NP különböző osztályok. Intuitíve: P-ben a gyorsan megoldható problémák, NP-ben a gyorsan leellenőrizhető megoldású problémák vannak. Tapasztalhattuk, hogy többnyire lényegesen nehezebb egy problémát megoldani, mint egy adott megoldás helyességét ellenőrizni. Ez arra „utal", hogy lehetnek NP-ben olyan nyelvek, melyek P-ben nincsenek benne. Van jobb okunk is, hogy elhiggyük a P $\neq$ NP sejtést: az NP-teljes nyelvek létezése. Ezzel az osztállyal a 34.3. alfejezetben fogunk foglalkozni.

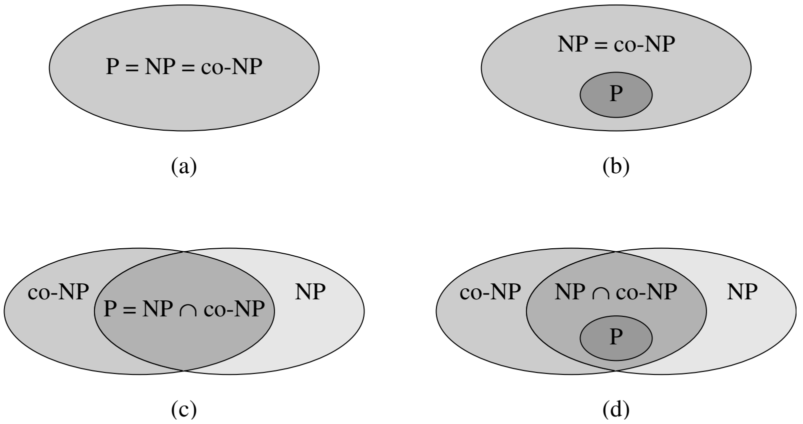
A témakörben a P $\neq$ NP sejtésen kívül is számos alapvető kérdés máig megoldatlan. A kutatók erőfeszítései ellenére senki sem tudja például, hogy az NP osztály zárt-e a komplementerképzésre. Definiáljuk a **co-NP *bonyolultsági osztályt*** a következőképp: $L \in$ co-NP pontosan akkor, ha $\overline{L} \in$ NP. NP komplementerképzésre való zártsága tehát azt jelentené, hogy NP = co-NP. Mivel P zárt a komplementálásra (34.1-6. gyakorlat), P $\subseteq$ NP $\cap$ co-NP. Itt sem tudjuk, hogy a tartalmazás valódi-e. A négy lehetséges változat a 34.3. ábrán látható.

A P és NP közti kapcsolatról való tudásunk tehát elkeserítően hiányos. Mindazonáltal az NP-teljesség elméletének beható vizsgálata során látni fogjuk, hogy lemaradásunk a problémák kezelhetetlenségének bizonyításában gyakorlati szempontból nem is olyan nagy, mint gondolnánk.

Gyakorlatok

34.2-1. Mutassuk meg, hogy a GRÁF-IZOMORFIZMUS = $\{\langle G_1, G_2\rangle : G_1$ és G_2 izomorf gráfok$\}$ nyelv NP-ben van.

[6]NP a „nondeterministic polinomial (time)" (nemdeterminisztikus polinomiális (idejű)) rövidítése. Az NP osztályt eredetileg a nemdeterminizmussal összefüggésben vizsgálták (és definiálták); jelen könyv a némiképp egyszerűbb, ekvivalens „ellenőrzéses" megközelítést részesíti előnyben. Az NP-teljesség nemdeterminisztikus számítási modellel való tárgyalása megtalálható például Hopcroft és Ullmann [156] művében.

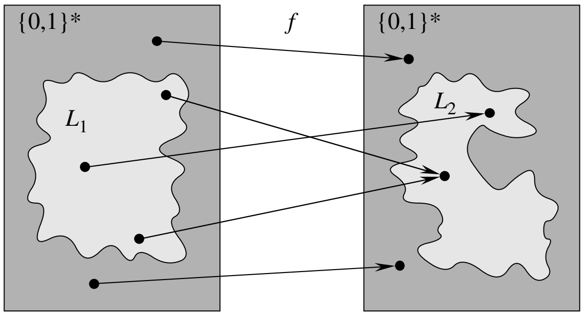


34.3. ábra. Az eddig megismert bonyolultsági osztályok közti négy lehetséges kapcsolat. **(a)** P = NP = co-NP. A kutatók többsége szerint ez a legkevésbé valószínű lehetőség. **(b)** NP zárt a komplementálásra, de P ≠ NP. **(c)** P = NP ∩ co-NP, de NP nem zárt a komplementálásra. **(d)** NP ≠ co-NP, és P ≠ NP ∩ co-NP. A tudósok többsége ezt tartja a legvalószínűbb esetnek.

34.2-2. Lássuk be, hogy ha egy páros gráfnak páratlan számú csúcsa van, akkor nem lehet Hamilton-köre.
34.2-3. Mutassuk meg, hogy ha HAM ∈ P, akkor polinom időben meg is lehet adni tetszőleges G gráf egy Hamilton-körét (ha van).
34.2-4. Bizonyítsuk be, hogy NP zárt az unióképzés, metszetképzés, konkatenáció és lezárás műveletekre. Gondolkozzunk el NP komplementálásra való zártságának kérdésén.
34.2-5. Mutassuk meg, hogy az NP-beli nyelvek eldönthetők $2^{O(n^k)}$ idejű algoritmussal, valamely k konstans mellett.
34.2-6. Egy G gráf ***Hamilton-útjának*** nevezzük az olyan egyszerű utakat, amelyek minden csúcsot pontosan egyszer érintenek. Mutassuk meg, hogy a HAM-ÚT = $\{\langle G, u, v\rangle$: van Hamilton-út u és v közt G-ben$\}$ nyelv NP-ben van.
34.2-7. Mutassuk meg, hogy a Hamilton-út probléma polinom időben megoldható irányított körmentes gráfokra. Adjunk minél gyorsabb algoritmust.
34.2-8. Legyen ϕ Boole-formula az $x_1, x_2, \ldots, x_n$ változókon. ϕ ***tautológia,*** ha a változók bármely behelyettesítésére 1 az értéke. Mutassuk meg, hogy TAU = $\{\langle\phi\rangle : \phi$ tautológia$\} \in$ co-NP.
34.2-9. Bizonyítsuk be, hogy P ⊆ co-NP.
34.2-10. Bizonyítsuk be, hogy ha NP ≠ co-NP, akkor P ≠ NP.
34.2-11. Legyen G legalább 3 csúcsú összefüggő, irányítatlan gráf. Jelöljük G^3-nel azt a gráfot, melyet úgy kapunk, hogy összekötjük G azon csúcsait, melyek legfeljebb 3 hosszú úton elérhetők egymásból. Mutassuk meg, hogy G^3-ben van Hamilton-kör. (*Útmutatás.* Induljunk ki G egy feszítőfájából, és használjunk indukciót.)

34.3. NP-teljesség és visszavezethetőség

A P ≠ NP sejtés melletti legnyomósabb érv talán az NP-teljes problémák létezése. Ezen problémák rendelkeznek a következő meglepő tulajdonsággal: ha közülük akár csak egy



34.4. ábra. Az L_1 nyelv polinomiális visszavezetése L_2-re az f visszavezető függvény segítségével. Bármely $x \in \{0,1\}^*$ bemenetre $x \in L_1$ pontosan akkor, ha $f(x) \in L_2$.

is megoldható polinom időben, akkor *valamennyi* NP-beli probléma megoldható polinom időben, azaz P = NP. Mint arról már szó esett, az évtizedek óta folyó kutatások ellenére sem sikerült egyelőre polinomiális algoritmust találni egyetlen NP-teljes problémára sem.

A HAM NP-teljes probléma. Ha el tudnánk dönteni HAM-et polinomiális idő alatt, akkor minden NP-beli problémát meg tudnánk oldani polinomiális idő alatt. Ha kiderülne, hogy NP − P nem üres, akkor teljes bizonyossággal mondhatnánk, hogy HAM ∈ NP − P.

Az NP-teljes nyelvek bizonyos értelemben a „legnehezebb" nyelvek NP-ben. Ebben az alfejezetben megmutatjuk, hogyan hasonlíthatjuk össze a nyelvek „nehézségét", a „polinomiális idejű visszavezethetőség" segítségével. Először is definiáljuk az NP-teljes nyelvek halmazát, majd vázoljuk annak bizonyítását, hogy a C-SAT nevű nyelv ide tartozik. A 34.4. és 34.5. alfejezetekben pedig számos más probléma NP-teljességét bizonyítjuk a visszavezetés segítségével.

Visszavezethetőség

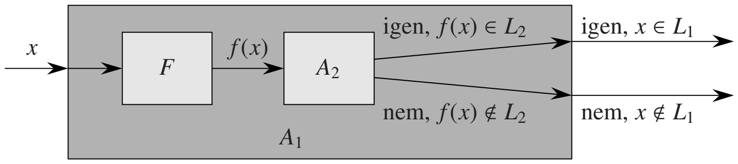
Intuitíve a P probléma visszavezethető Q-ra, ha P bármely x esete „könnyen átfogalmazható" Q egy olyan y esetévé, melynek megoldásából következik x megoldása. Például az elsőfokú egyismeretlenes egyenlet megoldásának problémája visszavezethető a másodfokú egyenlet megoldására. Az $ax + b = 0$ egyenletet ugyanis a $0y^2 + ay + b = 0$ egyenletté alakíthatjuk, s ennek megoldása kielégíti $ax + b = 0$-t is. Ez azt is jelenti, hogy ha P visszavezethető Q-ra, akkor P-t bizonyos értelemben „nem nehezebb megoldani", mint Q-t.

A döntési problémák nyelvelméleti tárgyalásához visszatérve, azt mondjuk, hogy az L_1 nyelv ***polinomiálisan visszavezethető*** az L_2 nyelvre (jelölése: $L_1 \leq_P L_2$), ha létezik $f : \{0,1\}^* \to \{0,1\}^*$ polinom időben kiszámítható függvény, melyre minden $x \in \{0,1\}^*$ esetén

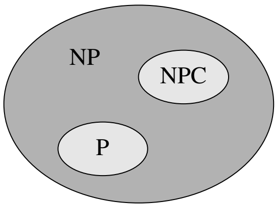
$$x \in L_1 \text{ pontosan akkor, ha } f(x) \in L_2. \tag{34.1}$$

Az f függvényt ***visszavezető függvénynek*** hívjuk, az f-et kiszámító polinomiális algoritmust pedig ***visszavezető algoritmusnak.***

A 34.4. ábrán látható egy L_1 nyelv L_2-re való visszavezetésének illusztrációja. A nyelvek természetesen $\{0,1\}^*$ részhalmazai, és jól látható, hogy az L_1-beli x-ek $f(x)$ képe L_2-ben van, az L_1-en kívülieké pedig L_2-n kívül. A visszavezetés fenti definíciója tehát megfelel intuitív elképzeléseinknek, az L_1 által reprezentált probléma eseteit megfeleltettük az L_2-vel reprezentált probléma eseteinek úgy, hogy az utóbbiakra kapott megoldás meghatározza az



34.5. ábra. A 34.3. lemma bizonyítása. F az L_1-et L_2-re visszavezető függvényt kiszámító polinomiális algoritmus, A_2 az L_2 nyelvet eldöntő polinomiális algoritmus. A képen látható A_1 algoritmus először előállítja $f(x)$-et x-ből F segítségével, majd A_2 segítségével eldönti, hogy $f(x) \in L_2$ vagy sem, ily módon eldöntve az L_1 nyelvet is.



34.6. ábra. A legtöbb számítógéptudós így képzeli el a P, NP és NPC osztályok viszonyát.

előbbiek megoldásait.

A visszavezetési módszer jól használható nyelvek polinomialitásának bizonyítására.

34.3. lemma. *Ha* $L_1, L_2 \subseteq \{0,1\}^*$, $L_1 \le_P L_2$, *és* $L_2 \in \mathrm{P}$, *akkor* $L_1 \in \mathrm{P}$.

Bizonyítás. Legyen A_2 polinomiális algoritmus, ami eldönti L_2-t, és legyen F az L_1-et L_2-re visszavezető f függvényt kiszámító polinomiális algoritmus. Megadunk egy L_1-et polinom időben eldöntő A_1 algoritmust.

A_1 konstrukcióját a 34.5. ábra szemlélteti. Az $x \in \{0,1\}^*$ bemenetből A_1 először F segítségével előállítja $f(x)$-et, majd A_2 felhasználásával eldönti,hogy $f(x)$ L_2-ben van-e, és az erre kapott választ adja ki eredményként.

Az, hogy A_1 helyes választ ad, a (34.1.) feltételből következik. A_1 polinomiális, hiszen F és A_2 is az. (Lásd a 34.1-5. gyakorlatot.) ∎

NP-teljesség

A polinomiális visszavezetés arra is kiválóan felhasználható, hogy belássuk: egy adott probléma legalább olyan nehéz, mint egy másik, legalábbis polinom tényező erejéig. Ha ugyanis $L_1 \le_P L_2$, akkor L_1 legfeljebb polinomszor „nehezebb” L_2-nél, ami megmagyarázza a visszavezethetőségre használt $\le_P$ jelölést. Most már készen állunk az NP-teljes nyelvek definiálására, amelyek a legnehezebb NP-beli nyelvek.

Az $L \subseteq \{0,1\}^*$ nyelv ***NP-teljes,*** ha

1. $L \in \mathrm{NP}$ és
2. Minden $L' \in \mathrm{NP}$-re $L' \le_P L$.

Ha egy nyelv rendelkezik a második tulajdonsággal, de az elsővel nem feltétlenül, akkor ***NP-nehéznek*** nevezzük. Az NP-teljes nyelvek halmazát NPC-vel jelöljük.

Mint a következő tétel mutatja, az NP-teljességnek döntő szerepe van P és NP viszonyának eldöntésében.

34.4. tétel. *Ha létezik polinomiális időben megoldható NP-teljes probléma, akkor* P = NP. *Más szóval: ha létezik* NP*-ben polinom időben nem megoldható probléma, akkor egyetlen* NP*-teljes probléma sem polinomiális.*

Bizonyítás. Tegyük fel, hogy $L \in \mathrm{P} \cap \mathrm{NPC}$. Tudjuk, hogy bármely $L' \in \mathrm{NP}$ esetén $L' \le_{\mathrm{P}} L$, az NP-teljesség definíciójának 2. feltétele miatt. Mivel $L \in \mathrm{P}$, a 34.3. lemma szerint $L' \in \mathrm{P}$, ezzel az első állítást beláttuk. A második állítás nyilván ekvivalens az elsővel. ∎

Érthető tehát, hogy a P ≠ NP sejtéssel kapcsolatos kutatások középpontjában az NP-teljes problémák állnak. A számítógéptudósok többsége úgy gondolja, hogy P ≠ NP, ami az iménti tétel szerint a P, NP és NPC osztályok 34.6. ábra szerinti viszonyát jelenti. Jelenlegi tudásunk szerint persze az sem elképzelhetetlen, hogy valaki előáll egy NP-teljes probléma polinomiális megoldásával, bizonyítva ezzel, hogy P = NP. Minthogy ilyesmi eddig nem történt, és nem is igen várható, egy probléma NP-teljessége jó ok annak feltételezésére, hogy a probléma nehezen kezelhető.

Boole-hálózatok kielégíthetősége

Most már tudjuk, hogy mit is értünk NP-teljes problémán, de nem tudjuk, hogy ilyen egyáltalán létezik-e. Amint egy problémáról belátjuk, hogy NP-teljes, a polinomiális visszavezetést használva már számos más probléma NP-teljességét is igazolni tudjuk. Elsőként tehát mutatnunk kell egy NP-teljes problémát. Ez a Boole-hálózatok kielégíthetőségének problémája (circuit-satisfiability, C-SAT) lesz.

A formális bizonyítás sajnos olyan eszközöket igényel, amelyek meghaladják e fejezet lehetőségeit. Ehelyett egy olyan informális bizonyítást adunk, mely csak a Boole-hálózatokkal kapcsolatos alapvető ismeretekre épít.

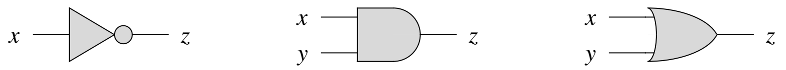
A Boole-hálózatok huzallal összekötött elemekből állnak. A ***logikai áramkör*** a hálózat olyan eleme, amelynek konstans számú be-, illetve kimenete van, és valamilyen jól meghatározott függvényt számít ki. A Boole-változók lehetséges értékei a $\{0, 1\}$ halmaz elemei, ahol a 0 jelenti a HAMIS, az 1 pedig az IGAZ értéket.

Azok a logikai áramkörök, amiket a C-SAT probléma esetében használni fogunk, az úgynevezett ***logikai kapuk,*** egyszerű Boole-függvényeket számítanak ki. A 34.7. ábrán látható a három alapvető logikai kapu: a ***NEM kapu,*** vagy ***megfordító***, az ***ÉS kapu*** és a ***VAGY kapu.*** A NEM kapunak egy x bináris bemenete és egy y bináris kimenete van, a 0 bemenetre 1-et, az 1 bemenetre 0-t ad kimenetként. A másik két kapunak két bináris bemenete (x és y) és egy z bináris kimenete van.

Ezen logikai kapuk működését (csakúgy, mint bármely kapu működését) az ***igazságtáblázat*** segítségével írhatjuk le, melyet a 34.7. ábrán az egyes kapuk alatt találunk. Az igazságtáblázat megadja a kapu kimenetét a bemenetek minden lehetséges értékére. Például a VAGY kapu igazságtáblázata elárulja nekünk, hogy ha a bemenetek $x = 0$ és $y = 1$, akkor a kimenet $z = 1$. A NEM függvényt a $\neg$, az ÉS függvényt a $\wedge$, a VAGY függvényt a $\vee$ szimbólummal jelöljük. Így például $0 \vee 1 = 1$.

Az ÉS és VAGY kapukat általánosíthatjuk, hogy kettőnél több bemenetet is elfogadjanak. Az általánosított ÉS kapu kimenete pontosan akkor 1, ha minden bemenete 1, egyébként 0. Az általánosított VAGY kapu kimenete pontosan akkor 1, ha legalább egy bemenete 1, egyébként 0.

A ***Boole-hálózatok*** kapukból állnak, melyeket ***huzalok*** kötnek össze. Egy huzal össze-



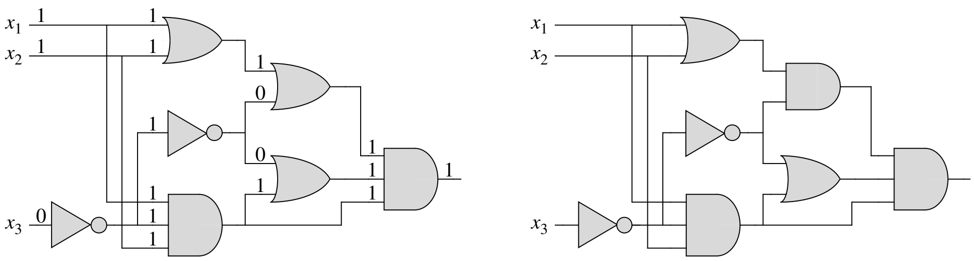
x	$\neg x$
0	1
1	0

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

(a) (b) (c)

34.7. ábra. A három alapvető logikai kapu. A kapuk alatt találhatók az őket leíró igazságtáblázatok. **(a)** A NEM kapu. **(b)** Az ÉS kapu. **(c)** A VAGY kapu.



(a) (b)

34.8. ábra. A C-SAT probléma két esete. **(a)** Az $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ bemenetekre az eredmény 1, a hálózat tehát kielégíthető. **(b)** A bemenetek semmilyen értékére sem lesz 1 az eredmény, vagyis ez a hálózat nem kielégíthető.

kötheti egy kapu kimenetét egy másik kapu bemenetével, így az első kapunál megjelenő kimeneti érték lesz a második kapu egyik bemeneti értéke. A 34.8. ábrán két olyan Boole-hálózat látható, melyek csak egyetlen kapuban különböznek. Az ábra (a) részén az egyes huzalokon megjelenő értékek is láthatók az $\langle x_1 = 1, x_2 = 1, x_3 = 0\rangle$ bemenet mellett. Egy huzal természetesen legfeljebb egy kimenethez csatlakozhat, a hozzá csatlakozó bemenetek száma azonban tetszőleges nemnegatív egész lehet. Ez utóbbi számot a huzal ***ki-fokának*** hívjuk. Ha egy huzal nem kapcsolódik kimenethez, akkor ***hálózati bemenetnek*** hívjuk, míg ha a ki-foka 0, akkor a neve ***hálózati kimenet.*** A hálózati kimenetek továbbítják a hálózat számításainak eredményeit a külvilágba. (Az is elképzelhető ugyanakkor, hogy egy belső huzalon lévő érték is hozzáférhető a külvilág számára.) A C-SAT probléma definiálásához korlátozzuk most a huzalok ki-fokát, legyen minden kifok legfeljebb 1.

A Boole-hálózatok nem tartalmaznak köröket. Más szóval, készítsünk minden hálózathoz egy irányított gráfot a következőképp: legyenek a kapuk a gráf csúcsai, a k ki-fokú huzaloknak pedig feleljen meg k irányított él, oly módon, hogy az u kapunak megfelelő csúcsból a v kapunak megfelelő csúcsba mutasson él, ha a huzal kimenete u-nak és bemenete v-nek; az így kapott gráfnak kell körmentesnek lennie.

Egy Boole-hálózat ***behelyettesítése*** Boole-változók egy, a bemenetek számával megegyező elemszámú sorozata. Azt mondjuk, hogy egy egy kimenetű Boole-hálózat ***kielégíthető,*** ha van ***kielégítő behelyettesítése,*** azaz olyan behelyettesítése, melyre a kimenet 1.

Például, a 34.8(a) ábrán látható hálózat az $x_1 = 1,\ x_2 = 1,\ x_3 = 0$ bemenetekre 1-et ad kimenetként, tehát kielégíthető. A 34.3-1. gyakorlatban viszont azt kell megmutatnunk, hogy a 34.8(b) ábrán látható hálózat az x_1, x_2, x_3 változók egyetlen behelyettesítésére sem ad 1-et, tehát nem kielégíthető.

A ***Boole-hálózatok kielégíthetőségének problémája:*** „Adott – ÉS, VAGY és NEM kapukból álló – Boole-hálózat kielégíthető-e?”. A kérdés formalizálásához meg kell állapodnunk a hálózatok valamilyen szabványos kódolásában. A gráfokéhoz hasonló kódolás alkalmas lesz (a hálózat tulajdonképpen nem más, mint egy speciális, irányított gráf), ekkor $\langle C \rangle$ hossza nem sokkal nagyobb, mint a C hálózat mérete. Most már definiálhatjuk a

$$\text{C-SAT} = \{\langle C \rangle : \ C \text{ kielégíthető Boole-hálózat}\}$$

formális nyelvet.

A C-SAT probléma igen jelentős szerepet játszik a hardver-optimalizációban. Ha ugyanis egy hálózat minden bemenetre 0-t ad, akkor helyettesíthető egy konstans 0-t adó kapuval, időt és erőforrásokat takarítva meg. Egy polinomiális algoritmusnak tehát számottevő gyakorlati haszna lenne.

Ha adott a C hálózat, megpróbálhatjuk a kielégíthetőség kérdését úgy eldönteni, hogy minden lehetséges bemenetre kiszámítjuk az eredményt. k bemenet esetén a lehetséges behelyettesítések száma 2^k. Ha például C mérete k polinomja. akkor minden eset ellenőrzése $\Omega(2^k)$ időt igényel, azaz szuperpolinomiális az áramkör méretében.[7] Persze, mint arról már szót ejtettünk, C-SAT-ra valószínűleg nincs is polinomiális algoritmus, hiszen C-SAT NP-teljes. A bizonyítás két lépésben történik, először a definíció első, majd a második feltételét igazoljuk.

34.5. lemma. C-SAT $\in$ NP.

Bizonyítás. Megadunk egy C-SAT-ot ellenőrző kétbemenetű algoritmust. Az A algoritmus egyik bemenete a C hálózat (szabványos kódolása), a másik egy t 0-1 sorozat, ami a huzalokon szereplő értékeknek felel meg. (Aki rövidebb tanúsítványt szeretne látni, oldja meg a 34.3-4. gyakorlatot.)

Az algoritmus minden kapura ellenőrzi, hogy az adott bemenetekhez az adott kimenet helyesen van-e kiszámítva, majd, ha a hálózat kimenete 1, akkor 1-et ad kimenetként, hiszen ekkor a hálózat bemenetei kielégítő behelyettesítést adnak. Minden más esetben az algoritmus 0-t ad kimenetként.

Ha C kielégíthető, akkor létezik C hosszúságában polinomiális hosszúságú t sorozat, amire $A(C, t) = 1$. Ha viszont C nem kielégíthető, akkor ez egyetlen t sorozatra sem teljesülhet. Az A algoritmus nyilván polinomiális (akár lineáris időben is megvalósítható). Mindezek együttesen azt jelentik, hogy az A algoritmus polinomiális tanúval, polinom időben ellenőrzi a C-SAT nyelvet, azaz C-SAT $\in$ NP. ■

[7] Másfelől, ha C mérete $\Theta(2^k)$, akkor egy $O(2^k)$ futásidejű algoritmus polinomiális a hálózat méretében. Ez még akkor sem mond ellent C-SAT NP-teljességének, ha P $\neq$ NP: egy speciális esetre adott polinomiális algoritmus nem jelenti azt, hogy létezik polinomiális algoritmus minden esetre.

A következő lépés annak bizonyítása, hogy C-SAT NP-nehéz, azaz minden NP-beli nyelv polinomiálisan visszavezethető rá. A tényleges bizonyítás – mely a számítógépek működésének alapelveire épül – számos technikai bonyodalmat tartalmaz, ezért vázlatos ismertetésére szorítkozunk.

A számítógépes programok a gép memóriájában vannak elhelyezve, utasítások sorozataiként. Egy tipikus utasítás a következő három dolog kódolása: egy végrehajtandó művelet, a művelet argumentumainak címei a memóriában és az a memóriacím, ahová az eredményt kell írni. Egy speciális memóriahely, az úgynevezett ***programszámláló*** nyomon követi, hogy melyik utasítás következik. A programszámláló automatikusan növekszik, valahányszor egy utasítás végrehajtódik, ami azt eredményezi, hogy a számítógép a műveleteket sorban egymás után hajtja végre. Egy művelet a programszámláló átírását is okozhatja, így a végrehajtás módosulhat, lehetővé téve ezzel ciklusok futását és a feltételes elágazásokat.

A program futása során a számítás pillanatnyi állapotát a gép memóriájában tároljuk. A memória tartalmazza magát a programot, a programszámlálót, a munkaterületet és a számtalan állapotjelző bitet, amiket a számítógép fenntart a „könyveléshez”. A memória egyes állapotait ***konfigurációknak*** hívjuk. Egy utasítás végrehajtását tekinthetjük olyan leképezésnek, amely egy konfigurációt egy másikba visz. Fontos, hogy a hardver, ami ezt a leképezést végrehajtja, megvalósítható Boole-hálózatként, amit M-mel fogunk jelölni a következő lemmában.

34.6. lemma. C-SAT NP-*nehéz.*

Bizonyítás. Legyen L tetszőleges NP-beli nyelv. Megadunk egy F polinomiális algoritmust, mely az alábbi f visszavezető függvényt számítja ki: minden $x \in \{0,1\}^*$ szó képe egy olyan $f(x) = \langle C \rangle$ kódolt hálózat, amelyre $x \in L$ pontosan akkor teljesül, ha $\langle C \rangle \in$ C-SAT.

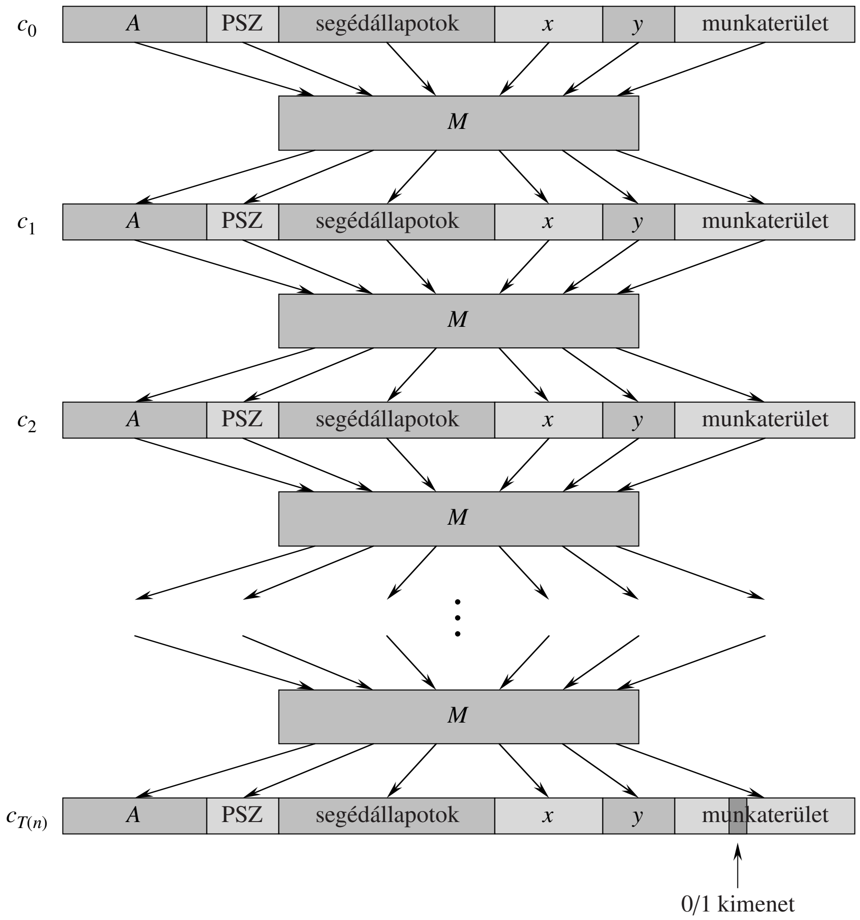
$L \in$ NP, így létezik olyan algoritmus, mely L-et polinom időben ellenőrzi. Megkonstruálunk egy olyan F algoritmust, amely a kétbemenetű A algoritmust fogja felhasználni az f visszavezetőfüggvény kiszámítására.

Jelöljük $T(n)$-nel A legrosszabb futási idejét az n hosszúságú bemeneti sztringeken, és legyen $k \geq 1$ olyan szám, melyre igaz, hogy $T(n) = O(n^k)$ és a tanú hossza is $O(n^k)$. (A futási ideje a két bemenet együttes hosszában polinomiális, de mivel a tanú maga is polinom n-ben, ami egy adott eset kódolásának hossza, a futási idő n-ben is polinomiális.)

A bizonyítás alapgondolata, hogy A számításait konfigurációk sorozataként jelenítjük meg. Amint az a 34.9. ábrán látható, minden egyes konfiguráció az alábbi részekre bontható: az A-nak megfelelő program, a programszámláló, a segédállapotok, az x bemenet (egy eset kódolása), az y tanú és a munkaterület. A c_0 kezdeti konfigurációból indulva, a c_i konfigurációkat c_{i+1}-be képezzük, az M Boole-hálózat segítségével, amely a hardver tevékenységét reprezentálja. Az algoritmus kimenete – 0 vagy 1 – a munkaterület egy kijelölt helyén található, amikor A futása befejeződik (legfeljebb $T(n)$ lépés), így az eredmény $c_{T(n)}$ egy kijelölt bitjével lesz azonos.

Az F algoritmus létrehoz egy olyan Boole-hálózatot, amely egy adott kezdő konfigurációhoz tartozó összes konfigurációt kiszámítja az alábbi módon. Az M hálózat $T(n)$ darab példányát összefűzzük, ami azt jelenti, hogy az i-edik hálózat kimenetét (azaz a c_i konfigurációt) betápláljuk az $(i+1)$-edikbe.

Mi is az F algoritmus feladata? Egy adott x bemenetre olyan $f(x) = C$ hálózatot kell adnia, amely pontosan akkor kielégíthető, ha létezik y tanú, melyre $A(x, y) = 1$. F először kiszámítja a bemenet $n = |x|$ hosszát, majd létrehozza a C' hálózatot, mely az M hálózat $T(n)$



34.9. ábra. Az A algoritmus (x, y) bemeneten való futása során kapott konfigurációk sorozata. A c_0 konfiguráció az y tanút leszámítva állandó. A konfigurációkat az M Boole-hálózat képezi a következő konfigurációba. Az eredmény a munkaterület egy kitüntetett bitje.

darab összekapcsolt példányából áll. C' bemenete az $A(x, y)$ számításához tartozó kezdő konfiguráció, kimenete pedig a $C_{T(n)}$ konfiguráció.

Az F által konstruálandó $C = f(x)$ hálózatot C' kis módosításával kapjuk. Először is C'-nek az A-t megvalósító programhoz tartozó bemeneteit – a programszámlálót, a kezdőállapotokat és az x bemenetet – beállítjuk a megfelelő ismert értékekre. Hálózatunk „igazi" bemenetei tehát valamennyien az y tanúhoz tartoznak. Másodszor, a hálózat kimeneteit a $C_{T(n)}$ bit kivételével figyelmen kívül hagyjuk. Az ily módon kapott C hálózat minden $O(n^k)$ hosszúságú y tanúhoz kiszámolja $C(y) = A(x, y)$-t.

Meg kell mutatnunk, hogy F visszavezető függvényt számít ki (azaz, hogy C pontosan akkor kielégíthető, ha létezik y tanú, melyre $A(x, y) = 1$), és hogy ez a számítás polinom időben véget ér.

Tegyük fel először, hogy létezik $O(n^k)$ hosszúságú y tanú, melyre $A(x, y) = 1$. Adjuk most y bitjeit C-nek bemenetként, ekkor $C(y) = A(x, y) = 1$. Ha tehát a megfelelő tanú létezik, akkor C kielégíthető. Tegyük fel ármost, hogy C kielégíthető. Ez azt jelenti, hogy létezik y, melyre $C(y) = 1$, így $A(x, y) = C(y) = 1$. F tehát valóban visszavezető függvényt számít ki.

Hátravan még annak bizonyítása, hogy F polinomiális $|x| = n$-ben. Vegyük észre, hogy a konfigurációk leírásához n-ben polinomiális számú bit elég, hiszen az A-t megvalósító program konstans méretű, a bemenet mérete n, az y tanú $O(n^k)$ hosszú és a munkaterület is polinomiális számú bitet használ, mivel az algoritmus $O(n^k)$ lépésben véget ér. (Feltesszük, hogy a memória összefüggő; a 34.3-5. gyakorlatban ki kell terjesztenünk a bizonyítást arra az esetre, amikor a program által elérni kívánt címek a memória egy nagyobb részén vannak szétszórva, és ez a szétszórás bemenetenként különböző lehet.)

A hardver működését megvalósító M hálózat polinomiális nagyságú a konfigurációk méretében, így n-ben is. A C hálózat M-nek legfeljebb $t = O(n^k)$ darab példányából áll, így szintén polinomiális méretű n-ben és konstrukciója polinomiális számú lépést igényel. (Ennek a hálózatnak a nagyobb része a memóriarendszert valósítja meg.) Az x bemenetből $f(x) = C$-t előállító F visszavezető algoritmus tehát polinomiális, hiszen a konstrukció minden lépése megvalósítható polinom időben. ■

A C-SAT nyelv ezek szerint legalább olyan nehéz, mint bármely NP-beli nyelv, és mivel NP-hez tartozik, NP-teljes.

34.7. tétel. *A* C-SAT *nyelv* NP-*teljes.*

Bizonyítás. A 34.5. és 34.6. lemmák és az NP-teljesség definíciójának közvetlen következménye. ■

Gyakorlatok

34.3-1. Igazoljuk, hogy a 34.8(b) ábrán látható hálózat nem kielégíthető.

34.3-2. Mutassuk meg, hogy a nyelveken értelmezett $\leq_P$ reláció tranzitív, azaz ha $L_1 \leq_P L_2$ és $L_2 \leq_P L_3$, akkor $L_1 \leq_P L_3$.

34.3-3. Bizonyítsuk be, hogy $L \leq_P \overline{L}$ akkor és csak akkor, ha $\overline{L} \leq_P L$.

34.3-4. Mutassuk meg, hogy egy kielégítő behelyettesítés felhasználható tanúként a 34.5. lemma egy másik bizonyításához. Az új vagy a már látott tanú esetén kapunk egyszerűbb bizonyítást?

34.3-5. A 34.6. lemma bizonyításában feltettük, hogy a munkaterület a memória egy összefüggő, polinomiális méretű része. Hol használtuk fel a bizonyítás során ezt? Mutassuk meg, hogy e feltevés nem megy az általánosság rovására.

34.3-6. Az L nyelv ***teljes*** a C nyelvosztályra nézve (a polinomiális visszavezetés tekintetében), ha $L \in C$ és minden $L' \in C$ esetén $L' \leq_P L$. Mutassuk meg, hogy az üres halmaz, és $\{0, 1\}^*$ az egyedüli P-beli nyelvek, amelyek nem teljesek P-re.

34.3-7. Lássuk be, hogy L pontosan akkor teljes NP-re nézve, ha $\overline{L}$ teljes co-NP-re nézve.

34.3-8. A 34.6. lemma bizonyításában szereplő F algoritmus létrehoz egy $C = f(x)$ hálózatot az x, az A és a k ismeretében. Sartre professzor megfigyelte, hogy az x szó valóban bemenete F-nek, de F A-nak és k-nak csupán a létezéséről tud, konkrétan nem ismeri őket. Ebből azt a következtetést vonja le, hogy F nem lehet képes C megkonstruálására, tehát a C-SAT nyelv nem feltétlenül NP-nehéz. Magyarázzuk el neki, hol a hiba az okoskodásában.

34.4. NP-teljességi bizonyítások

C-SAT NP-teljességét közvetlenül a definícióból láttuk be, azaz minden NP-beli L nyelvre igazoltuk, hogy $L \leq_P$ C-SAT. Ebben az alfejezetben megmutatjuk, hogyan látható be nyelvek NP-teljessége anélkül, hogy minden egyes NP-beli nyelvről igazolnánk, hogy visszavezethető az adott nyelvre. A technika illusztrációjaként két, a Boole-formulák kielégíthetőségével kapcsolatos problémáról látjuk be, hogy NP-teljes. A 34.5. alfejezetben számos más példát is látunk majd.

Az alábbi lemma minden további NP-teljességi bizonyítás alapja.

34.8. lemma. *Ha L olyan nyelv, melyhez létezik $L' \in$ NPC, hogy $L' \leq_P L$, akkor L NP-nehéz. Ha $L \in$ NP is teljesül, akkor $L \in$ NPC.*

Bizonyítás. Mivel $L' \in$ NPC, ezért minden $L'' \in$ NP-re igaz, hogy $L'' \leq_P L'$. Ha tehát $L' \leq_P L$, akkor a tranzitivitás miatt (34.3-2. feladat) $L'' \leq_P L$ minden $L'' \in$ NP nyelvre, vagyis L NP-nehéz. Ha $L \in$ NP is, akkor persze $L \in$ NPC. ■

Más szavakkal, egy ismert L' NP-teljes probléma visszavezetése L-re közvetve minden NP-beli problémát visszavezet L-re. A 34.8. lemma szerint tehát a következő módszerrel bizonyíthatjuk egy L nyelv NP-teljességét.

1. Belátjuk, hogy $L \in$ NP.
2. Választunk egy ismert L' NP-teljes nyelvet.
3. Megadunk egy F algoritmust, mely kiszámít egy olyan f függvényt, ami L' eseteinek L eseteit felelteti meg.
4. Bebizonyítjuk, hogy tetszőleges $x \in \{0, 1\}^*$ esetén $x \in L'$ akkor és csak akkor teljesül, ha $f(x) \in L$.
5. Igazoljuk, hogy F polinomiális.

(A 2–5. lépések bizonyítják azt, hogy L NP-nehéz.) Ez a módszer természetesen jóval egyszerűbb, mint az NP-teljesség közvetlen igazolása. Azzal, hogy C-SAT $\in$ NPC-t bebizonyítottuk, megtettük a legfontosabb lépést; az NP-teljességi bizonyítások ezentúl lényegesen könnyebbek lesznek. Sőt, ahogy egyre több problémáról látjuk be, hogy NP-teljes, úgy lesz egyre könnyebb dolgunk, hiszen módszerünk 2. lépésénél az NP-teljes problémák egyre népesebb táborából válogathatunk.

Boole-formulák kielégíthetősége

A visszavezetési technika első alkalmazásaként belátjuk, hogy a Boole-formulák kielégíthetőségének problémája (satisfiability problem, SAT) NP-teljes. E problémának történelmi jelentősége (is) van, róla bizonyították elsőként az NP-teljességet.

A ***(formula) kielégíthetőségi problémát*** a SAT nyelv segítségével fogalmazzuk meg. SAT esetei a ϕ Boole-formulák, melyek az alábbi elemekből épülnek fel:

1. $x_1, x_2, x_3, \ldots$ Boole-változók;
2. egy- vagy kétváltozós Boole-függvények, mint például $\wedge$ (ÉS), $\vee$ (VAGY), $\neg$ (NEM), $\rightarrow$ (implikáció), $\leftrightarrow$ (ekvivalencia);

3. zárójelek. (Anélkül, hogy ez az általánosság rovására menne, feltehetjük, hogy nincsenek felesleges zárójelek, azaz minden Boole-függvényhez legfeljebb egy zárójelpár tartozik.)

A ϕ Boole-formulát könnyűszerrel kódolhatjuk olyan hosszúságban, mely $n + m$ polinomja. Az olyan Boole-formulákat, melyeknek csak az elején és a végén van zárójel, ***klózoknak*** nevezzük. Csakúgy, mint a Boole-hálózatoknál, ϕ egy behelyettesítésének egy megfelelő hosszúságú 0-1 sorozatot nevezünk. Egy behelyettesítés kielégítő, ha az elemeit rendre az $x_1, x_2, \ldots$ változók helyébe írva ϕ értéke 1. Egy formula ***kielégíthető,*** ha létezik kielégítő behelyettesítése. A SAT probléma: „Egy adott Boole-formula kielégíthető-e vagy sem?”. Formális nyelvként:

$$\text{SAT} = \{\langle\phi\rangle : \ \phi \text{ kielégíthető Boole-formula}\}.$$

Vegyük például a

$$\phi = ((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

formulát; ezt az $(x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1)$ behelyettesítés kielégíti, hiszen

$$\begin{aligned} \phi &= ((0 \to 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg(1 \vee 1)) \wedge 1 \\ &= (1 \vee 0) \wedge 1 \\ &= 1, \end{aligned} \tag{34.2}$$

így $\langle\phi\rangle \in$ SAT.

Ugyanúgy, mint C-SAT esetében, az összes eset naiv végigpróbálása 2^n lépést igényel n változó esetén, így ha $\langle\phi\rangle$ mérete polinomiális n-ben, akkor minden behelyettesítés $\Omega(2^n)$ lépést igényel, azaz szuperpolinomiális $\langle\phi\rangle$ méretének függvényében. A következő tétel szerint polinomiális algoritmus nem is igen várható SAT-ra.

34.9. tétel. SAT NP-*teljes.*

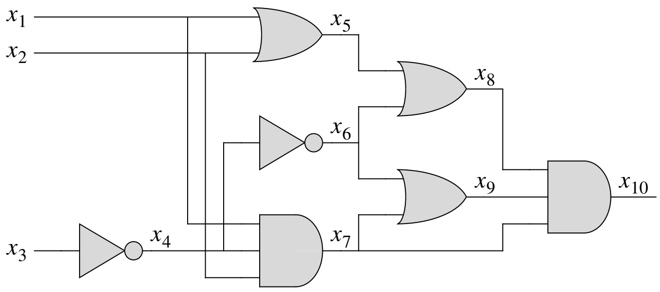
Bizonyítás. Először belátjuk, hogy SAT $\in$ NP, majd igazoljuk, hogy C-SAT $\leq_P$ SAT, amiből a 34.8. lemma szerint a tétel következik.

SAT $\in$ NP bizonyításához megmutatjuk, hogy a ϕ formula egy kielégítő behelyettesítése mint tanú, polinom időben ellenőrizhető. Az ellenőrző algoritmus egyszerűen minden változó helyére beírja a megfelelő értéket, és a kapott kifejezést kiszámítja, nagyjából úgy, ahogy azt (34.2) esetében tettük. Ez az eljárás könnyűszerrel elvégezhető polinom időben. Ha a kifejezés értéke (valamely tanúra) 1, akkor a formula kielégíthető, különben nem.

Ezzel SAT $\in$ NP-t beláttuk, térjünk rá C-SAT-ra való visszavezetésére. A hálózatok tulajdonképpen könnyen formulákká alakíthatók: tekintjük a hálózat kimenetét szolgáltató kaput, és felírjuk a neki megfelelő műveletet a bemeneteire, amiket rekurzívan, ugyanezen módszerrel alakítunk formulákká.

Sajnos, ez a kézenfekvő módszer nem mindig lesz polinomiális: a 34.4-1. gyakorlatban azt kell megmutatnunk, hogy olyan kapuk, melyek kimenete legalább kettő, a kapott formula méretének exponenciális növekedését okozhatják. Így tehát valamilyen ügyesebb visszavezetést kell találnunk.

A 34.10. ábrán egy ilyen jobb visszavezetés alapgondolata látható, a 34.8(a) ábra hálózatára alkalmazva. A C hálózat minden huzaljának (azaz a bemeneteknek és a kapuk



34.10. ábra. C-SAT visszavezetése SAT-ra. A visszavezető algoritmus a hálózat bemenetein kívül a hálózat minden kapujának kimenetéhez is a formula különböző változóit rendeli.

kimeneteinek) megfeleltetünk egy x_i változót. A kapuk működése ekkor leírható a hozzájuk csatlakozó huzaloknak megfelelő változók alkalmas formulájaként. Például az ábrán látható hálózat kimeneti ÉS kapuját leíró formula: $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$.

A teljes hálózatnak megfelelő formula a kimeneti kapuhoz tartozó változó és az egyes kapukat leíró formulák konjunkciója. Például a 34.10. ábra hálózatára:

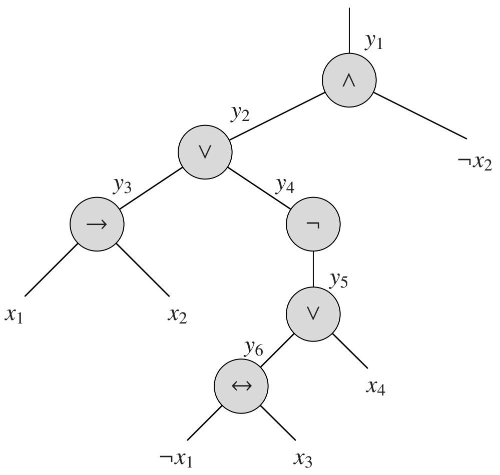
$$\begin{aligned}
\phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \\
&\wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\
&\wedge (x_6 \leftrightarrow \neg x_4) \\
&\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)).
\end{aligned}$$

Egy adott C hálózatra a megfelelő formula nyilván előállítható polinomiális időben.

Miért lesz a C hálózat, és a neki megfelelő ϕ formula ugyanakkor kielégíthető? Tekintsünk egy C-t kielégítő behelyettesítést. Ekkor a huzalokhoz tartozó változók jól definiált értékeket kapnak, melyek – a konstrukció értelmében – kielégítik a kapuknak megfelelő formulákat. Mivel a kimeneti kapuhoz tartozó változó értéke is 1 (hiszen a behelyettesítés kielégíti C-t), a ϕ változóira kapott értékek kielégítik ϕ-t. Megfordítva, ha ϕ-t kielégíti egy adott behelyettesítés, akkor a kimeneti változó 1, és a kapuknak megfelelő formulák értékének is egynek kell lennie, így ϕ azon változói, melyek C bemeneteihez tartoznak, kielégítik C-t. Beláttuk tehát, hogy C-SAT $\leq_P$ SAT, ezzel a bizonyítást befejeztük. ■

A 3-SAT probléma

Igen sok probléma NP-teljességét bizonyíthatjuk azzal, hogy visszavezetjük rá SAT-ot. A visszavezető algoritmusnak ilyenkor persze minden formulát tudnia kell bemenetként kezelni, ami nagyon sok eset áttekintését teszi szükségessé. Az esetek számának csökkentése érdekében sokszor kívánatos lenne a Boole-formulák egy szűkebb nyelvére szorítkozni. Annyira természetesen nem szabad leszűkíteni SAT-ot, hogy a kapott nyelv polinomiálisan eldönthető legyen (sőt, persze azt szeretnénk, ha még a szűkebb nyelv is NP-teljes lenne).



34.11. ábra. A $\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$ formulához tartozó fa.

Céljainknak a 3-SAT elnevezésű nyelv tökéletesen meg fog felelni; leírásához az alábbi definíciók szükségesek. ***Literálnak*** nevezzük egy változónak vagy negáltjának megjelenését egy Boole-formulában. Egy Boole-formula ***konjunktív normálformájú,*** ha olyan, ÉS-ekkel összekapcsolt klózokból áll, melyeket egy vagy több, VAGY-okkal összekapcsolt literál alkot. Ha mindegyik klózban pontosan 3 különböző literál van, akkor ***3-konjunktív normálformáról*** beszélünk. A 3-konjunktív normálformák halmazát 3-CNF (3-conjunctive normal form) jelöli.

Például

$$(x_1 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

egy 3-konjunktív normálforma, amely három klózból áll, ezek közül az első az x_1, $\neg x_1$, x_2 literálokat tartalmazza.

A 3-SAT probléma az, hogy adott, 3-CNF-beli ϕ Boole-formula kielégíthető-e. A következő tétel azt mutatja, hogy Boole-formulák kielégíthetőségének eldöntésére valószínűleg még akkor sem létezik polinomiális algoritmus, ha ebben az egyszerű normálformában vannak felírva.

34.10. tétel. 3-SAT NP-*teljes.*

Bizonyítás. A 34.9. tételben SAT $\in$ NP-re adott bizonyítás egy az egyben alkalmazható 3-SAT esetében is, így 3-SAT $\in$ NP. Meg kell még mutatnunk, hogy 3-SAT NP-nehéz. Ehhez belátjuk, hogy SAT $\leq_P$ 3-SAT, amiből a 34.8. lemma szerint az állítás következik.

A visszavezető algoritmus három fő részből áll, mindegyik lépés közelebb viszi a bemeneti ϕ formulát a kívánt 3-konjunktív normálformájú alakhoz.

Az első lépés hasonló ahhoz, amit C-SAT $\leq_P$ SAT bizonyításánál használtunk. Először felépítünk egy bináris „elemző” fát a ϕ formulához, melynek levelei a literálok, belső csúcsai pedig a műveletek. A 34.11. ábrán a

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2 \tag{34.3}$$

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

34.12. ábra. Az $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ formula igazságtáblázata.

formula elemző fája látható. Az elemző fák valóban binárisak, hiszen a műveletek egy- vagy kétváltozósak (az ÉS és VAGY műveletek többváltozósak is lehetnek, ezeket az asszociativitás felhasználásával több kétváltozós műveletté írhatjuk át), így minden csúcsnak legfeljebb két gyereke lehet. A bináris elemző fát szemügyre véve láthatjuk, hogy nem más, mint egy hálózat, ami a ϕ formulát számítja ki.

Hasonlóan a 34.9. tétel bizonyításához, a belső csúcsok (azaz a műveletek) kimeneteihez (az ősök felé eső élekhez) rendeljük az y_i változókat. Ezt követően a ϕ formulát átírjuk, ismét csak úgy, mint 34.9. bizonyításában, tehát tekintjük a gyökérhez tartozó kimeneti változó, és a belső csúcsok működését leíró formulák konjunkcióját. A (34.3) formula esetén a kapott kifejezés a

$$
\begin{aligned}
\phi' = y_1 \quad & \wedge \quad (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \wedge \quad (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \wedge \quad (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
& \wedge \quad (y_4 \leftrightarrow \neg y_5) \\
& \wedge \quad (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \wedge \quad (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))
\end{aligned}
$$

alakot ölti. Vegyük észre, hogy a kapott ϕ' formula (nem csak most, hanem mindig) olyan ϕ_i' részformulák konjunkciója, melyek legfeljebb három literált tartalmaznak.

A visszavezetés második lépése a ϕ_i' klózok konjunktív normálformába írása. Ehhez elkészítjük minden ϕ_i' igazságtáblázatát. Az igazságtáblázat soraiban a klózban lévő változók lehetséges értékei, és a klóz ezen behelyettesítés mellett felvett értéke szerepelnek. ϕ_i' igazságtáblázatának 0-t adó sorait felhasználva megadunk egy ***diszjunktív normálformát (DNF)*** (ez a konjunktív normálforma „fordítottja”: kívül vannak ÉS-ek, belül pedig VAGY-ok), mely ekvivalens $\neg\phi_i'$-vel. Ezt azután a DeMorgan-azonosságok ((B.2.) azonosságok) segítségével konjunktív normálformává írjuk át, mely ekvivalens ϕ_i'-vel.

Lássuk, hogy is megy ez, például a

$$\phi_1' = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$$

formulára. Vegyük szemügyre igazságtáblázatát (34.12. ábra); eszerint ϕ_1' a 0 értéket veszi föl, ha $(y_1 = 1, y_2 = 1, x_2 = 1)$, vagy $(y_1 = 1, y_2 = 0, x_2 = 1)$, vagy $(y_1 = 1, y_2 = 0, x_2 = 0)$, vagy ha ($y_1 = 0, y_2 = 1, x_2 = 0$). A $\neg\phi_1{}'$-vel ekvivalens diszjunktív normálforma tehát:

$$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2).$$

E formula tagadása ekvivalens $\phi_1{}'$-vel, és a DeMorgan-azonosságok alkalmazásával a következő alakba írható:

$$\phi_1{}'' = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2).$$

konjunktív normálformulát kapjuk, amely ekvivalens $\phi_1{}'$-vel.

Ily módon a ϕ' formula mindegyik $\phi_i{}'$ részformuláját $\phi_i{}''$ konjunktív normálformává tudjuk alakítani, és így ϕ' ekvivalens a $\phi_i{}''$ klózok konjunkciójából álló ϕ' konjunktív normál formával. Továbbá a kapott ϕ'' formula minden klózát legfeljebb három literál alkotja.

A bizonyítás harmadik és egyben utolsó lépésében tovább alakítjuk a formulát úgy, hogy minden klózban *pontosan* 3 literál legyen. Két segédváltozót fogunk használni, legyenek ezek p és q. ϕ'' minden C_i klózához a következőképp veszünk be klózokat ϕ'''-be:

- Ha C_i-ben 3 különböző literál szerepel, akkor egyszerűen magát C_i-t vesszük be.
- Ha C_i-ben 2 különböző literál szerepel, azaz $C_i = (l_1 \vee l_2)$, akkor vegyük be ϕ'''-be az $(l_1 \vee l_2 \vee p)$ és az $(l_1 \vee l_2 \vee \neg p)$ klózokat. A p és $\neg p$ literálok csak azt a formai követelményt hivatottak garantálni, hogy klózonként pontosan 3 literál szerepeljen: $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ nyilván ekvivalens $(l_1 \vee l_2)$-vel.
- Ha C_i-ben egyedül az l literál szerepel, akkor vegyük be ϕ'''-be az $(l \vee p \vee q)$, az $(l \vee p \vee \neg q)$, az $(l \vee \neg p \vee q)$ és az $(l \vee \neg p \vee \neg q)$ klózokat. Nyilvánvaló, hogy e négy klóz konjunkciója ekvivalens l-lel.

A fenti három lépés végigkövetéséből kiderült, hogy a ϕ''' 3-CNF-beli formula pontosan akkor elégíthető ki, ha ϕ kielégíthető. Az, hogy a visszavezetés polinom időben végrehajtható, mindegyik lépés esetében magától értetődő. ■

Gyakorlatok

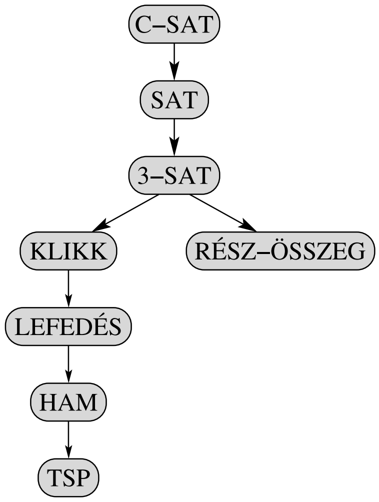
34.4-1. Adjunk meg olyan n méretű hálózatot, melynek a 34.9. tétel bizonyításában leírt kézenfekvő módszerrel való formulává alakítása n-ben exponenciális méretű formulához vezet.

34.4-2. Alakítsuk a (34.3) formulát a 34.10. tétel bizonyításában szereplő módszerrel 3-konjunktív normálformává.

34.4-3. Jagger professzor kizárólag az igazságtáblázat-technika felhasználásával szeretné megmutatni, hogy SAT $\leq_P$ 3-SAT, azaz a ϕ Boole-formula igazságtáblázata alapján szándékozik egy olyan 3-diszjunktív normálformát megadni, amely ekvivalens $\neg\phi$-vel, majd ezt negálva – a DeMorgan-szabályok alkalmazása után – megkapni a megfelelő, ϕ-vel ekvivalens 3-konjunktív normálformát. Mutassuk meg, hogy ez a módszer nem szolgáltat polinomiális visszavezetést.

34.4-4. Bizonyítsuk be, hogy annak eldöntése, hogy egy Boole-formula tautológia-e, teljes co-NP-re nézve. (*Útmutatás.* Lásd a 34.3-7. feladatot.)

34.4-5. Igazoljuk, hogy a diszjunktív normálformák kielégíthetőségének problémája polinom időben megoldható.



34.13. ábra. A 34.4. és 34.5. alfejezetek NP-teljességi bizonyításainak szerkezete. Valamennyi nyelv NP-teljességének bizonyítása C-SAT NP-teljességén alapul.

34.4-6. Tegyük fel, hogy rendelkezésünkre áll egy SAT-ot polinom időben eldöntő A algoritmus. Hogyan tudnánk ekkor kielégítő behelyettesítéseket találni polinom időben tetszőleges ϕ formulára?
34.4-7. Legyen 2-CNF-SAT azoknak a kielégíthető konjunktív normál formuláknak a halmaza, amelyekben minden klóz pontosan két literált tartalmaz. Mutassuk meg, hogy 2-SAT $\in$ P. Adjunk minél gyorsabb algoritmust! (*Útmutatás.* Vegyük észre, hogy $(x \vee y)$ ekvivalens $(\neg x \to y)$-nal. Vezessük vissza 2-SAT-ot egy olyan, irányított gráfokra vonatkozó problémára, mely gyorsan megoldható.)

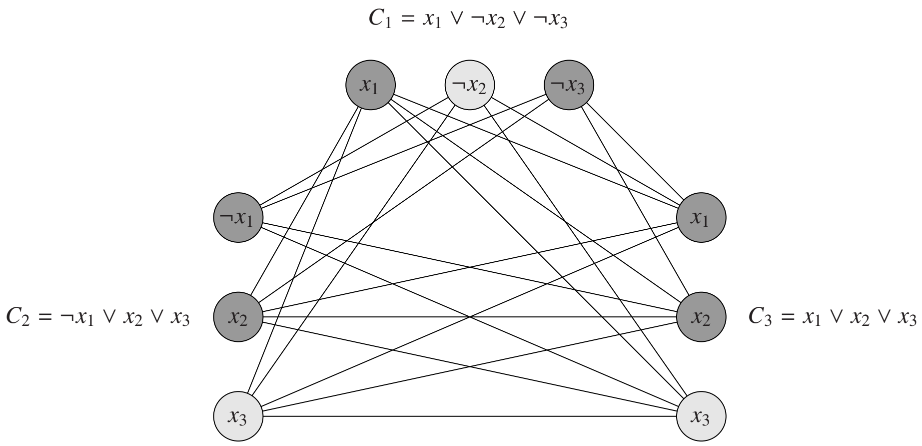
34.5. NP-teljes problémák

NP-teljes problémák a legváltozatosabb helyeken bukkannak fel: a logikában, a számtanban, a gráfelméletben, a számelméletben, az algebrában, a nyelv- és automataelméletben, az optimalizálásban, hálózatok tervezésénél, halmazok és partíciók kapcsán, tervezési és raktározási feladatoknál, játékok vizsgálatakor és még sorolhatnánk. Ebben az alfejezetben néhány gráfelméleti és halmazparticionálási feladat NP-teljességét bizonyítjuk, a visszavezetési technika segítségével.

A 34.13. ábra mutatja a visszavezetések szerkezetét; mindegyik nyelvet arra vezetjük vissza, amelyikből a nyíl rámutat.

34.5.1. A klikk probléma

Csúcsok egy V' halmaza a $G = (V, E)$ irányítatlan gráfban ***klikket*** alkot, ha V' bármely két csúcsa szomszédos G-ben. Egy klikk tehát nem más, mint G egy teljes részgráfja. A klikk ***mérete*** a benne lévő csúcsok száma. A ***klikk probléma*** „Mekkora a legnagyobb klikk egy G gráfban?”. Ez egy optimalizálási probléma, alakítsuk át döntésivé: „Létezik-e



34.14. ábra. A $\phi = C_1 \wedge C_2 \wedge C_3$ 3-CNF formulából kapott gráf, ahol $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee x_2 \vee x_3)$, $C_3 = (x_1 \vee x_2 \vee x_3)$. A formula egy kielégítő behelyettesítése $(x_1 = 0, x_2 = 0, x_3 = 1)$. Ekkor C_1-ben $\neg x_2$, C_2-ben és C_3-ban x_3 az a változó, amely miatt a részformulák értéke 1; a nekik megfelelő klikk csúcsai világosabb színűek.

a G gráfban k méretű klikk?" (k-as klikk ugyanis pontosan akkor létezik, amikor legalább k-as). A formális definíció:

$$\text{KLIKK} = \{\langle G, k\rangle : G \text{ olyan gráf, melyben van } k \text{ méretű klikk}\}.$$

Azt, hogy egy gráfban van-e k-as klikk, eldönthetjük egyszerűen úgy, hogy a csúcsok minden k elemű részhalmazáról megvizsgáljuk, hogy klikk-e vagy sem. Ezen egyszerű algoritmus futásideje $\Omega(k^2\binom{|V|}{k})$ (V a csúcsok halmaza), ami polinomiális, ha k konstans. k azonban lehet $|V|/2$-höz közeli érték, mely esetben a lépésszám már szuperpolinomiális. Valószínű, hogy a klikk probléma megoldására nem létezik hatékony algoritmus.

34.11. tétel. *A klikk probléma* NP*-teljes.*

Bizonyítás. A probléma NP-beli, hiszen a klikket alkotó csúcsok tanúként való ellenőrzése nyilván megoldható polinomiális időben: csak azt kell megnézni, hogy bármely kettő szomszédos-e.

Azt, hogy a klikk probléma NP-nehéz, 3-SAT-ra való visszavezetésével bizonyítjuk. Ez kicsit meglepően hangzik, hiszen első ránézésre a logikai formuláknak kevés közük van a gráfokhoz.

A visszavezető algoritmus 3-SAT egy esetéből indul ki. Legyen $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ egy k klózból álló 3-CNF-beli formula. A C_r klóz $r = 1, 2, \ldots, k$ esetén pontosan 3 literálból áll, legyenek ezek l_1^r, l_2^r és l_3^r.

Megadunk egy G gráfot, melyben pontosan akkor lesz k méretű klikk, ha ϕ kielégíthető. $G = (V, E)$ konstrukciója a következő. Minden $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ klóz esetén bevesszük V-be a v_1^r, v_2^r, v_3^r csúcsokat. A v_i^r és v_j^s csúcsok közé pontosan akkor húzunk élt, ha az alábbi két feltétel teljesül:

- $r \neq s$
- az l_i^r és l_j^s literálok nem egymás negáltjai.

ϕ ismeretében G nyilván megadható polinomiális időben. Nézzük meg a gráf konstrukcióját egy konkrét példán. Legyen

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

A kapott G gráf a 34.14. ábrán látható.

Meg kell mutatnunk, hogy ez a $\phi \mapsto G$ transzformáció visszavezetés. Tegyük fel először, hogy ϕ kielégíthető. Ekkor minden C_r-ben van legalább egy l_i^r literál, ami a kielégítő behelyettesítésnél 1-et vesz fel. Minden klózból egy ilyen „igaz” literált véve egy k elemű csúcshalmazt kapunk, amely nyilván klikket alkot G-ben.

Tegyük fel most, hogy G-ben van egy V' k-as klikk. Azonos hármasban lévő csúcsok nincsenek összekötve, így V' pontosan egy elemet tartalmaz klózonként. A $v_i^r \in V'$ csúcsokhoz tartozó l_i^r literálok értékét 1-nek adhatjuk meg, hiszen nem szerepel köztük változó a negáltjával együtt. Ily módon minden klóz értéke 1, így ϕ értéke is 1, tehát ϕ kielégíthető. (A klikkben nem szereplő csúcsokhoz tartozó változók értéke tetszőleges lehet.) ■

A 34.14. ábra példájában ϕ egy kielégítő behelyettesítésében $x_2 = 0$ és $x_3 = 1$. A megfelelő 3 elemű klikk az első klóz $\neg x_2$, a második klóz x_3 és a harmadik klóz x_3 literáljaihoz tartozó csúcsokból áll. Minthogy a klikk nem tartalmaz sem x_1-hez, sem $\neg x_1$-hez tartozó csúcsot, x_1 értéke lehet 0 és 1 is bármely kielégítő behelyettesítésben.

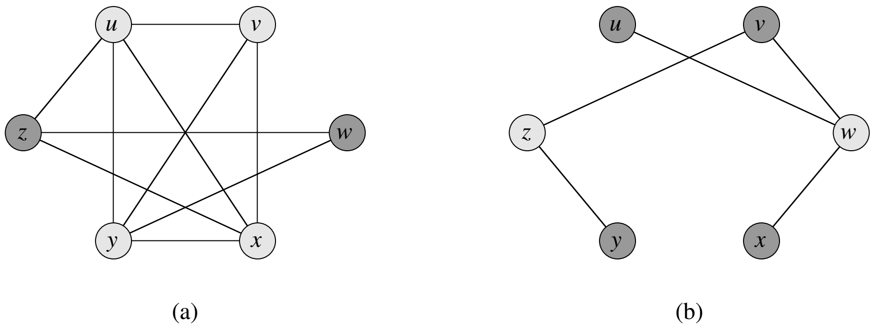
Vegyük észre, hogy a 34.11. tétel bizonyításakor 3-CNF egy tetszőleges esetét vezettük vissza KLIKK egy speciális szerkezettel bíró esetére. Ez alapján úgy tűnhet, hogy KLIKK NP-teljességét csak olyan gráfokra bizonyítottuk, melyekben a csúcsok olyan hármasokra oszthatók, melyeken belül nem mennek élek. Valóban igaz, hogy KLIKK NP-teljességét ilyen gráfokra bizonyítottuk, ebből azonban már következik, hogy a KLIKK probléma maga NP-teljes. Miért? Ha KLIKK-et meg tudjuk oldani általában, akkor nyilván meg tudjuk oldani minden speciális esetben is.

Nem volna ugyanakkor elegendő, ha 3-SAT-nak csak speciális eseteit vezetnénk vissza KLIKK eseteire. Miért? Előfordulhatna ugyanis az, hogy 3-SAT-nak csak „könnyű” eseteit vezetnénk vissza KLIKK eseteire, mely esetben a probléma, amelyet végeredményben visszavezetnénk KLIKK-re, nem lenne NP-teljes.

Érdemes azt is megfigyelni, hogy a visszavezetés csak 3-SAT eseteit használja, a megoldást magát nem. Polinomiális idejű visszavezetést ugyanis öreg hiba lenne arra a tudásra alapozni, hogy meg tudjuk mondani egy Boole-formuláról, hogy kielégíthető-e. Nem tudjuk ugyanis, hogy ezt a tudást hogyan szerezhetnénk meg polinom időben, sőt azt sem, hogy ez egyáltalán lehetséges-e.

34.5.2. A minimális lefedő csúcshalmaz probléma

Egy $G = (V, E)$ irányítatlan gráf ***lefedő csúcshalmazának*** nevezzük a $V' \subseteq V$ csúcshalmazt, ha minden $(u, v) \in E$ esetén $u \in V'$ vagy $v \in V'$ (esetleg mindkettő). Más szóval minden csúcs lefedi a hozzá tartozó éleket, G egy csúcshalmaza pedig akkor lefedő, ha minden élt legalább egy eleme lefed. Egy lefedő csúcshalmaz ***mérete*** a benne lévő csúcsok száma. Például a 34.15(b) ábra gráfjában $\{w, z\}$ egy kételemű lefedő csúcshalmaz.



34.15. ábra. KLIKK visszavezetése LEFEDÉS-re. **(a)** Egy hat csúcsú $G = (V, E)$ ($V = \{u, v, w, x, y, z\}$) irányítatlan gráf a $V' = \{u, v, x, y\}$ klikkel. **(b)** A $\overline{G}$ gráfban a $V - V' = \{w, z\}$ csúcshalmaz lefedi az éleket.

A ***minimális lefedő csúcshalmaz probléma:*** keressük meg egy adott gráf minimális lefedő csúcshalmazát. Döntési problémaként megfogalmazva: van-e egy adott gráfnak k elemű lefedő csúcshalmaza. A megfelelő nyelv:

$$\text{LEFEDÉS} = \{\langle G, k\rangle\ : \text{a } G \text{ gráfnak van } k \text{ méretű lefedő csúcshalmaza.}\}$$

34.12. tétel. LEFEDÉS NP-*teljes.*

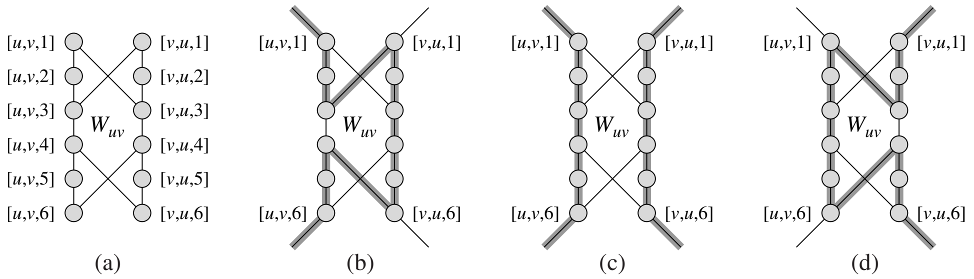
Bizonyítás. Először megmutatjuk, hogy LEFEDÉS $\in$ NP. Legyen adott a G gráf és a k egész szám. Tanúnak magát a $V' \subseteq V$ csúcshalmazt választjuk. Az ellenőrző algoritmus megnézi, hogy $|V'| = k$ teljesül-e, majd minden $(u, v) \in E$ élre megvizsgálja, hogy $(u \in V') \vee (u \in V')$ igaz-e. Mindez nyilván elvégezhető polinom időben.

A probléma NP-nehézségét a KLIKK $\leq_{\text{P}}$ LEFEDÉS visszavezetés segítségével mutatjuk meg, ehhez fel fogjuk használni a komplementer gráf fogalmát. A $G = (V, E)$ egyszerű gráf ***komplementere*** a $\overline{G} = (V, \overline{E})$ gráf, ahol $\overline{E} = \{(u, v) :\ (u, v) \notin E\}$. Más szóval $\overline{G}$ pontosan azokat az éleket tartalmazza, amik G-ből hiányoznak. A 34.15. ábra egy gráfot és a komplementerét mutatja, egyben illusztrálja a KLIKK-ről LEFEDÉS-re való visszavezetést.

A visszavezető algoritmus a KLIKK probléma egy $\langle G, k\rangle$ esetéből indul ki és meghatározza a $\overline{G}$ komplementert, ez könnyen végrehajtható polinom időben. Az algoritmus kimenete LEFEDÉS egy $\langle \overline{G}, |V| - k\rangle$ esete. A bizonyítást annak megmutatásával fejezzük be, hogy algoritmusunk valóban visszavezető függvényt számít ki, azaz a G gráfban akkor és csak akkor van k méretű klikk, ha a $\overline{G}$ gráfnak van $|V| - k$ méretű lefedő csúcshalmaza.

Tegyük fel, hogy G-ben $V' \subseteq V$ egy k méretű klikk. Azt állítjuk, hogy $V - V'$ lefedő csúcshalmaz $\overline{G}$-ban. Legyen (u, v) a $\overline{G}$ gráf tetszőleges éle. Ekkor $(u, v) \notin E$, ezért u és v közül legfeljebb az egyik tartozik V'-höz, hiszen bármely két V'-beli csúcs össze van kötve G-ben. Más szóval u és v közül legalább az egyik $(V - V')$-ben van, azaz $V - V'$ lefedi az (u, v) élt. Mivel (u, v) az $\overline{E}$ élhalmaz tetszőleges éle volt, $\overline{E}$ minden élét lefedi legalább egy $(V - V')$-beli csúcs, következésképp a $|V| - k$ méretű $V - V'$ halmaz lefedő $\overline{G}$-ban.

Megfordítva, tegyük fel, hogy $V - V'$ lefedő csúcshalmaz a $\overline{G}$ gráfban, ahol $|V'| = |V| - k$. Ekkor minden $u, v \in V$ esetén, ha $(u, v) \in \overline{E}$, akkor $u \in V'$ és $v \in V'$ közül legalább az egyik teljesül. Eszerint minden $(u, v) \in E$-re, ha $u, v \notin V'$, akkor $(u, v) \in E$. Azaz $V - V'$ klikk, és mérete $|V| - |V'| = k$. ■



34.16. ábra. A LEFEDÉS HAM-ra történő visszavezetése során használt segédgráf. A G gráf egy (u, v) éléhez tartozik a W_{uv} segédgráf a visszavezetés során előállított G' gráfban. **(a)** A segédgráf. **(b)–(d)** A vastagított utak az egyedüliek, melyek áthaladnak a segédgráf minden csúcsán, feltéve, hogy a segédgráfot G' többi részével csak az $[u, v, 1]$, $[u, v, 6]$, $[v, u, 1]$ és $[v, u, 6]$ csúcsokon keresztül kötjük össze.

Mivel LEFEDÉS NP-teljes, természetesen nem várható, hogy polinomiális algoritmust találunk rá. A 35.1. alfejezetben azonban bemutatunk egy polinomiális közelítő algoritmust, amely a minimálishoz képest legfeljebb kétszeres méretű lefedő csúcshalmazt ad meg.

Nem kell tehát azonnal feladni a reményt, ha egy probléma NP-teljesnek bizonyul; létezhet olyan polinomiális algoritmus, amely közel optimális megoldást talál. A 35. fejezet számos közelítő algoritmust mutat NP-teljes problémákra.

34.5.3. A Hamilton-kör probléma

Visszatérünk a 34.2. alfejezetben definiált Hamilton-kör problémához.

34.13. tétel. HAM NP-*teljes.*

Bizonyítás. Először megmutatjuk, hogy HAM $\in$ NP. A $G = (V, E)$ gráf esetén a tanú egy $|V|$ csúcsból álló sorozat. Az ellenőrző algoritmus megvizsgálja, hogy a sorozat V permutációja-e, és hogy a szomszédos csúcsok (az első és az utolsó is szomszédosnak számít) össze vannak-e kötve. Ez az ellenőrzés nyilván végrehajtható polinom időben.

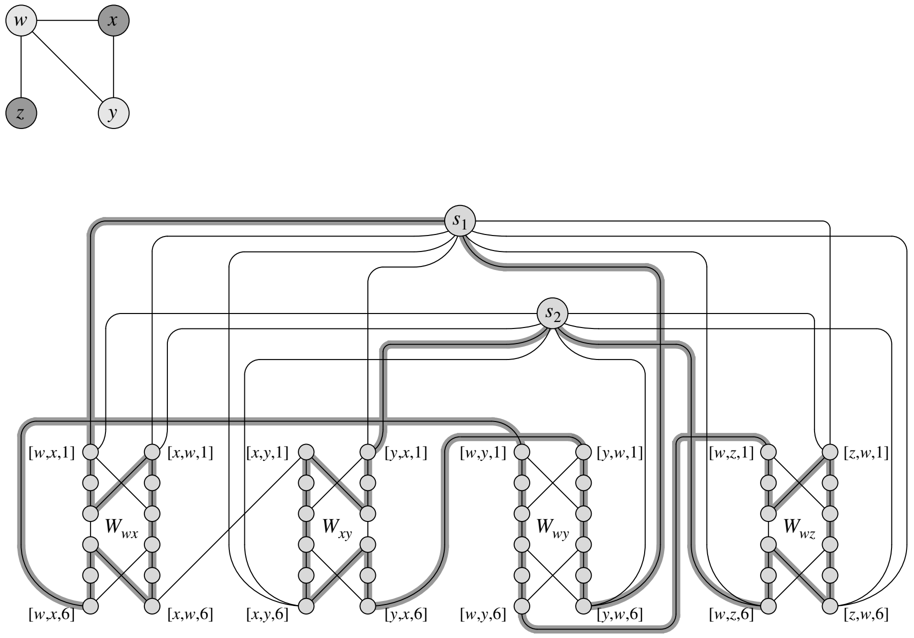
Most bebizonyítjuk, hogy a Hamilton-kör probléma NP-nehéz, visszavezetjük a LEFEDÉS problémát HAM-ra. Egy adott $G = (V, E)$ irányítatlan gráfhoz és adott k természetes számhoz konstruálunk egy olyan $G' = (V', E')$ irányítatlan gráfot, melynek pontosan akkor van Hamilton-köre, ha G-nek létezik k elemű lefedő csúcshalmaza.

A konstrukció egy speciális szerkezetű ***segédgráfon*** alapul, amelyet a 34.16(a) ábrán láthatunk. A G gráf minden (u, v) éléhez G' tartalmazni fogja a segédgráf egy példányát, melyet W_{uv}-vel fogunk jelölni. A W_{uv}-ben szereplő 12 csúcsot $[u, v, i]$-vel, illetve $[v, u, i]$-vel fogjuk jelölni, ahol $i = 1, 2, \ldots, 6$. A W_{uv}-ben szereplő 14 élt a 34.16(a) ábrán látható módon húzzuk be.

Az egyes segédgráfok csúcsai közül csak az $[u, v, 1]$, $[u, v, 6]$, $[v, u, 1]$ és $[v, u, 6]$ csúcsok lehetnek összekötve a G' gráf többi csúcsával, a többi 8 csúcsnak csak a segédgráfon belül vannak szomszédai. Ennek következtében G' bármely Hamilton-köre csak a 34.16(b)–(d) ábrákon látható három út valamelyikén haladhat keresztül W_{uv} csúcsain. Ha a Hamilton-kör az $[u, v, 1]$ csúcsban lép be a segédgráfba, akkor az $[u, v, 6]$ csúcsban kell kilépnie, és keresztülmegy vagy a segédgráf mind a 12 csúcsán (34.16(b) ábra), vagy az $[u, v, 1]$, $[u, v, 2]$, ..., $[u, v, 6]$ csúcsokon (34.16(c) ábra). Az utóbbi esetben a körnek természetesen vissza kell

(a)

(b)



34.17. ábra. LEFEDÉS egy esetének visszavezetése HAM egy esetére. **(a)** A G irányítatlan gráf éleit a w és y csúcsok lefedik. **(b)** A visszavezetés során kapott G' irányítatlan gráf. A vastagított élek alkotják a w és y csúcsokkal történő fedéshez tartozó Hamilton-kört.

térnie valamikor a segédgráfba, hogy átmehessen a $[v, u, 1]$, $[v, u, 2]$, …, $[v, u, 6]$ csúcsokon. Hasonlóképpen, ha a Hamilton-kör a $[v, u, 1]$ csúcsban lép be a segédgráfba, akkor a $[v, u, 6]$ csúcsban kell kilépnie, és keresztül kell mennie vagy a segédgráf mind a 12 csúcsán (34.16(d) ábra), vagy a $[v, u, 1]$, $[v, u, 2]$, …, $[v, u, 6]$ csúcsokon (34.16(c) ábra). A felsoroltakon kívül nem létezik más út, amely átmenne a segédgráf mind a 12 csúcsán.

A G' gráf többi (segédgráfban nem szereplő) csúcsai az úgynevezett ***választó*** csúcsok: $s_1, s_2, \ldots, s_k$. A választó csúcsokhoz csatlakozó éleket használjuk arra, hogy kiválasszuk a G gráf egy k elemű lefedő csúcshalmazát.

A segédgráfok élein kívül kétféle típusú él lehet G'-ben, melyek a 34.17. ábrán láthatók. Az első típusú élek arra szolgálnak, hogy a G gráf minden u csúcsához az u-hoz csatlakozó éleknek megfelelő segédgráfokat egy útra tudjuk felfűzni. Ezt a következőképpen valósítjuk meg. Minden $u \in V$-re tekintsük az u-val szomszédos éleket G-ben és rendezzük sorba őket tetszőlegesen. Legyenek az így kapott csúcsok $u^{(1)}, u^{(2)}, \ldots, u^{(d(u))}$, ahol $d(u)$ az u csúccsal szomszédos csúcsok száma. Létrehozunk G'-ben egy utat az u-val szomszédos éleknek megfelelő segédgráfokon keresztül oly módon, hogy hozzávesszük az E' élhalmazhoz az $([u, u^{(i)}, 6], [u, u^{(i+1)}, 1])$ éleket $i = 1, 2, \ldots, (k-1)$-re. A 34.17. ábrán például a w-vel szomszédos csúcsokat az x, y, z sorrendben nézzük, így a G' gráfba (amit a (b) ábrán láthatunk) a $([w, x, 6], [w, y, 1])$ és a $([w, y, 6], [w, z, 1])$ éleket vesszük be.

Az ily módon bevett élek arra a célra szolgálnak, hogy ha az u csúcs szerepel G egy lefedő csúcshalmazában, akkor meg tudjunk adni egy utat a G' gráf $[u, u^{(1)}, 1]$ csúcsából az $[u, u^{(d(u))}, 6]$ csúcsába, amely „lefedi” az u-hoz csatlakozó élekhez tartozó segédgráfokat. Ez azt jelenti, hogy bármely ilyen $W_{uu^{(i)}}$ segédgráfra az út tartalmazza vagy mind a 12 csúcsot (ha $u^{(i)}$ nincs benne a lefedő csúcshalmazban), vagy pontosan az $[u, u^{(i)}, 1]$, $[u, u^{(i)}, 2]$, …, $[u, u^{(i)}, 6]$ csúcsokat (ha $u^{(i)}$ is benne van a lefedő csúcshalmazban).

A másik típusú élek, amiket hozzáveszünk E'-höz az $[u, u^{(1)}, 1]$ és az $[u, u^{(d(u))}, 6]$ csúcsokat kötik össze az összes választó csúccsal, minden $u \in V$-re. Az E' élhalmazt tehát bővítjük az

$$(s_j, [u, u^{(1)}, 1]) \text{ és } (s_j, [u, u^{(d(u))}, 6])$$

élekkel minden $u \in V$-re és $j = 1, 2, \ldots, k$-ra.

Először is megmutatjuk, hogy G' mérete polinomiális G méretében, amiből már következik, hogy G'-t polinom időben meg tudjuk konstruálni G-ből, hiszen az egyes elemek (csúcsok, illetve élek) konstrukciója nyilván végrehajtható polinom időben. G minden éléhez létrehoztunk egy 12 csúcsú segédgráfot, G'-ben ezeken kívül szerepelnek még a választó csúcsok, így G' csúcsainak száma pontosan $12|E|+k \le 12|E|+|V|$, ami polinomiális G méretében. A G'-beli élek háromfélék lehetnek: a segédgráfokon belüliek, ezeknek a száma $14|E|$, segédgráfokat összekötők, ezekből minden $u \in V$ csúcsra éppen $d(u) - 1$ van, végül olyanok, melyek a segédgráfokat a választó csúcsokkal kötik össze, ilyen élből minden csúcsra $2k$ darab van. Így

$$\begin{aligned}
|E'| &= 14|E| + \left(\sum_{u \in V} d(u) - 1\right) + 2k|V| \\
&= 14|E| + (2|E| - |V|) + 2k|V| \\
&= 16|E| + (2k-1)|V| \le 16|E| + (2|V| - 1)|V|,
\end{aligned}$$

ami szintén polinomiális G méretében.

Most azt mutatjuk meg, hogy az átalakítás, aminek segítségével G'-t kaptuk G-ből, visszavezetés. Ehhez azt kell belátnunk, hogy G-ben pontosan akkor létezik k csúcsú lefedő halmaz, ha G'-ben létezik Hamilton-kör.

Tegyük fel, hogy a $G = (V, E)$ gráfnak létezik egy $V^* \subseteq V$ k méretű lefedő csúcshalmaza. Legyen $V^* = \{u_1, u_2, \ldots, u_k\}$. A 34.17. ábrán látható módon megadunk egy Hamilton-kört G'-ben, melyet a következő élek alkotnak.[8] Minden u_j-re az $\{([u_j, u_j^{(i)}, 6], [u_j, u_j^{(i+1)}, 1]) \ :\ 1 \le i \le d(u_j)\}$ élhalmaz, vagyis az u_j-hez csatlakozó éleknek megfelelő segédgráfokat összekötő élek, emellett az ezen segédgráfokban szereplő élek a 34.16(b)–(d) ábrákon látható módok valamelyike szerint, attól függően, hogy az élnek egy vagy két végpontja van benne V^*-ban, végül az

$$\{(s_j, [u_j, u_j^{(1)}, 1]) \ :\ 1 \le j \le k\},$$

$$\{(s_{j+1}, [u_j, u_j^{(d(u_j))}, 6]) \ :\ 1 \le j \le k-1\} \text{ és}$$

$$\{(s_1, [u_k, u_k^{(d(u_k))}, 6])\}$$

élhalmazok, minden $u_j \in V^*$-ra.

Nem nehéz ellenőrizni (és a 34.17. ábrán ezt érdemes is megtenni), hogy ezek az élek csakugyan kört alkotnak. A kör (mondjuk) s_1-ben kezdődik, végigmegy az u_1-hez csatlakozó élekhez tartozó segédgráfokon, ezt követően elmegy s_2-be, végigmegy az u_2-höz

[8] A köröket valójában csúcsok, és nem élek segítségével definiáljuk (lásd a B4. alfejezetet). Az egyszerűség kedvéért a Hamilton-kört most az éleivel adjuk meg.

csatlakozó élekhez tartozó segédgráfokon, és így tovább, egészen addig, míg vissza nem tér s_1-be. A kör minden segédgráfot egyszer vagy kétszer látogat meg, aszerint, hogy a megfelelő élnek egy vagy két végpontja van benne V^*-ban. Mivel V^* lefedő csúcshalmaz G-ben, G minden éle csatlakozik V^* valamelyik csúcsához, vagyis a kör az összes segédgráf összes csúcsán átmegy. Minthogy körünk a választó csúcsok mindegyikén is átmegy, valóban Hamilton-köre G'-nek.

Tegyük fel most, hogy a $G' = (V', E')$ gráfban létezik egy $C \subseteq E'$ Hamilton-kör. Azt állítjuk, hogy a k elemű

$$V^* = \{u \in V \ : \ \text{létezik } j \le k, \text{ hogy } (s_j, [u, u^{(1)}, 1]) \in C\} \tag{34.4}$$

csúcshalmaz lefedi G éleit. Ennek bizonyításához osszuk fel C-t olyan maximális utakra, melyek valamely s_i választó csúcsban kezdődnek, átmennek az $(s_i, [u, u^{(1)}, 1])$ élen valamely $u \in V$-re, és egy s_j választó csúcsban érnek véget anélkül, hogy bármely más választó csúcson átmennének. Nevezzük az ilyen utakat „fedőutaknak”. A G' gráf konstrukciójából látható, hogy az s_i választó csúcsban kezdődő fedőútnak, amely az $(s_i, [u, u^{(1)}, 1])$ élen megy át, át kell mennie az u csúcshoz G-ben csatlakozó összes élnek megfelelő segédgráfokon. Jelöljük ezt a fedőutat p_u-val. A V^* halmaz definíciója alapján $p_u \in V^*$. Tekintsünk egy tetszőleges, p_u által meglátogatott segédgráfot. Ez vagy W_{uv} vagy W_{vu} lesz, valamely $v \in V$-re. Az ennek megfelelő E-beli (u, v) élt a V^*-beli u és v csúcs is lefedi. Mivel minden segédgráfot meglátogat egy (vagy két) fedőút, a V^* csúcshalmaz valóban lefedi G minden élét. ■

34.5.4. Az utazóügynök probléma

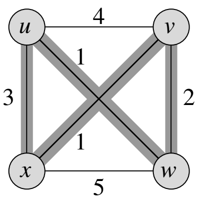
Az utazóügynök problémában (traveling salesman problem – TSP) – mely szoros kapcsolatban áll a Hamilton-kör problémával – szereplő ügynöknek n várost kell meglátogatnia tetszőleges sorrendben, de minél kisebb utazási költség mellett. (Ez negatív is lehet, ekkor az ügynöknek haszna van az útból, néha ez is előfordul.) Pontosabban: adott egy n csúcsú teljes gráf, melynek éleihez egész számokat rendelünk. (Ezek lesznek az utazási költségek az egyes városok között.) A feladat olyan Hamilton-kör megtalálása, melynek összköltsége (vagyis az érintett élekhez rendelt számok összege) minimális. A 34.18. ábrán látható esetben például (u, w, v, x, u) minimális (7) költségű Hamilton-kör. A megfelelő optimalizálási problémából nyert döntési problémához tartozó formális nyelv:

$$\begin{aligned} \text{TSP} = \{\langle G, c, k\rangle : \ & G = (V, E) \text{ teljes gráf}, \\ & c : V \times V \longrightarrow \mathbf{Z}, \\ & k \in \mathbf{Z} \\ & \text{és } G\text{-nek van legfeljebb } k \text{ költségű Hamilton-köre.}\} \end{aligned}$$

A következő tétel azt mutatja, hogy valószínűleg nincs gyors algoritmus az utazóügynök probléma megoldására.

34.14. tétel. TSP NP-*teljes.*

Bizonyítás. Először TSP $\in$ NP-t igazoljuk. Tanúnak egy megfelelő költségű Hamilton-kört használunk. Az ellenőrző algoritmus megnézi, hogy a megadott csúcssorozat valóban



34.18. ábra. Az utazóügynök probléma egy esete. A vastagított élek minimális költségű Hamilton-kört alkotnak.

Hamilton-kör-e (ilyen polinomiális algoritmust már láttunk), majd összeadja az élek költségeit és összehasonlítja k-val. Mindez nyilván végrehajtható polinom időben.

TSP NP-nehézségének bizonyításához megmutatjuk, hogy HAM $\leq_P$ TSP. Legyen $G = (V, E)$ HAM egy esete. Az ehhez tartozó TSP-beli eset konstrukciója a következő. Tekintsük a $G' = (V, E')$ teljes gráfot ($E' = \{(i, j) : i, j \in V$ és $i \neq j\}$). Legyenek az élek költségei

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

(Mivel G irányítatlan, ezért nem tartalmaz hurkot, és így $c(v, v) = 1$ minden $v \in V$ csúcsra.) Ekkor TSP egy esete $\langle G', c, 0\rangle$, ami könnyen előállítható polinomiális idő alatt.

Legyen ezek után a G-hez rendelt TSP-eset $\langle G, c, 0\rangle$. Ennek kiszámítása nyilván polinomiális idejű. Könnyen látható, hogy G-ben akkor és csak akkor van Hamilton-kör, ha G'-nek létezik legfeljebb 0 költségű Hamilton-köre. ■

34.5.5. A részletösszeg probléma

Ez az NP-teljes probléma aritmetikai jellegű. A ***részletösszeg problémánál*** adott egy $S \subseteq \mathbf{N}$ véges halmaz és egy $t \in \mathbf{N}$ szám. A kérdés az, hogy létezik-e olyan $S' \subseteq S$ halmaz, melyben az elemek összege t. Például ha $S = \{1, 2, 7, 14, 49, 98, 343, 686, 2409$ $2793, 16808, 17206, 117705, 117993\}$ és $t = 138457$, akkor az $S' = \{1, 2, 7, 98, 343, 686$ $2409, 17206, 117705\}$ részhalmaz ilyen.

A problémát szokás szerint nyelv formájában definiáljuk.

$$\text{RÉSZ-ÖSSZEG} = \{ \langle S, t\rangle \ : \ \text{létezik } S' \subseteq S : \sum_{s \in S'} s = t\}.$$

Mint minden aritmetikai problémánál, itt is fontos észben tartani, hogy a szabványos kódolás során az input egész számokat kettes számrendszerben adjuk meg. Ezt figyelembe véve meg tudjuk mutatni, hogy a részletösszeg problémára valószínűleg nincs gyors algoritmus.

34.15. tétel. RÉSZ-ÖSSZEG NP-*teljes.*

Bizonyítás. RÉSZ-ÖSSZEG $\in$ NP igazolásához egy $\langle S, t\rangle$ esethez az $S' \subseteq S$ részhalmazt választjuk tanúnak. $\sum_{s \in S'} s = t$ teljesülését egy alkalmas ellenőrző algoritmus nyilván meg tudja vizsgálni polinom időben.

Most megmutatjuk, hogy 3-SAT $\leq_P$ RÉSZ-ÖSSZEG. Legyen ϕ tetszőleges formula az $x_1, x_2, \ldots, x_n$ változókon, a $C_1, C_2, \ldots, C_k$ klózokkal, melyek mindannyian pontosan három literált tartalmaznak. A visszavezető algoritmus ϕ-hez megadja a RÉSZ-ÖSSZEG probléma

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

34.19. ábra. 3-SAT visszavezetése RÉSZ-ÖSSZEG-re. A felhasznált 3-CNF-beli formula $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, ahol $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3), C_3 = (\neg x_1 \vee \neg x_2 \vee x_3), C_4 = (x_1 \vee x_2 \vee x_3)$. ϕ egy kielégítő behelyettesítése $\langle x_1 = 0, x_2 = 0, x_3 = 1\rangle$. A visszavezetés során kapott S halmaz a következő tízes számrendszerbeli számokból áll: 1001001, 1000110, 100001, 101110, 10011, 11100, 1000, 2000, 100, 200, 10, 20, 1, 2. A t szám értéke 1114444. Az $S' \subseteq S$ halmaz elemei világosabb színnel vannak kiemelve.

egy $\langle S, t\rangle$ esetét úgy, hogy ϕ pontosan akkor lesz kielégíthető, ha létezik S-nek olyan részhalmaza, melyben az elemek összege k. Két feltételezéssel fogunk élni a ϕ formulát illetően, anélkül, hogy ez az általánosság rovására menne. Először is feltesszük, hogy egyetlen klózban sem szerepel egy változó a negáltjával együtt, hiszen az ilyen klózok minden behelyettesítésre 1-et vesznek fel, így nem befolyásolják a formula kielégíthetőségét. Másodszor, feltesszük, hogy minden változó megjelenik legalább egy klózban, ellenkező esetben az értéke nem befolyásolja a formula kielégíthetőségét.

A visszavezető algoritmus két számot hoz létre minden változóhoz és minden klózhoz is, ezek együttesen alkotják majd az S halmazt. A számokat tízes számrendszerben fogjuk létrehozni úgy, hogy minden számnak $n + k$ jegye legyen és minden számjegy vagy egy változóhoz, vagy egy klózhoz tartozzon.

Az S halmaz és a t szám konstrukciója az alábbi módon történik (lásd 34.19. ábra). Az $n + k$ darab helyi értéket (vagyis a számjegyek pozícióit) megcímkézzük a klózokkal vagy a változókkal: az utolsó k helyi értéket (vagyis a k legkisebb helyi értéket) a klózokkal, az első n helyi értéket a változókkal címkézzük.

- A t számban a változókhoz tartozó helyi értékekre 1-est írunk, a klózokhoz tartozó helyi értékekre 4-est.
- Minden x_i változóhoz két számot, v_i-t és $v_i{}'$-t tesszük S-be. Mindkettőnél az 1-es számjegy áll az x_i címkéjű helyi értéken és 0 a többi változóval címkézett helyi értéken. Ha az x_i literál megjelenik a C_j klózban, akkor a v_i szám C_j címkéjű helyi értékére az 1-es számjegyet írjuk és 0-t írunk a v_i szám minden más, klózokkal címkézett helyi értékére.

Ha a $\neg x_i$ literál megjelenik a C_j klózban, akkor a v_i' szám C_j címkéjű helyi értékére az 1-es számjegyet írjuk és 0-t írunk v_i' minden más, klózokkal címkézett helyi értékére.

Az ily módon kapott összes v_i és v_i' számok mind különbözők. Miért? Ha $l \neq i$, akkor a v_i és v_i' számok különböznek a v_l és v_l' számoktól, hiszen eltérés van köztük az első n számjegyben. Másrészt v_i sem lehet egyenlő v_i'-vel egyetlen i-re sem, hiszen ellenkező esetben az x_i és $\neg x_i$ literálok pontosan ugyanazokban a klózokban jelennének meg, holott feltettük, hogy egyetlen klóz sincs, amelyben mindketten megjelennének és legalább egyiküknek valamelyik klózban meg kell jelennie, mivel ezt is feltettük.

- Minden C_j klózhoz is két számot, s_j-t és s_j'-t tesszük S-be. A C_j-vel címkézett helyi érték kivételével minden számjegy 0 mindkettőjük esetében, míg az s_j szám C_j címkéjű helyi értékén 1-es, az s_j' szám C_j címkéjű helyi értékén 2-es áll. Ezek az egészek „felesleg” változók, amelyeket arra használunk, hogy a klózokkal címkézett pozíciókhoz hozzáadva megkapjuk a 4 célértéket.

  A 34.19. ábra átnézése mutatja, hogy az s_j és s_j' számok nemcsak egymástól különböznek, hanem az s_i, s_i' számoktól is, ha $i \neq j$.

Vegyük észre, hogy minden egyes helyi értékre az összes S-beli szám ilyen helyi értékű számjegyeinek összege legfeljebb 6 lehet (a klózokkal címkézett helyi értékek esetén három 1-es számjegy lesz a v_i és v_i' számok megfelelő helyi értékén és egy 1-es és egy 2-es az s_i és s_i' számok megfelelő helyi értékén). Következésképp akárhány S-beli számot adunk is össze, soha nem kell maradékot átvinni egyik helyi értékről a másikra az összeadás elvégzésekor.[9]

A visszavezetés nyilván végrehajtható polinomiális időben, hiszen az S halmaz $2n + 2k$ elemet tartalmaz, melyek mind $n + k$ jegyűek, s egy számjegy előállítása polinom időben végrehajtható, a t szám pedig $n + k$ konstans időben előállítható számjegyből áll.

Most megmutatjuk, hogy a 3-CNF-beli ϕ formula pontosan akkor kielégíthető, ha létezik olyan S' részhalmaza S-nek, melyben az elemek összege t. Tegyük fel először, hogy ϕ-nek létezik kielégítő behelyettesítése. Ha $x_i = 1$ ebben a behelyettesítésben, akkor vegyük be v_i-t az S' halmazba, ellenkező esetben vegyük be v_i'-t, $i = 1, 2, \ldots, n$-re. v_i és v_i' közül pontosan az egyiket vettük be tehát S'-be, így a változókkal címkézett helyi értékeken az S'-ben lévő elemek összege 1, ami azonos a t szám ilyen helyi értékein lévő számjegyekkel. Mivel a behelyettesítésben minden klóz értéke 1 kell legyen, minden klóz tartalmaz olyan literált, melynek az értéke 1. Így minden klózhoz egy, kettő vagy három olyan literál lesz, mely szerepel benne és az értéke 1. Mivel épp az ilyen literáloknak megfelelő v_i-k vagy v_i'-k kerülnek S'-be, a v_i és v_i' számok konstrukciója alapján minden klózzal címkézett helyi értéken 1,2 vagy 3 lesz az S'-ben lévő elemek összege. (A 34.19. ábrán például a $\neg x_1, \neg x_2$ és x_3 literálok értéke 1 a megadott kielégítő behelyettesítésben. A C_1 és C_4 klózok pontosan egyet tartalmaznak ezek közül, így v_1', v_2' és v_3 összege 1 a C_1 és C_4 helyi értékeken. A C_2 klóz pontosan kettőt tartalmaz az említett literálok közül, így a C_2 helyi értéken a v_1', v_2' és v_3 számok összege 2, míg a C_3 klóz a $\neg x_1, \neg x_2$ és x_3 literálok mindegyikét tartalmazza, tehát a C_3 helyi értéken a v_1', v_2' és v_3 számok összege 3.) Ahhoz, hogy a t számban szereplő 4-es számjegyet minden, klózokkal címkézett helyi értéken elérjük, már csak annyit kell tennünk, hogy minden klózzal címkézett helyi értékre a meglévő 1-es, 2-es vagy 3-as

[9] Természetesen tízes számrendszer helyett bármilyen, legalább hetes alapú számrendszer alkalmas lenne. A 34.19. ábrán látható számok hetes számrendszerben épp a részfejezet elején leírt S halmaz elemei és az ott megadott t szám lesznek.

számjegyet az s_j és s_j' számok közül a megfelelő(k) bevételével 4-re egészítjük ki. Precízebben: ha a C_j klózban az 1-es értéket felvevő literálok száma egy, akkor bevesszük S'-be az s_j és s_j' számokat, ha ez a szám kettő, akkor bevesszük S'-be s_j'-t, de s_j-t nem, ha pedig az 1-es értéket felvevő literálok száma három, akkor bevesszük S'-be s_j-t, de s_j'-t nem (a 34.19. ábrán S' a klózokhoz tartozó számok közül s_1-et, s_1'-t, s_2'-t, s_3-at, s_4-et és s_4'-t tartalmazza). Ily módon az S'-beli elemek összege a változókkal címkézett helyi értékeken 1 lesz, a klózokkal címkézett helyi értékeken pedig 4, azaz éppen a t számot kapjuk összegként.

Tegyük fel most, hogy létezik olyan $S' \subseteq S$ halmaz, melyben az elemek összege t. Az S' részhalmaz a v_i és v_i' számok közül pontosan egyet tartalmaz, minden $i = 1, 2, \ldots, n$ esetén, ellenkező esetben a változókkal címkézett helyi értékeken nem kaphatnánk 1-et összegként. Ha $v_i \in S'$, akkor legyen az x_i változó értéke 1, ha $v_i' \in S'$, akkor legyen x_i értéke 0. Azt állítjuk, hogy az így kapott behelyettesítés kielégíti ϕ-t. Ehhez be kell látnunk, hogy minden klóz értéke 1. Ennek bizonyításához vegyük észre, hogy a C_j-vel címkézett helyi értéken csak úgy tudjuk elérni a 4-es számjegyet, ha az S' halmaz tartalmaz olyan v_i vagy v_i' számot, amelynek C_j-vel címkézett helyi értékén 1-es áll, hiszen az s_j és s_j' számok együttes hozzájárulása ehhez a számjegyhez legfeljebb 3 lehet. Ha S' tartalmaz olyan v_i-t, melynek C_j címkéjű helyi értékén 1-es áll, akkor az x_i literál megjelenik a C_j klózban. Mivel $x_i = 1$ (hiszen $v_i \in S'$), a C_j klóz értéke 1. Hasonlóképp, ha S' tartalmaz olyan v_i'-t, melynek C_j címkéjű helyi értékén 1-es áll, akkor a $\neg x_i$ literál jelenik meg a C_j klózban. Mivel most $x_i = 0$ (hiszen $v_i' \in S'$), a C_j klóz értéke ismét 1. Beláttuk tehát, hogy minden klóz értéke 1 lesz, a bizonyítást ezzel befejeztük. ■

Gyakorlatok

34.5-1. Az ***izomorf részgráf probléma:*** adottak a G_1 és G_2 gráfok, kérdés, hogy G_1 izomorf-e G_2 egy részgráfjával. Mutassuk meg, hogy ez a probléma NP-teljes.

34.5-2. A ***0-1 egészértékű programozási probléma:*** adott az A $m \times n$-es egész mátrix és a b m-dimenziós egész vektor. Kérdés, hogy létezik-e $x \in \{0, 1\}^n$, melyre $Ax \leq b$. Bizonyítsuk be, hogy ez a probléma is NP-teljes. (*Útmutatás.* Vezessük vissza rá 3-SAT-ot.)

34.5-3. Az ***egészértékű lineáris programozási probléma*** az előző feladatban megadotthoz hasonló, a különbség annyi, hogy az x vektor koordinátái ezúttal nemcsak 0 és 1 lehetnek, hanem tetszőleges egész számok. Mutassuk meg, hogy ha a 0-1 egészértékű programozási probléma NP-nehéz, akkor az egészértékű lineáris programozási probléma is NP-nehéz.

34.5-4. Mutassuk meg, hogy a részletösszeg probléma unáris kódolás esetén polinom időben megoldható.

34.5-5. A ***halmaz-partíció probléma:*** adva van számok egy S halmaza, kérdés, hogy létezik-e $A \subseteq S$, melyre $\sum_{x \in A} x = \sum_{x \in S-A} x$. Mutassuk meg, hogy a halmaz-partíció probléma NP-teljes.

34.5-6. Mutassuk meg, hogy a Hamilton-út probléma NP-teljes.

34.5-7. A ***leghosszabb kör probléma:*** határozzuk meg egy gráf leghosszabb körének hosszát. Mutassuk meg, hogy ez a probléma NP-teljes.

34.5-8. A ***fél 3-SAT*** probléma az alábbi: adott egy ϕ 3-CNF-beli formula, melynek n változója és m klóza van, ahol m páros szám. Feladat annak eldöntése, hogy létezik-e olyan behelyettesítés, melyre ϕ klózainak pontosan a fele vesz fel 1-es értéket. Mutassuk meg, hogy ez a probléma NP-teljes.

Feladatok

34-1. Független halmaz

A $G = (V, E)$ gráfban egy $V' \subseteq V$ csúcshalmaz ***független,*** ha V' semelyik két csúcsa sincs összekötve. A ***független halmaz probléma:*** találjunk maximális méretű független csúcshalmazt egy adott G gráfban.

a. Fogalmazzuk meg a megfelelő optimalizálási és döntési problémát, és igazoljuk hogy az utóbbi NP-teljes. (*Útmutatás.* Vezessük vissza rá a KLIKK problémát.)

b. Tegyük fel, hogy rendelkezésünkre áll egy szubrutin, amely megoldja az a. pontban definiált döntési problémát. Adjunk algoritmust egy maximális méretű független ponthalmaz megtalálására. A futási idő legyen polinomiális $|V|$-ben és $|E|$-ben. (A szubrutin hívását egy lépésnek tekintjük.)

Noha a független halmaz döntési probléma NP-teljes, bizonyos speciális esetei polinom időben megoldhatók.

c. Adjunk minél jobb algoritmust a probléma megoldására olyan gráfok esetében, melyekben minden csúcs foka 2.

d. Adjunk minél jobb algoritmust a probléma megoldására páros gráfok esetében. (*Útmutatás.* Használjuk a 26.3. alfejezet eredményeit.)

34-2. Bonnie és Clyde

Bonnie és Clyde éppen most raboltak ki egy bankot. Zsákmányuk egy értékekkel teli táska, melynek tartalmát szeretnék elosztani. Az alábbi eshetőségek mindegyikére vagy adjunk egy polinomiális algoritmust, vagy bizonyítsuk be, hogy a probléma NP-teljes. A bemenet minden esetben a táskában lévő tárgyak n hosszúságú listája, az egyes tárgyak mellett fel van tüntetve azok értéke is.

a. A táska tartalma n pénzérme, kétféle típusból: egy részük x dolláros, a többi y dolláros. Az új tulajdonosok pontosan egyenlően szeretnének osztozni.

b. A táskában most is n érme van, ezek azonban tetszőleges kettő-hatvány dollárt érhetnek (1 dollár, 2 dollár, 4 dollár stb.). Bonnie és Clyde ez esetben is pontosan szeretné kettéosztani a zsákmányt.

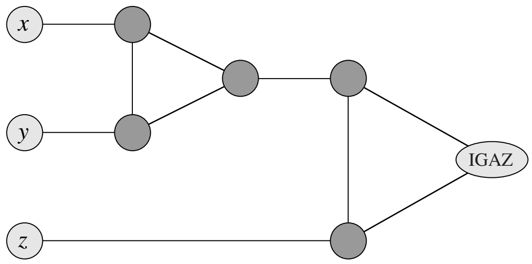
c. A táska n darab csekket rejt, melyeket valami különös véletlen folytán „Bonnie vagy Clyde veheti fel” felirattal láttak el. Az osztozkodás során ismét pontosan szeretnének felezni.

d. Megint ugyanazt az n csekket találják meg, de ezúttal nagyvonalúbb megoldással is beérik: úgy szeretnék elosztani a pénzt, hogy a két rész közti különbség ne legyen nagyobb 100 dollárnál.

34-3. Gráfszínezések

A $G = (V, E)$ gráf k-színezése egy $c : V \longrightarrow \{1, 2, \ldots, k\}$ függvény, amelyre $c(u) \neq c(v)$, ha $(u, v) \in E$. Más szavakkal: minden csúcshoz hozzárendeljük az $1, 2, \ldots, k$ színek valamelyikét úgy, hogy szomszédos csúcsok nem kaphatnak azonos színt. A ***gráfszínezés probléma:*** határozzuk meg a legkisebb k-t, amelyre létezik egy adott G gráfnak k-színezése.

a. Adjunk minél jobb algoritmust egy 2-színezhető gráf 2-színezésére.



34.20. ábra. Az $(x \vee y \vee z)$ klózhoz tartozó segédgráf a 34-3. feladatban.

b. Fogalmazzuk át a gráfszínezés problémát döntési problémává. Mutassuk meg, hogy a kapott döntési probléma pontosan akkor oldható meg polinom időben, ha az eredeti probléma megoldható polinom időben.

c. Legyen 3-SZÍN a 3-színezhető gráfok szabványos kódolásainak nyelve. Mutassuk meg, hogy ha 3-SZÍN NP-teljes, akkor a b. feladat döntési problémája is NP-teljes.

Most bebizonyítjuk, hogy 3-SZÍN NP-teljes. 3-SZÍN $\in$ NP nyilván igaz. Az alábbiakban visszavezetjük 3-SAT-ot 3-SZÍN-re. Az $x_1, x_2, \ldots, x_n$ változókon értelmezett, k klózból álló 3-CNF-beli ϕ formulához megadunk egy $G = (V, E)$ gráfot a következőképp: legyen V-ben egy csúcs minden változóhoz és minden változó negáltjához, 5 csúcs minden klózhoz, és végül 3 speciális csúcs: IGAZ, HAMIS, és PIROS. A gráf élei két típusból kerülnek ki: „literál” élek, melyek nem függnek a klózoktól, és „klóz” élek (ezek – mint sejthető – függenek a klózoktól). A literál élek összekötik a speciális csúcsokat egymással, x_i-t $\neg x_i$-vel és PIROS-sal, továbbá $\neg x_i$-t PIROS-sal, $i = 1, 2, \ldots,$ az $(x_i,\ \neg x_i,$ PIROS) hármasokon.)

d. Igazoljuk, hogy egy, a literál éleket tartalmazó gráf c 3-színezésében egy változó és a negáltja közül az egyik a c(IGAZ), a másik a c(HAMIS) színt kapja. Igazoljuk továbbá, hogy ϕ bármely behelyettesítésére a csak a literál éleket tartalmazó gráf 3-színezhető úgy, hogy az 1-et felvevő változókhoz tartozó szín c(IGAZ), a 0-t felvevőkhöz tartozó pedig c(HAMIS).

A klózok és a változók kapcsolatát most is segédgráfokkal biztosítjuk. Az $(x \vee y \vee z)$ klózhoz tartozó segédgráf látható a 34.20. ábrán. Minden klóz igényli az öt csúcs egy másolatát, melyeket az ábrán sötét színnel jelöltünk. Ezek a literálokat és a speciális IGAZ csúcsot kötik össze.

e. Igazoljuk, hogy ha a 34.20. ábra gráfjának x, y, z csúcsait vagy c(IGAZ)-ra, vagy c(HAMIS)-ra színezzük, akkor segédgráfunk pontosan abban az esetben lesz 3-színezhető, ha x, y, z valamelyikének a c(IGAZ) színt adjuk.

f. Fejezzük be 3-SZÍN NP-teljességének bizonyítását.

34-4. Ütemezés: profitok és határidők

Tegyük fel, hogy van egy gépünk, amivel az $a_1, a_2, \ldots, a_n$ feladatokat szeretnénk elvégezni. Az a_j feladat elvégzéséhez t_j időegység szükséges, befejezésének határideje d_j, a realizálható nyereség pedig p_j. A gépen egyszerre csak egy feladatot tudunk végezni és a feladatokat nem lehet megszakítani. A p_j nyereséget akkor söpörhetjük be, ha az a_j feladatot a d_j határidőre elvégezzük. Ha késünk, akkor nem termelődik nyereség. Feladatunk olyan ütemezés elkészítése, amelyben az összes feladatot elvégezzük (nem feltétlenül határidőre), és amely a lehetséges maximális nyereséget termeli.

a. A feladat nyilván egy optimalizálási probléma. Fogalmazzuk meg a megfelelő döntési problémát.

b. Mutassuk meg, hogy a kapott döntési probléma NP-teljes.

c. Adjunk polinomiális algoritmust a döntési problémára abban az esetben, amikor a feladatok elvégzéséhez szükséges idők 1 és n közötti egész számok. (*Útmutatás.* Használjunk dinamikus programozást.)

d. Adjunk polinomiális algoritmust az optimalizálási problémára abban az esetben, amikor a feladatok elvégzéséhez szükséges idők 1 és n közötti egész számok.

Megjegyzések a fejezethez

Garey és Johnson könyve [110] az NP-teljesség elméletének kiváló összefoglalását adja; a téma részletes tárgyalása mellett az 1979-ig NP-teljesnek bizonyult problémák széles skáláját is bemutatja. A 34.13. tétel bizonyítása ebből a könyvből való, csakúgy, mint az NP-teljes problémák előfordulási helyeinek felsorolása a 34.5. alfejezet elején. Johnson 1981 és 1992 között a *Journal of Algorithms* hasábjain tudósított az NP-teljes problémákkal kapcsolatos új fejleményekről. Hopcroft, Motwani és Ullmann [153], Lewis és Papadimitriou [204], Papadimitriou [236], illetve Sipser [279] művei jó bevezetést adnak az NP-teljesség elméletébe a bonyolultságelmélet összefüggésében. Aho, Hopcroft és Ullmann [5] ezen túlmenően jó néhány visszavezetést is ad, köztük a lefedési probléma HAM-ra való visszavezetését.

A P osztályt egymástól függetlenül bevezette 1964-ben Cobham [64] és 1965-ben Edmonds [84] is. Edmonds bevezette az NP osztályt is, és megfogalmazta a P $\neq$ NP sejtést. Az NP-teljesség fogalma Cooktól származik (1971-ből) [67], aki az első NP-teljességi bizonyításokat is adta (SAT-ra és 3-SAT-ra). A fogalmat tőle függetlenül felfedezte Levin is [203], aki egy parkettázási feladat NP-teljességét bizonyította. A visszavezetési módszer Karp [173] nevéhez fűződik (1972). Az ő cikkében szerepel a KLIKK probléma, a lefedési probléma és a Hamilton-kör probléma NP-teljességének első bizonyítása. Azóta problémák százai bizonyultak NP-teljesnek. A Karp hatvanadik születésnapja tiszteletére rendezett összejövetelen Papadimitriou megjegyezte: „Minden évben körülbelül 6000 cikk születik, amelynek címében, összefoglalójában vagy a kulcsszavak között szerepel az *NP-teljes* kifejezés. Ez több, mint ahányszor az *adatbázis*, *operációs rendszer*, *fordító*, *szakértő* vagy *neurális hálózat* kifejezések bármelyike előfordul."

A legújabb bonyolultságelméleti kutatások fényt derítettek számos, közelítő eljárások bonyolultságával kapcsolatos problémára. Új definíció is született az NP osztályra, a „valószínűségi alapon ellenőrizhető bizonyítások" használatával. Kiderült, hogy bizonyos problémák – mint például KLIKK, LEFEDÉS, TSP – esetében jó közelítő megoldások megadása is NP-nehéz. E témába nyerhet betekintést az olvasó Arora disszertációja [19] segítségével vagy az Arora és Lund által írt fejezetből [149], esetleg Mayr, Prömel és Steger könyve [214], illetve Arora [20] és Johnson [167] összefoglaló jellegű cikkei elolvasásával.

Irodalomjegyzék

[1] M. Abramowitz, I. A. Stegun (szerk.). *Handbook of Mathematical Functions*. Dover, 1965.

[2] G. M. Adelszon-Velszkij, E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

[3] L. M. Adleman, C. Pomerance, R. S. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, 117(1):173–206, 1983.

[4] A. Aggarwal, J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[5] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. 862

[6] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[7] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[8] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(1):213–223, 1990.

[9] R. K. Ahuja, J. B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

[10] R. K. Ahuja, J. B. Orlin, R. E. Tarjan. Improved time bounds for the maximum flow problem. *SIAM Journal on Computing*, 18(5):939–954, 1989.

[11] M. Ajtai, J. Komlós, E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[12] M. Ajtai, N. Megiddo, O. Waarts. Improved algorithms and analysis for secretary problems and generalizations. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 473–482. o., 1995.

[13] M. Akra, L. Bazzi. On the solution of linear recurrence equations. *Computational Optimization and Applications*, 10(2):195–210, 1998.

[14] N. Alon. Generating pseudo-random permutations and maximum flow algorithms. *Information Processing Letters*, 35(1):201–204, 1990.

[15] A. Andersson. Balanced search trees made simple. In *Proceedings of the Third Workshop on Algorithms and Data Structures*, *Lecture Notes in Computer Science* 709. kötete, 60–71. o. Springer-Verlag, 1993.

[16] A. Andersson. Faster deterministic sorting and searching in linear space. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, 135–141. o., 1996.

[17] A. Andersson, T. Hagerup, S. Nilsson, R. Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, 1998.

[18] T. M. Apostol. *Calculus*, 1. kötet. Blaisdell Publishing Company, 2. kiadás, 1967.

[19] S. Arora. *Probabilistic checking of proofs and the hardness of approximation problems*. PhD thesis, University of California, Berkeley, 1994. 862

[20] S. Arora. The approximability of NP-hard problems. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 337–348. o., 1998. 862

[21] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[22] S. Arora, C. Lund. Hardness of approximations. In D. S. Hochbaum (szerk.), *Approximation Algorithms for NP-Hard Problems*, 399-446. o. PWS Publishing Company, 1997.

[23] J. A. Aslam. A simple bound on the expected height of a randomly built binary search tree. Technical Report TR2001-387, Dartmouth College Department of Computer Science, 2001.

[24] M. J. Atallah (szerk.). *Algorithms and Theory of Computation Handbook*. CRC Press, 1999.

[25] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, 1999.

[26] S. Baase A. Van Gelder. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, 3. kiadás, 2000.

[27] E. Bach. Szóbeli közlés, 1989.

[28] E. Bach. Number-theoretic algorithms. In *Annual Review of Computer Science*, 4. kötet, 119–172. o. Annual Reviews, Inc., 1990.

[29] E. Bach J., Shallit. *Efficient Algorithms*, *Algorithmic Number Theory* 1. kötete. The MIT Press, 1996.

[30] D. H. Bailey, K. Lee, H. D. Simon. Using Strassen's algorithm to accelerate the solution of linear systems. *The Journal of Supercomputing*, 4(4):357–371, 1990.

[31] R. Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, 1972.

[32] R. Bayer, E. M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[33] P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, C. Pomerance. The generation of random numbers that are probably prime. *Journal of Cryptology*, 1:53–64, 1988.

[34] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[35] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[36] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, 80–86. o., 1983.

[37] M. A. Bender, E. D. Demaine, M. Farach-Colton. Cache-oblivious B-trees. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 399–409. o., 2000.

[38] S. W. Bent, J. W. John. Finding the median requires $2n$ comparisons. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, 213–216. o., 1985.

[39] J. L. Bentley. *Writing Efficient Programs*. Prentice Hall, 1982.

[40] J. L. Bentley. *Programming Pearls*. Addison-Wesley, 1986.

[41] J. L. Bentley, D. Haken, J. B. Saxe. A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(3):36–44, 1980.

[42] P. Billingsley. *Probability and Measure*. John Wiley & Sons, 2. kiadás, 1986.

[43] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[44] B. Bollobás. *Random Graphs*. Academic Press, 1985.

[45] J. A. Bondy, U. S. R. Murty. *Graph Theory with Applications*. American Elsevier, 1976. 830

[46] G. Brassard, P. Bratley. *Algorithmics: Theory and Practice*. Prentice Hall, 1988.

[47] G. Brassard, P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[48] R. P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20(2):176–184, 1980.

[49] M. R. Brown. *The Analysis of a Practical and Nearly Optimal Priority Queue*. PhD thesis, Computer Science Department, Stanford University, 1977. Technical Report STAN-CS-77-600.

[50] M. R. Brown. Implementation and analysis of binomial queue algorithms. *SIAM Journal on Computing*, 7(3):298–319, 1978.

[51] J. P. Buhler, H. W. Lenstra, Jr., C. Pomerance. Factoring integers with the number field sieve. In A. K. Lenstra, H. W. Lenstra, Jr. (szerk.), *The Development of the Number Field Sieve*, *Lecture Notes in Mathematics* 1554. kötete, 50–94. o. Springer-Verlag, 1993.

[52] J. L. Carter, M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[53] B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

[54] J. Cheriyan, T. Hagerup. A randomized maximum-flow algorithm. *SIAM Journal on Computing*, 24(2):203–226, 1995.

[55] J. Cheriyan and S. N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. *SIAM Journal on Computing*, 18(6):1057–1086, 1989.

[56] B. V. Cherkassky, A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[57] B. V. Cherkassky, A. V. Goldberg, T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73(2):129–174, 1996.

[58] B. V. Cherkassky, A. V. Goldberg, C. Silverstein. Buckets, heaps, lists and monotone priority queues. *SIAM Journal on Computing*, 28(4):1326–1346, 1999.

[59] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[60] K. L. Chung. *Elementary Probability Theory with Stochastic Processes*. Springer-Verlag, 1974.

[61] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[62] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, 1983.

[63] V. Chvátal, D. A. Klarner, D. E. Knuth. Selected combinatorial research problems. Technical Report STAN-CS-72-292, Computer Science Department, Stanford University, 1972.

[64] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 Congress for Logic, Methodology, and the Philosophy of Science*, 24–30. o. North-Holland, 1964. 862

[65] H. Cohen, H. W. Lenstra, Jr.. Primality testing and Jacobi sums. *Mathematics of Computation*, 42(165):297–330, 1984.

[66] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[67] S. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158. o., 1971. 862

[68] J. W. Cooley, J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[69] D. Coppersmith. Modifications to the number field sieve. *Cryptology*, 6(1):169–180, 1993.

[70] D. Coppersmith, S. Winograd. Matrix multiplication via arithmetic progression. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[71] T. H. Cormen. *Virtual Memory for Data-Parallel Computing*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[72] E. V. Denardo, B. L. Fox. Shortest-route methods: 1. Reaching, pruning, and buckets. *Operations Research*, 27(1):161–186, 1979.

[73] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[74] W. Diffie, M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[75] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(3):269–271, 1959.

[76] E. A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Mathematics Doklady*, 11(5):1277–1280, 1970.

[77] B. Dixon, M. Rauch, R. E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

[78] J. D. Dixon. Factorization and primality tests. *The American Mathematical Monthly*, 91(6):333–352, 1984.

[79] D. Dor, U. Zwick. Selecting the median. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, 28–37. o., 1995.

[80] A. W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, 1967.

[81] J. R. Driscoll, H. N. Gabow, R. Shrairman, R. E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11):1343–1354, 1988.

[82] J. R. Driscoll, N. Sarnak, D. D. Sleator, R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

[83] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, *EATCS Monographs on Theoretical Computer Science* 10. kötete. Springer-Verlag, 1987.

[84] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. 862

[85] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:126–136, 1971.

[86] J. Edmonds, R. M. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[87] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[88] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 3. kiadás, 1968.

[89] R. W. Floyd. Algorithm 97 (SHORTEST PATH). *Communications of ACM*, 5(6):345, 1962.

[90] R. W. Floyd. Algorithm 245 (TREESORT). *Communications of ACM*, 7(12):701, 1964.

[91] R. W. Floyd. Permuting information in idealized two-level storage. In R. E. Miller, J. W. Thatcher (szerk.), *Complexity of Computer Computations*, 105–109. o. Plenum Press, 1972.

[92] R. W. Floyd, R. L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[93] L. R. Ford, Jr., D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[94] L. R. Ford, Jr., S. M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66:387–389, 1959.

[95] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976.

[96] M. L. Fredman, J. Komlós, E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[97] M. L. Fredman, M. E. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, 345–354. o., 1989.

[98] M. L. Fredman, R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[99] M. L. Fredman, D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[100] M. L. Fredman, D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, 1994.

[101] H. N. Gabow. Path-based depth-first search for strong and biconnected components. *Information Processing Letters*, 74(3-4):107–114, 2000.

[102] H. N. Gabow, Z. Galil, T. Spencer, R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[103] H. N. Gabow, R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

[104] H. N. Gabow, R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[105] Z. Galil, O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.

[106] Z. Galil, O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, 1997.

[107] Z. Galil, J. Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983.

[108] I. Galperin, R. L. Rivest. Scapegoat trees. In *Proceedings of the 4st ACM-SIAM Symposium on Discrete Algorithms*, 165–174. o., 1993.

[109] M. R. Garey, R. L. Graham, J. D. Ullman. Worst-case analyis of memory allocation algorithms. In *Proceedings of the Fouth Annual ACM Symposium on Theory of Computing*, 143–150. o., 1972.

[110] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 862

[111] S. Gass. *Linear Programming: Methods and Applications*. International Thomson Publishing, 4. kiadás, 1975.

[112] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[113] A. George, J. W-H Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, 1981.

[114] E. N. Gilbert, E. F. Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.

[115] M. X. Goemans, D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[116] M. X. Goemans, D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum (szerk.), *Approximation Algorithms for NP-Hard Problems*, 144–191. o. PWS Publishing Company, 1997.

[117] A. V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1987.

[118] A. V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[119] A. V. Goldberg, S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998.

[120] A. V. Goldberg, É. Tardos, R. E. Tarjan. Network flow algorithms. In B. Korte, L. Lovász, H. J. Prömel, A. Schrijver (szerk.), *Paths, Flows, and VLSI-Layout*, 101–164. o. Springer-Verlag, 1990.

[121] A. V. Goldberg, R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[122] D. Goldfarb, M. J. Todd. Linear programming. In G. L. Nemhauser, A. H. G. Rinnooy-Kan, M. J. Todd (szerk.), *Handbook in Operations Research and Management Science, Vol. 1, Optimization*, 73–170. o. Elsevier Science Publishers, 1989.

[123] S. Goldwasser, S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[124] S. Goldwasser, S. Micali, R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[125] G. H. Golub, C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3. kiadás, 1996.

[126] G. H. Gonnet. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1984.

[127] R. C. Gonzalez, R. E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[128] M. T. Goodrich, R. Tamassia. *Data Structures and Algorithms in Java*. John Wiley & Sons, 1998.

[129] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[130] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[131] R. L. Graham, P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[132] R. L. Graham, D. E. Knuth, O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2. kiadás, 1994.

[133] D. Gries. *The Science of Programming*. Springer-Verlag, 1981.

[134] M. Grötschel, L. Lovász, A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[135] L. J. Guibas, R. Sedgewick. A dichromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, 8–21. o. IEEE Computer Society, 1978.

[136] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[137] Y. Han. Improved fast integer sorting in linear space. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, 793–796. o., 2001.

[138] F. Harary. *Graph Theory*. Addison-Wesley, 1969.

[139] G. C. Harfst, E. M. Reingold. A potential-based amortized analysis of the union-find data structure. *ACM SIGACT News*, 31(3):86–95, 2000.

[140] J. Hartmanis, R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965. 829

[141] M. T. Heideman, D. H. Johnson, C. S. Burrus. Gauss and the history of the Fast Fourier Transform. *IEEE ASSP Magazine*, 14–21. o., 1984.

[142] M. R. Henzinger, V. King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 664–672. o., 1995.

[143] M. R. Henzinger, V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[144] M. R. Henzinger, S. Rao, H. N. Gabow. Computing vertex connectivity: New bounds from old techniques. *Journal of Algorithms*, 34(2):222–250, 2000.

[145] N. J. Higham. Exploiting fast matrix multiplication within the level 3 BLAS. *ACM Transactions on Mathematical Software*, 16(4):352–368, 1990.

[146] C. A. R. Hoare. Algorithm 63 (PARTITION) and algorithm 65 (FIND). *Communications of ACM*, 4(7):321–322, 1961.

[147] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[148] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243–254, 1983.

[149] D. S. Hochbaum (szerk.). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997. 862

[150] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713–721, 1956.

[151] M. Hofri. *Probabilistic Analysis of Algorithms*. Springer-Verlag, 1987.

[152] J. E. Hopcroft, R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[153] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2. kiadás, 2001. 862

[154] J. E. Hopcroft, R. E. Tarjan. Efficient algorithms for graph manipulation. *Journal of the ACM*, 16(6):372–378, 1973.

[155] J. E. Hopcroft, J. D. Ullman. Set merging algorithms. *SIAM Journal on Computing*, 2(4):294–303, 1973.

[156] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979. 825, 832

[157] E. Horowitz, S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[158] E. Horowitz, S. Sahni, S. Rajasekaran. *Computer Algorithms*. Computer Science Press, 1998.

[159] T. C. Hu, M. T. Shing. Computation of matrix chain products. Part I. *SIAM Journal on Computing*, 11(2):362–373, 1982.

[160] T. C. Hu, M. T. Shing. Computation of matrix chain products. Part II. *SIAM Journal on Computing*, 13(2):228–251, 1984.

[161] T. C. Hu, A. C. Tucker. Optimal computer search trees and variable-length alphabetic codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.

[162] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[163] S. Huss-Lederman, E. M. Jacobson, J. R. Johnson, A. Tsao, T. Turnbull. Implementation of Strassen's algorithm for matrix multiplication. In *SC96 Technical Papers*, 1996.

[164] O. H. Ibarra, C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal ot the ACM*, 22(4):463–468, 1975.

[165] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.

[166] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[167] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974. 862

[168] D. S. Johnson. The NP-completeness column: An ongoing guide the tale of the second prover. *Journal of Algorithms*, 13(3):502–524, 1992.

[169] D. R. Karger, P. N. Klein, R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

[170] D. R. Karger, D. Koller, S. J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199–1217, 1993.

[171] H. Karloff. *Linear Programming*. Birkhäuser, 1991.

[172] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[173] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher (szerk.), *Complexity of Computer Computations*, 85–103. o. Plenum Press, 1972. 862

[174] R. M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, 34(1-3):165–201, 1991.

[175] R. M. Karp, M. O. Rabin. Efficient randomized pattern-matching algorithms. Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University, 1981.

[176] A. V. Karzanov. Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics Doklady*, 15:434–437, 1974.

[177] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997.

[178] V. King, S. Rao, R. E. Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17:447–474, 1994.

[179] J. H. Kingston. *Algorithms and Data Structures: Design, Correctness, Analysis*. Addison-Wesley, 1990.

[180] D. G. Kirkpatrick, R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(2):287–299, 1986.

[181] P. N. Klein, N. E. Young. Approximation algorithms for NP-hard optimization problems. In *CRC Handbook on Algorithms*, 341–3419. o. CRC Press, 1999.

[182] D. E. Knuth. *Fundamental Algorithms*, *The Art of Computer Programming* 1. kötete. Addison-Wesley, 1968. 3. kiadás, 1997.

[183] D. E. Knuth. *Seminumerical Algorithms*, *The Art of Computer Programming* 2. kötete. Addison-Wesley, 1969. 3. kiadás, 1998.

[184] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[185] D. E. Knuth. *Sorting and Searching*, *The Art of Computer Programming* 3. kötete. Addison-Wesley, 1973. 2. kiadás, 1998.

[186] D. E. Knuth. Big omicron and big omega and big theta. *ACM SIGACT News*, 8(2):18–23, 1976.

[187] D. E. Knuth, J. H. Morris Jr., V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[188] J. Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985.

[189] B. Korte, L. Lovász. Mathematical structures underlying greedy algorithms. In F. Gécseg (szerk.), *Fundamentals of Computation Theory*, *Lecture Notes in Computer Science* 117. kötete, 205–209. o. Springer-Verlag, 1981.

[190] B. Korte, L. Lovász. Structural properties of greedoids. *Combinatorica*, 3(3-4):359–374, 1983.

[191] B. Korte, L. Lovász. Greedoids – a structural framework for the greedy algorithm. In W. Pulleybank (szerk.), *Progress in Combinatorial Optimization*, 221–243. o. Academic Press, 1984.

[192] B. Korte, L. Lovász. Greedoids and linear objective functions. *SIAM Journal on Algebraic and Discrete Methods*, 5(2):229–238, 1984.

[193] D. C. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.

[194] D. W. Krumme, G. Cybenko, K. N. Venkataraman. Gossiping in minimal time. *SIAM Journal on Computing*, 21(1):111–139, 1992.

[195] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[196] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, 1976.

[197] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys (szerk.). *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[198] C. Y. Lee. An algorithm for path connection and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.

[199] F. T. Leighton, S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, 422–431. o., 1988.

[200] D. A. Lelewer, D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–296, 1987.

[201] A. K. Lenstra, Lenstra, Jr., H. W., M. S. Manasse, J. M. Pollard. The number field sieve. In A. K. Lenstra, H. W. Lenstra, Jr. (szerk.), *The Development of the Number Field Sieve*, *Lecture Notes in Mathematics* 1554. kötete, 11–42. o. Springer-Verlag, 1993.

[202] H. W. Lenstra, Jr.. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[203] L. A. Levin. Universal sorting problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. Oroszul. 862

[204] H. R. Lewis, C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, 2. kiadás, 1998. 825, 862

[205] C. L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1968.

[206] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975.

[207] L. Lovász, M. D. Plummer. *Matching Theory*, *Annals of Discrete Mathematics* 121. kötete. North-Holland, 1986.

[208] B. M. Maggs, S. A. Plotkin. Minimum-cost spanning tree as a path-finding problem. *Information Processing Letters*, 26(6):291–293, 1988.

[209] M. Main. *Data Structures and Other Objects Using Java*. Addison-Wesley, 1999.

[210] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989.

[211] C. Martínez, S. Roura. Randomized binary search trees. *Journal of the ACM*, 45(2):288–323, 1998.

[212] W. J. Masek, M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[213] H. A. Maurer, T. Ottmann, H.-W. Six. Implementing dictionaries using binary trees of very small height. *Information Processing Letters*, 5(1):11–14, 1976.

[214] E. W. Mayr, H. J. Prömel, A. Steger (szerk.). *Lectures on Proof Verification and Approximation Algorithms*, *Lecture Notes in Computer Science* 1367. kötete. Springer-Verlag, 1998. 862

[215] C. C. McGeoch. All pairs shortest paths and the essential subgraph. *Algorithmica*, 13(5):426–441, 1995.

[216] M. D. McIlroy. A killer adversary for quicksort. *Software Practice and Experience*, 29(4):341–344, 1999.

[217] K. Mehlhorn. *Sorting and Searching*, *Data Structures and Algorithms* 1. kötete. Springer-Verlag, 1984.

[218] K. Mehlhorn. *Graph Algorithms and NP-Completeness*, *Data Structures and Algorithms* 2. kötete. Springer-Verlag, 1984.

[219] K. Mehlhorn. *Multidimensional Searching and Computational Geometry*, *Data Structures and Algorithms* 3. kötete. Springer-Verlag, 1984.

[220] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[221] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[222] J. C. Mitchell. *Foundations for Programming Languages*. The MIT Press, 1996.

[223] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[224] L. Monier. *Algorithmes de Factorisation D'Entiers*. PhD thesis, L'Université Paris-Sud, 1980.

[225] L. Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theoretical Computer Science*, 12(1):97–108, 1980.

[226] E. F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, 285–292. o. Harvard University Press, 1959.

[227] R. Motwani, J. (S.) Naor, P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In D. S. Hochbaum (szerk.), *Approximation Algorithms for NP-Hard Problems*, 447–481. o. PWS Publishing Company, 1997.

[228] R. Motwani, P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[229] J. I. Munro, V. Raman. Fast stable in-place sorting with $O(n)$ data moves. *Algorithmica*, 16(2):151–160, 1996.

[230] J. Nievergelt, E. M. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

[231] I. Niven, H. S. Zuckerman. *An Introduction to the Theory of Numbers*. John Wiley & Sons, 4. kiadás, 1980.

[232] A. V. Oppenheim, R. W. Schafer, J. R. Buck. *Discrete-Time Signal Processing*. Prentice Hall, 2. kiadás, 1998.

[233] A. V. Oppenheim, A. S. Willsky, S. Hamid Nawab. *Signals and Systems*. Prentice Hall, 2. kiadás, 1997.

[234] J. B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(1):109–129, 1997.

[235] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1993.

[236] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. 862

[237] C. H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[238] M. S. Paterson, 1974. Publikálatlan előadás, Ile de Berder, France.

[239] M. S. Paterson. Progress in selection. In *Proceedings of the Fifth Scandinavian Workshop on Algorithm Theory*, 368–379. o., 1996.

[240] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000.

[241] S. Phillips, J. Westbrook. Online load balancing and network flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 402–411. o., 1993.

[242] J. M. Pollard. A Monte Carlo method for factorization. *BIT*, 15(3):331–334, 1975.

[243] J. M. Pollard. Factoring with cubic integers. In A. K. Lenstra, Jr. H. W. Lenstra (szerk.), *The Development of the Number Field Sieve*, *Lecture Notes in Mathematics* 1554. kötete, 4–10. o. Springer-Verlag, 1993.

[244] C. Pomerance. On the distribution of pseudoprimes. *Mathematics of Computation*, 37(156):587–593, 1981.

[245] C. Pomerance (szerk.). *Proceedings of the AMS Symposia in Applied Mathematics: Computational Number Theory and Cryptography*. American Mathematical Society, 1990.

[246] W. K. Pratt. *Digital Image Processing*. John Wiley & Sons, 2. kiadás, 1991.

[247] F. P. Preparata, M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[248] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[249] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[250] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[251] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Journal of the ACM*, 33(6):668–676, 1990.

[252] P. W. Purdom, Jr., C. A. Brown. *The Analysis of Algorithms*. Holt, Rinehart, and Winston, 1985.

[253] M. O. Rabin. Probabilistic algorithms. In J. F. Traub (szerk.), *Algorithms and Complexity: New Directions and Recent Results*, 21–39. o. Academic Press, 1976.

[254] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[255] P. Raghavan, C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[256] R. Raman. Recent results on the single-source shortest paths problem. *ACM SIGACT News*, 28(2):81–87, 1997.

[257] E. M. Reingold, J. Nievergelt, N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, 1977.

[258] H. Riesel. *Prime Numbers and Computer Methods for Factorization*. Progress in Mathematics. Birkhäuser, 1985.

[259] R. L. Rivest, A. Shamir, L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communication of the ACM*, 21(2):120–126, 1978. Lásd még az U.S. Patent 4 405 829 számú szabadalmat.

[260] H. Robbins. A remark on Stirling's formula. *American Mathematical Monthly*, 62(1):26–29, 1955.

[261] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6:563–581, 1977.

[262] S. Roura. An improved master theorem for divide-and-conquer recurrences. In *Proceedings of Automata, Languages and Programming, 24th International Colloquium, ICALP'97*, 449–459. o., 1997.

[263] Y. A. Rozanov. *Probability Theory: A Concise Course*. Dover, 1969.

[264] S. Sahni, T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[265] A. Schönhage, M. Paterson, N. Pippenger. Finding the median. *Journal of Computer and System Sciences*, 13(2):184–199, 1976.

[266] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[267] A. Schrijver. Paths and flows a historical survey. *CWI Quarterly*, 6(3):169–183, 1993.

[268] R. Sedgewick. Implementing quicksort programs. *Communications of the ACM*, 21(10):847–857, 1978.

[269] R. Sedgewick. *Algorithms*. Addison-Wesley, 2. kiadás, 1988.

[270] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[271] R. Seidel, C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4-5):464–497, 1996.

[272] J. Setubal, J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.

[273] C. A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis*. Prentice Hall, 2. kiadás, 2001.

[274] J. Shallit. Origins of the analysis of the Euclidean algorithm. *Historia Mathematica*, 21(4):401–419, 1994.

[275] M. I. Shamos, D. Hoey. Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, 208–215. o., 1976.

[276] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers and Mathematics with Applications*, 7(1):67–72, 1981.

[277] D. B. Shmoys. Computing near-optimal solutions to combinatorial optimization problems. In W. Cook, L. Lovász, P. Seymour (szerk.), *Combinatorial Optimization*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 20. kötete. American Mathematical Society, 1995.

[278] A. Shoshan, U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 605–614. o., 1999.

[279] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. 862

[280] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 1998.

[281] D. D. Sleator, R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[282] D. D. Sleator, R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[283] J. Spencer. *Ten Lectures on the Probabilistic Method*. Regional Conference Series on Applied Mathematics (No. 52). SIAM, 1987.

[284] D. A. Spielman, S.-H. Teng. The simplex algorithm usually takes a polynomial number of steps. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[285] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[286] G. Strang. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, 3. kiadás, 1988.

[287] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.

[288] T. G. Szymanski. A special case of the maximal common subsequence problem. Technical Report TR-170, Computer Science Laboratory, Princeton University, 1975.

[289] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[290] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[291] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, 1979.

[292] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[293] R. E. Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):306–318, 1985.

[294] R. E. Tarjan. Class notes: Disjoint set union. COS 423, Princeton University, 1999.

[295] R. E. Tarjan, J. van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, 1984.

[296] G. B. Thomas, Jr., R. L. Finney. *Calculus and Analytic Geometry*. Addison-Wesley, 7. kiadás, 1988.

[297] M. Thorup. Faster determinstic sorting and priority queues in linear space. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 550–555. o., 1998.

[298] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999.

[299] M. Thorup. On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000.

[300] R. Tolimieri, M. An, C. Lu. *Mathematics of Multidimensional Fourier Transform Algorithms*. Springer-Verlag, 2. kiadás, 1997.

[301] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, 75–84. o. IEEE Computer Society, 1975.

[302] J. van Leeuwen (szerk.). *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. Elsevier Science Publishers és The MIT Press, 1990.

[303] C. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, 1992.

[304] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 1996.

[305] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[306] R. M. Verma. General techniques for analyzing recursive algorithms with applications. *SIAM Journal on Computing*, 26(2):568–581, 1997.

[307] J. Vuillemin. A data structure for manipulating priority queues. *Journal of the ACM*, 21(4):309–315, 1978.

[308] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[309] M. S. Waterman. *Introduction to Computational Biology, Maps, Sequences and Genomes*. Chapman & Hall, 1995.

[310] M. A. Weiss. *Data Structures and Algorithm Analysis in C++*. Addison-Wesley, 1994.

[311] M. A. Weiss. *Algorithms, Data Structures and Problem Solving with C++*. Addison-Wesley, 1996.

[312] M. A. Weiss. *Data Structures and Problem Solving Using Java*. Addison-Wesley, 1998.

[313] M. A. Weiss. *Data Structures and Algorithm Analysis in Java*. Addison-Wesley, 1999.

[314] H. Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533, 1935.

[315] H. S. Wilf. *Algorithms and Complexity*. Prentice Hall, 1986.

[316] J. W. J. Williams. Algorithm 232 (HEAPSORT). *Communications of the ACM*, 7(6):347–348, 1964.

[317] S. Winograd. On the algebraic complexity of functions. In *Actes du Congrès International des Mathématiciens*, 3. kötet, 283–288. o., 1970.

[318] A. C.-C. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28(4):780–787, 1981.

[319] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. John Wiley & Sons, 1997.

[320] D. Zwillinger (szerk.). *CRC Standard Mathematical Tables and Formulae*. CRC Press, 30. kiadás, 1996.

Névmutató

Tárgymutató