9.

Néhány általános algoritmus-tervezési módszer

Minden felfedezett igazságot szabályként használva fel más igazságok felfedezésére ... számos olyan problémát oldottam meg, amelyet azelőtt igen nehéznek tartottam... RENÉ DESCARTES

Problémákat megoldani, algoritmusokat tervezni kreatív folyamat. Ezért nem is várhatunk biztos recepteket, amelyek segítségével mindig elboldogulunk. De nem is vagyunk teljesen fegyvertelenek: vannak olyan ötletek, módszerek, megközelítések, amelyek sok esetben sikeresnek bizonyultak hatékony algoritmusok tervezésében. Ezek a módszerek konkrét feladatok hatékony megoldásaiból kristályosodtak ki, *váltak szabállyá* a descartes-i értelemben. Olyan általános technikák ezek, amelyek feladatok széles körére alkalmazhatók. Már az eddigiek során is találkoztunk néhány ilyen elvvel.

Az **oszd meg és uralkodj** stratégia lényege, hogy a feladatot megpróbáljuk visszavezetni, „felvágni" ugyanezen feladat sokkal kisebb példányaira úgy, hogy ezek megoldásaiból az eredeti probléma megoldása gyorsan összerakható legyen.

Példák:

1. A bináris keresés során egy kérdéssel a keresési tartomány méretét a felére csökkentjük. A hatékony keresőfák alkalmazásakor is ilyesmi történik. Egy összehasonlítással visszavezetjük a feladatot egy jóval kisebb méretű hasonlóra (feleakkorára, vagy még kisebbre, mint pl. a B-fáknál).
2. Az összefésüléses rendezésnél két feleakkora méretű részfeladatra vágjuk az eredetit. Ezek megoldásából gyorsan adódik – összefésüléssel – a globális megoldás.

3. A gyorsrendezés partíciós lépésének a célja a probléma hatékony kettévágása.

Az oszd meg és uralkodj elv különösen akkor tud eredményes lenni, ha a vágás eléggé egyenletes, más szóval a kapott részek *nagyjából egyenlő méretűek*. Jól mutatja ezt a gyorsrendezés, ami akkor bizonyul lassúnak, ha a partíciós lépésekben kapott vágások nem eléggé egyenletesek. Azt is érdemes meggondolni, hogy az 1. és 2. példák módszereinek elemzésében, az igen kedvező korlátok levezetésében fontos szerepet játszott a vágások egyenletessége.

A **mohó módszer** néhány alkalmazásával is találkoztunk már. Az elv valami olyasmit mond, hogy a feladat megoldása során a következő lépés legyen a lehető legjobb a rendelkezésre állók közül; itt a *legjobb* valami egyszerű, könnyen ellenőrizhető kritérium szerint értendő. Úgy is mondhatjuk, hogy mindig valamiféle „helyi", messze nem a teljes képet figyelembe vevő optimumot választunk, és reméljük, hogy ezek a választások összességükben jó megoldáshoz vezetnek. Ezt az egyszerűségre törekvést két nem teljesen független tényező motiválja. Egyrészt a teljes kép általában túl bonyolult ahhoz, hogy egy pillantásra megértsük. Másfelől egyszerűbb szempontok alapján könnyebben, gyorsabban tudunk választani. Láttunk szép példákat olyan esetekre, amikor a mohó megközelítés megalapozott – ide sorolhatók Dijkstra, Kruskal és Prim módszerei. Mindhárom módszernél megfigyelhető, hogy egyszerű mohó választások sorozatával érik el a kívánt megoldást.

Természetesen ennek, és a többi tervezési szemléletnek is megvannak a korlátai. A mohó módszer kritikájaként gyakran említik a következő metaforát: képzeljük el, hogy a célunk a Mount Everest megmászása. Azt a mohó megközelítést alkalmazzuk, hogy a következő lépésünket mindig abba az irányba tesszük, amerre legmeredekebb az emelkedés. Világos, hogy ily módon legtöbbször valami kis dombtetőn akadunk el. Mindez arra mutat, hogy a módszer alkalmazhatóságát, az általa elérhető eredmény minőségét körültekintően meg kell vizsgálni.

A gyorsrendezés meggyőzően példázza a **véletlent használó módszerek** erejét. Az algoritmus gyakorlati sikerének a kulcsa a partícionáló elem véletlen választása. Az ilyen módszerekkel – kiemelkedő fontosságuk miatt – részletesebben foglalkozunk ebben a fejezetben.

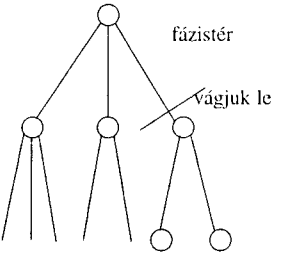
Eddig példaként csupa hatékonyan kezelhető problémát említettünk. A bemutatásra kerülő további technikák nehéz problémák kezelésére is alkalmasak. Mit várhatunk az ilyen esetekben? Szélsebes megoldásokat nem, hiszen nehéz feladatokról van szó. Például a tudomány mai állása szerint kevés a remény, hogy NP-nehéz feladatok megoldására igazán hatékony (polinom idejű) módszereket találjunk. Mégis, számos gyakorlati probléma megköveteli, hogy ilyen (vagy még nehezebb) feladatokra is adjunk valamilyen, természetesen minél jobb megoldó módszert. Lehetséges és gyakran értelmes célkitűzés az, hogy a kézenfekvő, minden lehetőséget végignéző (igen gyakran butának, bambának, stb. aposztrofált)

módszereknél jobbat találjunk. Másféle viszonyulást jelentenek a **közelítő módszerek**, melyek akkor jönnek szóba, ha a pontos megoldásnál kevesebbel is beérjük. Az **elágazás és korlátozás** elnevezésű módszert kifejezetten a nehéz feladatok esetén használják. A **dinamikus programozás** könnyű és nehéz problémákra egyaránt hasznos lehet. Végezetül itt tárgyaljuk a **prekondícionálás** módszerét. Ez leggyakrabban afféle munkaszervezési ötletnek tekinthető, amivel egy feladat egyszerre több példányának a megoldását lehet gyorsítani.

9.1. Elágazás és korlátozás

A módszer angol neve *branch-and-bound*. Olyan esetekben szokás alkalmazni, amikor a megoldandó feladat természetes módon vezet egy óriási gyökeres irányított fa teljes, vagy részleges bejáráshoz. A fát gyakran nevezik a feladat *fázisterének*. A fa általában olyan hatalmas, hogy nem érdemes (olykor nem is lehetne) egyszerre tárolni minden csúcsát. Vannak viszont szabályaink, amelyekkel egy csúcs leszármazottait elő tudjuk állítani. Ezt szokásos kifejezéssel generálásnak nevezzük. A feladat általában azt követeli, hogy a fa igen sok csúcsáról rendelkezzünk információval. Az elágazás és korlátozás módszere ezt az információgyűjtést igyekszik gazdaságosan megszervezni.

Első példaként nézzük egy sakkállás értékelésének a problémáját. A cél annak a meghatározása, hogy melyik fél – világos vagy sötét – áll jobban, és mi(k) a helyes lépés(ek). Képzeljünk el ehhez egy fát; ennek csúcsai sakkállások, megjelölve azzal, hogy melyik félen van a lépés sora. Egy csúcsból (állásból) él vezet a belőle egyetlen lépéssel megkapható állásokba. Több sakkozó program használja ezt a szemléletet. Az állás értékének megállapításához az alatta levő részfa csúcsait kellene értékelni. Ez azonban túl nagy, túlságosan időigényes vállalkozás lenne.



Az *elágazás* azt jelenti, hogy vesszük (generáljuk) az éppen vizsgált csúcs fiait, bizonyos esetekben egy korlátos mélységig a további leszármazottait is. A generált csúcsok vizsgálata alapján döntjük el, hogy merre menjünk tovább lefelé a

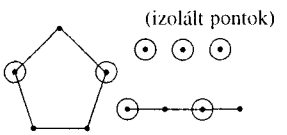
fában. Ebben fontos szerepet játszanak a *korlátozó heurisztikák*. Ezek a feladat tulajdonságaiból eredő megfontolások, amelyekkel igyekszünk minél több csúcsot és egyszerre azok leszármazottait is kizárni a további keresésből.

A sakk esetén az elágazás azt jelenti, hogy generáljuk a néhány lépésben elérhető állásokat, és azokat valahogy értékeljük. Korlátozó heurisztika lehet, hogy nem nézzük azokat az ágakat, ahol vezért vesztünk. Természetesen a komoly sakkprogramok többféle, a fentinél sokkal finomabb és bonyolultabb korlátozó heurisztikát használnak. A korlátozó heurisztikák alkalmazásával ágakat tudunk levágni a fáról, és csak az esélyes, a túlélő irányokkal foglalkozunk tovább.

A következőkben egy kevésbé bonyolult példát veszünk szemügyre: maximális független halmaz keresését gráfokban. A probléma eldöntendő változatáról tudjuk, hogy NP-teljes.

Legyen $G = (V, E)$ egy n pontú egyszerű irányítatlan gráf. A célunk az, hogy találjunk egy maximális méretű $H \subseteq V$ független halmazt. A kézenfekvő módszer az lenne, hogy végignézzük V összes részhalmazát, mindegyikről eldöntve, hogy független-e. Ez 2^n részhalmaz végigbogarászását jelentené. Próbáljuk meg ennél ügyesebben csinálni. Az esetek végignézésében alkalmazzuk a következő, igen egyszerű korlátozó heurisztikát:

Észrevétel: *Ha G-ben minden pont foka legfeljebb kettő, akkor a feladat lineáris időben megoldható, hiszen ekkor G izolált pontok, utak és körök diszjunkt uniója. Minden ilyen komponensben gyorsan találhatunk maximális független halmazt: komponensenként minden „második" pontot bevesszük a halmazba.*



Ezt használja az MF() algoritmus, aminek az inputja a G gráf.

MF(G)
1. Ha G-ben minden pont foka ≤ 2, akkor MF(G) az előbbi eljárás által adott maximális független halmaz, és a munkát befejeztük.
2. Legyen $x \in G$, $fok(x) \geq 3$.
   $S_1 :=$ MF($G \setminus \{x\}$)
   $S_2 := \{x\} \cup$ MF($G \setminus \{x$ és szomszédai$\}$).
3. Legyen S az S_1 és S_2 közül a nagyobb méretű, illetve akármelyik, ha $|S_1| = |S_2|$.
4. MF(G) $:= S$.

A módszer helyességének megértéséhez elég meggondolni, hogy egy maximális független halmaz vagy tartalmazza x-et, és ekkor a szomszédait biztosan nem (ilyen halmaz kerül S_2-be), vagy nem tartalmazza x-et (maximális méretű ilyen halmaz kerül S_1-be).

Nézzük ezután az eljárás költségét. Legyen $T(n)$ az MF(G)-n ($\mid V(G) \mid \leq n$) belüli MF hívások maximális száma, beleértve MF(G)-t magát is.

Állítás: *Van olyan c állandó, hogy $T(n) \leq c\gamma^n$, tetszőleges n természetes számra, ahol γ a $\gamma^4 - \gamma^3 - 1 = 0$ egyenlet pozitív gyöke ($\gamma \approx 1,381$).*

Bizonyítás: Legyen $t(n) := T(n) + 1$. Az eljárás szerkezetéből azonnal kapjuk, hogy $T(n) \leq T(n-1) + T(n-4) + 1$, ha $n > 4$. Innen adódik, hogy $t(n) \leq t(n-1) + t(n-4)$, ha $n > 4$. Ezután elég megmutatni, hogy $t(n) \leq c\gamma^n$ teljesül alkalmas c-vel, mert $T(n) < t(n)$. Indukciót használunk. A kívánt egyenlőtlenség $n < 5$-re alkalmas (elég nagy) c-vel elérhető. Ezután, ha $n \geq 5$, akkor alkalmazható az indukciós feltevés:

$$t(n) \leq t(n-1)+t(n-4) \leq c\gamma^{n-1}+c\gamma^{n-4} = c\gamma^{n-4}(\gamma^3+1) = c\gamma^{n-4}\gamma^4 = c\gamma^n.$$

Az utolsó előtti egyenlőségnél használtuk a γ definiáló egyenletét. □

A hívások előtti és utáni munka $O(n^d)$ alkalmas d-re. Ezt a költséget az MF(G) hívásra terheljük. Így becsülve az összes munka $O(n^d T(n)) = O(n^d \gamma^n) = O(1,381^n)$. Itt figyelembe vettük, hogy $\gamma < 1,381$. A kapott módszer tehát, bár a korlát itt is exponenciális, jóval gyorsabb, mint az összes eset végignézése. Ennél a példánál a fa csúcsainak az MF()-hívásokat tekinthetjük. Az MF(G) fiai a 2. lépésben megadott hívások. A G gráfhoz tartozó csúcsnak nincs további fia, ha az 1. lépésben végzünk (sikeres korlátozás), és két fia van különben.

Az elágazás és korlátozás módszerének másik alkalmazásaként vegyük szemügyre a 3 színnel való színezés feladatát. Az eldöntési probléma itt is NP-teljes. Mint előbb, legyen $G = (V, E)$ egy n pontú egyszerű irányítatlan gráf. Szeretnénk megadni G egy jó 3-színezését, ha egyáltalán létezik ilyen. A kézenfekvő módszer a csúcsok 3^n színezésének a végignézése lehetne. Korlátozó heurisztikát ad a következő észrevétel: ha kijelöltük az egyik színhez tartozó csúcsokat, mondjuk a pirosakat, akkor polinom időben ellenőrizhető, hogy ez a részleges színezés kiteljesíthető-e jó 3-színezéssé. Valóban, mindössze arról kell megbizonyosodni, hogy a piros pontok között nem fut él, továbbá, hogy a színezetlen pontok által feszített gráf kiszínezhető két színnel. Mindkét részfeladat legyűrhető polinom időben.

Feladat: Mutassuk meg, hogy az előbbi korlátozó heurisztika alkalmazásával a 3-színezés feladata megoldható $O(2^n n^c)$ időben, ahol $c > 0$ állandó.

9.2. Dinamikus programozás

A dinamikus programozás módszere gyakran használatos valamilyen numerikus paraméterektől függő érték optimumának a meghatározására. Lényege, hogy az optimumot (optimális megoldást) alkalmas kisebb részfeladatok optimumából (optimális megoldásából) próbáljuk előállítani. A dinamikus programozást használó algoritmusok vezérlési szerkezete gyakran emlékeztet egy *táblázat* szisztematikus (pl. sorról sorra haladó) kitöltésére. A táblázat kitöltését általában egy *rekurzív összefüggés* teszi lehetővé, ami alapján a bejárási sorrend szerint korábbi elemekből meghatározhatók a későbbiek. A kapott módszer költségét többnyire a kitöltendő táblázat mérete határozza meg. A rekurzív összefüggés megtalálásában sokszor a segítségünkre van az *optimalitás elve*, amivel a Bellman–Ford- és a Floyd-módszernél már találkoztunk.

A dinamikus programozásnak vannak erőteljes alkalmazásai a könnyű és nehéz problémák körében egyaránt. Működését néhány példán keresztül szeretnénk bemutatni.

1. Binomiális együtthatók számítása

Tegyük fel, hogy az $\binom{n}{k}$ binomiális együttható értékére vagyunk kíváncsiak. Lehetséges utat jelent a jól ismert

$$(*) \qquad \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

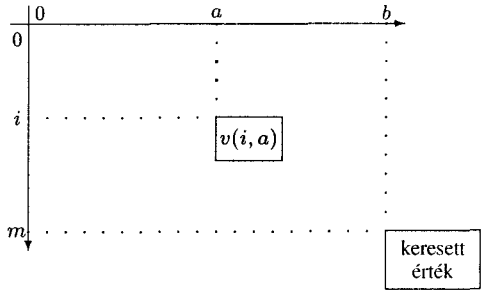
azonosság használata. Ennek segítségével a kisebb n értékektől a nagyobbak felé haladva adódnak a binomiális együtthatók: ha az összes $\binom{n-1}{j}$ értéket ismerjük ($0 \leq j \leq n-1$), akkor az $\binom{n}{k}$ alakú együtthatók egy-egy összeadással megkaphatók. Az elindulás lehetőségét az $\binom{m}{0} = \binom{m}{m} = 1$ értékek biztosítják. Tulajdonképpen a nevezetes Pascal-háromszög (egy háromszög alakú táblázat) kitöltéséről van szó. Az algoritmus csak összeadásokat használ, ezért gyakorlati szempontból is érdekes lehet olyan aritmetikai környezetben, ahol az összeadás sokkal gyorsabb, mint a szorzás.

A dinamikus programozás *alulról építkező*, a kisebb esetektől a nagyobbak felé menő stratégiája gyakran hatékonyabb alternatívát kínál a rekurzív eljárások felülről lefelé haladó felfogásával szemben. A binomiális együtthatók számítása jól mutatja ezt. A $(*)$ összefüggés alapján megírt rekurzív eljárás árnyoldala, hogy többször is kiszámítja ugyanazokat a (közbülső) binomiális együtthatókat. Például az $\binom{n}{k}$ számításakor az $\binom{n-2}{k-1}$ együtthatót kétszer is kiszámoljuk. A dinamikus programozás gondolatát követő módszer kiküszöböli ezeket az ismétlődéseket.

2. A Hátizsák probléma

Adottak az $s_1, \ldots, s_m$ súlyok, a b súlykorlát, a $v_1, \ldots, v_m$ értékek és a k értékkorlát. A kérdés, hogy van-e olyan $I \subseteq \{1, \ldots, m\}$ részhalmaz, melyre teljesül, hogy $\sum_{i \in I} s_i \leq b$ és $\sum_{i \in I} v_i \geq k$. Feltesszük még, hogy a szereplő mennyiségek mind pozitív egészek. A feladatról láttuk, hogy NP-teljes.

Itt most egy dinamikus programozást használó megoldást ismertetünk. Szeretnénk a feladatot visszavezetni kisebb hasonló problémákra. Ilyenkor gyakran segít, ha a feladatban megadott bizonyos paramétereket változónak tekintjük, és egy értelmes tartományban „futni hagyjuk". (Ez az ötlet erős eszköz a programozáselméletben, amikor ciklusinvariánst keresünk.) A mi esetünkben az m és a b lesznek ezek a paraméterek. Pontosabban fogalmazva legyen $v(i, a)$ a maximális elérhető érték az $s_1, \ldots, s_i$ súlyokkal, $v_1, \ldots, v_i$ értékekkel és a súlykorláttal megadott feladatra (mivel a maximális értéket keressük, nincs szükség értékkorlátokra). Ekkor $v(0, a) = v(i, 0) = 0$ tetszőleges a és i természetes számokra, és célunk a $v(m, b)$ mennyiség meghatározása, illetve annak eldöntése, hogy fennáll-e a $v(m, b) \geq k$ egyenlőtlenség. A feladat úgy is felfogható, hogy meg akarjuk határozni az $m + 1$ sorból és $b + 1$ oszlopból álló $[v(i, a)]$ táblázat (m, b) pozíciójú elemét.



A táblázat 0 indexű sorában, illetve oszlopában az értékek ismertek. Az érdekesebb helyeken levő számok meghatározásában segít a következő egyszerű összefüggés:

$$v(i, a) = \max\{v(i-1, a); v_i + v(i-1, a - s_i)\}.$$

Indoklásul megjegyezzük, hogy a jobb oldalon az első mennyiség az i-edik súlyt nem tartalmazó választások optimális értéke, a második pedig az i-edik súlyt tartalmazó választások optimális értéke (mindkét esetben a súlykorlát mellett). Itt is érvényesül az optimalitás elve: ha a $v(i, a)$ értéket adó kitöltésben az s_i súly szerepel, akkor a zsákban levő többi (az $s_1, s_2, \ldots, s_{i-1}$ közül kikerülő) súlynak optimális kitöltést kell adnia az $a - s_i$ súlykorláttal.

Az összefüggés alapján a táblázat kitölthető úgy, hogy vesszük rendre az $1, 2, \ldots, m$ indexű sorokat, ezeken belül pedig az a indexet növelve haladunk. A táblázatnak mb eleme van. A módszer logaritmikus költsége $O(bL)$, ahol L az input hossza. A módszer *nem polinomiális idejű*, mert b mérete $\lceil \log_2(b+1) \rceil$, az input hossza pedig

$$L = \sum_{i=1}^{m} (\lceil \log_2(s_i + 1) \rceil + \lceil \log_2(v_i + 1) \rceil) + \lceil \log_2(k+1) \rceil + \lceil \log_2(b+1) \rceil.$$

Másfelől már a táblázat mérete is legalább mb, ami L-hez képest exponenciálisan nagy is lehet. Ha viszont b nem túl nagy a többi input paraméterhez képest, akkor a módszer akár polinom idejű is lehet. Ezt most pontosabban is megfogalmazzuk.

Definíció: *A b egész* unáris ábrázolása: $0^b := 0 \cdots 0$ *(összesen b darab 0 egymás után).*

Definíció: *Egy feladat egy egész input paramétere* apró, *ha unárisan számítjuk bele az input hosszába.*

Ha a b paraméter apró, akkor az input méretéhez való hozzájárulása $|b|$ és nem $\lceil \log_2(b+1) \rceil$, mint a bináris megadás esetén. A b-t akkor érdemes aprónak tekinteni, amikor az unáris ábrázolása az inputban nem növelné meg az input méretének *nagyságrendjét*. Ez teljesül, ha $|b|$ felülről becsülhető az input más összetevői méretének egy polinomjával. Például a Hátizsák feladatnál ha $b \leq m^5$, akkor b-t szemlélhetjük úgy, mint egy apró paramétert. Ez természetesen nem jelenti azt, hogy b-t unárisan kell ábrázolnunk az inputban; arról van csupán szó, hogy a költségszámításnál az unáris hosszát vesszük figyelembe.

Következmény: *A Hátizsák probléma apró súlykorlát esetén megoldható polinom időben.*

Igazolásul elég annyi, hogy ekkor $L \geq b$, tehát a futási idő az input hosszának polinomjával becsülhető: $O(L^2)$.

Feladat: A Hátizsák probléma apró értékkorlát esetén megoldható polinom időben.

3. Legrövidebb utak gráfokban

Itt egy korábban megismert algoritmusra pillantunk ismét, felismerve rajta a dinamikus programozás gondolati jegyeit. Ez Floyd módszere az összes pontpár közötti távolság meghatározására: adott a C adjacencia mátrixával egy súlyozott élű

G irányított gráf, aminek csúcshalmaza $V = \{1, 2, ..., n\}$. Feltesszük, hogy G-ben nincs negatív összsúlyú irányított kör. A módszer $k = 1, 2, \ldots, n$-re sorban meghatározza az $F_k[i, j]$ értékeket, ahol az $F_k[i, j]$ a legrövidebb olyan $i \rightsquigarrow j$ utak hossza, melyeknek a közbülső pontjai az $1, 2, \ldots, k$ csúcsok közül valók. A számítás alapja az

$$F_k[i, j] := \min\{F_{k-1}[i, j], F_{k-1}[i, k] + F_{k-1}[k, j]\}$$

rekurzió. A kezdőértékeket a C mátrix adja: $F_0[i, j] := C[i, j]$. Itt is arról van szó, hogy az egyre összetettebb (egyre több közbülső pontot megengedő) optimumokat az egyszerűbbekből származtatjuk. Az algoritmus felfogható egy térbeli táblázat lapról lapra haladó kitöltésének. Legyen ugyanis $B[k, i, j] = F_k[i, j]$. Tulajdonképpen a B tömb értékeit határozzuk meg a $B[0, i, j]$ $(1 \leq i, j \leq n)$ laptól indulva. Floyd módszerének elegáns, helytakarékos vonása, hogy ez kivitelezhető egyetlen n-szer n-es tömb alkalmazásával (amit az algoritmus leírásakor $F[i, j]$-vel jelöltünk).

Feladat: Adott az $A[1 : n, 1 : n]$ kétdimenziós Boole-tömb. Adjunk $O(n^2)$ uniform költségű módszert az A-beli legnagyobb csupa egyesből álló négyzetek egyikének a megkeresésére. Pontosabban: határozzuk meg a legnagyobb olyan $0 \leq k < n$ egészet, melyhez vannak olyan i, j indexek, hogy az $A[i : i + k, j : j + k]$ résztömb minden eleme egyes.

9.3. Közelítő algoritmusok

Ne ránts tőrt, ha egy pofon is megteszi.
SKÓT KÖZMONDÁS

Előfordul, hogy egy nehéz algoritmikus problémával kerülünk szembe, de elegendő a céljainkhoz a megoldás (tipikusan valamilyen optimum) elég jó közelítése. Bizonyos esetekben igen egyszerű és hatékony módszerekkel kaphatunk az optimálishoz közeli eredményt. Érdemes lehet közelítő módszert használni akkor is, ha a problémára van ugyan gyors módszer, de egy a céloknak megfelelő közelítést még gyorsabban ki lehet számítani. Lássunk ezután néhány példát!

1. Sok független él kiválasztása egy gráfból
Legyen adott éllistával az n pontú és e élű $G = (V, E)$ irányítatlan gráf. Szeretnénk minél nagyobb méretű $E' \subseteq E$ független élhalmazt találni. (Az E' élhalmaz

független, ha semelyik két E'-beli élnek sincs közös végpontja.) Ez kezelhető feladat. Egy maximális méretű E' a korábban használt terminológiánk szerint maximális párosítás, ami polinom időben található. A páros gráfok esetét részletesen tárgyaltuk a gráfalgoritmusokról szóló részben. Maximális párosítás (tetszőleges, tehát nem feltétlenül páros gráfban is) építhető ugyan polinom időben, de az ismert algoritmusok elég időigényesek. Előfordulhat másfelől, hogy elegendő egy nagy független élhalmaz, ami esetleg nem maximális. Szerencsét próbálhatunk például a következő mohó ötlettel:

Kezdetben legyen $E' = \emptyset$. Vegyünk egy $e \in E$ élet, tegyük E'-be, majd töröljük E-ből azokat az éleket, amelyeknek van e-vel közös végpontjuk. A megmaradó élek közül tegyünk egy f élet E'-be, majd töröljük az f-fel közös végpontú éleket, stb. egészen addig, amíg E el nem fogy.

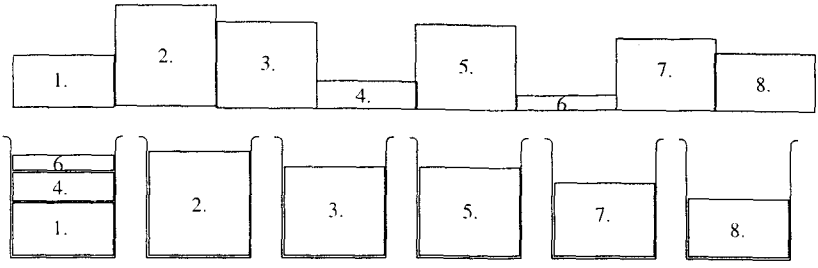
A fenti igen egyszerű módszer nagyon gyorsan, $O(n + e)$ költséggel megvalósítható. A kapott E' tovább már nem bővíthető független élhalmaz. Ha egy ilyen halmaz is megteszi, akkor felesleges az időigényesebb módszerekhez fordulni.

Feladat: Legyen a G-beli maximális független élhalmazok elemszáma k. Mutassuk meg, hogy a fenti algoritmus által kiválasztott E' élhalmaz mérete legalább $k/2$.

2. Ládapakolás

A következőkben a Ládapakolás feladat (angol nevén: Bin packing) néhány egyszerű és nevezetes közelítő megoldását mutatjuk be. Inputként adottak az $s_1, \ldots, s_m$ (racionális) súlyok, $0 \leq s_i \leq 1$. A cél a súlyok elhelyezése minél kevesebb 1 súlykapacitású ládába. Tudjuk, hogy a probléma döntési feladatként megfogalmazva NP-teljes.

Igen egyszerű és természetes közelítő algoritmus az FF-módszer (*first fit*, magyarul *első alkalmas*). Vegyünk először üres ládákat, és számozzuk meg őket az $1, 2, \ldots, k$ egészekkel. Ennyi előkészület után ismertethetjük a módszer általános lépését: tegyük fel, hogy az $s_1, \ldots, s_{i-1}$ súlyokat már elhelyeztük. Ekkor s_i kerüljön az első (legkisebb sorszámú) olyan ládába, amelybe még befér. Az ábra az FF-algoritmus működését illusztrálja.



Jelölje a Ládapakolás probléma egy I inputjára $OPT(I)$ az optimális (minimálisan elegendő), $FF(I)$ pedig az FF-módszer által eredményezett ládaszámot.

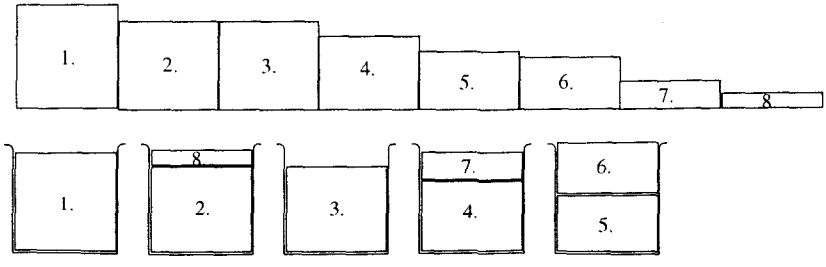
Világos, hogy $\lceil \sum_{i=1}^{m} s_i \rceil \leq OPT(I)$, továbbá $FF(I) \leq \lceil 2 \sum_{i=1}^{m} s_i \rceil$, hiszen az FF elhelyezésnél legfeljebb egy olyan használt láda lehet, amely nincs félig kitöltve. A két egyenlőtlenséget összefűzhetjük, figyelembe véve, hogy tetszőleges valós x-re $\lceil 2x \rceil \leq 2\lceil x \rceil$.

Következmény: *A probléma tetszőleges I inputjára teljesül, hogy* $FF(I) \leq 2OPT(I)$.

Valójában ennél sokkal több mondható (nem bizonyítjuk, nehéz).

Tétel (D. S. Johnson és munkatársai, 1976): *A probléma tetszőleges I inputjára teljesül, hogy* $FF(I) \leq \lceil \frac{17}{10} OPT(I) \rceil$. *Továbbá vannak tetszőlegesen nagy méretű I inputok, melyekre* $FF(I) \geq \frac{17}{10}(OPT(I) - 1)$.

Az FF tehát igen egyszerű és gyors módszer, amely eléri az optimum 17/10-ét. Egy kis javítással még jobb eredményt kaphatunk. Az FFD- (first fit decreasing, magyarul első alkalmas csökkenő) módszer receptje a következő: először rendezzük a súlyokat nem növő sorrendbe, utána alkalmazzuk az FF-módszert. Másképpen fogalmazva: az FF alkalmazásánál feltesszük, hogy $s_1 \geq s_2 \geq \ldots \geq s_m$. Az előző példa súlyai így eggyel kevesebb ládába bepakolhatók.



Általában érvényes a következő (nem bizonyítjuk, nagyon nehéz). Itt $FFD(I)$ jelöli az I inputon az FFD-módszer által adott ládaszámot.

Tétel (D. S. Johnson, 1973): *Tetszőleges I inputra teljesül, hogy $FFD(I) \leq \frac{11}{9}OPT(I) + 4$, és tetszőlegesen nagy méretű I inputok vannak, melyekre $FFD(I) \geq \frac{11}{9}OPT(I)$.*

Az FFD-módszerrel elérhetjük tehát az optimális érték mintegy $11/9$-ét. Az algoritmus rendkívül egyszerű, ugyanakkor jó közelítést ad. Megemlítjük még, hogy az FFD-nél valamivel jobb eredményt adó módszerek is vannak, ezek azonban egyszersmind jóval bonyolultabbak is.

3. Euklideszi utazó ügynök

Ez a probléma az *Utazó ügynök* feladat egy változata. Az n pontú K_n teljes gráf élein adott a nemnegatív értékű d súlyfüggvény. Erre teljesül a háromszög-egyenlőtlenség: tetszőleges különböző u, v, w csúcsokra érvényes a

$$d(u, w) \leq d(u, v) + d(v, w)$$

egyenlőtlenség (az euklideszi feltétel). A cél egy minimális összsúlyú Hamilton-kör keresése.

Feladat: Mutassuk meg, hogy az *Euklideszi utazó ügynök* eldöntési változata NP-teljes. (A H nyelvet, a Hamilton-kört tartalmazó gráfok nyelvét redukáljuk erre a nyelvre. Legyen a G gráf a H egy inputja. A G éleinek a súlya legyen 1, a G komplementerébe tartozó éleké pedig legyen 2.)

Jó közelítő megoldást kaphatunk a következők szerint. Vegyük először a súlyozott K_n gráf egy T minimális összsúlyú feszítőfáját. Jelölje s a T fa költségét (súlyát). Egy ilyen T fa hatékonyan kapható Kruskal vagy Prim módszerével. Helyettesítsük a T éleit egy-egy pár ellentétesen irányított éllel: az (u, v) irányítatlan él helyett az $u \to v$ és a $v \to u$ éleket vesszük. Az így kapott irányított gráf legyen T'. A T'-ben minden pontnak a be-foka megegyezik a ki-fokával, tehát van zárt Euler-bejárása. Egy ilyen bejárás gyorsan számítható. Írjuk le ezután a K_n csúcsait egy olyan sorrendben, ahogy az Euler-séta során végiglátogatjuk őket:

$$(*) \qquad u = v_1, v_2, \ldots, v_{2n-2}, v_{2n-1} = u.$$

Ebből a sétából a kitérők levágásával fogunk Hamilton-kört kapni. Először is megjegyezzük, hogy a listán K_n minden csúcsa szerepel, és a listán szomszédos két csúcs között él fut T-ben. Az is igaz, hogy a T egy éle pontosan kétszer fordul elő mint a körsétán szomszédos csúcsok közötti él. Ebből következik, hogy a $(*)$

körséta összsúlya éppen kétszerese a T összsúlyának, azaz $2s$. Ezután a $(*)$ mentén haladva megadjuk a K_n egy F Hamilton-körét. Legyen a kiindulópont u. Elég megadni azt a sorrendet, ahogy a K_n pontjait F-ben az u pontból indulva végigjárjuk. Tegyük fel, hogy az $u = u_1, u_2, \ldots, u_k$ pontokat már kiválasztottuk, és $u_k = v_i$. Legyen $j > i$ a legkisebb olyan index, melyre teljesül, hogy v_j nincs az $u_1, u_2, \ldots, u_k$ pontok között. Ha van ilyen j, akkor legyen $u_{k+1} := v_j$. Ha nincs ilyen j, akkor szükségképpen $k = n$ és a munkát befejezzük az $u_{n+1} := u$ választással. Minden csúcs szerepelni fog F-ben, hiszen a $(*)$ lista tartalmazza a K_n összes pontját. Az u-t kivéve minden pont egyszer szerepel, tehát F tényleg Hamilton-kör.

Mit mondhatunk az F Hamilton-kör összsúlyáról? Ez a mennyiség legfeljebb akkora mint a $(*)$ séta összsúlya, hiszen – az előző jelöléseket megtartva – a háromszög-egyenlőtlenség miatt

$$d(u_k, u_{k+1}) \leq d(v_i, v_{i+1}) + d(v_{i+1}, v_{i+2}) + \cdots + d(v_{j-1}, v_j).$$

Az F összsúlya ezért legfeljebb $2s$. Mennyire lesz ez jó közelítése a minimumnak? Egy Hamilton-körből egy élet elhagyva feszítőfát kapunk, aminek az összsúlya legalább s. Ebből következően bármely Hamilton-kör súlya is legalább s. A módszerünk tehát kettes szorzó erejéig közelíti az optimális értéket.

Megjegyzés: A módszer lelke a $(*)$ bejárás konstrukciója, amit a T' gráf segítségével vittünk végbe. Ez a lépés szemléltethető úgy, hogy a T fát lerajzoljuk a síkba, majd az u csúcstól elindulva végigsétálunk az ágai mellett. Képzeljük el, hogy a bal kezünkben egy ecsetet tartunk, és ezt a séta során végighúzzuk a T élein, a leveleknél visszafordulva a „másik oldalra". A séta akkor ér véget, amikor minden él mindkét oldalát befestettük. Ekkor éppen az u pontba érünk vissza. A sorrend, ahogy a gráf csúcsait elérjük, megfelel a T' egy Euler-körútjának.

A vázolt fabejáró módszer neve *Lindström-bejárás*. Bizonyos csúcsokat esetleg többször is meglátogat, de igen hatékonyan megvalósítható. Elsősorban a következő tulajdonsága miatt használják belső memóriabeli listák kezelésére.

Feladat: Legyen T egy bináris fa, u a gyökere. Mutassuk meg, hogy az u-ból induló Lindström-bejárás megvalósítható lineáris idejű, konstans munkaterületet használó algoritmussal (így pl. akkor is, ha nem használhatunk vermet, és nem írhatunk a fa csúcsaiba).

9.4. Véletlent használó módszerek

És mikor előállatta volna a Sámuel az Izráelnek minden nemzetségit, találák ki sors által az Benjámin nemzetségét. Azután az Benjámin nemzetségét előállatá, mindent az ő háza népével, és találák kivenni az Mátri nemzetségének sorsát, és azok közül találák az Sault, az Kisnek fiát... SÁMUEL ELSŐ KÖNYVE, 10: 20-21.

Itt olyan számításokról lesz szó, melyekben megengedünk valamiféle véletlen választásokat. Az ilyen algoritmusok furcsa sajátossága, hogy a bemenet és az algoritmus maga nem határozzák meg egyértelműen a tényleges lépéseket, a számítási időt, olykor magát a végeredményt sem. Mindezekért a bizonytalanságokért bőségesen kárpótol bennünket a véletlent használó (szokásos szakkifejezéssel: randomizált) módszerek hatékonysága és eleganciája.

Egy példával már találkoztunk. A gyorsrendezés ideje nagymértékben függ attól, hogy a partícionáló elemek mennyire vágják egyforma darabokra a tömböt. A partícionáló elemek véletlen választásával az esetek jó részében elég egyenletes vágást kapunk. Gyakorlati szempontból a gyorsrendezés randomizált változata a legjobb az ismert általános rendező módszerek közül. A randomizálás nem csak a könnyű feladatok megoldásakor jön szóba. Látni fogjuk, hogy így olyan problémák kezelésére kaphatunk hatékony módszereket, melyekre nem ismert gyors hagyományos algoritmus (Turing-gép, RAM program).

A randomizált módszerekkel szemben két érvet szokás felhozni. Az egyiket már érintettük: a bizonytalanság, a hiba lehetősége mintegy bele van tervezve ezekbe a módszerekbe. Mint látni fogjuk, ettől nem kell igazán félnünk. Gondoljunk arra, hogy a számítógépek, amelyeken a determinisztikus módszereink futnak, szintén nem mentesek a hiba lehetőségétől. Azt mondhatjuk, hogy egy bizonyos kis pozitív valószínűséggel hibásan működnek. A kérdés ezután már csak az, hogy le tudjuk-e szorítani erre a szintre a randomizált módszereink hibázási valószínűségét. (Sokkal ez alá menni nem érdemes, hiszen ezek a módszerek is a mi gyarló számítógépeinken futnak). A válasz a kérdésre sok érdekes esetben igenlő: a hibavalószínűség hatékonyan tetszőlegesen kicsivé tehető.

A másik ellenérv elvi szempontból súlyosabb. A randomizált módszerek elemzésénél, viselkedésük vizsgálatakor hallgatólagosan feltesszük, hogy valamiféle „igazán véletlen" bitek kellően hosszú sorozatával rendelkezünk, mégpedig általában nulla költséggel. Valójában nem világos, hogy mit tekinthetünk véletlen sorozatnak, és az még kevésbé, hogy nyerhetünk-e ilyet egyáltalán valamilyen természeti folyamatból. Nem célunk itt ezeket a filozófiába nyúló kérdéseket taglalni;

izgalmasak ugyan, de tudomásunk szerint kevés algoritmikus tanulsággal szolgálnak. Gyakorlati szempontból mindez sokkal biztatóbban néz ki: a randomizált módszerek nagyon jól működnek a rendelkezésre álló egyszerű, olcsó álvéletlen sorozatokkal. Ennek a széles körben megfigyelt kedvező jelenségnek az okát még nem érti a számítástudomány.

Az első példánk egy olyan feladat lesz, melyre nem ismert hatékony hagyományos algoritmus. Ez a probléma a polinomazonosságok tesztelése.

Probléma: Adott behelyettesítéssel egy n-változós $f \in \mathbb{Z}[x_1, \ldots, x_n]$ egész együtthatós polinom. Tudjuk, hogy $\deg f \leq d$. El akarjuk dönteni, hogy f azonosan nulla-e.

A feltétel, hogy f behelyettesítéssel adott, azt jelenti, hogy f-fel egyetlen dolgot tehetünk: értékeket adhatunk a változóinak, és ekkor valamilyen rendelkezésünkre álló eljárás megadja f értékét az adott behelyettesítésnél. Az eljárást úgy tekinthetjük, mint egy fekete dobozt. Bemenetként adunk neki egy egész komponensekből álló $\alpha = (\alpha_1, \ldots, \alpha_n)$ vektort; válaszul a doboz kiadja az $f(\alpha_1, \ldots, \alpha_n)$ értéket. Ez az első látásra furcsának tűnő helyzet könnyen előfordulhat. Megeshet például, hogy f olyan kifejezéssel adott, amit nem lehet gyorsan kifejteni (vagy más, céljainknak megfelelő módon átalakítani), ugyanakkor könnyű a változók adott értékeire f-et kiszámítani. Például tekintsük az

$$f(x_1, x_2, \ldots, x_{2n}) = (x_1 + x_2)(x_3 + x_4) \cdots (x_{2n-1} + x_{2n})$$

polinomot. Ezt kifejtve 2^n tagot kapunk, viszont egy helyettesítési érték n összeadással és $n-1$ szorzással megkapható. Hasonló a helyzet a következő, változókból álló determinánssal:

$$D = \det \begin{pmatrix} x_{11} & \ldots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \ldots & x_{nn} \end{pmatrix}.$$

A D-t kifejtve $n!$ tagot kapunk, ugyanakkor bármely helyettesítési értéke $O(n^3)$ aritmetikai művelettel megkapható (Gauss-elimináció). Mindkét polinom foka n, a behelyettesítés gyors és könnyű, a kifejtés nehéz, hiszen túl nagy a végeredményben a tagok száma.

Problémánkat megfogalmazhatjuk úgy is, hogy a d fok függvényében minél kevesebb behelyettesítéssel *tanút* szeretnénk arra, hogy $f \not\equiv 0$. Tanún egy $\alpha = (\alpha_1, \ldots, \alpha_n)$ egészekből álló vektort értünk, melyre $f(\alpha) \neq 0$.

A következő tétel annak pontos megfogalmazása, hogy ha $f \not\equiv 0$, akkor egy véletlenül választott α jó eséllyel tanú lesz. Ez szemléletesen nyilvánvaló. Képzeljük el az $n = 2$ esetet, amikor is f-nek két változója van. Ha $f \not\equiv 0$, akkor a

$\{(\beta_1, \beta_2) \in \mathbb{R}^2; f(\beta_1, \beta_2) = 0\}$ halmaz egy görbe a síkon. Világos, hogy a görbe a sík majdnem minden pontját elkerüli. Egy véletlen pont óriási eséllyel tanú lesz. A szokásos módon $Prob(A)$ jelöli az A esemény valószínűségét.

Tétel (J. Schwartz lemmája): *Ha* $\deg f \leq d$, *és* $\alpha_1, \ldots, \alpha_n$ *egyenletes eloszlású, egymástól független véletlen elemei az* $\{1, \ldots, N\}$ *számhalmaznak, akkor* $f \not\equiv 0$ *esetén* $Prob(f(\alpha) = 0) \leq \frac{d}{N}$.

Bizonyítás: Indukció n szerint. $n = 1$ esetén f-nek legfeljebb d gyöke van, így $Prob(f(\alpha_1) = 0) \leq \frac{d}{N}$. Legyen $n > 1$. Fejtsük ki f-et x_1 szerint, és legyen $y = (x_2, \ldots, x_n)$. Ezzel a jelöléssel

$$f(x) = h_0(y) + h_1(y)x_1 + \cdots + h_k(y)x_1^k,$$

ahol $k \leq d$ és $h_k \not\equiv 0$. Ha $f(\alpha) = 0$, akkor vagy az igaz, hogy $h_k(\alpha_2, \ldots, \alpha_n) = 0$, vagy pedig az, hogy $h_k(\alpha_2, \ldots, \alpha_n) \neq 0$ és $f(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$. Mit mondhatunk e két nemkívánatos esemény valószínűségéről?

Tekintetbe véve, hogy $\deg h_k \leq d - k$, az indukciós feltevés szerint $Prob(h_k(\alpha_2, \ldots, \alpha_n) = 0) \leq \frac{d-k}{N}$. Ha pedig $(\alpha_2, \ldots, \alpha_n)$ rögzített úgy, hogy $h_k(\alpha_2, \ldots, \alpha_n) \neq 0$, akkor $Prob(f(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0) \leq \frac{k}{N}$, hiszen az $\alpha_2, \ldots, \alpha_n$, értékek behelyettesítése után kapott egyváltozós polinom foka k.

Annak a valószínűsége, hogy a két nemkívánatos esemény valamelyike bekövetkezik, legfeljebb a két esemény valószínűségének az összege, ami nem több, mint $\frac{d-k}{N} + \frac{k}{N} = \frac{d}{N}$. □

Következmény: *Az* $\{1, 2, \ldots, 2d\}$ *halmazból vett véletlen* n*-komponensű* α *vektor esetén* $Prob(f(\alpha) \neq 0) \geq 1/2$, *ha* $f \not\equiv 0$. *Ekkora halmazból választva tehát legalább* $1/2$ *valószínűséggel adódik tanú. Ha* t*-szer függetlenül választunk ilyen helyettesítést, akkor legalább* $1 - \frac{1}{2^t}$ *valószínűséggel kapunk tanút.* □

Ahhoz, hogy találjunk egy tanút, várhatóan 2 kísérlet elegendő. Ha az f polinom T időben kiértékelhető $\{1, 2, \ldots, 2d\}$-beli α_i értékekkel, akkor várhatóan $2T$ időben adódik tanú. Általában véve tT időköltséggel $1 - \frac{1}{2^t}$ valószínűséggel találhatunk tanút.

Térjünk itt vissza egy kicsit a randomizált módszerek megbízhatóságával kapcsolatos kifogáshoz! Képzeljünk el egy roppant megbízható számítógépet. Mondjuk olyat, ami ezer esztendőnyi folyamatos működés során csak egyszer hibázik. (Ennyi idő alatt csak történik valami, ami megzavarja: közeli villámcsapás, üstökös felbukkanása, tatárdúlás, stb.) Tegyük fel, hogy ez a hiba egy adott időintervallumba annak hosszával arányos eséllyel kerül. Ekkor annak a valószínűsége, hogy a hiba a következő ezredmásodpercben lép fel, nagyvonalúan becsülve is legalább

$(1/2)^{50}$. Az előző eljárást 60-szor ismételve a hibás döntés esélye ennél sokkal kisebbé tehető. A t alkalmas (és nem túl nagy) választásával tehát elérhetjük a hagyományos algoritmusok biztonságát.

Nézzük ezután a Schwartz-lemma egy alkalmazását. Legyen $G = (L, U; E)$ páros gráf, $L = \{l_1, \ldots, l_n\}$ és $U = \{u_1, \ldots, u_n\}$. Az $M = (m_{ij})$ n-szer n-es mátrixot az alábbi módon értelmezzük:

$$m_{ij} = \begin{cases} x_{ij} & \text{ha } (l_i, u_j) \in E, \\ 0 & \text{különben.} \end{cases}$$

Az M mátrix elemei tehát nullák és független változók. Annyi különböző változó szerepel, ahány éle van G-nek.

Tétel: *G-ben akkor és csak akkor van teljes párosítás, ha* $\det M \neq 0$.

Bizonyítás: A determináns egy tagja kifejtés után $\pm m_{1\pi(1)} m_{2\pi(2)} \cdots m_{n\pi(n)}$ alakú, ahol π az $1, \ldots, n$ számok egy permutációja. Ha egy ilyen tag nem 0, akkor $(l_i, u_{\pi(i)}) \in E$, $i = 1, \ldots, n$, így ezek az élek teljes párosítást adnak. Ha tehát G-ben nincs teljes párosítás, akkor $\det M = 0$. Ha viszont van G-ben teljes párosítás, akkor annak egy nem 0 kifejtési tag felel meg. Különböző kifejtési tagok nem ejthetik ki egymást, mert bármely kettőben van két különböző változó. Tehát ekkor $\det M \neq 0$. □

A $\det M$ polinomra alkalmazhatjuk az előbbieket. Véletlent használó (randomizált) módszert kapunk annak eldöntésére, hogy van-e G-ben teljes párosítás. Nézzük az eljárás költségét. Itt $\deg \det M = n$, az α_i elemeket az $\{1, 2, \ldots, 2n\}$ halmazból választhatjuk. A Schwartz-lemma szerint ha $\det M \not\equiv 0$, akkor egy helyettesítés legalább $(1/2)$ valószínűséggel tanút ad. Egy kiértékelés (Gauss-elimináció) $O(n^3)$ aritmetikai művelettel $(+, -, *, /)$ elvégezhető. A fellépő számok mérete a számítás alkalmas szervezése mellett legfeljebb $\log_2 n!(2n)^n$, ami $O(n \log n)$. A módszer tehát *várhatóan polinom időben*[1] eldönti, hogy van-e G-ben teljes párosítás.

Érdekességként megemlítjük az előző tétel általánosítását (bizonyítás nélkül) tetszőleges irányítatlan gráfokra.

[1] Az Olvasóban felmerülhet a kérdés, hogy miként viszonyul az itt bemutatott randomizált algoritmus hatékonysága a magyar módszeréhez. Bizonyítás nélkül megemlítjük, hogy – további ötletek alkalmazásával – a módszer várható futási ideje levihető $O(n^{2,38})$-ra. Ez sűrű gráfok esetén elvben versenyképes a König-módszer idejével, sőt a Hopcroft–Karp-algoritmuséval is. A gyakorlatban azonban erre a feladatra ma jobbnak tűnnek a gráfelméleti hátterű algoritmusok.

Tétel (Tutte tétele): *Legyen* $G = (V, E)$ *egy irányítatlan gráf,* $V = \{v_1, \ldots, v_n\}$. *Legyen* $T = (t_{ij})$ *a következő mátrix:*

$$t_{ij} = \begin{cases} x_{ij} & \text{ha } (v_i, v_j) \in E \text{ és } i < j, \\ -x_{ij} & \text{ha } (v_i, v_j) \in E \text{ és } i > j, \\ 0 & \text{különben.} \end{cases}$$

A G *gráfban pontosan akkor van teljes párosítás, ha* $\det T \not\equiv 0$.

A páros gráfokra javasolt randomizált módszer minden nehézség nélkül működik az általános esetben is. Ekkor $\det T$ lesz a polinom, amihez tanút keresünk.

9.4.1. Az RP nyelvosztály

Köszönöm, kedves véletlen, fogadd hálás köszönetemet!
SØREN A. KIERKEGAARD: Vagy-vagy[2]

A hagyományos számítások hatékonyságának értelmezésében hasznosak voltak az idő- és tárkorlátokkal megadott nevezetes nyelvosztályok. Most abból szeretnénk rövid ízelítőt adni, hogy a randomizált számítások miként férnek ebbe a keretbe. Olyan nyelvosztályt definiálunk – ez lesz az RP osztály –, melynek a nyelvei hatékonyan felismerhetők randomizált (véletlent használó) algoritmusokkal.

Definíció: *Az* $L \subseteq I^*$ *nyelv az* RP *nyelvosztályba tartozik, ha van olyan* $L_1 \in$ P *nyelv és* $c \geq 0$ *állandó, hogy*
(i) $L = \{x \in I^*,$ *és van olyan* $y \in I^*$ *szó, melyre* $\mid y \mid = \mid x \mid^c$ *és* $(x, y) \in L_1\}$,
(ii) ha $x \in L$, *akkor az* $\mid x \mid^c$ *hosszú* $y \in I^*$ *szavaknak legalább a felére teljesül, hogy* $(x, y) \in L_1$.

Vegyük észre, hogy az (i) tulajdonság éppen azt fejezi ki, hogy $L \in$ NP. Az RP-beli nyelvek tehát egyben NP-beliek is: RP $\subseteq$ NP. Ha viszont az L nyelv polinom időben felismerhető, akkor az $L_1 = \{(x, 1); x \in L\}$ nyelvvel és a $c = 0$ állandóval teljesülnek a definíció kikötései, vagyis ekkor $L \in$ RP is igaz. A P $\subseteq$ RP $\subseteq$ NP tartalmazási relációkról azt gondoljuk, hogy valódiak; széles körben elfogadott sejtés szerint az RP nem esik egybe sem a P, sem pedig az NP nyelvosztállyal.

Legyen $L \in$ RP. Az L_1 nyelv és egy neki megfelelő polinom idejű algoritmus ismeretében az L felismerésére a következő egyszerű randomizált (véletlent

[2] Johannes szavai amikor hosszú keresgélés után végre rábukkant Cordéliára.

használó) algoritmus kínálkozik. Legyen $x \in I^*$ egy input szó. El kell döntenünk, hogy $x \in L$ teljesül-e. Válasszunk egy véletlen $y \in I^*$ szót, melynek hossza $|x|^c$, és döntsük el az $((x, y) \in L_1?)$ kérdést. Ha a válasz igenlő, akkor $x \in L$, ellenkező esetben a következtetésünk: „valószínűleg $x \notin L$". Vegyük észre, hogy az (i) feltétel miatt az igenlő válasz mindig helyes. Ha van tanú, akkor az x szó az L nyelvbe tartozik. A nemleges válasz lehet hibás. Előfordulhat, hogy $x \in L$ teljesül ugyan, de rossz tanújelölt akadt a kezünkbe. Az (ii) feltétel szerint azonban ennek a balszerencsének a valószínűsége legfeljebb $1/2$. Nevezzük az itt vázolt módszert RP-tesztnek. A hibázás valószínűsége tetszőlegesen kicsivé tehető a már megismert módon: t darab függetlenül választott y kipróbálása után legfeljebb $\frac{1}{2^t}$ a téves következtetés valószínűsége. Mivel $L_1 \in$ P és y hossza az x hosszában polinomiális, egy ilyen teszt polinom időben elvégezhető. Az $(x \in L?)$ kérdés tehát várhatóan polinom időben megválaszolható.

Megjegyzés: A randomizált módszerek költségének a tanulmányozásakor nem vettük számításba a „véletlen választások" költségét. A gyakorlatban ezek a választások álvéletlen objektumok (számok, bitek) generálását jelentik. Itt csak megemlítjük, hogy vannak gyors módszerek, melyek elég jó (megfigyelt) viselkedésű álvéletlen objektumokat eredményeznek.

Igen érdekesek a $Las\ Vegas :=$ RP $\cap$ coRP osztályba tartozó nyelvek. Ha $L \in Las\ Vegas$, akkor definíció szerint $L \in$ RP és $L' = I^* \setminus L \in$ RP. Nézzük, mi történik, ha egy $x \in I^*$ bemenő szóra lefuttatunk egy-egy RP-tesztet mind az $(x \in L?)$, mind pedig az $(x \in L'?)$ kérdésre. Ezt megtehetjük, mert L és L' is RP-beli nyelv. Ezek közül csak az egyik adhat igenlő eredményt, hiszen x pontosan az egyik nyelvnek eleme. (Az (i) feltétel miatt az RP-teszt nem tud úgy lódítani, hogy a nyelvbe nem tartozó szóról azt állítja, hogy benne van.) Ha valamelyik tesztre *igen* a válasz, akkor készen vagyunk. Mi van akkor, ha egyik teszt sem adott igenlő választ? Az (ii) feltételt alkalmazva az L és L' közül arra, amelyiknek x eleme, látjuk, hogy ennek a valószínűsége legfeljebb $1/2$. Két *nem* esetén tehát arra következtethetünk, hogy a tesztek nem jártak eredménnyel. A tesztek ismétlésével ennek a valószínűsége gyorsan lecsökkenthető. A lényeges különbség a szimpla RP-teszthez képest, hogy itt sosem kapunk helytelen eredményt. Ha egyik teszt sem adott igenlő választ, akkor úgy vehetjük, hogy tovább kell próbálkoznunk az igazság kiderítésével. Az itt vázolt *Las Vegas-teszt* becsületes abban az értelemben, hogy sosem ad hamis választ a kérdésre. A Las Vegas-teszt jellemzőit így foglalhatjuk össze: *gyors (polinom idejű), nagy valószínűséggel válaszol az $(x \in L?)$ kérdésre, és a válasza mindig helyes.*

Az RP nyelvosztály definíciójában az (ii) feltételt úgy is fogalmazhatjuk, hogy ha $x \in L$, akkor az $|x|^c$ hosszú szavak közül választott véletlen y szó legalább

(1/2) valószínűséggel tanú lesz. Legyen p egy tetszőleges, rögzített, nem szélsőséges valószínűség, azaz legyen $0 < p < 1$. Az RP-beli nyelv definíciójában cseréljük ki az (ii) feltételt a következőre:

(ii') Ha $x \in L$, akkor egy véletlenül választott $|x|^c$ hosszú $y \in I^*$ szóra legalább p valószínűséggel igaz, hogy $(x, y) \in L_1$.

Feladat: Mutassuk meg, hogy ha a definícióban az (ii) feltétel helyett az (ii') feltételt használjuk, akkor is éppen az RP-beli nyelveket kapjuk.

Megemlítjük még, hogy a randomizált módszerek számítási ereje nagyobb a Turing-gépekénél. Tegyük fel ugyanis, hogy a (binárisan) adott n egész bemenetre találnunk kell egy olyan $x \in I^*$ szót, melyre $C(x) \geq n/2$ (itt $C(x)$ az x szó Kolmogorov bonyolultsága). Véletlen választással könnyen célt érünk: az olyan n hosszú x szavak száma, melyekre $C(x) \leq n - 2$, kisebb mint 2^{n-1}, tehát egy véletlenül választott n-hosszú x szóra legalább $1/2$ valószínűséggel teljesül, hogy $C(x) > n - 2$.

Másfelől nincs olyan M TG, mely az n bemenetre olyan $M(n)$ szót ad, melyre $C(M(n)) \geq n/2$. Ellenkező esetben az invariancia-tételből $C(M(n)) \leq C_M(M(n)) + c \leq \lceil \log_2(n+1) \rceil + c$. Ezek összevetéséből pedig $n/2 \leq \lceil \log_2(n+1) \rceil + c$ adódna, ami képtelenség, ha n elég nagy.

9.4.2. Prímtesztelés

Tegyük fel, hogy bemenő adatként adott (binárisan) egy m páratlan egész; szeretnénk eldönteni, hogy m prímszám-e. A feladat az elvi érdekességén túl gyakorlati szempontból is fontos. A biztonságos kommunikáció céljait szolgáló algoritmusok gyakran használnak igen hosszú – olykor többszáz-jegyű – prímeket. Hamar meggyőződhetünk róla, hogy ilyen nagy m-nél a kézenfekvő (exponenciális idejű) módszer, hogy m-et sorban elosztjuk a $\sqrt{m}$-nél kisebb természetes számokkal, nem fér bele az időnkbe.

A feladat tehát a Π nyelv felismerése. Erről eddig annyi derült ki, hogy jól karakterizált: $\Pi \in \mathrm{NP} \cap \mathrm{coNP}$. Azt is megemlítettük, hogy eddig nem találtak a megoldására polinom idejű módszert.

A prímszámok felismerésére szolgáló eljárásokat *prímteszteknek* szokás nevezni. Itt egy hatékony randomizált prímtesztet szeretnénk bemutatni. A módszer valójában egy RP-teszt lesz a prímtulajdonság komplementerének, az *összetettségnek* a felismerésére.

Először egy olyan módszert ismertetünk, ami szemléletes ugyan, de van egy kis fogyatékossága. Mint látni fogjuk, ez a fogyatékosság a módszer finomításával kiküszöbölhető. Az algoritmus (Fermat-teszt) a kis Fermat-tételen alapul. Az egyetlen bemenő paramétere a binárisan leírt $m > 2$ páratlan egész.

Fermat-teszt (m)
1. Válasszunk egy véletlen a egészet az $[1, m)$ intervallumból.
2. Ha $a^{m-1} \equiv 1 \pmod{m}$, akkor a válasz „$m$ valószínűleg prím", különben a válasz „m összetett".

A teszt gyorsan ($\log_2 m$-ben polinomiális időben) végrehajtható: az $a^{m-1} \pmod{m}$ hatványt a korábban említett *gyors hatványozással* kaphatjuk meg hatékonyan.

Vegyük észre, hogy ha az eredmény az, hogy m összetett, akkor ez biztosan helyes. Ezt a tényt a tanúsítja, hiszen a 2. lépésben kiderül, hogy a-ra nem teljesül a Fermat-kongruencia. Ezért ekkor azt mondjuk, hogy a az m összetettségének *Fermat-tanúja.*

Példa: Legyen $m = 21 = 7 \cdot 3$ és $a = 2$. Ekkor a az m Fermat-tanúja, hiszen $2^{20} \equiv 4 \pmod{21}$.

A Fermat-tanúk számáról szól a következő állítás.

Állítás: *Ha m-nek van olyan a Fermat-tanúja ($1 \leq a < m$ és $a^{m-1} \not\equiv 1 \pmod{m}$), melyre $lnko(a, m) = 1$, akkor az $[1, m)$ intervallum egészeinek legalább a fele Fermat-tanú.*

Az állítást úgy is értelmezhetjük, hogy ha m összetettségének van olyan a Fermat-tanúja, melyre $lnko(a, m) = 1$, akkor a teszt legalább $1/2$ valószínűséggel talál Fermat-tanút.

Bizonyítás: Ha $lnko(a, m) > 1$, akkor a nyilván Fermat-tanú. Elég ezért látni, hogy a redukált maradékosztályok (amelyekre teljesül, hogy $lnko(a, m) = 1$) legalább fele Fermat-tanú. Legyen G a mod m redukált maradékosztályok halmaza és

$$H := \text{nem Fermat-tanúk} = \{b \in G;\ b^{m-1} \equiv 1 \pmod{m}\}.$$

Legyen $a \in G$ egy (a feltételünk szerint létező) Fermat-tanú, és tekintsük a $Ha := \{ba;\ b \in H\} \subseteq G$ halmazt. Ha b és b' két különböző maradékosztály, akkor $ba \not\equiv b'a \pmod{m}$, mert a és m relatív prímek. Ebből arra jutunk, hogy $|\ H\ | = |\ Ha\ |$.

Másfelől $Ha \subseteq G \setminus H$, hiszen egy H-beli maradékosztályt Fermat-tanúval szorozva ismét tanút kapunk. Mindezek alapján

$$|\ H\ | = |\ Ha\ | \leq |\ G \setminus H\ |,$$

azaz $|\ H\ | \leq \frac{1}{2}\ |\ G\ |$. Tehát a G elemeinek legalább a fele Fermat-tanú. □

A módszer hátulütője, hogy vannak olyan m *összetett* számok, melyekre egyáltalán nincs olyan a Fermat-tanú, melyre $lnko(a, m) = 1$. Ezek az úgynevezett

pszeudoprímek, vagy Carmichael-számok. Ilyen például az $561 = 3 \cdot 11 \cdot 17$. Újabb eredmény (W. R. Alford, A. Granville, C. Pomerance, 1992), hogy végtelen sok ilyen szám van. Hogyan ellenőrizhetjük, hogy 561 tényleg egy Carmichael-szám? Emlékeztetünk itt a kínai maradéktételre.

Tétel (kínai maradéktétel): *Legyenek* $m_1, \ldots, m_k$ *páronként relatív prím,* $a_1, \ldots, a_k$ *tetszőleges egészek. Ekkor az*

$$x \equiv a_1 \pmod{m_1}, \ldots, x \equiv a_k \pmod{m_k}$$

kongruencia rendszernek van egész x *megoldása. Az* x *megoldás modulo* $m_1 m_2 \cdots m_k$ *egyértelműen meghatározott.*

Visszatérve példánkhoz: meg kell mutatni, hogy ha $lnko(a, 561) = 1$, akkor $a^{560} \equiv 1 \pmod{561}$. A kínai maradéktétel szerint elég, ha az $a^{560} \equiv 1 \pmod{17}$, $a^{560} \equiv 1 \pmod{11}$ és $a^{560} \equiv 1 \pmod{3}$ kongruenciákat igazoljuk, feltéve, hogy a relatív prím a szóban forgó modulushoz. Nézzük meg az elsőt; a többi hasonlóan kezelhető. Mivel $560 = 16 \cdot 35$,

$$a^{560} \equiv a^{16 \cdot 35} \equiv (a^{16})^{35} \equiv 1^{35} \equiv 1 \pmod{17}.$$

Az utolsó előtti kongruencia a kis Fermat-tétel következménye.

Érdemes megjegyezni azt a módot, ahogy itt a kínai maradéktételt használtuk. Modulo $m_1 m_2 \cdots m_k$ számítás helyett elegendő volt modulo m_i $(1 \leq i \leq k)$ számolni. Ezt az ötletet széles körben alkalmazzák a számítógépes aritmetikában, különösen a nagy pontosságú számítások területén.

E kis kitérő után kanyarodjunk vissza a prímteszteléshez. A Fermat-teszt fogyatékossága, hogy ha m Carmichael-szám, akkor m-et a teszt nagy eséllyel prímnek deklarálja. Most a Fermat-teszt olyan javítását ismertetjük, ami kiküszöböli ezt a problémát. A módszert *Rabin–Miller-tesztnek* nevezik.

Definíció: *Legyen* m *egy páratlan természetes szám. Írjuk fel* $m - 1$*-et* $m - 1 = 2^k n$ *alakban, ahol* n *páratlan. Az* $1 \leq a < m$ *egész* Rabin–Miller-tanú *(*m *összetettségére), ha az*

$$a^n - 1,\ a^n + 1,\ a^{2n} + 1, \ldots,\ a^{2^{k-1} n} + 1$$

számok egyike sem osztható m*-mel.*

Tétel: *Ha* m *prím, akkor* m*-hez nincs Rabin–Miller-tanú.*

Bizonyítás: Legyen a egy tetszőleges egész az $[1, m)$ intervallumból. Az

$$a^{m-1} - 1 = (a^n - 1)(a^n + 1)(a^{2n} + 1) \cdots (a^{2^{k-1}n} + 1)$$

azonosságot használjuk. Mivel m prím, a kis Fermat-tétel szerint m osztja a bal oldalon álló számot. Ugyancsak m prímvoltából következik, hogy m ekkor osztja a jobb oldal valamelyik tényezőjét, ami azt jelenti, hogy a nem Rabin–Miller-tanú. □

Az is látszik az azonosságból, hogy ha a nem Rabin–Miller-tanú, akkor nem lehet Fermat-tanú sem. Ugyanis ha a jobb oldali számok valamelyike osztható m-mel, akkor a bal oldal is osztható m-mel. Ezt pozitívba fordítva úgy is mondhatjuk, hogy egy Fermat-tanú egyben Rabin–Miller-tanú is. Az új definíció tehát tágítani igyekszik a lehetséges tanúk körét. A következő állítást úgy is olvashatjuk, hogy ez a bővítés sikerült: ha m összetett, akkor ennek a ténynek sok Rabin–Miller-tanúja van.

Tétel: *Ha m összetett, akkor az $1 \leq a < m$ feltételt teljesítő a egészeknek legalább a fele Rabin–Miller-tanú.*

A bizonyítást mellőzzük; csak annyit jegyzünk meg, hogy a lényeg annak az igazolása, hogy van egyáltalán m-hez relatív prím Rabin–Miller-tanú. Utána a Fermat-tesztnél látott érveléssel következik, hogy sok tanú van. A Rabin-Miller összetettségi teszt a Fermat-teszt kézenfekvő módosítása: Fermat-tanú helyett Rabin–Miller-tanút választunk.

$RM(m)$
1. Írjuk fel $m - 1$-et $m - 1 = 2^k n$ alakban, ahol n páratlan.
2. Válasszunk egy véletlen a egészet az $[1, m)$ intervallumból.
3. Ha az $a^n - 1$, $a^n + 1$, $a^{2n} + 1, \ldots, a^{2^{k-1}n} + 1$ számok egyike sem osztható m-mel, akkor megállunk azzal a válasszal, hogy „m összetett", különben megállunk azzal a válasszal, hogy „m valószínűleg prím".

Várhatóan 2 véletlen a érték választása után kapunk egy Rabin–Miller-tanút, ha m összetett. Ha t kísérlet után sem adódik Rabin–Miller-tanú, akkor m-et legfeljebb $\frac{1}{2^t}$ tévedési valószínűséggel prímnek nyilváníthatjuk. Az RM-teszt, ugyanúgy, mint a Fermat-teszt, gyors hatványozással valósítható meg hatékonyan.

Következmény: *Az összetett számok nyelve az* RP *osztályban van.*

Bizonyítás: Legyen

$$L_1 = \left\{ (m,a) \;\middle|\; \begin{array}{l} m > 2, 1 \leq a < m \text{ egészek, és ha } m \text{ páratlan,} \\ \text{akkor } a \text{ egy } m\text{-hez tartozó Rabin–Miller-tanú} \end{array} \right\}$$

és $c = 1$. Ezekkel a választásokkal teljesülnek az RP-beliség (i) és (ii') feltételei, utóbbi a $p = (1/4)$ valószínűséggel. Legyen ugyanis n az m bitjeinek száma. A tételből következik, hogy a legfeljebb n bittel leírható egészek legalább $1/4$-e Rabin–Miller-tanúja lesz m összetettségének. □

Könnyű meggondolni, hogy a módszer éppen az L_1 nyelven alapuló RP-teszt az összetett számok felismerésére. Sokkal mélyebb eredmény (L. A. Adleman, M. D. Huang, 1987, a bizonyítás másfélszáz oldalnyi és pogányul nehéz), hogy $\Pi \in$ RP is teljesül. Maga a teszt polinom idejű ugyan, de olyan bonyolult, hogy a gyakorlatilag még érdekes méretű inputokon nem érdemes futtatni. Ezért prímtesztelésre inkább összetettségi teszteket használnak. Ezek között a Rabin–Miller-teszt elég jónak számít.

A két eredmény együtt azt jelenti, hogy $\Pi \in Las\ Vegas$. És még ennél is cifrább a helyzet. Egy sokat tanulmányozott, de mindeddig nem igazolt számelméleti hipotézisből (az ún. általánosított Riemann-sejtés) következik, hogy ha m összetett, akkor van olyan a Rabin–Miller-tanú is, melyre $a \leq 2\log^2 m$ (E. Bach, 1990). Ha tehát a sejtés igaz, akkor $\Pi \in$ P, hiszen ekkor elég tanút keresni az első $2\log^2 m$ természetes szám között.

9.4.3. Nagy prímszám keresése

A későbbiekben szó lesz olyan módszerekről, amelyek nagy prímszámokat használnak. Itt megmutatjuk, hogy véletlen választással könnyen találhatunk ilyeneket.

Probléma: bemenő adatként adott egy n természetes szám. Keressünk egy n-jegyű prímszámot.

A módszer igen egyszerű: válasszunk egy véletlen p egészet a $[2^{n-1}, 2^n - 1]$ intervallumból. Az RM-teszttel ellenőrizzük, hogy p prím-e, mondjuk 2^{-60} hibavalószínűséggel. Ez legfeljebb 60 véletlen a kipróbálását jelenti. Ha kiderül, hogy p nem prím, akkor új p-t választunk.

Várhatóan hány p-t kell választani? Egy $x > 1$ valós számra jelölje $\pi(x)$ az $[1, x]$ intervallumba eső prímek számát. A prímszámtétel szerint, $\pi(x) \sim \frac{x}{\log x}$, ha $x \to \infty$. Innen könnyen adódik, hogy a számunkra érdekes intervallumban a prímek száma $\approx \frac{(\log_2 e)2^{n-1}}{n}$, ha n elég nagy. Annak a valószínűsége ezért, hogy egy véletlen p prím lesz, körülbelül $\frac{\log_2 e}{n}$, tehát $O(n)$ darab véletlen p kipróbálásával

várhatóan találunk egyet. A módszer várható költsége polinomiális az *eredmény hosszához mérve*.

9.5. Prekondícionálás

GUCUACGGCCAUACCACCUGAACGCGCCCGAUCUCGUCUG
AUCUCGGAAGCUAAGCAGGGUCGGGCCUGGUUAGUACUUG
GAUGGGAGACCGCCUGGGAAUACCGGGUGCUGUAGGCUU
Az emberi $5S$ rRNS genetikai kódja

Képzeljük el, hogy egy adott P algoritmikus problémának több példányát, mondjuk $i_1, i_2, \ldots, i_k$-t kell egy feladat részeként megoldanunk. A kézenfekvő megközelítés az lenne, hogy veszünk egy jó algoritmust a P megoldására, amit sorra lefuttatunk az $i_1, i_2, \ldots, i_k$ bemenetekkel. Sok esetben ennél számottevően hatékonyabb módszert kaphatunk, ha először egy kis előkészítő munkát végzünk – szokásos kifejezéssel *prekondícionáljuk* P-t. A prekondícionálás olyan előkészítő tevékenység, aminek az eredménye többször felhasználható. Bizonyos feladatoknál az előkészítő munka extra költsége több példány esetén megtérül, sőt az összköltség csökken. Gyakran egész egyszerűen arról van szó, hogy az $i_1, i_2, \ldots, i_k$ megoldásainak közös lépéseit kiemeljük, és előre elvégezzük. Ebben az értelemben a prekondícionálás hatékony munkaszervezési fogás. Előzetesen elvégezzük azokat a teendőket, amelyek a konkrét bemenetektől viszonylag függetlenek.

Az ilyen módszerek tervezésekor figyelnünk kell az előkészítő fázis költségeire is. Csak akkor érdemes belevágni, ha az $i_1, i_2, \ldots, i_k$ megoldásának plusz az előkészítésnek a költsége versenyképes a kézenfekvő megoldáshoz viszonyítva. Ennek mérlegelésekor érdemes meghatározni k-nak azt a nagyságrendjét, amitől kezdve már gazdaságos a módszer. Ennyi általánosság után nézzünk néhány példát:

1. Ősiség fákban
Tegyük fel, hogy T gyökeres, felfelé irányított fa (aminek az élei a gyökér felé mutatnak). A fa mint bemenet a csúcsaival és az apamutatókkal adott. A feladat T-beli (v, w) csúcspárokra eldönteni, hogy w leszármazottja-e v-nek. Egyetlen pár esetén a költségigény (uniform) lehet cn is (ahol n a T csúcsainak száma és $c > 0$ egy állandó). Ugyanakkor alkalmas $O(n)$ költségű prekondícionálás után minden egyes kérdés $O(1)$ költséggel kezelhető.

Egy lehetséges ötlet, hogy minden $v \in T$ csúcs mellé feljegyezzük a v T-beli $pre(v)$ preorder, illetve $post(v)$ postorder bejárás szerinti sorszámát. Nyilvánvaló

ugyanis, hogy a $v, w \in T$ csúcsokra $pre(v) \leq pre(w) \Leftrightarrow v$ őse w-nek vagy v a w-től balra van T-ben. Ugyanígy $post(v) \geq post(w) \Leftrightarrow v$ őse w-nek, vagy v a w-től jobbra van T-ben. A két tényből adódik, hogy v őse w-nek pontosan akkor, ha $pre(v) \leq pre(w)$ és $post(v) \geq post(w)$.

A kétfajta számozás $O(n)$ időben megkapható. A számok ismeretében egy (v, w) párra a kérdés két összehasonlítással megválaszolható. Ha a bemenetként adott párok száma (a bevezetőbeli k érték) nincs konstans korlát alatt, akkor érdemes ezt a bonyolultabb módszert használni. Például ha a párok száma $k = \log n$, akkor a kézenfekvő módszer költsége $cn \log n$ is lehet; a prekondícionált időigény viszont $O(n + \log n) = O(n)$.

2. Polinom ismételt kiértékelése

Adott az $f(x) = x^n + a_1 x^{n-1} + \cdots + a_n \in \mathbb{Z}[x]$ polinom, és a $b_1, b_2, \ldots, b_k$ egészek. Célunk az $f(b_1), \ldots, f(b_k)$ értékek kiszámítása minél kevesebb szorzással.

Mennyibe kerül itt a kézenfekvő módszer? Először kiszámítjuk a $b_i^2, b_i^3, \ldots, b_i^n$ hatványokat, $(n-1$ szorzás), majd az $a_{n-1}b_i, a_{n-2}b_i^2, \ldots, a_1 b_i^{n-1}$ számokat $(n-1$ szorzás). Összesen ez $2k(n-1)$ szorzás.

Javítást jelent a *Horner-elrendezés* alkalmazása: $f(x)$ felírható

$$f(x) = (\cdots(((x + a_1)x + a_2)x + a_3)\cdots)x + a_n$$

alakban (összesen $n-1$ zárójelpár van). Így végezve a számítást, $f(b_i)$-t $n-1$ szorzással kapjuk. Az összköltség $k(n-1)$ szorzás, ami fele az előzőnek. A Horner-elrendezés szerinti alakra való átalakítást is felfoghatjuk prekondícionálásnak.

Ezután egy még finomabb prekondícionálást használó megoldást mutatunk be. Az egyszerűség kedvéért tegyük fel, hogy $\deg f = n = 2^t - 1$, valamely t természetes számra. Írjuk fel f-et

$$f(x) = (x^{\frac{n+1}{2}} + a)f_1(x) + f_2(x)$$

alakban, ahol $a \in \mathbb{Z}$, az f_1, f_2 egész együtthatós, 1 főegyütthatós polinomok, és $\deg f_1 = \deg f_2 = 2^{t-1} - 1$. Az f_1 együtthatói, az a egész, majd pedig f_2 együtthatói egyszerű együttható összehasonlítással megkaphatók. Az a értékét tudjuk úgy választani, hogy f_2 főegyütthatója 1 legyen. Végezzük el ugyanezt a felbontást az f_1, f_2 polinomokra, majd a belőlük kapott újabb polinomokra, stb., amíg ez lehetséges. Végeredményben összesen t ilyen menetben kifejezzük $f(x)$-et $x^{2^j} + b$, (b egész) alakú polinomok szorzatainak összegeként. Egy menetben összesen n új együtthatót kell meghatározni.

Példa: Nézzük az $f(x) = x^7 + x^5 + 2x + 1$ polinom felbontását. Itt $n = 7$ és $t = 3$. Az első lépés után $f(x) = (x^4 - 1)(x^3 + x) + x^3 + 3x + 1$, azaz $f_1(x) = x^3 + x$,

és $f_2(x) = x^3 + 3x + 1$. E két polinomot tovább bontjuk ($t = 2$). Az eredmény: $f_1(x) = (x^2 + 1)x$ és $f_2(x) = (x^2 + 2)x + x + 1$. Végül

$$f(x) = (x^4 - 1)(x^2 + 1)x + (x^2 + 2)x + x + 1.$$

Az f-nek az így kapott felírását használva egy kiértékelés 5 szorzást jelent. Két szorzással kapjuk az x^2 és x^4 hatványokat, majd elvégezzük a felbontásban szereplő további három szorzást.

Vegyük szemügyre általánosabban is a felbontást használó kiértékelés költségét. Legyen $M(t) =$ az előbbi felbontás szerint a szorzások száma egy kiértékelésnél. Legyen továbbá $N(t) = M(t) - t + 1$. A kiértékelést a c helyen úgy fogjuk végezni, hogy először kiszámítjuk a $c^2, c^4, \ldots, c^{2^{t-1}}$ hatványokat (ez $t - 1$ szorzás), utána pedig a felbontásból adódó $c^{2^j} + b$ alakú tényezőket szorozzuk össze. Az ilyen szorzások száma lesz $N(t)$. Nyilvánvaló, hogy $N(1) = 0$. A felbontás definíciójából kiolvasható, hogy $N(t) \leq 2N(t-1) + 1$, ha $t > 1$, hiszen az $f_1(c)$ és $f_2(c)$ kiszámításához használt szorzásokon felül még egy további szorzás szükséges. A rekurzióból kézenfekvő indukcióval kapjuk, hogy $N(t) \leq 2^{t-1} - 1$. Innen az összes szorzások száma $M(t) \leq 2^{t-1} + t - 2 = \frac{n-3}{2} + \log_2(n+1)$. A k darab kiértékelés összesen $k(\frac{n-3}{2} + \log_2(n+1))$ szorzással elvégezhető.

Feladat: Mutassuk meg, hogy a fenti séma szerinti egyetlen kiértékelésnél legfeljebb $\frac{3n-1}{2}$ összeadás elegendő.

Megjegyzések:

1. V. Pan egyik tételéből következik hogy egy n-edfokú $f(x)$ polinom (f főegyütthatója tetszőleges) egyszeri kiértékeléséhez a legrosszabb esetben legalább n szorzás/osztás kell.

2. Ismeretes olyan prekondícionált algoritmus is, amellyel egy helyettesítés $\lfloor \frac{n}{2} \rfloor + 2$ szorzással kiszámolható. Másfelől bármilyen jól prekondícionáljuk is a polinomot, mindig lesz olyan helyettesítés, ami nem kapható meg kevesebb, mint $\lfloor \frac{n}{2} \rfloor$ szorzással (E. G. Belaga tétele).

3. Az egyszerűség kedvéért, hogy elég legyen csak a szorzásokra figyelnünk, nem foglalkoztunk az előkészítő munka költségével. A gazdaságos k értékek meghatározásánál erre is tekintettel kell lennünk.

3. Mintaillesztés szövegben

A feladat, amivel itt foglalkozunk, alapvető fontosságú a szövegeket tároló, kezelő rendszerek szemszögéből: egy adott jelsorozatot kell megkeresnünk egy hosszú jelsorozatban. Az ilyen és hasonló igények a közhelyszerű számítógépes alkalmazások mellett (pl. keresők, szerkesztők) fontos szerepet játszanak a genetikai kutatások egyik irányzatában, ahol a genetikai kódot roppant méretű, az A,G,C,U,T

jelekből formált szavakkal modellezik. A következőkben az egyik leghatékonyabb ötletet mutatjuk be, melynek lényeges része a keresett minta prekondícionálása. Először fogalmazzuk meg pontosan a feladatot:

Probléma: Adottak a Σ véges abc betűiből alkotott $s = s_1 \cdots s_n$ és $m = m_1 \cdots m_d$ szavak ($s_i, m_i \in \Sigma$). Találjuk meg m első részszóként való előfordulását az s szóban.

Úgy is fogalmazhatunk, hogy a legrövidebb olyan x szót keressük, melyre alkalmas y szóval teljesül, hogy $s = xmy$. Például az $s = balalajka$ szóban az $m = la$ minta a harmadik betűvel kezdődően fordul elő először részszóként, míg az $m = lak$ szó egyáltalán nem fordul elő.

Természetesen kínálkozik az a megoldás, hogy $i = 1, 2, \ldots, n - d + 1$-re sorban megvizsgáljuk, hogy az s-nek az i-edik betűjével kezdődő d hosszúságú részszava egyenlő-e m-mel. Ez az eljárás legrosszabb esetben $d(n - d + 1)$ betű-összehasonlítást igényel.

Első hallásra talán meglepő, hogy van a feladatra ennél jóval gyorsabb módszer is. A Knuth–Morris–Pratt-algoritmus, amit ismertetni fogunk, lineáris idejű, vagyis $O(n + d)$ összehasonlítást használ.

Definíció: *Az y szó az x szó* kezdőszelete (végszelete), *ha van olyan nem üres z szó, hogy $yz = x$ (illetve $zy = x$).*

Definíció: *Az y szó az x szó* lábfeje, *ha y az x leghosszabb olyan kezdőszelete, ami egyben végszelete is. (Az y tehát a leghosszabb olyan w szó, melyre alkalmas nem üres z, z' szavakkal $x = wz = z'w$ teljesül.)*

Például a *bababa* szó lábfeje *baba*. Egy szónak önmaga nem lehet a lábfeje. A továbbiakban a leírás szempontjából kényelmes lesz a C programozási nyelvből ellesett egyik jelölés. A szavakat egészekkel indexelt, Σ típusú tömböknek is tekintjük. Így például az s szó második betűje hivatkozható mint $s[2]$, a harmadik helytől az ötödikig terjedő részszava pedig mint $s[3:5]$.

Mielőtt forró fejjel elkezdenénk az m szót az s különböző részeivel hasonlítgatni, először elemezzük, feltárjuk az m szerkezetét. Ezáltal bizonyos összehasonlítások eredményét többször is felhasználhatjuk a keresésben, számottevően redukálva az összköltséget. A prekondícionálás az m szó előfeldolgozását fogja jelenteni. Pontosabban fogalmazva kitöltjük a $P[1:d]$ egész típusú tömböt, ahol

$$P[j] \stackrel{\text{def}}{=} \text{az } m[1:j] \text{ szó lábfejének a hossza.}$$

A P kitöltésére szolgáló módszer egyszersmind takaros példa a dinamikus programozás alkalmazására, amennyiben a P már ismert részét használjuk a még

ismeretlen értékek meghatározására. Nyilvánvalóan $P[1] = 0$. A módszer leírásakor kényelmes lesz még a $P[0] = 0$ megállapodás. Tegyük fel ezután, hogy $j > 1$ és a $P[1 : j-1]$ résztömböt már kitöltöttük. Ekkor $P[j]$ így számítható:

1. $i := j - 1$; $P[j] := 0$; $P[0] := 0$;
2. Ha $m[j] = m[P[i] + 1]$ akkor legyen $P[j] := P[i] + 1$, különben, ha $i > 0$, akkor legyen $i := P[i]$, és menjünk vissza a 2. lépésre.

Az eljárás megértéséhez jelölje w az $m[1 : j]$ szó lábfejét és l a w hosszát. Az l értéket kell meghatározni, ugyanis $P[j] = |w|$. Először megjegyezzük, hogy a $P[j]$ értéke nyilván legfeljebb $P[j-1]+1$ lehet, hiszen ha az $m[1 : l]$ szó azonos az $m[j-l+1 : j]$ szóval, akkor $m[1 : l-1] = m[j-l+1 : j-1]$ is igaz. Ennek a lehetőségét nézzük meg először a második sor összehasonlításával. Ha itt nemleges a válasz, akkor sem üres a kezünk. Tudjuk, hogy $P[j] \leq P[j-1]$, ami azt jelenti, hogy a $w[1 : l-1]$ szó kezdőszelete $m[1 : P[j-1]]$-nek és végszelete az $m[j - P[j-1] : j-1]$ szónak. De az utóbbi két szó a P tömb definíciója szerint azonos: $m[1 : P[j-1]] = m[j - P[j-1] : j-1]$. Ezzel az észrevétellel l meghatározását visszavezettük egy kisebb hasonló feladatra. A w ezután annak a szónak (is) a lábfeje, amit úgy kapunk, hogy az $m[1 : P[j-1]]$ szó végére fűzzük az $m[j]$ betűt. Ezt a visszavezetést valósítja meg az $i := P[i]$ értékadás.

Példa: Vegyük szemügyre a módszer működését az $m = ababaa$ bemenettel. Tegyük fel, hogy a $P[1 : 5]$ résztömböt már kitöltöttük, és éppen $P[6]$ meghatározásánál tartunk. Tudjuk tehát, hogy $P[1] = P[2] = 0$, $P[3] = 1$, $P[4] = 2$ és $P[5] = 3$. A keretes eljárás indításakor $j = 6$ és emiatt $i = 5$. A 2. sorban az $m[6]$ és $m[4]$ betűket hasonlítjuk össze. Mivel ezek nem egyeznek, az $i = 3$ értékkel térünk vissza a 2. sorba. A teszt eredménye újfent negatív: $m[6] \neq m[2]$. Ezáltal megint a *különben* ágra kerülünk, ahonnan az $i := 1$ értékadás után jutunk a 2. sorba. Az összehasonlítás végre valahára egyezést mutat ($m[1] = m[6] = a$), tehát megtörténik a $P[6] := P[1] + 1 = 1$ értékadás.

A $P[\ ]$ tömb kitöltésének időköltsége nyilvánvalóan arányos a szükséges betűösszehasonlítások számával, így csak az utóbbiakat számoljuk össze. Jelölje k_j a $P[j]$ számításához felhasznált betűösszehasonlítások számát. Ebbe $P[1 : j-1]$ költségét nem értjük bele. Először megmutatjuk, hogy $k_j \leq P[j-1]-P[j]+2$. Világos, hogy a 2. sor tesztjét pont k_j-szer hajtjuk végre, hiszen a kódban csak itt van betűösszehasonlítás. Vizsgáljuk meg, hogy mi történik ezalatt a $P[i]$ értékkel. Ez kezdetben $P[j-1]$, majd minden további ilyen teszt előtt legalább eggyel csökken. A végső értéke, amit f-fel jelölünk, ezért nem lehet több, mint $P[j-1] - k_j + 1$. Tudjuk azt is, hogy $P[j]$ értéke vagy $f + 1$ lesz, vagy f aszerint, hogy a legutolsó betűösszehasonlítás sikeres volt-e, vagy sem. (Utóbbi esetben $f = 0$.) Ezeket

összevetve $P[j] \leq P[j-1] - k_j + 2$, amiből $k_j \leq P[j-1] - P[j] + 2$.

Az összehasonlítások száma mármost

$$\sum_{j=2}^{d} k_j \leq \sum_{j=2}^{d} (P[j-1] - P[j] + 2) = P[1] - P[d] + 2(d-1) \leq 2(d-1).$$

Az előfeldolgozás tehát a minta hosszával arányos költséggel, vagyis lineáris időben elvégezhető.

Ezután a Knuth–Morris–Pratt-algoritmus általános lépését mutatjuk be. Tegyük fel, hogy az m mintát éppen az $s[i]$ betűvel kezdődően próbáljuk illeszteni, és már tudjuk, hogy $s[i] = m[1], \ldots, s[i+j-1] = m[j]$, azaz a minta első j betűjénél illeszkedést találtunk.

1. Ha $j = d$, akkor készen vagyunk, találtunk egy m-mel azonos részszót s-ben. Ha $j < d$, akkor (feltéve, hogy $i + j \leq n$) összehasonlítjuk az $m[j+1]$ és $s[i+j]$ betűket. Ha ezek egyenlők, akkor $j := j+1$, és a lépést befejeztük. Ha $m[j+1] \neq s[i+j]$, akkor két esetet különböztetünk meg:

2. (a) Ha $j = 0$ (azaz éppen m első betűjénél tartunk), akkor m-et egy hellyel jobbra csúsztatjuk: $i := i+1$, és a lépést befejeztük.

(b) Ha $j > 0$, akkor az m mintát $j - P[j]$ hellyel jobbra csúsztatjuk: $i := i + j - P[j]$ és $j := P[j]$, és a lépést befejeztük.

A lényeges ötlet a 2.(b) lépésben lapul. A $P[j]$ érték mondja meg, hogy mi az a legkisebb esélyes mennyiség (nevezetesen $j - P[j]$), amivel érdemes eltolni jobbra az m mintát. Vegyük észre, hogy az eltolás után az m első $P[j]$ betűje egyezik az s megfelelő betűivel. A fontos ebből, hogy az s szóban nem kell visszalépnünk. Az $i + j$ mennyiség ugyanaz a 2. (b) lépés előtt és után.

Példa: Az $s = cdcbabababa$ szövegben keressük az $m = bababc$ mintát. Tegyük fel, hogy m-et már az s negyedik betűjétől kezdve próbáljuk illeszteni, és éppen túl vagyunk az első öt betű hasonlításán. Mindez számokkal azt jelenti, hogy $i = 4$ és $j = 5$:

			i								
c	d	c	b	a	b	a	b	a	b	a
			b	a	b	a	b	c		
							$i+j$			

A következő összehasonlításnál kiderül, hogy $s[9] \neq m[6]$, és ezzel a 2. (b) ágba kerülünk. Hasznát vesszük az eddigi sikeres összehasonlításoknak. Ezekből tudjuk, hogy $s[4:8] = m[1:5]$. A következő esélyes illeszkedés helyét a P

tömbből tudhatjuk meg. Mivel $P[5] = 3$, a legkisebb esélyes eltolás az eddigi ismeretek alapján 2. A pusztán eggyel való eltolással nem érdemes próbálkozni; ha ott volna illeszkedés, akkor $m[1:5]$ lábfejének a hossza 4 lenne. A 2. (b) lépés után ez lesz a helyzet:

$$\begin{array}{cccccccccccc} & & & & & i & & & & & & \\ c & d & c & b & a & b & a & b & a & b & a \\ & & & & & b & a & b & a & b & c \\ & & & & & & & i+j & & & \end{array}$$

A lépés befejeztével $i = 6$ és $j = 3$. Az s szövegben nem kellett visszalépnünk, a mintát pedig jobbra mozdítottuk.

A módszer időigényének nagyságrendi becsléséhez itt is elég a betűösszehasonlítások számát vizsgálni. Ezt nagyban megkönnyíti az a tény, hogy az eljárás során az i és az $i + j$ mennyiségek *sohasem csökkennek*. Kezdetben $i = 1$ és $j = 0$, tehát $i + j = 1$. Egy lépés (ami legfeljebb egy betűösszehasonlítást jelent) megtétele után vagy i vagy pedig $i + j$ értéke nő. Mivel $i \leq n$ és $i + j \leq n$, a lépések és így a betűösszehasonlítások száma legfeljebb $2(n-1)$. A módszer tehát összegészében – a prekondícionálás idejét is beleértve – lineáris uniform költségű.