Prímfaktorizációk és alkalmazásaik

Szakdolgozat

Írta: Hammer Gergő

Matematika BSc

Alkalmazott matematikus szakirány

Témavezető:

Freud Róbert

egyetemi docens

Algebra és Számelmélet Tanszék

Eötvös Loránd Tudományegyetem

Természettudományi kar

2011

Tartalomjegyzék

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Freud Róbertnek, aki a félév során mindig szakított rám időt, kérdéseimre körültekintően és jól érthetően válaszolt, valamint észrevételeivel és hasznos tanácsaival segített a szakdolgozatom elkészítésében.
Köszönettel tartozom még csoporttársaimnak akik tanácsokat adtak a LaTeX használatához, valamint köszönöm családomnak, amiért támogatják tanulmányaimat.

Bevezető

A prímek a matematikán belül a számelmélet témakörébe tartoznak. Ez az ág az egyik legrégebbi, mégis a mai napig „helytáll”. Sőt, nemcsak, hogy helytáll, de olyan — egyszerűen megérthető — problémákat vet fel, melyeket az utca embere is megért, de ezekre — a néhol akár több ezer éves kérdésekre — a mai napig nem tudunk választ adni. És nemcsak hogy nem tudunk, de jelenlegi tudásunk szerint reménytelennek tűnik, hogy a közeljövőben ezeket megoldjuk. Csak, hogy néhányat soroljunk, ilyen az ikerprím sejtés, Mersenne/Fermat-prímekkel kapcsolatos sejtések, de például a nagy számok prímtényezős felbontásának megállapítása is az. Persze bárki tudna olyan algoritmust mondani, mely megtalálja a felbontást, itt az igazi feladat az, hogy gyorsan (polinomiális időben) végezzük el ezt. Ezen szakdolgozat ennek a problémának a részeredményeit és alkalmazását tárgyalja, hiszen lenyűgöző, hogy egy ókori problémának a kódolásban és titkosításban a mai napig széles körben elterjedt és használt alkalmazása van.

Tehát a téma a prímtényezős felbontások, ehhez vázolunk egy menetrendet. Először is jó lenne, ha el tudnánk gyorsan dönteni egy tetszőleges számnál, hogy az prím-e, ez nyilván hasznos, ha prímtényezőket akarunk megállapítani. Ennek során több prímtesztet is vizsgálunk megbízhatóság és gyorsaság szempontjából.

Ezt követően prímtényezőket meghatározó algoritmusokkal foglalkozunk. Bár a nagy számok felbontását nem tudjuk meghatározni gyorsan, de részeredmények vannak, amelyek az alkalmazás szempontjából fontosak lesznek. Ismertetjük, hogy ezek hogyan és miért működnek, illetve milyen hatékonysággal.

Rátérünk végül az alkalmazásokra. Ezek közül a legfontosabb az RSA-kódolás, mely napjainkban is használt titkosító algoritmus. Ez kihasználja, hogy gyorsan találunk nagy prímeket, és azt, hogy az ezek szorzatából kapott összetett számot nem tudjuk gyorsan tényezőire bontani. Látni fogjuk viszont, hogy nem mindegy, milyen prímeket választunk; ha ügyetlenek vagyunk, akkor létezik hatékony támadás a titkosítás ellen.

Szakdolgozatomban rövid elméleti ismertetők után bemutatom az egyes

tesztek és algoritmusok gyakorlati megvalósítását. A struktogrammok alapján elkészítettem a programokat is, ezt CD-n mellékelem a szakdolgozathoz. A prímteszteknél bebizonyítom, hogy a Miller–Rabin–Lenstra prímteszt az addigi legjobb, így ezt érdemes használni. A faktorizációs algoritmusoknál rövidebb feladatok megoldásával igyekszem érthetőbbé tenni ezek működési elvét. A Monte Carlo módszereknél az általam megvalósított program futási eredményeit is közlöm a program hatékonyságának alátámasztásaként.

1. fejezet

Prímtesztek

Ahogy azt a bevezetőben már említettem, nem tudunk tetszőlegesen nagy számot gyorsan faktorizálni, de szerencsére, ha csak az a feladat, hogy gyorsan eldöntsük egy számról, hogy prím-e, akkor pozitív választ tudunk adni. Ezt úgy tudjuk vizsgálni, hogy olyan feltételeket/tulajdonságokat ellenőrzünk, melyeket csak — vagy majdnem csak — a prímek teljesítenek.

1.1. A kis–Fermat–teszt

Az első ilyen teszt a kis Fermat-tétel eredményét használja fel, mely a következőképpen szól:

$$\forall\, p \text{ prímre és } \forall\, a \in \mathbb{Z},\ (a, p) = 1\text{-re} : a^{p-1} \equiv 1 \pmod{p}$$

Hogy kaphatunk ebből prímtesztet? Egy $n > 2$ számot véve, ha ez valamely a-ra $(a, n) = 1$ mellett nem teljesíti a fenti kongruenciát, akkor n biztosan összetett. Ha teljesíti, akkor több a-ra is megvizsgálhatjuk, így még biztosabbak lehetünk benne, hogy n prím. Rossz hír azonban, hogy vannak olyan összetett számok, melyek teljesítik a kis Fermat-tételt valamely a-ra, ezeket a-alapú álprímeknek hívjuk, de még olyan számok is vannak, melyek minden a-ra, $(a, n) = 1$ mellett teljesítik a tétel feltételét, ezeket univerzális álprímeknek hívjuk. További rossz hír, hogy 1992-ben igazolták, hogy végtelen sok univerzális álprím létezik, de a prímekhez képest jóval ritkábban

fordulnak elő. Tehát ha sok a-ra megvizsgáljuk, hogy teljesül-e a kongruencia, akkor valószínűleg prímet találtunk. Hogyan ellenőrizhető ez?
A válasz az ismételt négyzetre emelések módszere:

1. Kiszámítjuk $n-1$ kettes számrendszerbeli alakját.

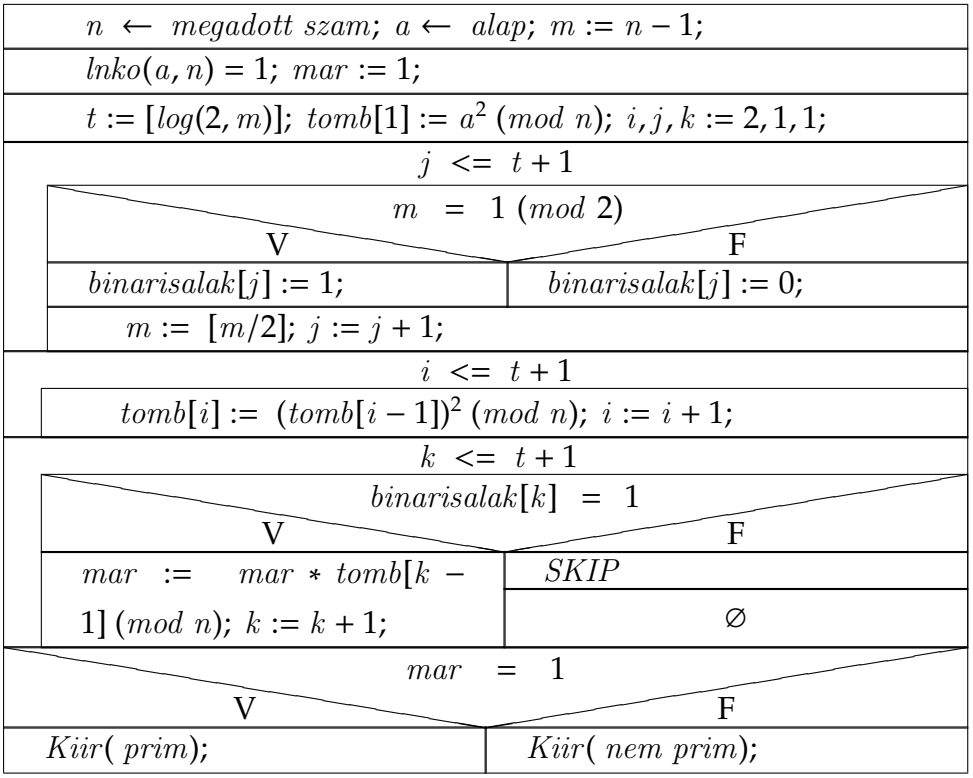
2. Kiszámítjuk a^2, a^4, a^8, a^{16}, ... (mod n) értékeket, úgy, hogy minden lépésben négyzetre emelünk, és vesszük az n-nel való osztási maradékot.

3. Összeszorozzuk a megfelelő a hatványokat, azaz azon indexűeket, melyek $n-1$ bináris alakjában 1-essel szerepelnek, és minden lépésben a szorzatot redukáljuk modulo n.

4. Ha így 1-et kapunk eredményül, akkor n-et prímnek nyilvánítjuk, különben n biztosan összetett szám.

Az algoritmus:

kis Fermat



Vizsgálat: Az algoritmusunk kész van, de kíváncsiak vagyunk, hogy milyen gyors.

1. A kettes számrendszerbeli alak $\log_2(n-1)$ lépésben véget ér, hiszen 2-vel osztunk maradékosan.

2. A struktogramm jelöléseivel itt t darab négyzetre emelést, és ennyi darab redukciót végzünk el, és mivel $2t \leqq 2\log_2(n-1)$, ezért ez legfeljebb $2\log_2(n-1)$ lépést vett igénybe.

3. Legfeljebb t darab szorzást és t darab redukciót végeztünk, ez szintén legfeljebb $2\log_2(n-1)$ lépés.

Tehát összesen legfeljebb $5\log_2(n-1)$ lépést hajtunk végre, ahol 1 lépés szorzás, négyzetre emelés, osztás vagy redukció. Egy 500-jegyű számra ez körülbelül 9000 lépés, ami számítógéppel gyorsan kiszámítható.
Megjegyzés: A második lépést végezhetnénk egy időben az elsővel. Eleve csak azokat a hatványokat számolnánk ki, ami nekünk kell, így még tovább csökkenne a lépésszám. A későbbiekben ezt felhasználjuk.
Kaptunk egy prímtesztet, ami gyors, és majdnem biztosan prímet talál, ugyanakkor végtelen sok számra tévedhet (univerzális álprímek). A következő prímteszt már az álprímeket is lebuktatja.

1.2. Solovay–Strassen prímteszt

A teszt feltalálóiról kapta a nevét, és a következőképpen szól:
Legyen $n > 1$ páratlan szám, és tekintsük az

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$$

kongruenciát, ahol $\left(\frac{a}{n}\right)$ a Jacobi–szimbólumot jelöli (lásd lentebb). Ha n prím, akkor ez a kongruencia minden $a \not\equiv 0 \pmod{n}$-re teljesül, ha n összetett, akkor egy modulo n vett teljes maradékrendszer kevesebb, mint a felére teljesül. Ismét feltesszük a kérdést, hogyan kapunk ebből prímtesztet.

Válasszunk véletlenszerűen 50 db $a \not\equiv 0 \pmod{n}$ értéket, melyek egymással is páronként inkongruensek. Ha a kongruencia legalább egy a-ra nem teljesül, akkor n biztosan összetett. Ha minden a-ra teljesül, akkor 2^{-50}-nél kisebb a valószínűsége, hogy a szám öszetett, hiszen egy teljes maradékrendszer elemeinek kevesebb, mint a felére teljesül a kongruencia, tehát egy próbára 1/2-nél kisebb az esélye, hogy teljesül a feltételünk, és ez mind az 50 független esetben így van. (50 helyett tetszőleges darabszámú a-ra próbálhatunk, így tetszőlegesen kicsivé téve a tévedés valószínűségét.)
Hogyan ellenőrizhető a feltétel? A kongruenciát könnyen láthatóan az előbb ismertetett ismételt négyzetre emelések módszerével gyorsan ki tudjuk számítani.

A Jacobi–szimbólumhoz először bevezetünk két fogalmat:

Legyen $p > 2$ prím és $(a, p) = 1$. Az a számot kvadratikus maradéknak, illetve kvadratikus nemmaradéknak nevezzük, ha az $x^2 \equiv a \pmod{p}$ kongruencia megoldható, illetve nem megoldható.

Az $\left(\frac{a}{p}\right)$ Legrende–szimbólumot a következőképp értelmezzük:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{ha } a \text{ kvadratikus maradék modulo } p; \\ -1, & \text{ha } a \text{ kvadratikus nemmaradék modulo } p; \end{cases}$$

A Jacobi–szimbólum definíciója a következő: $n > 1$ páratlan szám, $n = p_1 \cdot p_2 \cdot \dots \cdot p_r$, ahol p_i-k nem feltétlen különböző prímek, és $(a, n) = 1$. Ekkor

$$\left(\frac{a}{n}\right) := \left(\frac{a}{p_1}\right) \cdot \left(\frac{a}{p_2}\right) \dots \left(\frac{a}{p_r}\right)$$

szorzatként értelmezzük a Jacobi–szimbólumot, ahol $\left(\frac{a}{p_i}\right)$ Legrende-szimbólum. Azonnal látszik, hogy a definíciót nem használhatjuk fel a prímteszt készítése során, hiszen n prímtényezőire lenne szükség, ám ezeket tipikusan nem ismerjük.

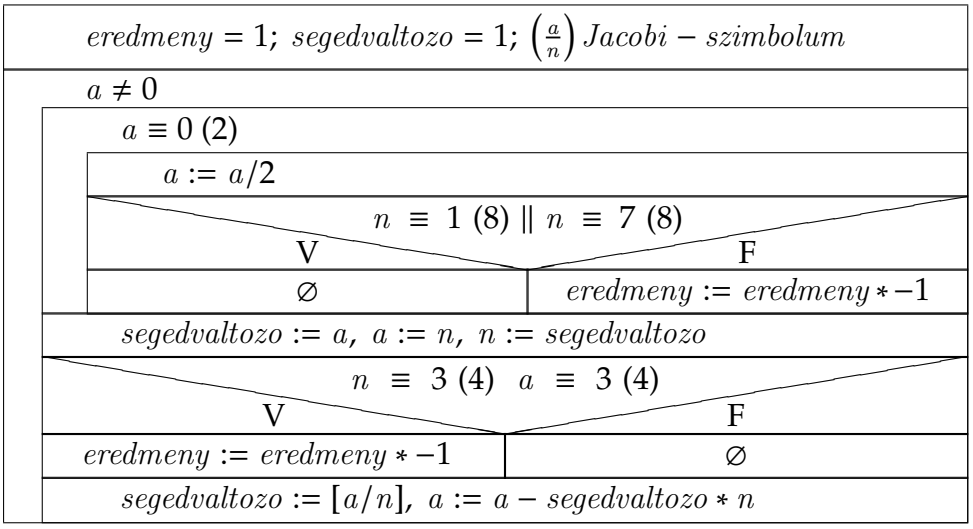
Ehelyett a Jacobi–szimbólumot tulajdonságai alapján könnyen számíthatjuk egy kibővített euklideszi algoritmussal. A tulajdonságok a következők:

1.2.1. Tétel (A Jacobi-szimbólum tulajdonságai). *Tegyük fel, hogy az alábbi Jacobi-szimbólumok értelmesek. Ekkor:*

1. $a \equiv b \ (mod\ n) \Rightarrow \left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$
2. $\left(\frac{ab}{m}\right) = \left(\frac{a}{m}\right)\cdot\left(\frac{b}{m}\right)$; $\quad \left(\frac{a}{mn}\right) = \left(\frac{a}{n}\right)\cdot\left(\frac{a}{m}\right)$;
3. $\left(\frac{-1}{m}\right) = \begin{cases} 1, & ha\ m \equiv 1 \pmod 4; \\ -1, & ha\ m \equiv -1 \pmod 4; \end{cases}$
4. $\left(\frac{2}{m}\right) = \begin{cases} 1, & ha\ m \equiv \pm 1 \pmod 8; \\ -1, & ha\ m \equiv \pm 3 \pmod 8; \end{cases}$
5. $\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right), & ha\ n \equiv m \equiv -1 \pmod 4; \\ \left(\frac{n}{m}\right), & \textit{egyébként, } (n,\ m \textit{ páratlan}). \end{cases}$

Az algoritmus során mindig arra törekszünk, hogy a számlálóban álljon a kisebb szám (ha nem így lenne, akkor az ötödik tulajdonságot használnánk ezek felcserélésére). Valójában egymással és 0-val inkongruens a-kat próbálunk, melyek egy teljes maradékrendszer elemeit alkotják, így ezeket érdemes n-nél kisebbre választani, tehát az $\left(\frac{a}{n}\right)$ Jacobi–szimbólumban tipikusan a nevező mindig nagyobb lesz. Ezután ha a számláló páros, akkor a negyedik és második tulajdonság alapján a kettes szorzókat leválasztjuk a számlálóból, és az eredményt ennek megfelelően változtatjuk. Ekkor a számlálóban garantáltan kisebb páratlan szám áll, tehát cseréljünk az ötödik tulajdonság alapján, és az első felhasználásával redukáljuk a számlálót, hogy ismét kisebb legyen a nevezőnél (maradékos osztást végzünk). Ezt addig csináljuk, amíg $a \neq 0$. A struktogrammban pontosan ez történik:

Kibovitett euklideszi algoritmus



A teszt vizsgálata:

1. Az ismételt négyzetre emelések módszeréről az előbb beláttuk, hogy $O(log_2 n)$ lépésszámú.

2. A Jacobi–szimbólum számítása is gyorsan megy. Ahogy említettük, ez egy kibővített euklideszi algoritmus. Részletesen, $\left(\frac{a}{b}\right)$ számolásánál, az a-t b-vel osztva maradékosan kapjuk, hogy

$$\left(\frac{a}{b}\right) = \left(\frac{r_1}{b}\right), \qquad \text{ahol} \qquad |r_1| < \frac{b}{2}.$$

Ha r_1 páros, akkor $\left(\frac{2}{b}\right)$ kiemelésével (ahányszor csak lehetséges) a számláló ismét feleződik. Ha r_1 páratlan, akkor r_1 a nevezőbe kerül, a számláló pedig b-nek r_1 szerinti r_2 maradéka, ahol ismét $|r_2| < \frac{r_1}{2}$. Tehát a számláló mindig feleződik. Tegyük fel, hogy az l-edik lépésben ér véget az algoritmus, azaz r_{l+1} már 0. Ekkor legfeljebb $l + 1$ lépést végeztünk, és mivel minden lépésben a számláló feleződik, így

$$1 \leq |r_l| \leq \frac{b}{2^l}$$

ezért

$$2^l \leq b, \quad \text{tehát} \quad l \leq \log_2 b.$$

Kapjuk, hogy a lépésszám legfeljebb $\log_2 b$. Ehhez még hozzájön a $\left(\frac{2}{m}\right)$ kiszámításához szükséges lépésszám, ami egy-egy maradékos osztást jelent 8-cal. Durván becsülve ebből is legfeljebb l darab van. A számláló és nevező cseréjével is változik az eredmény, ekkor az ötödik tulajdonság alapján egy-egy 4-gyel való maradékos osztást kell végeznünk. Könnyen láthatóan, legrosszabb esetben ebből is legfeljebb l darabot végeztünk. Ez összesen $3\log_2 b$ lépés.

3. Láttuk, hogy az ismételt négyzetre emelések módszere $5\log_2 n$ lépésben elvégezhető, ehhez hozzájön $3\log_2 n$ lépés a Jacobi–szimbólum számításakor. Tehát a Solovay–Strassen prímteszt lépésszáma $O(\log_2 n)$, ami számítógéppel szintén gyorsan számítható.

Tehát a második prímtesztünk gyors, előnye az elsővel szemben, hogy itt nincsenek álprímek, ugyanis ha egy teljes maradékrendszer modulo n elemeinek több, mint a felére ellenőriznénk Solovay-Strassen feltételét, akkor biztosan „lebukna" egy összetett szám. Persze nagy n-re ez időigényes lenne, ám erre nincs is szükség, hiszen például, ha csak ezer darab a-ra ellenőrzünk, az a lépésszám ezerszerese, ami még gyors, ám ekkor már gyakorlatilag 0 a valószínűsége a tévedésnek.

1.3. Miller–Rabin–Lenstra prímteszt

A teszt alapja, hogy ha p prím, és $x^2 \equiv 1 \pmod p$, akkor csak $x \equiv \pm 1 \pmod p$ lehetséges. Így ha veszünk egy $a \not\equiv 0 \pmod p$ számot, akkor az

$$a^{p-1},\ a^{\frac{p-1}{2}},\ a^{\frac{p-1}{4}},\ a^{\frac{p-1}{8}},\ \ldots$$

számok p-vel való osztási maradéka vagy mind 1, vagy szükségképpen egyszer -1 követi az 1-eseket, sőt a^{p-1}-nek szükségképpen 1-nek kell lennie (kis Fermat–tétel). Azonban, ha p nem prím volt, akkor megmutatható, hogy sok a-ra már másfajta sorozatot kapunk. Ezek alapján a Miller–Rabin–Lenstra teszt a következő:
Legyen $n > 1$ páratlan szám, $n - 1 = 2^k \cdot r$, ahol r páratlan. Tekintsük az

$$a^r,\ a^{2r},\ a^{4r}\ \dots\ a^{2^{k-2}r},\ a^{2^{k-1}r}$$

sorozatot. Ha a^r maradéka 1 modulo p, vagy a sorozat elemeinek n-nel való osztási maradékai közt előfordul a -1, akkor ezt jó sorozatnak hívjuk.

- Ha n prím, akkor minden $a \not\equiv 0 \pmod{n}$-re jó sorozatot kapunk.
- Ha n összetett, akkor egy modulo n vett teljes maradékrendszer elemeinek kevesebb, mint a felére alkot jó sorozatot.

Miben más ez, mint a Solovay–Strassen prímteszt? A sorozat elemeinek modulo n vett maradékai szintén ismételt négyzetre emelésekkel gyorsan számíthatók, sőt, itt nincs szükség Jacobi–szimbólumok kiszámítására. Továbbá bonyolultabb módszerekkel megmutatható, hogy egy teljes maradékrendszer elemeinek nemcsak hogy legfeljebb a felére, de kevesebb mint a negyedére alkot jó sorozatot egy összetett n. Az is megmutatható, hogy a Miller–Rabin–Lenstra "eredményesebb". Pontosítva:

1.3.1. Tétel (MRL eredményessége). *Egy adott n-re, ha egy a jó sorozatot alkot, akkor az kielégíti a Solovay–Strassen feltételt.*

Bizonyítás: Szeretnénk belátni, hogy ha egy a-ra jó sorozatot kapunk, akkor

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$$

is teljesül.

Ha n prím, akkor ez triviálisan teljesül, hiszen mindkét teszt lényege, hogy a prímek jó sorozatot alkotnak, és teljesítik a fenti kongruenciát minden a-ra. Tegyük fel tehát, hogy n összetett. Ekkor két részre bontjuk a bizonyítást, aszerint, hogy a^r maradéka 1 modulo p, vagy a sorozat elemeinek n-nel való osztási maradékai közt előfordul a -1.

1. Tekintsük az $a^r \equiv 1 \pmod{n}$ esetet. Mivel $n-1 = 2^k \cdot r$, így $\frac{n-1}{2} = 2^{k-1} \cdot r$, amit beírva a bal oldalra kapjuk, hogy

$$a^{\frac{n-1}{2}} = a^{2^{k-1}r} = (a^r)^{2^{k-1}} \equiv 1^{2^{k-1}} \equiv 1 \pmod{n}.$$

Tehát a bal oldalon 1 áll. Legyenek n prímosztói $p_1, p_2, \ldots, p_s$. Ekkor a jobb oldal definíció alapján

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \cdot \left(\frac{a}{p_2}\right) \ldots \left(\frac{a}{p_s}\right)$$

ahol $\left(\frac{a}{p_i}\right)$ Legrende–szimbólum. Tekintsük valamely i-re az $\left(\frac{a}{p_i}\right)$ tényezőt. Ez definíció szerint 1, ha a kvadratikus maradék modulo p_i, egyébként -1. Egy tétel szerint, a kvadratikus maradék modulo p_i pontosan akkor, ha

$$a^{\frac{p_i-1}{2}} \equiv 1 \pmod{p_i}.$$

Mivel a Jacobi–szimbólum ± 1 értéket vehet fel, és mivel r páratlan, így ha a jobb oldal r-edik hatványát vesszük, akkor a szimbólum és a hatvány egyszerre lesz $+1$ vagy -1. Ugyanez a Legrende–szimbólumokra is igaz. Azt állítom, hogy minden Legrende–szimbólum r-edik hatványa 1. Ehhez egy rövid állítást igazolunk:

1.3.1. Lemma. $\left(\frac{a}{p_i}\right)^r = 1 \iff \left(a^{\frac{p_i-1}{2}}\right)^r \equiv 1 \pmod{p_i}$

Bizonyítás:

=> $\left(\frac{a}{p_i}\right)^r = 1 \Longrightarrow \left(\frac{a}{p_i}\right) = 1$, hiszen r páratlan és a Legrende–szimbólum ± 1. Ekkor viszont az említett tétel szerint $a^{\frac{p_i-1}{2}} \equiv 1 \pmod{p_i}$. Ezt r-edik hatványra emelve pont a jobb oldalt kapjuk.

<= A kis Fermat–tételből tudjuk, hogy $a^{p_i-1} \equiv 1 \pmod{p_i}$. Ezért $a^{\frac{p_i-1}{2}} \equiv \pm 1 \pmod{p_i}$ lehetséges. De ha ez -1 lenne, akkor — mivel r páratlan — az r-edik hatványa is -1 lenne, ami ellentmondana a jobb oldalnak, és most ezt tettük fel. Tehát $a^{\frac{p_i-1}{2}} \equiv 1 \pmod{p_i}$, ez viszont a tétel alapján ekvivalens azzal, hogy a Legrende–szimbólum értéke 1. Így ennek r-edik hatványa is 1.

■

Ha tekintjük az $\left(a^{\frac{p_i-1}{2}}\right)^r$ kifejezést, akkor a feltételünk miatt

$$\left(a^{\frac{p_i-1}{2}}\right)^r \equiv (a^r)^{\frac{p_i-1}{2}} \equiv 1^{\frac{p_i-1}{2}} \equiv 1 \pmod{p_i}.$$

Az 1.3.1 lemma miatt így a Legrende–szimbólumok r-edik hatványai is 1-ek, tehát a Jacobi–szimbólum r-edik hatványa is 1. Mivel r páratlan, így a jobb oldalon is 1 áll.

2. A sorozat elemeinek n-nel való osztási maradékai közt előfordul a -1, azaz

$$\exists j : a^{2^j r} \equiv -1 \ (\mathrm{mod}\ n)$$

A bal oldal kiszámítása itt is könnyen megy, felhasználva, hogy $\frac{n-1}{2} = 2^{k-1} \cdot r$:

$$a^{\frac{n-1}{2}} = a^{2^{k-1}r} = a^{2^j 2^{k-j-1}r} = \left(a^{2^j r}\right)^{2^{k-j-1}} \equiv (-1)^{2^{k-j-1}} \equiv 1 \ (\mathrm{mod}\ n)$$

ha $j < k-1$. Be kell tehát látnunk, hogy $\left(\frac{a}{n}\right)$=1.
Vegyünk egy p-t, mely prímosztója n-nek. Ekkor persze p-re is teljesül a feltétel, vagyis $a^{2^j r} \equiv -1 \pmod{p}$. Ha ezt négyzetre emeljük, kapjuk, hogy $a^{2^{j+1}r} \equiv 1 \pmod{p}$. Ebből következik, hogy a modulo p rendjére fennáll

$$o_p(a) \nmid 2^j r \quad \text{és} \quad o_p(a) \mid 2^{j+1} r.$$

Tehát a rendje 2^{j+1} többszöröse lehet, sőt csak egy alkalmas l páratlan többszöröse, hiszen r-ben nem lehet 2-es prímtényező. De a kis Fermat–tételből tudjuk, hogy $a^{p-1} \equiv 1 \pmod{p}$, és egy tétel alapján a rend osztója minden k pozitív egész számnak, melyre $a^k \equiv 1 \pmod{p}$ teljesül, ezért $o_p(a) \mid p-1$ is fennáll. Az oszthatóság definíciója alapján

$$p - 1 = 2^{j+1} lh, \quad \text{azaz} \quad \frac{p-1}{2} = 2^j lh, \quad \text{egy alkalmas } h\text{-val.}$$

Ezt és a fent említett kvadratikus maradékokra vonatkozó tételt felhasználva az

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} = a^{2^j lh} = \left(a^{2^j l}\right)^h \pmod{p}$$

összefüggést kapjuk. Mivel $o_p(a) = 2^{j+1}l$, ezért $a^{2^{j+1}l} \equiv 1 \pmod{p}$ és $2^{j+1}l$ a legkisebb ilyen pozitív egész, és mivel p prím, ezért ha „gyököt vonunk" a kongruenciából, csak $a^{2^j l} \equiv -1 \pmod{p}$ lehetséges. Adódik tehát, hogy $\left(\frac{a}{p}\right)$ értéke $h = \frac{p-1}{2^{j+1}l}$ paritásától függ, ahol l páratlan.
Ezt minden prímosztóra elvégezzük. Legyen $n = p_1 \cdot p_2 \dots p_s$, ahol p_i-k nem feltétlen különböző prímek. Minden p_i-re kapunk h_i-ket, és a Jacobi–szimbólum definíciója alapján

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \cdot \left(\frac{a}{p_2}\right) \dots \left(\frac{a}{p_s}\right) = (-1)^{(h_1 + \dots + h_s)}.$$

Kellene tehát, hogy $\sum h_i$ páros. A fentiekből kifejezve p_i-ket kapjuk, hogy $p_i = 1 + 2^{j+1}l_i h_i$. Ezt beírjuk n kanonikus alakjába:

$$n = \prod(1 + 2^{j+1}l_i h_i) = 1 + 2^{j+1}(h_1 l_1 + \dots + h_s l_s) + \dots$$

A maradék osztható lesz 2^{j+2}-vel, ezért nem írtuk őket ki. Felhasználva még, hogy $n = 2^k r + 1$, és 2^{j+1}-gyel osztva kapjuk, hogy

$$2^{k-j-1}r = \sum h_i l_i + 2 \cdot (\dots)$$

Ebből adódik, hogy $\sum h_i l_i$ páros.

1.3.2. Lemma. *Legyen $h_i, l_i \in \mathbb{Z}^+$ $i = 1, 2 \dots s$, és l_i páratlan $\forall i$-re. Ekkor*

$$h_1 + h_2 + \dots + h_s \text{ páros} \iff h_1 l_1 + h_2 l_2 + \dots + h_s l_s \text{ páros.}$$

Bizonyítás:

=> Ha h_i-k összege páros, akkor volt valamilyen paritás sorrendjük (pl.: páros, páratlan, páratlan . . .). Ezeket páratlan számmal szorozva, a párosok párosok maradnak, a páratlanok pedig páratlanok, tehát a paritás nem változik, így az összeg sem.

<= Hasonlóan, a $h_i l_i$ szorzatoknak volt valamilyen paritás sorrendje. Ha tagonként leosztunk a megfelelő l_i-vel, akkor ugyanolyan marad a paritás, hiszen a szorzatokban a h_i-k voltak a meghatározók.

■

Az 1.3.2. lemmából adódóan tehát a h_i-k összege is páros, ezzel a tételünket beláttuk $j < k - 1$ esetén. A $j = k - 1$-re hasonlóan megy a bizonyítás. A feltételbe helyettesítve kapjuk, hogy

$$a^{2^j r} = a^{2^{k-1} r} = a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$$

tehát

$$\left(\frac{a}{n}\right) = -1\text{-et}$$

kell belátni. A bizonyítás ugyanúgy elmondható, mint a $j < k - 1$ esetben, csak a végén $j = k - 1$-et helyettesítve kapjuk

$$2^{k-j-1} r = 2^0 r = r = \sum h_i l_i + 2 \cdot (\dots).$$

Mivel r páratlan, így ebből egy páros számot kivonva kapjuk, hogy $\sum h_i l_i$ páratlan, ebből — az előbbivel hasonló módon — következik hogy $\sum h_i$ is páratlan, azaz

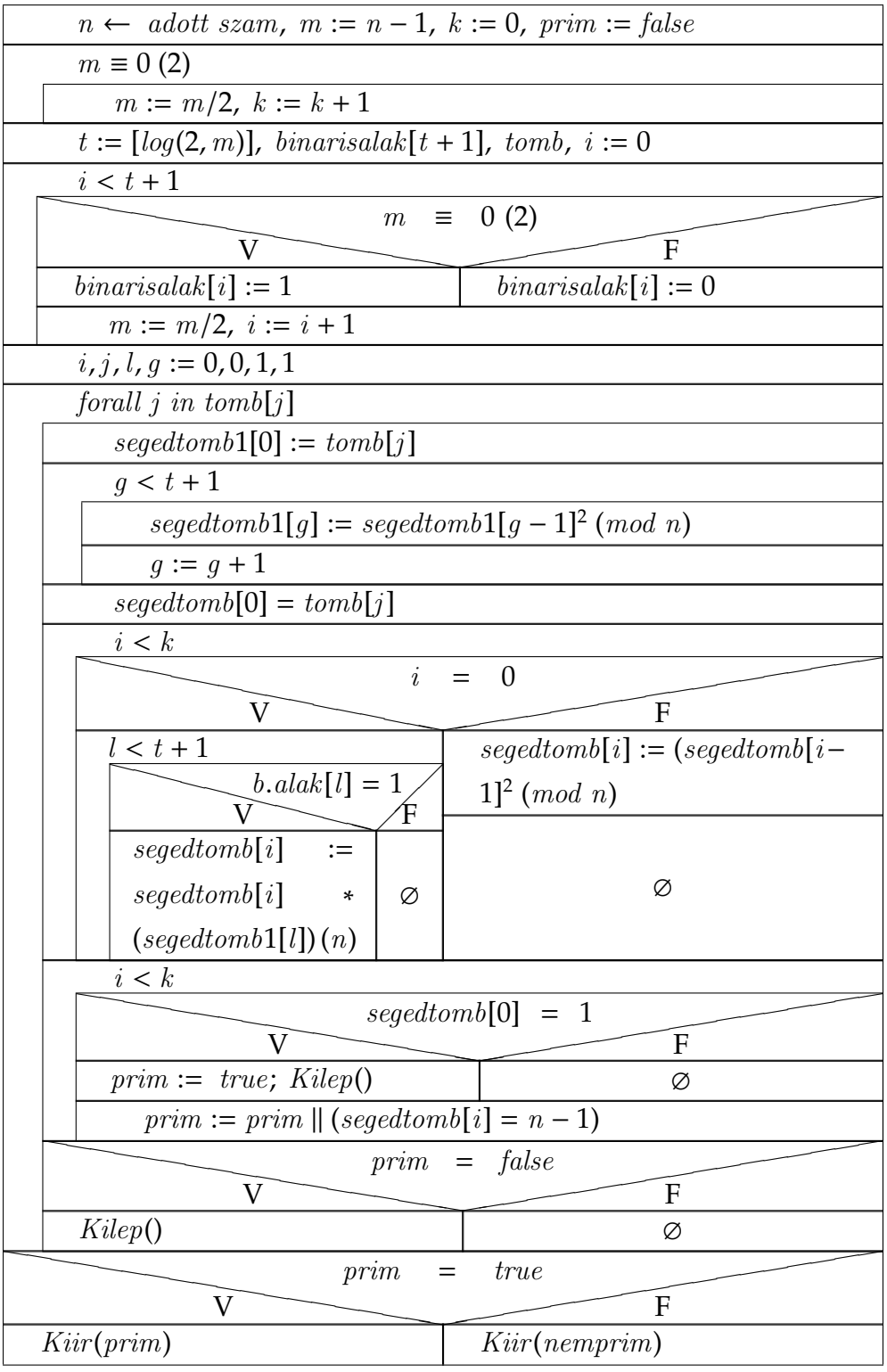
$$\left(\frac{a}{n}\right) = (-1)^{\sum h_i} = -1.$$

Ezzel a tételünket beláttuk.

■

Érdemes tehát megvalósítani:

Miller-Rabin-Lenstra



A $tomb$ nevű vektor tartalmazza azokat az a-kat, melyek egymással inkongruensek és $a \not\equiv 0 \pmod{n}$. A $binarisalak$ nevű vektorban $n-1$ kettes számrendszerbeli alakja található. A $segedtomb$-ben számítódnak az a hatványok maradékai modulo n, a $segedtomb1$ pedig az ismételt négyzetre emelések módszeréhez használt segítség. Ha a futás végén a $prim$ nevű logikai változó igaz, akkor n-et prímnek nyilvánítjuk, különben összetett számnak. A program elején a kezdeti értékadások után, kiszámítjuk r kettes számrendszerbeli alakját. Ezután a kettőhatvány hatványait meghatározzuk modulo n. A főciklusban előbb a^r értékét számoljuk ki, majd ebből a sorozat többi elemének értékét modulo n, és a ciklus végén ellenőrizzük, hogy jó sorozatot kaptunk-e (ha ez valahol nem teljesül, akkor megáll a program).

Vizsgálat:

1. A $segedtomb1$-ben kiszámítjuk az a kettőhatvány kitevőjű hatványait modulo n, ez t-szer fut, ami kevesebb, mint $\log_2 m$, tehát ez legfeljebb $2 \cdot \log_2 n$ lépés.

2. Itt is használjuk az ismételt négyzetre emelések módszerét, hogy meghatározzuk a^r maradékát modulo n. Ennek — ahogy azt az 1.1 -es részben beláttuk — a lépésszáma $O(\log_2 n)$.

3. Ezután $k-1$ darab négyzetre emelést és maradékos osztást végzünk, hogy meghatározzuk a sorozat többi tagjának maradékát. A legrosszabb esetben $k \log_2 m$-mel egyenlő, így az ehhez a részhez szükséges lépésszámot is felülről korlátozhatjuk $\log_2 n$-nel (sőt ez nagyon nagyvonalú becslés, hiszen ekkor az előbbi pont lépésszáma csupán 1 maradékos osztásból állna).

4. A végső lépésben csak ellenőrzéseket végzünk, legfeljebb k darabot, és átállítjuk a $prim$ logikai változó értékét, ha szükséges.

Tehát összesen ez is legfeljebb $const \cdot \log_2 n$ lépésszámú algoritmus, így ez az eddigi leghatékonyabb prímtesztünk.

1.4. Agrawal-Kayal-Saxena prímteszt

Eddig találtunk tetszőlegesen pontos, gyorsan ellenőrizhető prímtesztet. Felmerül a kérdés azonban, hogy van-e olyan prímteszt, ami nem csak tetszőleges, de 100 százalékos pontossággal eldönti egy számról, hogy az prím-e, és persze ezt gyorsan (logaritmikus lépésszámmal) számítja. A válasz pozitív, ugyanis 2002-ben három indiai matematikus, Agrawal, Kayal és Saxena talált egy ilyet. A teszt ötlete a következő:
Legyen $(c, n) = 1$. Ha tekintjük az $f_c = x^n - c$, $g_c = (x - c)^n$ polinomokat $\mathbb{Z}_n$ felett, akkor ha n prím, $f_c = g_c$ teljesül, egyébként van olyan k, amelyre $\binom{n}{k}$ nem osztható n-nel, így $f_c \neq g_c$. Az ötlet az, hogy $f_c = g_c$ helyett csak azt ellenőrizzük, hogy f_c és g_c egy alkamas $h \in \mathbb{Z}_n[x]$ polinommal osztva azonos maradékot ad-e. Ha h foka (n-hez képest) alacsony, akkor ez kivitelezhető ismételt négyzetre emelésekkel. Ha n prím, akkor bármely h szerinti maradékaik is egyenlőek, azonban az összetett számok ezt nem teljesítik. A teszt nehézsége megfelelő h polinomot találni.
Megjegyezzük, hogy ennek a módszernek egy továbbfejlesztett változata (2006-os eredmény) $(\log n)^6$ lépésszámú, ami a hatodik hatvány miatt lényegesen lassabb, mint az előbbi prímtesztek (bár biztosan prímet talál). Azonban az alkalmazásnál nagy véletlen prímszámot szeretnénk találni, ami azt jelenti, hogy a prímek számossága miatt átlagosan körülbelül ezer darab $200-300$ jegyű számról kell megvizsgálni, hogy prímek-e. Ez a Miller–Rabin–Lenstra teszttel sokkal gyorsabban elvégezhető, és mivel a tévedés valószínűsége gyakorlatilag 0, inkább ezt alkalmazzák.

2. fejezet

Prímfaktorizációk

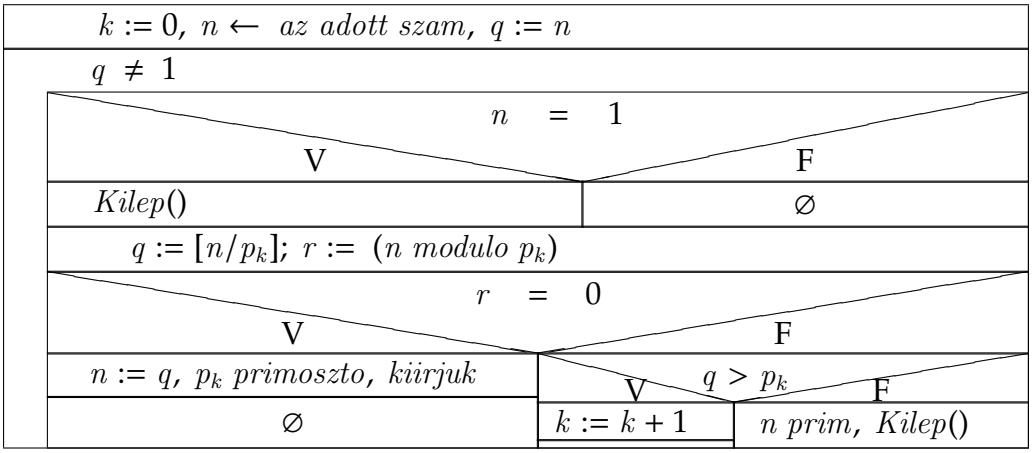
Az előző fejezetben láttuk, hogy tetszőlegesen nagy prímeket is gyorsan és tévedés nélkül tudunk találni. Ezek után szeretnénk tudni, hogy egy nagy tetszőleges összetett szám prímtényezős felbontását milyen módszerekkel tudjuk meghatározni, és ezek milyen hatékonyak.

2.1. Tényezőkre bontás osztással

Az első és egyben legtermészetesebb ötlet — amit középiskolában is használ az ember — a prímekkel való osztások próbálgatása. Ehhez persze, ha N a számunk, szükségünk van egy, a prímeket $\sqrt{N}$-ig tartalmazó táblázatra, ezek legyenek rendre $p_1, \dots p_r$. Ha ez megvan, akkor N-et megpróbáljuk elosztani p_i-kkel $i = 1, 2, \dots r$-re. Ha valamely i-re az osztási maradék 0, akkor ezt a p_i-t felírjuk a prímosztók közé, és N-et lecseréljük N/p_i-re, majd erre ugyanezt az eljárást elvégezzük, persze itt már elég az első p_i osztóra, és az annál nagyobbakra ellenőrizni. Ezt addig csináljuk, míg N-et p_i-kkel osztva egy $\sqrt{N}$-nél nagyobb prímet vagy 1-et kapunk. Így megkaptuk N prímtényezős felbontását.
Az algoritmus elején egy már megvalósított prímteszttel ellenőrizhetjük, hogy N prím-e, ekkor nem kell megkísérelni a felbontást. Ezért a továbbiakban mindig feltesszük, hogy N összetett szám.

Felvetődik egy probléma, mégpedig a prímeket tartalmazó táblázat elkészítése. Bár egy-egy számról gyorsan el tudjuk dönteni, hogy prím-e, de ha veszünk például egy 500-jegyű összetett számot, akkor ennek gyöke kb. 250 jegy körül van. Elég persze a páratlan számokat ellenőrizni, azok közül is az 5-tel végződőeket kizárhatjuk, de még így is nagyjából 10^{250} darab szám $\frac{4}{10}$-ét kellene ellenőriznünk, ami időigényes lenne, ezért egy picit módosítjuk az algoritmust. A táblázatunkban nemcsak prímek szerepelhetnek, hanem összetett számok is. Például válasszuk a $2, 3, 5, 7, 11, 13, 17, 19 \ldots$ számokat, ahol az első három tag után felváltva 2-vel, illetve 4-gyel növelünk (ez 2 és 3 többesein kívül minden páratlan számot tartalmaz). Ekkor a következő az algoritmusunk:

Tenyezokre bontas osztassal



Felmerülhet a kérdés, nem okoz-e problémát, hogy a táblázatunkban nem csak prímek szerepelnek, nem lehet-e, hogy az outputban nemprímek is megjelennek? Azt állítom, hogy nem.

2.1.1. Lemma. *Legyen N összetett szám és legyen $2 = p_1, p_2, \ldots p_r$ próbaosztók sorozata, mely sorozat $\sqrt{N}$-ig tartalmazza a prímeket, és még esetleg összetett számokat. Ekkor a fenti algoritmus nem írhat ki az outputra összetett számot.*

Bizonyítás: Tegyük fel indirekt, hogy az outputban szerepelt egy $h = p_1 \cdot p_2 \dots p_s$ összetett szám. Mivel $\sqrt{N}$-ig minden prímet tartalmaz a táblázatunk, így h prímtényezőit is legfeljebb 1 kivétellel. Az algoritmus futása során úgy kerülhetett h az outputra, hogy vele osztottunk maradékosan, és a maradék 0 lett, vagy az algoritmus végén megmaradt. De mivel a táblázatban növekvő sorrendben vannak a számok, így h prímosztóival is meg kellett próbálnunk osztani, mire eljutottunk h-ig. Például, ha h egy prímtényezője 2 volt, akkor 2-vel már megkíséreltünk osztani, és ha h-val osztható N, akkor persze 2-vel is. Az algoritmus felépítése miatt azonban 2-vel addig próbálkozunk, míg az összes 2-es prímtényezőit ki nem osztjuk N-ből, így az már h-val nem lesz osztható. Tehát a feltevésünk hibás volt, az algoritmus csak prímeket ír ki az outputra.

■

Az algoritmusunk valóban meghatározza N prímtényezős felbontását. A próbaosztásos módszer tovább javítható, ha a próbaosztások előtt végzünk egy prímtesztet, hiszen ha N-nek egy $\sqrt{N}$-nél jóval nagyobb prímosztója is van, akkor végigpróbálnánk az összes osztót $\sqrt{N}$-ig, ám ha gyorsan ellenőrizzük minden lépésben, hogy a maradék prím-e, akkor nem kell tovább próbálgatni. Már csak az a kérdés, hogy milyen gyorsan teszi ezt?

Vizsgálat: Hány próbaosztásra van szükségünk az algoritmusban? Ehhez nem árt tudni, hogy körülbelül hány prímszám van egy N pozitív egész számig. Legyen π(x) az x-ig terjedő prímek száma. Ekkor a tanultak alapján π(x) értékét $\frac{x}{\ln x}$-szel közelíthetjük.
Az nyilvánvaló, hogy az algoritmus futni fog, amíg p_{r-1}-et, azaz az utolsó előtti prímosztót meg nem találja. És ennél jobbat nem is tudunk mondani, hiszen csak szerencsés esetben nem kell sokáig futnia. A későbbiekben, az alkalmazásnál tipikusan két nagy prím szorzatából álló számokkal fogunk foglalkozni, így tehát közel $\sqrt{N}$-ig fog futni.
Vegyünk egy 500-jegyű számot. Ekkor ennek gyöke körülbelül 250-jegyű. Hány prím van eddig?

$$\frac{10^{250}}{\ln 10^{250}} = \frac{10^{250}}{\log_{10} 10^{250}} \cdot \log_{10} e \geq \frac{10^{250}}{2500} \geq 10^{246}$$

Tehát ebből látjuk, hogy ennek a lépésszáma reménytelenül sok, nagy számokra nem tudjuk meghatározni emberi időben.

2.2. Fermat-módszer

A próbaosztásos módszernél már előrevetítettük, hogy tipikusan nagy prímtényezekből álló számokat fogunk használni a gyakorlatban. Mielőtt a próbaosztásnál hatékonyabb algoritmusokat vizsgálnánk, előbb közelítsük meg más oldalról a problémát. A kis osztók helyett a nagyok megkeresésével próbálkozzunk. A módszert Fermat használta, és a következőképpen szól: Legyen $N = u \cdot v$, ahol $u \leq v$, és tegyük fel, hogy N páratlan (ekkor u és v is páratlan). Legyen

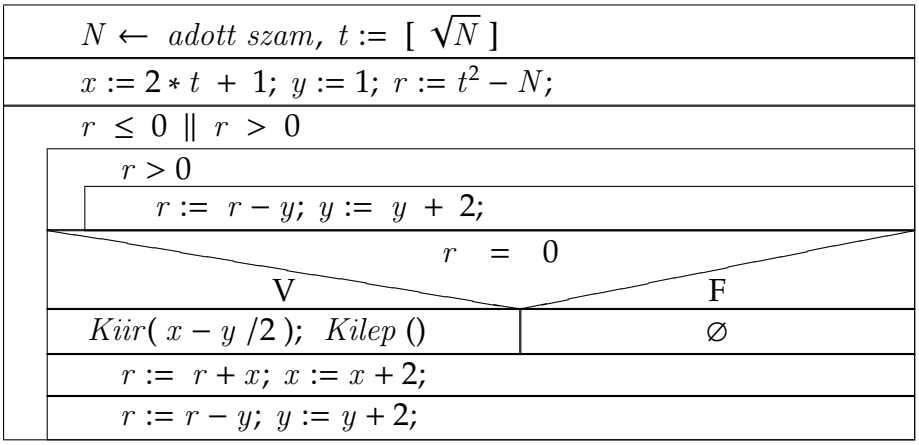
$$x := \frac{u+v}{2} \qquad \text{és} \qquad y := \frac{v-u}{2}.$$

Ekkor nyilván

$$N = x^2 - y^2, \qquad 0 \leq y < x \leq N.$$

Fermat módszere abban áll, hogy szisztematikusan keresünk olyan x-et és y-t, melyek kielégítik a fenti egyenletet. A módszer egy adott páratlan N-hez megkeresi N legnagyobb olyan osztóját, amely kisebb vagy egyenlő, mint $\sqrt{N}$. Az algoritmus az alábbi:

Tenyezokre bontas osszeadassal es kivonassal



Az algoritmus futása során kezdetben $x = [\sqrt{N}]$ és $y = 0$, majd a $2x + 1$, $2y + 1$ értékkel kezdjük a keresést. Az r az $x^2 - y^2 - N$ értéknek felel meg, vagyis méri, hogy milyen távol vagyunk N-től. Természetes módon, ha $r > 0$, akkor csak y értéket növeljük, egyébként mindkettőt.
A vizsgálatra nem térünk ki részletesen, csak az eredményeket közöljük. Az eddigi jelölések mellett, a szükséges lépésszám $v - [\sqrt{N}]$ értékkel arányos, ahol v a nagyobb prímosztója volt N-nek. Ha v távol van $\sqrt{N}$-től, akkor ez nagyon időigényes lehet.

2.3. Szita módszer

A következő módszer szintén egy $\sqrt{N}$-hez közeli prímosztót keres, lényegében a Fermat módszer egy módosított változata. Ha az előbb látott r-et, ami a távolságot mérte, átrendezzük, kapjuk hogy $x^2 - N = y^2$, tehát nem feltétlenül kell megtartani y-t, elég ha $x^2 - N$ -ről megvizsgáljuk, hogy az teljes négyzet-e.
Hogyan tudjuk ezt eldönteni egy számról? Vegyünk különböző m modulusokat. Ha egy szám teljes négyzet, akkor minden m-re az m-mel vett maradékának ezzel összhangban kell lennie. Például, ha m = 3-at választunk,

akkor egy szám négyzete modulo 3 csak 0 és 1 maradékot vehet fel, ugyanis egy szám eredetileg $0, 1, 2$ maradékot adhat 3-mal osztva, azaz $3k, 3k \pm 1$ alakú lehet, ezeket négyzetre emelve kapjuk, hogy milyen maradékot adhat egy négyzetszám modulo 3. Ugyanezt minden m modulusra megvizsgálhatjuk. Mivel N egy adott fix szám, ennek maradékát bármilyen számmal osztva meg tudjuk mondani, így ha megvizsgáljuk x maradékát is, akkor sok számot kizárhatunk. Például, ha $N \equiv 2 \pmod 3$ és $x \not\equiv 0 \pmod 3$, akkor ahogy az előbb láttuk $x^2 \equiv 1 \pmod 3$, így $x^2 - N \equiv 2 \pmod 3$ adódik, viszont akkor $x^2 - N$ nem lehet teljes négyzet, hiszen csak $0, 1$ maradékot vehet fel egy négyzetszám modulo 3. Tehát x-nek oszthatónak kell lennie 3-mal. Ugyanilyen vizsgálatokat minden m modulusra el tudunk végezni, ezzel egy adott N-hez kapunk egy-egy szitatáblázatot. Válasszunk tehát prímeket, melyek N-hez relatív prímek. Ekkor a kínai maradéktétel szerint a sziták egymástól függetlenek lesznek. Sőt, még az is igaz, hogy minden szita körülbelül a megmaradó értékek felét szűri ki, így ha veszünk l darab prímet, akkor 2^l érték közül 1-ről kell megvizsgálni csupán, hogy az teljes négyzet-e. Megjegyezzük, hogy túl sok prímet nem érdemes választani, hiszen ekkor a szitatáblázat elkészítése lenne időigényes.

Tehát az algoritmusunk egy N páratlan számhoz meghatározza N legnagyobb osztóját, amely kisebb vagy egyenlő, mint $\sqrt{N}$. Vegyünk $m_1, m_2, \ldots, m_r$ modulusokat, melyek páronként és N-hez relatív prímek. Elkészítünk r darab $S[i, j]$ szitatáblázatot, ahol $1 \leq j < r$, $0 \leq j \leq m_i$ és

$$S[i,j] = \begin{cases} 1, & \text{ha } j^2 - N \equiv y^2 \text{ (modulo } m_i\text{) megoldható ;} \\ 0, & \text{egyébként.} \end{cases}$$

Ha kis prímeket választunk modulusoknak, akkor ezek a megoldhatósági feltételek akár kézzel is ellenőrizhetőek, de sokkal egyszerűbb egy tétel felhasználásával kiszámítani a szitatáblázatot:

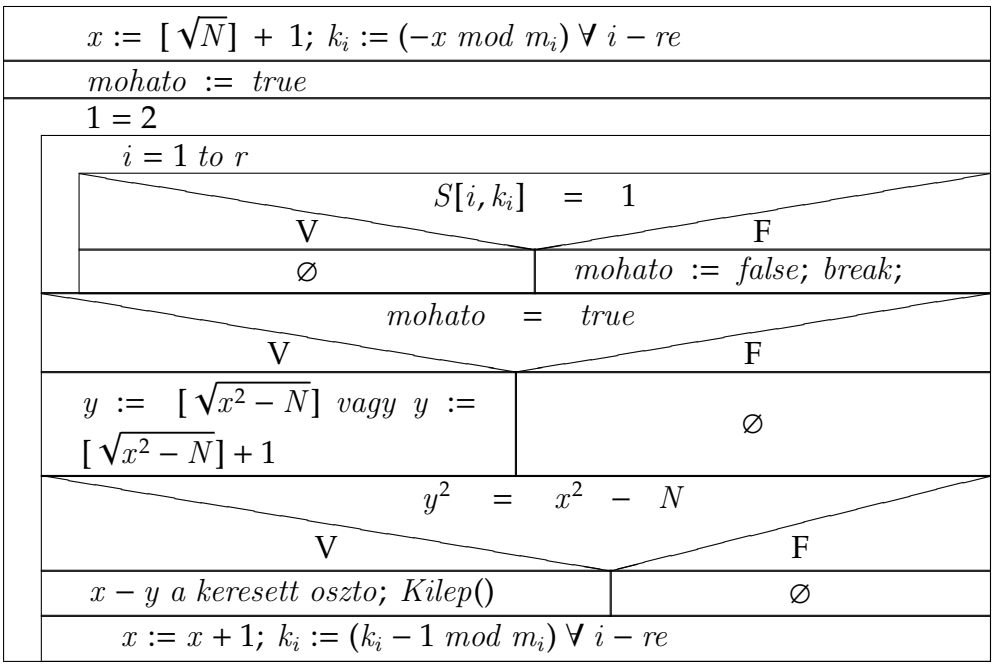
2.3.1. Tétel. *Tegyük fel, hogy p páratlan prím és $(a, p) = 1$, ahol $a \in \mathbb{Z}$. Ekkor az*

$$x^2 \equiv a \ (mod\ p) \text{ kongruencia megoldható} \Leftrightarrow a^{\frac{p-1}{2}} \equiv 1 \ (mod\ p)$$

teljesül, és a megoldásszám 2.

Tehát, ha $j^2 - N$ nem osztható m_i-vel, akkor csak a $(j^2 - N)^{\frac{m_i - 1}{2}}$ kifejezést kell kiszámolni modulo m_i, ami — a már ismert — ismételt négyzetre emelések módszerével gyorsan számítható.
Tehát ez alapján el tudjuk készíteni az $S[i, j]$ szitatáblázatokat, melyek alapján az algoritmus az alábbi:

Tenyezokre bontas szitalassal



2.4. Tényezőkre bontás Monte Carlo módszerrel

Most térjünk vissza az eredeti problémánkra, mely a prímtényezők meghatározása volt. Az ötletünk — mivel 2.1-ben láttuk, hogy minden prímet végigpróbálni $\sqrt{N}$-ig reménytelenül sokáig tart — az, hogy ne próbálgassunk ki minden prímet, hanem valamilyen véletlen módon lépkedjünk a prímek között. Látni fogjuk majd, hogy azért mégsem véletlen lépkedünk, már csak azért sem, mert nem tudunk a mai napig sem tökéletes véletlen-

szám generátort készíteni. Először azonban megoldunk két feladatot.

1.feladat: Adott x_1, x_2, x_3 ... egész számok sorozata, úgy, hogy $0 \leq x_n < m$ minden n-re teljesül. Legyen $f(x)$ egy egészértékű függvény, mely $[0, m)$-ből $[0, m)$-be képez. A sorozatunkat, valamilyen kezdőérték mellett, az $x_{n+1} = f(x_n)$ szabály szerint képezzük. Mutassuk meg, hogy a sorozat valahonnan kezdve periodikus, abban az értelemben, hogy léteznek μ és λ számok, amelyekre $x_0, x_1, \ldots, x_\mu, \ldots, x_{\mu+\lambda-1}$ mind különböző, de $x_{n+\lambda} = x_n$ ha $n \geq \mu$.

Megoldás: Tegyük fel, hogy a sorozat nem periodikus és $x_0 = a$ valamely $a \in [0, m)$-re. Mivel f függvény, ezért egyértelmű hozzárendelés, tehát a kezdőérték mellett minden további x_i egyértelműen meg van határozva. Legyen $b = x_1 = f(x_0)$. Ekkor ha $f(b) = a$ lenne, akkor azonnal periodikus lenne a sorozat, hiszen ezután a-ból ismét b-be kellene menni f egyértelműsége miatt. Tehát a sorozat minden új tagjánál f-nek egy addig még nem szerepelt értéket kell felvennie. De mivel f egész értékű és $[0, m)$-be képez, ezért csak véges sok értéket vehet fel, tehát előbb-utóbb szükségképpen egy már szerepelt számhoz jut a sorozat. Innentől pedig periodikus lesz a sorozat f egyértelműsége miatt, hiszen ugyanúgy kell képződnie a sorozatelemeknek, mint korábban.
Tehát tegyük fel, hogy az $x_n = k$ értéket másodjára vesz fel a sorozat. A k érték első sorozatbeli előfordulásának indexe jó lesz μ-nek, $n - \mu$ pedig jó lesz λ-nak. Ezen értékekre teljesül a megkívánt periodicitás.

2.feladat: Legyen $l(n)$ a legnagyobb, n-nél nem nagyobb kettőhatvány. Az előző feladat jelölései mellett bizonyítsuk be, hogy létezik $n > 0$, amelyre $x_n = x_{l(n)-1}$.

Megoldás: Indirekt tegyük fel, hogy minden n-re $x_n \neq x_{l(n)-1}$. Tekintsünk $\mu + \lambda$-nál nagyobb indexeket, és egy periódus hossza legyen l. Az $l(n) - 1$ számok rendre $2^s - 1$-gyel egyenlőek, $s = 0, 1, 2, \ldots$. Vegyünk olyan nagy s-et, hogy a $2^s - 1$ és $2^{s+1} - 1$ indexek között lévő távolság $2l$-nél nagyobb legyen. Ekkor a $2^s - 1$ és $2^{s+1} - 1$ között lévő sorozatelemek legalább egy egész

periódust tartalmaznak. A feltevésünk erre az s-re is igaz, azaz $x_{2^s-1} \neq x_i$, ahol $i = 2^s, 2^s + 1, \ldots, 2^{s+1} - 2$. Ám ezen i-k között biztosan van egy egész periódusnyi sorozatérték, hiszen olyan nagyra választottuk s-et, hogy biztosan beleessen. Mivel még $s > \mu + \lambda$, ezért a sorozat csak a perióduson belül előforduló értékeket vehet fel, tehát x_{2^s-1} értéke is csak egy perióduson belül előforduló érték lehet. Ekkor viszont valamely i-re $x_{2^s-1} = x_i$ kell legyen, ami ellentmond a feltevésünknek, ezzel beláttuk az állítást.

Legyen most $f(x)$ egy egész együtthatós polinom, és tekintsük az alábbi két sorozatot:

$$x_0 = y_0 = A, \qquad x_{m+1} = f(x_m) \pmod{N}, \qquad y_{m+1} = f(y_m) \pmod{p},$$

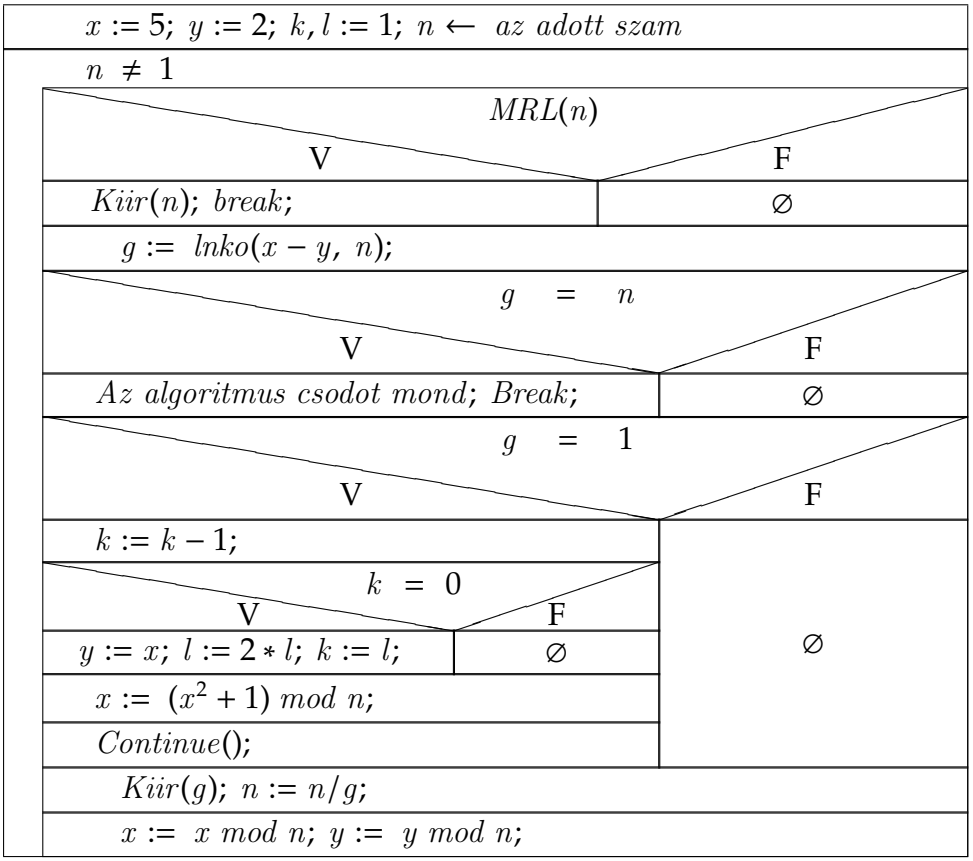
ahol p az N egyik prímosztója. Ekkor

$$y_m = x_m \text{ modulo } p \qquad \text{teljesül minden} \qquad m \geq 1\text{-re.}$$

Ekkor a feladatok feltételei nyilván teljesülnek az y_m sorozatra, így a 2. feladat alapján létezik $m \geq 1$, melyre $y_m = y_{l(m)-1}$. Így ha tekintjük az $x_m - x_{l(m)-1}$ kifejezést, akkor az osztható lesz p-vel. Megmutatható az is, hogy ha $f(y) \pmod{p}$ úgy viselkedik, mint a $[0, p-1]$ halmazt önmagába képező véletlen leképezés, akkor az ilyen m várható értéke $\sqrt{p}$ nagyságrendű.
Hogyan lesz ebből prímfaktorizáció? Ha N különböző prímosztói különböző m értékeknek felelnek meg, és ahogy azt már említettük, tipikusan ez lesz a helyzet, akkor úgy kaphatjuk meg ezeket az osztókat, hogy $m = 1, 2, 3 \ldots$ -ra kiszámítjuk $lnko(x_m - x_{l(m)-1}, N)$-et. Addig csináljuk ezt, amíg a felbontatlan rész prím nem lesz.
Felmerül a kérdés, hogy milyen f polinomot válasszunk, hogy az elég véletlen legyen? Belátható, hogy ha f elsőfokú, az nem lesz elég véletlen. És sajnos nem is tudjuk, hogy milyet lenne érdemes választani, a következő amit természetes módon kipróbálunk a másodfokúak közül a legegyszerűbb, az $f(x) = x^2 + 1$. Ha ezzel próbálkozunk, azt tapasztaljuk, hogy ez a polinom jónak tűnik, ám nem tudjuk, hogy nincs-e ennél jobb. Ezek után lássuk magát az algoritmust, mely egy N számnak meghatározza a prímtényezőit, azoban elképzelhető, hogy csődöt mond (az 1. feladatban látott

periodicitás miatt végtelen ciklusba esik, a legnagyobb közös osztónak a felbontatlan rész adódna, ekkor megáll a program):

Tenyezokre bontas Monte Carlo modszerrel



Az $MRL(n)$ kifejezés azt jelöli, hogy futtatjuk a Miller–Rabin–Lenstra tesztet N-re, hiszen ha N prím, akkor nincs mit faktorizálni. Megjegyezzük továbbá, hogy mielőtt a struktogramm végén kiírjuk g-t mint prímtényezőt, ellenőrizni kell, hogy g tényleg prím-e, hiszen lehet, hogy nem az. Ha összetett, akkor például futtathatjuk rá az első próbaosztásos módszert, ha viszont túl nagy szám g, akkor valószínűleg nem sikerül ezzel az algoritmussal meghatározni a felbontást.

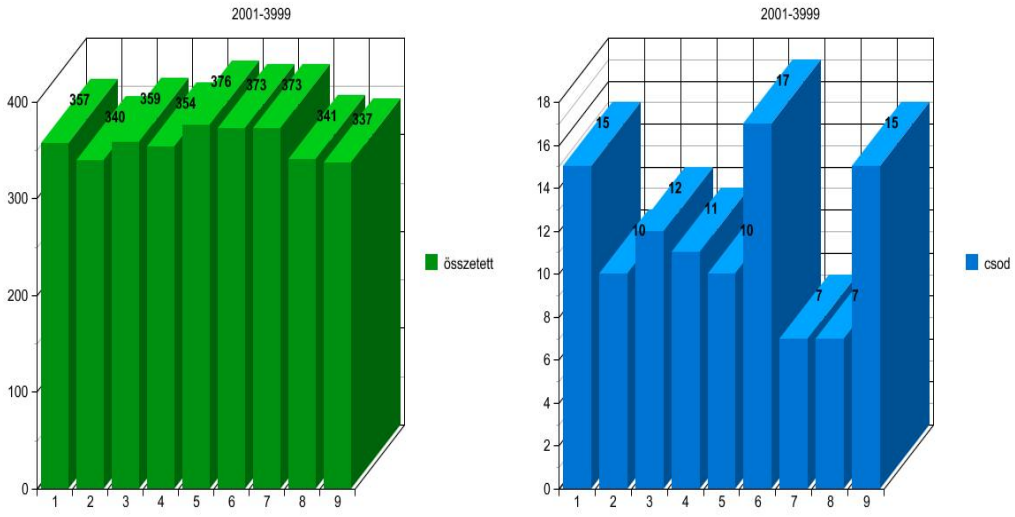
A Monte Carlo módszer tipikusan nagyobb prímtényezők elfogására alkalmas, a gyakorlatban érdemes egy darabig futtatni a próbaosztásos módszert,

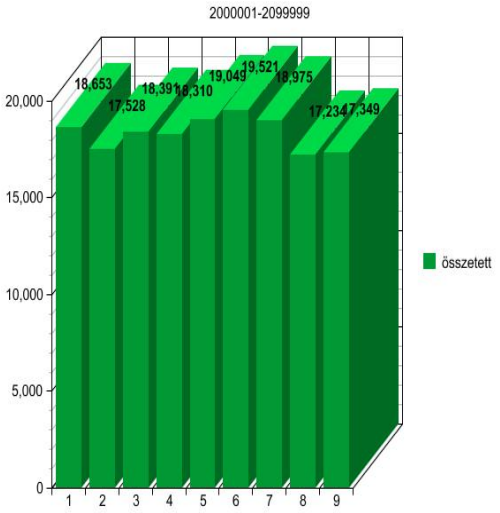
majd áttérni erre.

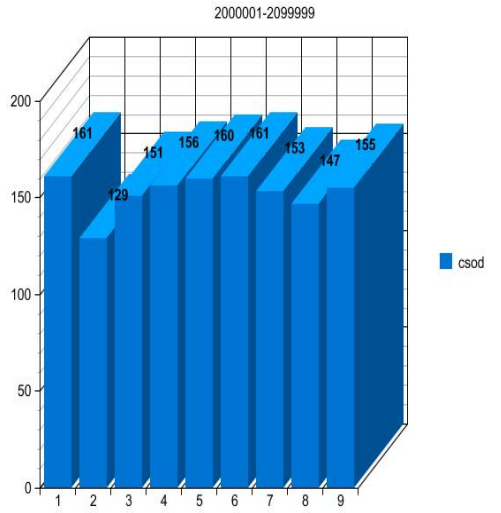
Ezt igazolják az alábbi diagramok is, melyek a megvalósított program futási eredményeit foglalják össze. A tesztelést 4 kategóriában végeztem el:

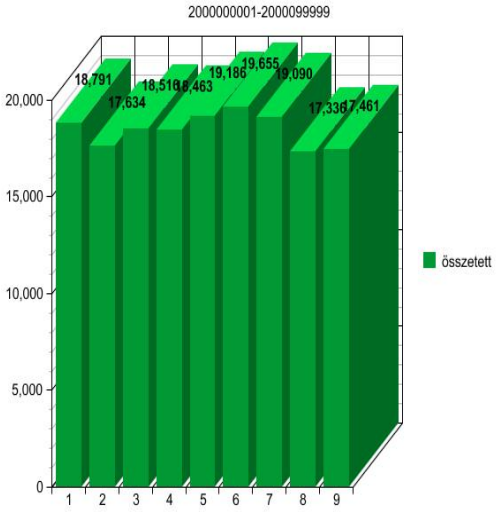
1. 2001-től 3999-ig a páratlan számok, 1000 darab
2. 2000001-től 2099999-ig a páratlan számok, 50000 darab
3. 2000000001-től 2000099999-ig a páratlan számok, 50000 darab
4. $2 \cdot 10^{13} + 1$-től $2 \cdot 10^{13} + 999$-ig a páratlan számok, 500 darab

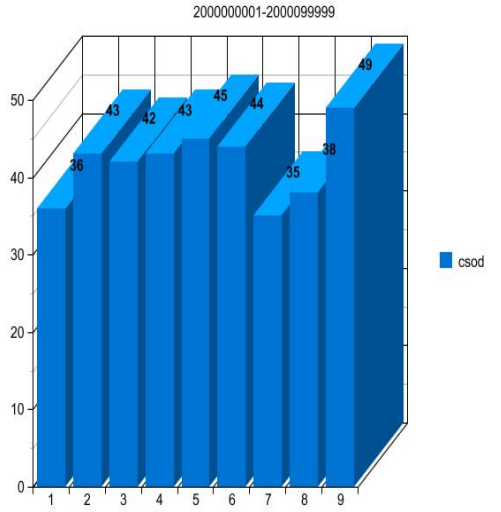
A páros számokkal nem érdemes foglalkozni, a program elején ajánlott kiosztani a kettes szorzókat, ha vannak. Minden kategóriában megvizsgáltam, hogy hány esetben mond csődöt a felbontás és hány esetben ír ki összetett számot az outputra. Továbbá, nemcsak az $x^2 + 1$ polinomot vizsgáltam, hanem $x^2 + c$ -t is, ahol $c = 1, 2, \ldots 9$, mivel nem tudni, hogy nincs-e jobb a $c = 1$ esetnél. Az alábbi eredményeket kaptam:
Az x-tengelyen a c konstansok vannak, az y-tengelyen a felvett értékek az egyes kategóriákra.

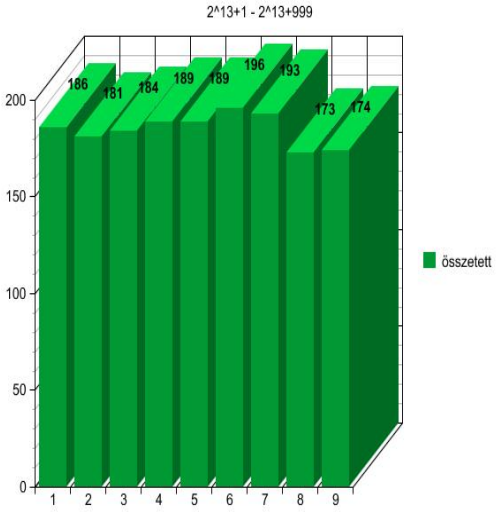


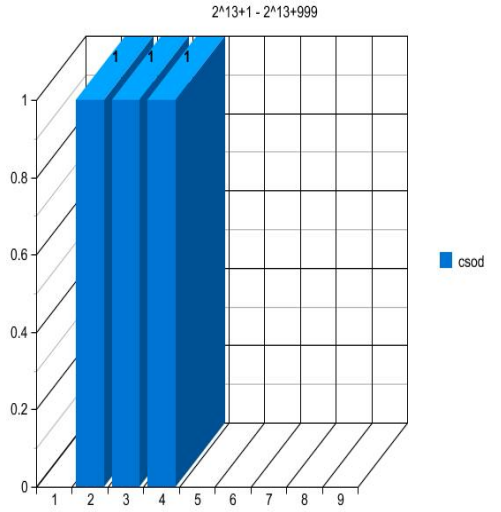

2000001-2099999
20,000
15,000
10,000
5,000
0
18,653
17,528
18,391
18,310
19,049
19,521
18,975
17,234
17,349
1 2 3 4 5 6 7 8 9
összetett


2000001-2099999
200
150
100
50
0
161
129
151
156
160
161
153
147
155
1 2 3 4 5 6 7 8 9
csod

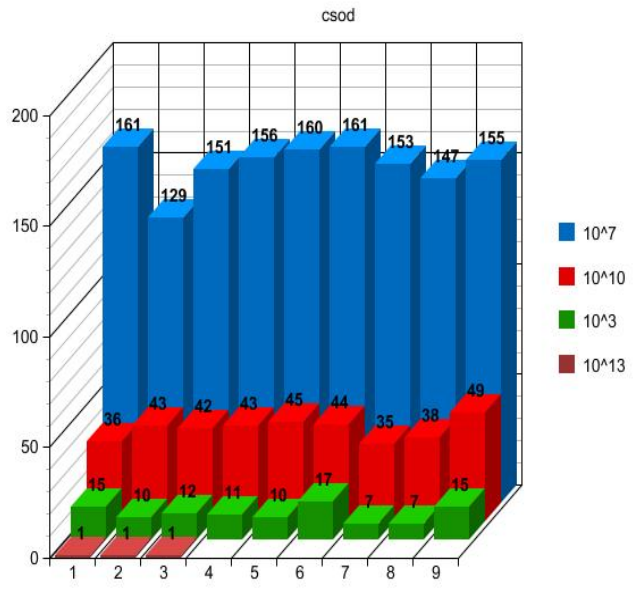

2000000001-2000099999
20,000
15,000
10,000
5,000
0
18,791
17,634
18,516
18,463
19,186
19,655
19,090
17,336
17,461
1 2 3 4 5 6 7 8 9
összetett


2000000001-2000099999
50
40
30
20
10
0
36
43
42
43
45
44
35
38
49
1 2 3 4 5 6 7 8 9
csod


2^13+1 - 2^13+999
200
150
100
50
0
186
181
184
189
189
196
193
173
174
1 2 3 4 5 6 7 8 9
összetett


2^13+1 - 2^13+999
1
0.8
0.6
0.4
0.2
0
1
1
1
1 2 3 4 5 6 7 8 9
csod

Vizsgáljuk meg először, hogy hány esetben ír ki a program összetett számot. Láthatóan jelentős eltérés nincs az egyes konstansok között, így mind a négy kategóriában körülbelüli értékkel számolok. Ezek — osztva a darabszámmal — rendre 350/1000, 18000/50000, 18000/50000, 180/500. Láthatóan az arányok nem sokat változnak a konstansok és a felbontandó számok nagyságának változásával, így azt lehet mondani, hogy körülbelül 36 százalékban írna ki az outputra összetett számot, tehát valóban indokolt ezekre is futtatni egy prímtesztet és megkísérelni felbontani őket.
Nézzük most, hogy hány esetben mond csődöt az algoritmus. Ha ismét az arányokat tekintjük, akkor már eltérő értékeket kapunk. Az ezres nagyságrendben 15/1000 körülbelül ez az érték. Ha viszont a 10^6 és 10^9 nagyságrendet nézzük akkor jelentős a különbség, hiszen az elsőben körülbelül 155/50000 a kudarc esélye, ezzel szemben a valamivel nagyobb számoknál már csak 40/50000. Sőt, ha megvizsgáljuk a legnagyobb kategóriát, ott azt látjuk, hogy 500 számból gyakorlatilag mindnél meg tudtuk határozni a felbontást. Ez is jól mutatja, hogy a Monte Carlo módszer tipikusan nagyobb számoknál eredményesebb.
Most hasonlítsuk össze a különböző polinomok által elért eredményt egy diagramban:



Azt vesszük észre, hogy a $c = 7,8$ esetben a csőd aránya minden kategóriában a legalacsonyabbak közt van, míg a többi c-re csak $1 - 1$ „jó eredmény" van (például $c = 1,2$). Ha 7-re és 8-ra megnézzük, hogy hány esetben írtak ki összetett számot az outputra, akkor azt tapasztaljuk, hogy a $c = 8$-ra ez szinte mindig a legalacsonyabb érték. Ebből úgy tűnik, hogy az $x^2 + 8$ polinom egy jó választásnak tűnik, ezért megmértem a futásidőket az egyes konstansokra. Ebből az adódott, hogy $c = 8$-cal futtatva semmivel sem lassabb a felbontás. A 10^6 nagyságrendnél a második leggyorsabb volt $9,23$ másodperccel ($9,17$ mögött, mely a $c = 9$ esetben adódott), sőt a 10^{13} nagyságrendben $3,67$ másodperc futásidővel ez szerepelt a legjobban, és a másik két kategóriában sem volt szignifikánsan lassabb a többinél. Ezen eredmények alapján az $x^2 + 8$ polinomot ajánlanám a Monte Carlo felbontáshoz, hiszen sebesség, összetett számok aránya, és csődök aránya szempontjából is előkelő helyen végzett (azonban megjegyzem, hogy nagyságrendbeli különbségek nincsenek, aszimptotikusan egyformán jónak tűnnek az egyes polinomok).
Vizsgáltam még más nagyobb c értékeket is, például $50, 51, 100, 101$ stb. de ezek láthatóan rosszabb eredményt adtak. Az összetett számok aránya 40 százalék körül volt és többször mondott csődöt a program. Ezután — mivel nem tudjuk milyen polinom lenne megfelelő — próbálkoztam magasabb fokú polinomokkal, de $x^2 + x + c$-vel is. Ezekre sem kaptam semmilyen biztató eredményt, sőt néhol meglepően rosszat. Úgy tűnik, valóban az $x^2 + c$ alakú polinomokkal érdemes próbálkozni, legalábbis nem találtam jobbat náluk.

Pollard belátta, hogy a módszer $\sqrt{p_{t-1}}$ lépésszámú, ahol p_{t-1} a második legnagyobb prímosztója N-nek, azonban gyakorlati eredmények alapján a futásidő általában jóval $N^{\frac{1}{4}}$ alá esik.

3. fejezet

Alkalmazás

3.1. RSA-kódolás

Ahogy azt az előző fejezetben láttuk, gyors prímfaktorizációs eljárást a mai napig nem találtunk. Erre a hiányosságra épít az RSA–titkosítás. Az RSA egy nyilvános kulcsú titkosírás, mely a következőt jelenti:

- Adott 3 személy, legyenek ők A,B és C. A és B kommunikálni szeretnének egymással, de úgy, hogy az egymásnak küldött üzeneteket C ne tudja elolvasni, ha az üzenetnek a birtokába jut.
- Adott tehát egy x üzenet, egy f nyilvános kulcs, és kell egy g titkos kulcs mindkét szereplőnek.
- A titkosítás azt használja ki, hogy egy művelet inverzét nehéz kiszámolni, tehát a g titkos kulcs f inverze lesz. Az esetünkben a nehéz művelet a faktorizálás lesz.
- Az x üzenetre A alkalmazza az ő titkos kulcsát, majd erre B nyilvános kulcsát, azaz kiszámítja $g_B(f_A(x))$-et, és ezt küldi el B-nek.
- Ezt az üzenetet B úgy tudja elolvasni, ha — ugyanúgy, mint A — alkalmazza a kapott üzenetre a saját titkos kulcsát, majd erre A nyilvános kulcsát, ugyanis $f_A(g_B(f_B(g_A(x)))) = f_A(g_A(x)) = x$.

- Ekkor C nem tudja elolvasni az üzenetet a g_B függvény hiányában, sőt hamis üzenetet sem tud küldeni, hiszen a kódoláshoz szükséges g_A-t sem ismeri.

Hogy néz ez ki pontosan az RSA-kódolásnál? Vegyünk két nagy véletlenül választott prímszámot. Ilyeneket például a Miller–Rabin–Lenstra prímteszttel kereshetünk, úgy, hogy véletlen páratlan számokat próbálgatunk. A prímek számossága és a teszt lépésszáma miatt gyorsan jutunk prímekhez. Legyen a két prím p és q, ezeket titokban tartja a tulajdonos. Kiszámítjuk $n = pq$-t, ami nyilvánosságra hozható. Mivel a tulajdonos ismeri p-t és q-t, ezért könnyen kiszámíthatja $\varphi(n)$-et. Ezután keres egy e számot melyre $(e, \varphi(n)) = 1$. Ekkor az ő f nyilvános kulcsa

$$f(y) = y^e \text{ legkisebb maradéka modulo } n \text{ lesz.}$$

Ekkor a g függvényt könnyen meghatározhatjuk, ha hasonló alakban keressük, hiszen mivel ezek egymás inverzei, ezért egymás után alkalmazva őket egy y-ra visszakapjuk y-t, másrészt ha $g(y) = y^s$, akkor $g(f(y)) = y^{es}$, vagyis minden y-ra teljesülnie kell a

$$y^{se} = y \pmod{N}$$

kongruenciának. A kis Fermat–tétel egy módosított változatából tudjuk, hogy minden y-ra $y^p \equiv y \pmod{p}$. Ugyanez q-ra is fennáll. Ezekből adódik, hogy minden y-ra $y^{t\varphi(n)+1} \equiv y \pmod{n}$, így csupán az

$$se = 1 + t\varphi(n)$$

diofantikus egyenletet kell megoldanunk s-re és t-re. Ez az euklideszi algoritmus segítségével gyorsan megoldható, így megkaptuk g titkos kulcsunkat. Azonban ezt más nem tudja megoldani, hiszen $\varphi(n)$ értékére szüksége lenne a számításhoz, ehhez viszont n prímtényezőit lenne kénytelen meghatározni. Tehát a kódolás feltöréséhez tudnia kellene C-nek gyorsan prímfaktorizálni, ami viszont jelenleg megoldatlan probléma.
Miután A és B is meghatározta a saját nyilvános és titkos kulcsát, akkor úgy tudnak egymással beszélgetni, hogy a betűknek és írásjeleknek megfeleltetnek számokat, ennek megfelelően az elküldeni kívánt szövegből egy számot

készítenek. Ezeket blokkokra osztják — csupán kényelmi szempontból — és a fent ismertetett f és g függvényeket alkalmazva minden blokkra elküldik egymásnak a kapott számokat. Ezt dekódolva visszakapják az eredeti szövegből készült számot, amit egyszerűen visszafejtenek.
A gyakorlatban tipikusan nagy prímeket kell keresni, hogy működjön a kódolás, hiszen kis számokat tudunk faktorizálni. Arra is vigyázni kell, hogy ne egymáshoz közeli számokat válasszunk, hiszen ekkor a 2. fejezetben ismertetett Fermat–módszer, vagy szitamódszer alkalmazásával hamar megtalálnánk a prímtényezőket, hiszen ekkor a szám gyöke közel lenne a prímekhez, viszont túl messzire sem érdemes menni a gyöktől, hiszen ekkor a Monte Carlo módszer megtalálná a „kicsi" prímtényezőt gyorsan. Arra is vigyázni kell, hogy a $\varphi(n)$-hez választott e számot ne válasszuk kicsinek, ugyanis ekkor is létezik hatékony támadás a kódolás ellen. Tehát egy 200-jegyű és 300-jegyű véletlen prímszám választásával nyugodtak lehetünk adataink biztonsága felől.

Irodalomjegyzék

[1] *D. E. Knuth: A számítógép programozás művészete II*

[2] *Freud Róbert, Gyarmati Edit: Számelmélet*