

# Nagyhatékonyságú számítások: faktorizálás

Vatai Emil

PhD hallgató, ELTE IK, Budapest, Komputeralgebra tanszék

2009. november 20.

## 1. Faktorizálás

Az oszthatóság vitathatatlanul a matematika egyik legősibb kérdése. Ősi civilizációk mint például az Egyiptomiak és a Maják 10.000 évvel ezelőtt már bevezették a 30 napból álló hónapokat és a 12 hónapból álló évet, melynek 360 napja van, ugyanis könnyen oszthatóak kis számokkal (ezeket sima számoknak nevezik a matematikában). Azonban vannak olyan számok, melyek csak saját magukkal és eggyel oszthatóak, ezeket felbonthatatlan illetve prímszámoknak nevezzük, továbbá nagy prím számok szorzatai nehezen faktorizálhatóak és a dolgozat eme kérdés hatékony megoldásait ismerteti.

A prímtesztelés és a faktorizálás a számelmélet két alapvető kérdése. A prímtesztelés arra a kérdésre ad választ, hogy egy szám prím vagy összetett, míg a faktorizálás egy összetett számnak a prímtényezőit, prímfaktorait, határozza meg.

### 1.1. A probléma

Legyen a továbbiakban  $N$  egész szám amit faktorizálni szeretnénk. Mivel ez előjel nem okoz gondot és bináris számrendszerben, amit a mai számítógépek használnak a számok ábrázolására, a kettővel való osztás nem okoz problémát, az általánosság megszorítása nélkül feltehetjük, hogy  $N$  páratlan pozitív. Továbbá feltehetjük, hogy  $N = p \cdot q$ , ahol  $p$  és  $q$  „nagy” prímelek. (Amikor azt mondjuk egy számra, hogy „nagy”, akkor például azt értjük alatta, hogy nem fér el egy-két gépi szóban egy számítógépen.)

### 1.2. A megoldás

Lényegében véve, kis számoknak tekinthetjük az 5 decimális számjegynél nem nagyobb számokat, közepméretűnek az 5–25 decimális számjegűeket és nagyoknak a 25 decimális számjegynél nagyobb

számokat. A kis számokat próbaosztással is hatékonyan faktorizálhatjuk, a középmeretű számok faktorizálását nem tárgyaljuk, ugyanis ezek több módon is hatékonyan felbonthatóak, például Pollard  $\rho$  vagy  $p - 1$  algoritmusával, vagy Lenstra ECM algoritmusával (elliptikus görbék módszere). Egy tetszőleges számról, az imént említett algoritmusokkal, leválasztva a kis és középmeretű prímtényezőket csupa nagy prímek szorzatából álló szám marad. Ezeket a számokat csak szitálással lehet faktorizálni és a dolgozat ezeknek az algoritmusoknak a működésébe ad rövid betekintést.

### 1.2.1. Próbaosztás

A faktorizálás legegyszerűbb, de ugyanakkor leglassabb megoldása a *próbaosztás*, azaz amikor a faktorizálandó  $N$  számot sorba leosztják a nála kisebb számokkal és ha nulla a maradék, akkor megvan az  $N$  egy prímtényezője. A próbaosztást fel lehet gyorsítani, ha az algoritmus csak  $< \sqrt{N}$  számokat és azokból is csak prímeket veszi figyelembe, de ezzel a megközelítéssel is csak kis számokat tud faktorizálni (egy milliótól nagyobb számokat már érdemesebb más módszerekkel faktorizálni). Ugyanakkor, ha  $N$ -ről nem tudunk semmit, érdemes végrehajtani a próbaosztást és így eltávolítani a kis faktorokat.

## 1.3. Négyzetek különbsége

Pierre de Fermat a 17. századi híres francia matematikus (hivatása szerint valójában ügyvéd volt, matematikával „csak” amatőr szinten foglalkozott) észrevette, hogy ha talál olyan  $x$  és  $y$  egész számokat, hogy  $N = x^2 - y^2$ , akkor  $N$  felírható mint,  $x + y$  és  $x - y$  szintén egészek, szorzata. Maurice Kraitchik (1882-1957) felfigyelt arra a tényre is, hogy nem feltétlenül szükséges az egyenlőség, elég ha  $N$  osztója  $x^2 - y^2$ -nek, vagyis  $x^2 \equiv y^2 \pmod{N}$ . Ha  $N = pq$  két nagy prím szorzata és  $x^2 \equiv y^2 \pmod{N}$ , akkor  $2/3$  valószínűséggel  $(x + y, N)$  vagy  $(x - y, N)$  eredménye  $p$  vagy  $q$  lesz (ahol  $(a, b)$   $a$  és  $b$  legnagyobb közös osztóját jelöli). A problémát arra vezettük vissza, hogy viszonylag hatékonyan állítsunk elő  $(x, y)$  párokat, melyre  $x^2 \equiv y^2 \pmod{N}$ . Ezen az elven alapszik a nagy számok faktorizálására leghatékonyabb algoritmusok, a szitáló algoritmusok.

## 2. Sziták

A két leghatékonyabb faktorizáló algoritmus az általános számtest szita (a továbbiakban GNFS, az angol General Number Field Sieve elnevezésből) és a négyzetes szita (QS, a Quadratic Sieve elnevezésből), illetve a több polinomos változata. 1996-óta minden faktorizáló rekordot GNFS-szev döntöttek meg és ez a ma ismert aszimptotikusan leggyorsabb faktorizáló algoritmus.

## 2.1. Négyzetes szita: QS

A GNFS a QS elvei alapján működik és a QS-ből fejlődött ki, ezért a megértéséhez célszerű a QS rövid áttekintése.

### 2.1.1. Faktorbázis előkészítése

**Definíció 1 (Faktorbázis)** Legyen  $U$  egész számok halmaza és  $p \in U$  akkor és csak akkor, ha  $p = -1$  vagy ha  $p$  prímszám és  $\left(\frac{N}{p}\right) = 1$ , vagyis  $N$  négyzetes maradék mod  $p$ .

A továbbiakban legyen  $n = \lfloor \sqrt{N} \rfloor$  és  $I \subset \mathbb{Z}$  egész számok egy intervalluma. Ha  $i \in I$  és  $x_i = n + i$  egész, elég közel van  $n$ -hez, akkor  $r_i = x_i^2 - N$  abszolút értékben elég kicsi és így gyorsan faktorizálhatóak.  $I$ -vel lesz a szita indexelve és általában úgy választják  $I$ -t, hogy  $I - E$ -től fut  $E - 1$ -ig, így a szita mérete  $2E$ . Legyen  $K \subset I$ , olyan indexhalmaz, melyre ha  $j \in K$ , akkor  $r_j$  faktorizálható  $U$  felet (vagyis minden  $p$  prímtényezője  $r_j$ -nek eleme  $U$ -nak). Legyen  $\mathbf{e}_j \in \mathbb{N}^{|U|}$  olyan kitevők vektora, hogy  $r_j = \prod_{p \in U} p^{e_{p,j}}$ , ahol  $e_{p,j}$  az  $\mathbf{e}_j$   $p$ -hez tartozó komponense. (Legyen  $e_{-1,j} = 0$  ha  $r_j \geq 0$  és  $e_{-1,j} = 1$  ha  $r_j < 0$ ).

### 2.1.2. Szitálás és próbaosztás

Az a cél, hogy előálljanak  $(x, y)$  párok, hogy  $x^2 \equiv y^2 \pmod{N}$ . Mivel  $r_i = x_i^2 - N$ , ezért  $r_i \equiv x_i^2 \pmod{N}$  mindig teljesül. Találni kell egy olyan  $K \subset I$  indexhalmazt, amelyre  $r = \prod_{k \in K} r_k$  négyzet szám, ugyanis akkor előáll a négyzetek különbsége. Ha  $r$  négyzetszám, akkor felírható  $r = y^2$  alakban is, ugyanakkor  $x = \prod_{k \in K} x_k$  jelöléssel:

$$x^2 = \prod_{k \in K} x_k^2 \equiv \prod_{k \in K} r_k = r = y^2 \pmod{N} \quad (1)$$

Minden  $p \in U$  prímszámhoz meghatározható a legkisebb  $t_p$  szám, amelyre  $t_p^2 \equiv N \pmod{p}$  teljesül (ilyen  $t_p$  létezik, ugyanis  $p \in U$ -ból következik, hogy  $N$  négyzetes maradék mod  $p$ ). Ha  $t_p$ -hez  $l \in \mathbb{Z}$ -szer hozzáadunk  $p$ -t, akkor az is teljesül, hogy

$$(t_p + lp)^2 = t_p^2 + 2t_plp + l^2p^2 \equiv t_p^2 \equiv N \pmod{N} \quad (2)$$

A szitálás abból áll, hogy egy  $I$  elemeivel indexelt  $S$  tömböt nullákkal inicializálunk. Utána minden  $p \in U$ -ra, ki kell számolni  $\log p$ -t és hozzáadni  $S$  tömb  $p + (t_p - n \pmod{p})$  pozíciótól kezdve, minden  $p$ -edik pozíciójához (ez (2) miatt lehet megtenni). Ez azt eredményezi, hogy  $S$   $i$ -edik pozíciójában, az  $r_i$   $U$ -beli prímfaktorainak a logaritmusai összegződnek, vagyis

$$S[i] = \sum_{\substack{p \in U \\ p | (n+i)^2 - N}} \log p = \sum_{\substack{p \in U \\ p | r_i}} \log p$$

Az elképzelés az, hogy  $S[i]$  értéke ha megegyezik  $\log r_i$ -vel akkor  $r_i$  faktorizálható  $U$  felett. A fent leírt módszer, azonban nem teljesíti ezt a tulajdonságot, ugyanis nem veszi figyelembe a hatványokat:  $\sum \log p$  helyett,  $\sum e_p \log p = \sum \log p^{e_p} = \log \prod p^{e_p}$  kellene, hogy legyen  $S[i]$  értéke, ha az aktuális  $r_i = \prod p^{e_p}$ . Ez úgy pótolható, hogy ha az algoritmus nem várja el, hogy  $S[i] = \log r_i$  legyen, hanem már akkor is elfogad egy számot, ha a megfelelő  $S[i]$  értéke elég közel van  $\log r_i$ -hez, sőt azzal is gyorsítani lehet az eljárást, ha  $S[i]$ ,  $\log r_i$  helyet az  $r_i$  logaritmusainak átlagával van összevetve, még alacsonyabbra vett küszöbvel, mely feletti  $S[i]$ -k elfogadottak (így nem kell minden  $r_i$  logaritmusát kiszámolni).

Így kiszitálható elegendő szám, melyek nagy valószínűséggel nem faktorizálhatók  $U$  felet és a szitán átmegy kevés olyan  $r_i$ -t amely nem faktorizálható  $U$  felett. A szitálás után próbaosztással meg kell határozni a fennmaradt  $r_i$  prímfelbontásait és ezzel együtt a megfelelő  $\mathbf{e}_i$  vektorokat.

### 2.1.3. Lineáris algebra

A szitálást (és próbaosztást), addig kell folytatni amíg nem áll elő több  $U$  felett faktorizált  $r_i$  mint  $|U|$ . Ekkor minden  $\mathbf{e}_i$  vektorhoz, megfeleltethető egy  $\mathbf{w}_i = \mathbf{e}_i \pmod 2$  vektor, mely az  $\mathbf{e}_i$  vektor, komponenseit tartalmazza  $\pmod 2$ .

$\mathbf{e}_i$  definíciójából adódik, hogy  $r_i \cdot r_j$  kitevőiből álló vektor  $\mathbf{e}_i + \mathbf{e}_j$ . Hasonló összefüggés érvényes  $\mathbf{w}_i + \mathbf{w}_j \pmod 2$ , ezért a cél az, hogy előálljon a  $K \subset I$  halmaz, amelyre  $\sum_{k \in K} \mathbf{w}_k \pmod 2$  a nullvektor adja. Ekkor a  $k \in K$  indexekre,  $r_k$ -k szorzata teljesíti a (1) összefüggést.

Ezzel a lépéssel nem foglalkozunk, ugyanis több hatékony megoldása is létezik. Lényegében egy Gauss eliminációt kell végrehajtani  $\mathbb{Z}/2\mathbb{Z}$  felett.

Az így kapott vektorok, meghatározzák a  $K$  halmazt, melyre  $y^2 = \prod_{k \in K} r_k$  és így  $y$  a kitevők vektoraiból könnyen előállítható:

$$y = \prod_{p \in U} p^{e_p} \text{ ahol } e_p\text{-k az } \mathbf{e} = \left( \sum_{k \in K} \mathbf{e}_k \right) / 2 \text{ vektor elemei}$$

A négyzetes szita részletesebb leírása megtalálható [BRES89] könyvben.

## 2.2. Általános számtest szita: GNFS

Az algoritmus futtatása előtt, a faktorbázisok és a szita méretein kívül, meg kell még adni paraméterként egy  $N$ -nél kisebb  $m$  pozitív (vagyis  $m \in \mathbb{Z}/N\mathbb{Z}$ ) egész és egy  $d$  fokú  $f \in \mathbb{Z}[x]$  főpolinomot, ahol  $d > 0$  páratlan egész és a  $f(m) \equiv 0 \pmod N$  teljesül. Feltehetjük, hogy  $f$  felbonthatatlan, ugyanis ha találunk egy  $\mathbb{Z}/N\mathbb{Z}$  felett felbontható polinomot, akkor  $N$  is könnyen faktorizálható. A továbbiakban  $f$ ,  $d$  és  $m$  a fent definiált fogalmakat jelölik és  $\theta$  egy olyan komplex szám, amelyre  $f(\theta) = 0 \pmod N$  felett.

A GNFS is hasonlóan működik mint a QS. Az  $r_i$ -ket előállító  $x \mapsto x^2 - N$  leképezést felfoghatjuk úgy is mint egy  $\mathbb{Z} \rightarrow \mathbb{Z}/N\mathbb{Z}$  gyűrű homomorfizmus. Az GNFS ezt az ötletet általánosítja.

**Definíció 2** ( $\mathbb{Z}[\theta]$  gyűrű) A fenti jelölések mellett,  $\mathbb{Z}[\theta]$  halmazt, a  $\{1, \theta, \theta^2, \dots, \theta^{d-1}\}$  összes  $\mathbb{Z}$ -lineáris kombinációjaként definiált, vagyis

$$\mathbb{Z}[\theta] = \left\{ \sum_{t=0}^{d-1} c_t \theta^t : c_t \in \mathbb{Z} \right\}$$

A GNFS-ben  $\mathbb{Z}$  helyett,  $\mathbb{Z}[\theta]$  lesz  $\mathbb{Z}/N\mathbb{Z}$ -re leképezve, a következő tételben szereplő  $\varphi$  leképezéssel.

**Tétel 1** A fenti jelölésekkel, létezik egy  $\varphi : \mathbb{Z}[\theta] \rightarrow \mathbb{Z}/N\mathbb{Z}$  leképezés, amelyre  $\varphi(1) = 1 \pmod{N}$  és  $\varphi(\theta) = m$ , és  $\varphi$  egy szürjektív gyűrűhomomorfizmus (gyűrűepimorfizmus).

Egy olyan  $T$  halmazt meghatározása a cél, amelyre

$$\prod_{(a,b) \in T} (a + b\theta) = \beta^2 \text{ és } \prod_{(a,b) \in T} (a + bm) = y^2 \quad (3)$$

teljesül, ahol  $\beta \in \mathbb{Z}[\theta]$ , ugyanis ekkor  $\varphi(\beta) = x$  jelöléssel

$$x^2 = \varphi(\beta)^2 \equiv \varphi(\beta^2) = \varphi \left( \prod_{(a,b) \in T} (a + b\theta) \right) \equiv \prod_{(a,b) \in T} (a + bm) \equiv y^2 \pmod{N}$$

és így előáll az  $x^2 \equiv y^2 \pmod{N}$  négyzetek különbsége.

### 2.2.1. A $\mathbb{Z}[\theta]$ -ről

$\mathbb{Q}(\theta)$ , vagyis a racionális számok teste  $\theta$  gyökkel bővítve, szintén testet ad, melyben a  $\mathfrak{D}$  algebrai egészek halmaza egy részgyűrű,  $\mathbb{Z}[\theta]$  pedig  $\mathfrak{D}$  egy részgyűrűje.

Hasonlóan, mint QS, a GNFS is sima számokból állítja elő a négyzetek különbségét, vagyis a fent említett  $T$  halmazt. Ezért szükség van a  $\mathbb{Z}[\theta]$  feletti „sima szám” fogalmára. Ha  $\beta \in \mathbb{Z}[\theta]$ , akkor a  $\langle \beta \rangle$  főideál egyértelműen faktorizálható  $\mathfrak{p}_i$  prímeideálok szorzatára, vagyis:

$$\langle \beta \rangle = \prod_{i=1}^r \mathfrak{p}_i^{e_i}$$

Tehát az algoritmus  $\mathbb{Z}[\theta]$  főideáljait faktorizálja prím ideálokra és pontosan azért lett a  $\mathbb{Z}[\theta]$  választva mint gyűrű amelyen a GNFS dolgozik, mivel  $\mathbb{Z}[\theta]$  prím ideáljai könnyen ábrázolhatóak:

**Tétel 2** Az eddigi jelölésekkel,  $\mathbb{Z}[\theta]$  minden elsőfokú prím ideálja bijektív kapcsolatban áll egy  $(r, p)$  párral, ahol  $p$  prím,  $r \in \mathbb{Z}/p\mathbb{Z}$  és  $f(r) \equiv 0 \pmod{p}$ .

Bevezethető az  $N$  norma, amely értelmezhető  $\mathbb{Z}[\theta]$  elemeire és  $\mathbb{Z}[\theta]$  elemei által generált fő-ideálokra is, és ekkor  $|N(\alpha)| = N(\langle \alpha \rangle)$  ha  $\alpha \in \mathbb{Z}[\theta]$ . Az  $a + b\theta \in \mathbb{Z}[\theta]$  elemnek a normája  $N(a + b\theta) = (-b)^d f(-a/b)$  is könnyen számolható. Továbbá ha egy  $\mathfrak{p}$  egy prímszorzó ideál, amely az  $(r, p)$  párral ábrázolható, akkor  $p \mid N(\langle \mathfrak{p} \rangle)$  és mivel  $N$  egy multiplikatív függvény, ezért ha  $p \nmid N(\alpha)$ , akkor  $(r, p)$ -vel ábrázolt  $\mathfrak{p}$  prímszorzó ideál sem osztója  $\langle \alpha \rangle$ -nak, ahol  $\alpha \in \mathbb{Z}[\theta]$ .

A  $\mathbb{Z}[\theta]$  -vel kapcsolatos állítások bizonyított számelméleti tételek következményei, melyek szükséges és elégséges feltételt is adnak az oszthatóságra ezekben az absztrakt gyűrűkben és általában  $\mathfrak{D}$  algebrai egészekre érvényesek. A tételek pontos kimondása és a GNFS többi részlete, itt nem tárgyalt részlete, mint például a  $\mathbb{Z}[\theta]$  feletti gyökkvonás és a kvadratikus karakterek, megtalálható a [BRIGGS] cikkben.

### 2.2.2. Szitálás

**Definíció 3 (Racionális és Algebraifaktorbázis)** Az  $R$  és  $P$  halmazokat rendre racionális illetve algebrai faktorbázisnak nevezik, és a következő módon definiáltak:

$$R = \{(r, p) : p \text{ prím és } r = (m \pmod p)\}$$

$$P = \{(r, p) : p \text{ prím és } f(r) \equiv 0 \pmod p\}$$

A szitálás hasonlóan történik mint QS-ben, annyi különbséggel, hogy most  $S$  nem egy, hanem két dimenziós tömb lesz, amelynek az  $S[i][j]$  értéke az  $a + bm$  illetve az  $a + b\theta$  számokat ábrázolja, ahol  $-I \leq i \leq I$  és  $0 \leq j \leq J$  valamilyen  $I$  és  $J$  korlátokra. Soronként szitálva, minden  $(r, p)$  párra, a  $j$ -edik oszlop szitálása úgy történik, hogy  $i = jr$  sortól kezdve, a  $j$ -edik oszlop minden  $p$ -edik sorát  $\log p$ -vel növeljük. A racionális számokhoz tartozó  $S[i][j]$  értékeket  $\log a + bm$ -mel, míg az algebrai számokhoz tartozó értékeket pedig  $N(a + b\theta)$ -val kell összevetni.

A szitálás után jön a próbaosztás, ami előállítja az  $\mathbf{e}_{(a,b)}$  vektorokat, amiből triviálisan adódnak a  $\mathbf{w}_{(a,b)}$  vektorok és ezekből pedig egy bináris Gauss eliminációnak megfelelő eljárással megkapjuk a  $T$  halmazt amelyre teljesül az (3) összefüggés.

## 3. A szitálás gyorsítása

Manapság a technológia fejlődése és a mikroprocesszorok órajelének a növekedése miatt, egy hatékony program tervezésekor, nem csak a műveletek számát és gyorsaságát kell figyelembe venni, ugyanis nem ez a domináló tényező egy program végrehajtási sebességében. A mai számítógépek teljesítményei, akkora adatmennyiségek mozgatását és feldolgozását teszik lehetővé, hogy egy program gyorsaságát többnyire a hatékony memóriakezelés határozza meg.

A hatékony memória kezelés a memória hierarchia megfelelő kihasználásával oldható meg. A memória hierarchia arra a felépítésre utal, hogy a processzor regisztereitől a merevlemezig a memóriák méretei növekednek míg az átviteli sebességük csökken és az elérési sebességük növekszik. A szokásos PC számítógépek memóriahierarchiájának felépítése a következő:

- Regiszterek: 32 vagy 64bit
- CPU cache: néhány kilobyte-tól pár megabyte-ig
- RAM: 1-4 gigabyte
- Merevlemez: manapság egy terabyte sem szokatlan

Az lenne az elképzelés, hogy a szítálás hatékonyan végre lehessen hajtani, szekvenciálisan, durván L1 cache méretű szakaszonként. A továbbiakban a szítálás problémájára javasolunk egy jobb megoldást, feltéve hogy az  $S$  szita tömb jóval nagyobb a cache memóriánál (sőt előfordul, hogy a RAM memóriánál is nagyobb), és egy  $F$  halmaz, minden  $p$  elemére, amely egy prímszám, az  $r_p$  pozíciótól kezdve, minden  $p$ -edik pozícióra végre kell hajtani valamilyen  $p$ -től függő  $g_p$  eljárást (a fenti algoritmusokban ez a  $\log p$ -vel való növelés volt). A  $g_p$  eljárás végrehajtását, egy tömb  $i$ -edik elemén  $g_p(i)$ -vel fogjuk jelölni.

### 3.1. Adatstruktúrák: körök és edények

Egy konkrét implementációban, az  $F$  halmaz elemei természetesen egy tömbben lesznek tárolva. Ezért célszerű bevezetni a  $p_i = p$  és a  $r_i = r_{p_i}$  jelölést ha a  $(r, p) \in F$  a tömb  $i$ -vel indexelt pozícióján található, továbbá a tömbre is  $F$ -el lehet hivatkozni, vagyis az  $F$  tömb  $i$ -edik eleme, legyen az  $(r_i, p_i)$  pár.

A szítát szekvenciálisan, darabonként fogjuk beolvasni és ezeket a darabokat nevezzük *blokkok*-nak és jelöljük  $B$ -vel a blokk méretét (byte-okban). Továbbá vezessük be az  $I_k$  jelölést a  $[kB, (k+1)B)$  egészek intervallumára ( $k \in \mathbb{Z}$ ).

**Definíció 4** Ha  $F (r_i, p_i)$  rendezett párok halmaza, akkor legyenek

$$C_k = (\mathbb{Z} \times I_k) \cap F$$

a  $k$ -edik kör.

Másképp: a  $k$ -edik kör, tartalmazza az összes olyan  $F$ -beli  $(r_i, p_i)$  párokat, amelyekre  $kB \leq p_i < (k+1)B$  teljesül.

A kör elnevezés az adatstruktúra ciklikus természetéből adódik, amit majd később részletezünk. A körök száma kiszámolható az  $F$  méretéből és  $B$ -ből. A körök halmaza mutatópárok tömbjeként implementálható, amit  $C$ -vel fogunk jelölni. Az első mutató az  $F$  azon elemére mutat, ahol az adott kör kezdődik. Általában a  $F$  tömb  $p_i$  szerint rendezett ha nem az, akkor rendezés után, könnyen meghatározhatók a körök határai. A körök második mutatója az *aktuális edényre* mutat.

**Definíció 5** A  $C_k$  kör,  $j$ -edik edénye  $b_j^{(k)} = C_k \cap (I_k \times \mathbb{Z})$  ahol  $0 \leq j \leq k$ .

*Másképp: most egy körön belül,  $r_i$  szerint csoportosulnak a  $(r_i, p_i)$  párok edényekbe.*

**Definíció 6** A  $C_k$  kör, aktuális edénye  $b^{(k)} = b_j^{(k)}$  valamilyen  $0 \leq j \leq k$  indexre.

Az edények halmaza is  $F$  elemeire mutatók tömbjeként implementálható. Miután a körök be lettek határolva, körönként kell rendezni  $F$  elemeit  $r_i$  szerint és hasonló algoritmussal mit amit a körökre alkalmaztunk, el kell menteni az edények első elemének memóriacímét. Az összes edény kezdőelemeinek címei egy nagy tömbben tárolhatók és mivel a  $k$ -edik körben  $k + 1$  edény van, ezért az összes edények száma  $\frac{M(M-1)}{2}$ , ahol  $M$  a körök számát jelöli. A köröket ábrázoló tömb elemeinek a második mutatóját, az edények tömbjében az adott kör első edényt ábrázoló mutató címére kell inicializálni.

## 3.2. Szítálás

A szita gyorsításának lényegét a következő állítás foglalja magában:

**Állítás 1** Ha  $p \in I_k$  és a  $t$ -edik blokkban a  $0 \leq r < p$  pozíción kell  $g_p$ -vel szítálni, akkor a következő szítálásra vagy  $t + k$ -edik vagy  $t + k + 1$ -edik blokkban fog sor kerülni.

Az állítás a definíciókból (pontosabban  $I_k$  definíciójából) következik.

Feltehetjük, hogy a szita tömb mérete  $B$  többszöröse. Ekkor a szita egy  $S_t$  sorozatként is kezelhető, ahol az algoritmus sorba az  $S_1, S_2 \dots$  blokkokat olvassa be a memóriába. A fenti állítás azt mondja ki, hogyha  $p$ -vel az  $S_t$  blokkban szitáltunk és  $p \in Bk$  és  $B(k + 1)$  között van, akkor ezzel a  $p$ -vel nem kell szítálni  $S_{t+k}$  vagy  $S_{t+k+1}$ -ig. Ez azt teszi lehetővé, hogy, úgymond, „végig guruljunk” a szitán blokkonként illetve edényenként (ugyanis minden blokkot egy-egy edénnyel kell végigszítálni).

A körként definiált adatstruktúrákat geometriai körökként lehet elképzelni, amelyeknek az elemeit ciklikusan tudjuk lekérdezni, vagyis úgy hogy az utolsó elem rákövetkezője az első. Ezekon a körökön, meg vannak határozva az edények határai, vagyis minden kör fel van osztva  $k + 1$  körszeletre. A szítálás úgy történik, hogy elkezdjük az első blokkal,  $S_1$ -gyel és szítáljuk minden  $C_k$  körrel. A  $C_k$  körrel, az  $S_t$  blokkot úgy szítáljuk, hogy a  $b^{(k)}$  aktuális edény  $(r, p)$  elemeire, végrehajtjuk az  $S_t$



$r$ -edik pozícióján a  $g_p(r)$  műveletet és kiszámolva az új  $r$  értéket, a  $(r, p)$  párt vagy a  $b^{(k)}$  edényben hagyjuk vagy az előtte lévő edénybe helyezzük. Ezután a  $b^{(k)}$  utáni blokk lesz az aktuális és amikor ezt az eljárást végrehajtjuk minden körre, a következő  $S_{t+1}$  blokkot olvassuk be a memóriába és így folytatjuk az utolsó blokkig.

Sorban minden  $S_t$  blokkon végrehajtjuk, minden  $C_k$  szerint a szitálást. Egy adott  $S_t$  blokkra a szitálás a következő módon történik:

$k = 0$  **eset:** Ha  $(r_i, p_i) \in C_0$ , akkor minden  $S_t$ -re, végig kell szitálni minden  $p_i$ -vel  $r_i$ -től kezdve. A szitálás úgy történik, hogy végrehajtjuk  $g_{p_i}(r_i)$ -t és  $r_i$ -t kicseréljük  $r_i + p_i$ -vel amíg  $r_i < B$ .

Amikor a ciklus lefut,  $r_i \geq B$  fog teljesülni és ekkor  $r_i$ -t kicseréljük  $r_i - B$ -vel (így a  $S_{t+1}$  blokkot, 0-tól indexelhetjük).

$k > 0$  **eset:** A  $b^{(k)}$  edényből két oldalról szitálunk. Nyilván kell tartani az edény elején és végén meddig jutottunk, vagyis mely elemek lettek feldolgozva az edény végéről és elejéről.

Egy  $(r, p)$  párral való szitálás után, ki kell számolni az új  $r$  értékét, vagyis hogy a  $S_{t+k}$  vagy  $S_{t+k+1}$  blokkba esik és azon belül melyik pozícióra. Mivel mindig egy-egy elemet tartunk számon az edény elejéről és végéről, négy eset lehetséges. Amikor az első elem az  $S_{t+k}$ -ba és az utolsó pedig az  $S_{t+k+1}$ -be esik, akkor mindkét elemet feldolgozottként tekintjük és az elején és a végén is lépünk egyet (vagyis az első pár helyett a következőt választjuk, az utolsó helyett az előzőt). Amikor fordított a helyzet, első pár esik  $S_{t+k+1}$ -be és az utolsó  $S_{t+k}$ -ba, akkor megcseréljük a két elem helyét és megint mindkettőt feldolgozottként tekintjük. Amikor mindkettő pár az  $S_{t+k}$ -ba esik akkor csak az elsőt tekintjük feldolgozottként, amikor meg mindkettő az  $S_{t+k+1}$ -be esik, akkor az utolsót tekintjük feldolgozottként. Itt lényegében egy edényrendezés történik, vagyis ha az aktuális edény  $b_j^{(k)}$  volt, akkor az  $S_{t+k}$  blokkra eső  $(r, p)$  párok a  $b_{j-1}^{(k)}$  edénybe kerülnek, az új  $r$  értékkel, míg az  $S_{t+k+1}$ -re eső elemek maradnak a  $b_j^{(k)}$  edényben, csak az  $r$  értéke változik meg.

**Megjegyzés** Amikor az edény elején lévő elemet feldolgozottként tekintjük, akkor elég csak a  $b^{(k)}$  mutatót eggyel növelni. Az edény végén feldolgozott elemekre új mutatót kell bevezetni és ezt visszafelé léptetni. Az eljárás addig tart amíg a két mutató össze nem ér.

A hatékony implementáció és az adatstruktúrák helyessége végett, minden körnél nyilván kell tartani, hogy melyik a törött edény, vagyis melyik az a  $b_j^{(k)}$ , amelyik kezdőcíme nagyobb mint  $b_{j+1}^{(k)}$  kezdőcíme. Amikor elkezdünk a  $C_k$  körrel szitálni, akkor leellenőrizhető, hogy az aktuális edény törött-e, és ha nem az, akkor hatékonyabb rutint tudunk végrehajtani.

## 4. További kutatások

Minél hamarabb el kellene készíteni egy versenyképes implementációját a GNFS-nek. Az algoritmus bonyolultsága nagy kihívássá teszi ezt. Másrészt figyelembe kell venni a létező implementációkat is. Valószínűleg ma a *GGNFS* és a *msieve* nyilvános forrású programok a legnépszerűbb és talán a leghatékonyabb kivitelezései a GNFS-nek. Ezek viszont a GNFS szitáját rács szitával gyorsítják és mindenféleképp meg kell vizsgálni, melyik módszer a gyorsabb. Esetleg a két szitálás összeolvasztásával is meg lehetne próbálkozni.

Továbbá manapság talán ár/hatékonyág arányban, az egyik legjobb processzor a *Cell BE* ami a Sony Playstation 3 játékkonzoljaiban is megtalálható. Ennek az architektúrának a segédprocesszorain, bevezettek egy új szintet a memória hierarchiába a cache és a RAM közé, amit Local Store-nak hívnak. Ez a 256 KiB méretű, gyors memóriát, a program explicit kell, hogy kezelje és a feltöltése párhuzamosítható a SPU-k működésével.

Még egy érdekes terület, ahol a fenti módszerek talán alkalmazhatóak a GPGPU, vagyis a grafikus kártyák programozása. NVidia kifejlesztett egy bővítést a C programozási nyelvnek, amellyel az nVidia 8000 grafikus processzorok és utódai könnyen programozhatóak. A GNFS nem minden része alkalmas GPU-kon való végrehajtásra, ugyanis a GPU-k nagyon gyorsak ha sok párhuzamos szorzást vagy összeadást kell végrehajtani, viszont az elágazásokat és az osztásokat rosszul kezelik.

Ezen kívül, a szita fent említett gyorsítása nem csak faktorizálásnál alkalmazható. A Goldbach sejtésben és más számelméleti problémában is előfordul a szita és ezekben az algoritmusokban is talán alkalmazható.

## Hivatkozások

- [BRES89] Bressoud, David M., *Factorization And Primality Testing*, Springer-Verlag, 1989 Number of Pages: 260 Language: English
- [BRIGGS] Briggs, Matthew E., *An Introduction to the General Number Field Sieve*, 1998, <http://scholar.lib.vt.edu/theses/available/etd-32298-93111/unrestricted/etd.pdf> [2009 augusztus]
- [CELL] IBM, *Cell Broadband Engine resource center*, „Documentation”, <http://www.ibm.com/developerworks/power/cell/>, [2009 augusztus]
- [CUDA] nVidia, *Cuda zone*, „Developing with Cuda”, „Documentation” [http://www.nvidia.com/object/cuda\\_home.html](http://www.nvidia.com/object/cuda_home.html), [2009 november]