Diszkrét logaritmus

Diszkrét modellek alkalmazásai

1. Moduláris hatványozás

Az előző jegyzetben (Kongruenciák) szerepelt a hatványozásról a következő tulajdonság:

1.1. Tétel. *Ha* $a \equiv b \pmod{m}$*, akkor* $\forall n \in \mathbb{N} : a^n \equiv b^n \pmod{m}$.

Példa. $5 \equiv 12 \pmod{7} \implies 5^3 \equiv 12^3 \pmod{7}$, azaz mindegy, hogy a kongruens alapok (5 és 12) közül melyiket választjuk, amikor hatványozunk. Célszerű tehát először az alap maradékát venni.

(De vigyázat: ez csak az alapra igaz, a kitevőre nem! Tehát például $5^{10} \not\equiv 5^3 \ (7)$. Erről később még lesz szó.)

Ha egy kicsi moduláris hatványt szeretnénk kiszámítani, akkor megtehetjük, hogy először egész számként számítjuk ki a hatványt, majd vesszük az eredmény maradékát. Például 5^5 mod 7-re:

$$5^5 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 3125 \implies 5^5 \equiv 3125 \equiv 3 \pmod{7}.$$

Ez azonban nagy kitevőkre kezelhetetlenül nagy számokat eredményezhet. Ezért okosabban járunk el, ha minden egyes szorzás után maradékot veszünk, mert így mindvégig kicsi számokkal kell számolni:

$$\begin{aligned} 5^2 &\equiv 5 \cdot 5 \equiv 25 \equiv 4 \pmod{7}, \\ 5^3 &\equiv 5^2 \cdot 5 \equiv 4 \cdot 5 \equiv 20 \equiv 6 \pmod{7}, \\ 5^4 &\equiv 5^3 \cdot 5 \equiv 6 \cdot 5 \equiv 30 \equiv 2 \pmod{7}, \\ 5^5 &\equiv 5^4 \cdot 5 \equiv 2 \cdot 5 \equiv 10 \equiv 3 \pmod{7}. \end{aligned}$$

Sage-ben is kerüljük a naiv számítást, `a^n % m`-et, mert az először kiszámítja `a^n`-t, amely óriási is lehet. Használjuk inkább a beépített `power_mod()` függvényt, amely sokkal okosabban számol:

```
sage: 5^999999999 % 7              # LASSÚ!
6
sage: power_mod(5, 999999999, 7)   # gyors
6
```

Ebben a jegyzetben a moduláris hatványozásnak azzal a speciális esetével foglalkozunk, amikor a modulus prímszám. (Az általánosítás a következő jegyzetben lesz.)

Ilyenkor bizonyos műveletek leegyszerűsödnek, például az előző jegyzetben láttuk, hogy ekkor minden nemnulla maradék invertálható. De nemcsak az inverz, hanem a szorzás is jól viselkedik prímszám modulusokra:

1.2. Tétel. *Ha* p *prímszám,* $a \not\equiv 0 \ (p)$ *és* $b \not\equiv 0 \ (p)$*, akkor* $a \cdot b \not\equiv 0 \ (p)$.

Vagyis nemnulla maradékok szorzata sosem ad 0 maradékot. Ez megfelel annak, hogy ha két nemnulla valós számot összeszorzunk, akkor a szorzat sem lesz nulla. Ez nagyon természetesnek tűnik, és nagyon hasznos is lesz, viszont csak prímszám modulus esetén igaz! Például ha a modulus $m = 10$, akkor $4 \cdot 5 \equiv 20 \equiv 0 \pmod{10}$. (Az ilyeneket fogjuk később *nullosztók*nak nevezni.)

És mivel a hatványozás nem más, mint ismételt szorzás, ezért értelemszerűen hatványozáskor sem kapunk 0-t:

1.3. Tétel. *Ha* p *prímszám és* $a \not\equiv 0 \ (p)$*, akkor* $\forall n \in \mathbb{N} : a^n \not\equiv 0 \ (p)$.

(De itt is fontos, hogy a modulus prímszám legyen, különben például $6^2 \equiv 0 \pmod{12}$.)

2. Gyorshatványozás

Most megismerünk egy nagyon hatékony eljárást az $a^n \bmod m$ moduláris hatvány kiszámítására: az ún. **gyorshatványozás**t. (A Sage `power_mod()` függvénye is ennek egy változatát használja.)

Először tekintsünk el a modulustól. Például hogyan számítjuk ki a^8-t? A naiv módszerrel így: $a^8 = a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a$, azaz hét szorzással. Vajon lehet ennél kevesebb szorzást használni? Igen, lehet: $a^8 = \left(a^4\right)^2 = \left(\left(a^2\right)^2\right)^2$, ami mindössze három szorzás (három négyzetreemelés). Ezt általánosíthatjuk a következőképpen: ha a kitevő páros, akkor az első lépés ugyanaz: $a^{2n} = (a^n)^2$, és ezt addig folytatjuk, amíg a belső n is páros marad; ha pedig páratlan, akkor először elkülönítünk egy a-t: $a^{2n+1} = (a^n)^2 \cdot a$, és innen folytatjuk tovább. Például:

$$a^{22} = \left(a^{11}\right)^2 = \left(\left(a^5\right)^2 \cdot a\right)^2 = \left(\left(\left(a^2\right)^2 \cdot a\right)^2 \cdot a\right)^2.$$

Így tehát a^{22}-t nem 21, hanem mindössze 6 szorzással ki tudjuk számítani.

A fenti felírásnak szoros kapcsolata van a kitevő kettes számrendszerbeli (*bináris*) alakjához. Minden lépésben van egy négyzetreemelés, ami után néha megszorozzuk az eredményt a-val, néha nem. Ha minden a-val való szorzáskor írunk egy 1-est, máskor pedig egy 0-t, majd az egész elejére írunk még egy 1-est (a legbelső a-hoz), akkor éppen a kitevő bináris felírását kapjuk, pl. $22 = 10110_2$.

Mivel modulárisan számolunk, ezért venni kell az eredmény maradékát – méghozzá minden szorzás után, ellenkező esetben a részeredmény túl nagyra nőhetne, és a "gyorshatványozás" lassúvá válna.

Gyorshatványozás(a, n, m)**:**

- **Bemenet:** $a \in \mathbb{Z}$ alap, $n \in \mathbb{N}^+$ kitevő, $m \in \mathbb{N}^+$ modulus
- $B :=$ Kettes számrendszerbe(n) *(lásd lent)*
- $a := a \bmod m$
- $r := a$
- **ciklus** $b := B$ *elemei, kivéve az elsőt (ami mindig* 1*)*
  - $r := r \cdot r \bmod m$
  - **ha** $b = 1$
    - $r := r \cdot a \bmod m$
- **Kimenet:** r

Kettes számrendszerbe(n)**:**

- **Bemenet:** $n \in \mathbb{N}^+$
- $B :=$ *(üres lista)*
- **ciklus amíg** $n \neq 0$
  - B végére beszúrjuk $(n \bmod 2)$-t
  - $n := n \operatorname{div} 2$
- **Kimenet:** B fordított sorrendben

Példa. Kettes számrendszerbe$(8) = [1, 0, 0, 0]$ és Kettes számrendszerbe$(22) = [1, 0, 1, 1, 0]$.

Az algoritmus nagyon hatékony, mert legfeljebb $2L$ szorzást végez, ahol L a kitevő hossza kettes számrendszerben, azaz $L \approx \log_2 n$. Például ha $n \approx 1000000$, akkor ez legfeljebb kb. 40 szorzást jelent.

Megjegyzés: a gyorshatványozás nemcsak moduláris hatványozásra működik, hanem bármely olyan matematikai objektum hatványozására, amelyet lehet szorozni (pl. egész szám, valós szám, mátrix stb.). Ekkor a maradékképzéseket $(\bmod m)$ értelemszerűen elhagyjuk.

3. A hatványok struktúrája

Most vizsgáljuk meg különböző számok moduláris hatványait az $m = 7$ modulusra:

```
sage: matrix([[power_mod(a, n, 7) for n in range(30)] for a in range(1, 7)])
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[1 2 4 1 2 4 1 2 4 1 2 4 1 2 4 1 2 4 1 2 4 1 2 4 1 2 4 1 2 4]
[1 3 2 6 4 5 1 3 2 6 4 5 1 3 2 6 4 5 1 3 2 6 4 5 1 3 2 6 4 5]
[1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2]
[1 5 4 6 2 3 1 5 4 6 2 3 1 5 4 6 2 3 1 5 4 6 2 3 1 5 4 6 2 3]
[1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6]
```

Látható, hogy a hatványok periodikusan ismétlődnek. (Mivel csak véges sok maradék van, ezért nem is lehet minden hatvány különböző.) Az is látható, hogy legelőször mindig az 1 ismétlődik.

De miért éppen az 1? Mert ha van bármilyen más ismétlés, pl. $a^8 \equiv a^2 \pmod 7$, akkor ezt leosztva a^2-tel – pontosabban megszorozva a^2 moduláris inverzével – kapunk egy korábbi ismétlést: $a^6 \equiv 1 \pmod 7$.

Most tegyük fel azt a kérdést, hogy *mikor* van a legelső ismétlés.

3.1. Definíció. $a \in \mathbb{Z}$ **rendje** modulo m az a legkisebb $n \in \mathbb{N}^+$ kitevő, amelyre $a^n \equiv 1 \pmod m$. [Angolul: *order*.] Jelölés: $\mathrm{ord}_m(a)$.

Példa. $\mathrm{ord}_7(4) = 3$, azaz 4 rendje 3 modulo 7, mert $4^3 \equiv 1\ (7)$, de $4^2 \not\equiv 1\ (7)$ és $4^1 \not\equiv 1\ (7)$.

Példa. $\mathrm{ord}_7(5) = 6$, ahogy az a táblázatból is leolvasható.

Egy szám rendje tehát az a legkisebb pozitív kitevő, amelyre emelve 1 maradékot ad. Mivel láttuk, hogy az 1 az első ismétlődő hatvány, ezért a rend megegyezik az ismétlés periódusának hosszával:

$$a^{n+\mathrm{ord}_m(a)} \equiv a^n a^{\mathrm{ord}_m(a)} \equiv a^n \cdot 1 \equiv a^n \pmod m.$$

A táblázatból az is látszik, hogy minden nemnulla maradék hatodik hatványa 1 (de nem mindenütt ez az első 1-es). Ez általában is igaz:

3.2. Tétel (Kis Fermat-tétel)**.** *Ha p prímszám és $a \not\equiv 0 \pmod p$, akkor*

$$a^{p-1} \equiv 1 \pmod p.$$

Ebből tehát látható, hogy minden szám rendje legfeljebb $p-1$ lehet, azaz $\mathrm{ord}_p(a) \le p-1$. Sőt:

3.3. Tétel (Lagrange-tétel)**.** *Ha p prímszám és $a \not\equiv 0 \pmod p$, akkor* $\mathrm{ord}_p(a) \mid p-1$.

És valóban, leolvasható, hogy a számok rendje 1, 2, 3 vagy 6, amelyek éppen a $7-1 = 6$ osztói.

Külön felhívjuk a figyelmet két speciális esetre. Egyrészt természetesen bármely modulusra $\mathrm{ord}_m(1) = 1$, hiszen 1 bármely hatványa 1. Másrészt $\mathrm{ord}_7(6) = 2$, és ez is igaz általában is: $\mathrm{ord}_m(m-1) = 2$, mert $m-1$ kongruens -1-gyel, és $(-1)^2 = 1$, azaz $(m-1)^2 \equiv (-1)^2 \equiv 1 \pmod m$ (kivéve $m = 2$ esetén, ahol nem ez az első 1-es, mert $-1 \equiv 1 \pmod 2$).

Megfigyelhető az is, hogy bármely szám moduláris hatványai között szerepel a moduláris inverze, méghozzá éppen az 1-es előtt. Miért? Mert a kis Fermat-tétel szerint $a^{p-1} \equiv 1\ (p)$, és ezt megszorozva a-nak az inverzével azt kapjuk, hogy $a^{p-2} \equiv a^{-1}\ (p)$. A legelső előfordulása a^{-1}-nek $a^{-1} \equiv a^{\mathrm{ord}_p(a)-1}\ (p)$. (Például $p = 7$ esetén 4 inverze 2, rendje 3, és valóban: $4^{3-1} \equiv 2\ (7)$.) Ezzel tehát kaptunk egy újabb módszert a moduláris inverz kiszámítására (az előző jegyzetben tárgyaltakon felül).

Különösen érdekesek azok a számok, amelyek rendje a lehető legmagasabb érték:

3.4. Definíció. Egy p prímszám esetén $g \in \mathbb{Z}$ **primitív gyök** modulo p, ha $\mathrm{ord}_p(g) = p-1$.
(Másik elnevezése: **generátor**.)

Példa. $p = 7$ esetén primitív gyök a 3 és az 5, mert azok rendje 6 (lásd a fenti táblázatot).

Miért érdekesek a primitív gyökök? Nézzük például 5 hatványait modulo 7: $1, 5, 4, 6, 2, 3, 1, 5, \ldots$. Vegyük észre, hogy minden nemnulla maradék szerepel közöttük. Ez igaz általában is:

3.5. Tétel. *Ha p prímszám és g primitív gyök modulo p, akkor $\forall b \not\equiv 0 \ (p) : \exists n \in \mathbb{N} : g^n \equiv b \ (p)$.*

Más szóval: egy primitív gyök a hatványaival előállítja ("generálja") az összes létező maradékot.

(A bizonyítás egyszerű: a primitív gyök definíciója miatt az első ismétlés a $p-1$-edik hatványban lesz, tehát az első $p-1$ hatványnak különbözőnek kell lennie, viszont mivel éppen $p-1$-féle maradék van, ezért mindegyiknek szerepelnie kell az első $p-1$ hatvány között.)

Vajon minden p prímszámra találunk primitív gyököt? A válasz igenlő (de nehéz bizonyítani):

3.6. Tétel. *Ha p prímszám, akkor létezik primitív gyök modulo p.*

4. Diszkrét logaritmus

Most vizsgáljuk meg a moduláris hatványozás inverzműveletét: egy adott érték egy adott alapnak hányadik hatványaként áll elő? Ez nagyon hasonlít a valós számoknál ismert logaritmusra: az $a^x = b$ valós egyenlet megoldását $x = \log_a b$ adja meg, például $2^x = 256$ megoldása $x = \log_2 256 = 8$.

4.1. Definíció. $a, b \in \mathbb{Z}$ esetén b-nek az a-alapú **diszkrét logaritmusa** (vagy **indexe**) modulo m az a legkisebb $n \in \mathbb{N}$ kitevő, amelyre $a^n \equiv b \pmod{m}$. Jelölés: $n = \operatorname{ind}_a b \pmod{m}$.

Példa. $\operatorname{ind}_5 6 = 3 \pmod 7$, azaz 6-nak az 5-ös alapú diszkrét logaritmusa 3 modulo 7, mert $5^3 \equiv 6 \pmod 7$, és 5-nek semelyik korábbi hatványa sem 6 (az első néhány hatványa: $1, 5, 4, 6, \ldots$).

Példa. Mennyi $\operatorname{ind}_4 3 \pmod 7$, azaz 3-nak a 4-es alapú diszkrét logaritmusa modulo 7? Nézzük 4 hatványait: $1, 4, 2, 1, 4, 2, 1, \ldots$ – egyik sem 3, vagyis $\operatorname{ind}_4 3 \pmod 7$ nem létezik.

Ha g egy primitív gyök modulo p, akkor a fenti 3.5. Tétel alapján $\forall b \not\equiv 0 \ (p)$ számra van $\operatorname{ind}_g b$.

A diszkrét logaritmust legegyszerűbben úgy számíthatjuk ki, ha előállítjuk rendre az $1, a, a^2, a^3, \ldots$ hatványokat (modulo m), és amelyik egyenlő b-vel, annak a kitevője lesz az eredmény. Ha nincs ilyen, akkor a következő 1-esnél megállhatunk, mert onnantól már csak ismétlődni fognak a számok. A hatványokat nem kell külön-külön kiszámítani, hanem mindegyiket megkaphatjuk az előző hatványból egy szorzással és egy maradékos osztással. A kis Fermat-tétel miatt az algoritmus legfeljebb $p-1$ lépést tesz, vagyis *lineáris*. Ezt azt jelenti, hogy nagy p-kre nagyon lassú, például $p \approx 1000000$ esetén kb. egymillió lépésre van szükség.

Ennél a lineáris keresésnél vannak *valamelyest* hatékonyabb módszerek (egy ilyet bemutatunk a következő szakaszban), de egyik sem közelíti meg a gyorshatványozás sebességét. Vagyis a moduláris hatványozás műveletét el tudjuk végezni gyorsan, de az inverzét, a diszkrét logaritmust nem – ez mind a mai napig a számításelmélet nyitott problémái közé tartozik.

De ez nem baj. Sőt, mint majd látni fogjuk, kifejezetten hasznos!

A Sage a diszkrét logaritmust a `discrete_log()` függvénnyel tudja kiszámítani (ami sokkal általánosabb, mint amire szükségünk van, ezért kissé körülményes a használata):

```
sage: discrete_log(mod(6, 7), mod(5, 7))   # 5-ös alapú log 6 (mod 7)
3
sage: power_mod(5, 3, 7)    # ellenőrzés
6
```

De nagyon nagy modulusokra persze a `discrete_log()`-ot is hiába próbáljuk:

```
sage: p = next_prime(2^1000)
sage: b = power_mod(7, randrange(p), p)
sage: discrete_log(mod(b, p), mod(7, p))    # nem fog menni
...
```

Vigyázat: a `mod(b, p)`-t ne keverjük össze a `(b % p)`-vel, mert más típusú értékeket adnak vissza (erről egy későbbi jegyzetben lesz szó). A `discrete_log()`-hoz viszont pont a `mod(b, p)` kell.

4.1. Baby-step giant-step

Most nézzünk a diszkrét logaritmusra egy hatékonyabb módszert, mint a lineáris keresés. Az ötlet az, hogy vegyünk egy közbülső M számot, amelyre kb. $M^2 \approx p-1$, és keressük az $a^n \equiv b\ (p)$ hatvány ismeretlen n kitevőjét $n = x + My$ alakban, ahol $0 \le x, y < M$ (például ha $M = 10$ és $n = 76$, akkor $n = 7M + 6$, azaz $x = 6$ és $y = 7$). Először számítsuk ki az első néhány a^x hatványt (modulo p): $1,\ a,\ a^2,\ \ldots,\ a^{M-1}$ ("baby-step"), és tároljuk el őket. Ezután vegyük a keresett hatványt (b), és kezdjük el szorozgatni a^{-M}-nel: $b,\ a^{-M}b,\ a^{-2M}b,\ \ldots$ ("giant-step"). Ez azért jó, mert a két sorozat (a "baby-step" és a "giant-step") előbb-utóbb összeér: amikor b-t éppen a^{-yM}-nel szoroztunk meg (ahol y az $n = x + My$ felírás egyik ismeretlene), akkor $a^{-yM}b = a^{-yM+n} = a^x$, ami szerepel az eltárolt a^x értékek között. Ha ezt megtaláltuk, akkor megvan x és y, és abból összerakhatjuk n-et.

Baby-step giant-step algoritmus:

Bemenet: p prímszám, $a \in \{1, \ldots, p-1\}$ alap, $b \in \{1, \ldots, p-1\}$ hatványérték
$M := \left\lceil \sqrt{p-1} \right\rceil$
$c := 1$
ciklus $x = 0, 1, \ldots, M-1$
  Tároljuk el (c, x)-et egy táblában. (Megjegyzés: $c \equiv a^x\ (p)$.)
  $c := c \cdot a \bmod p$
$d := c^{-1} \bmod p\ \ (= a^{-M} \bmod p)$
$c := b$
ciklus $y = 0, 1, \ldots, M-1$
  ha *c benne van a táblában valamely x-szel*
    megtaláltuk: $\rightarrow n := x + My$
  $c := c \cdot d \bmod p$

Ahhoz, hogy ez hatékony legyen, az kell, hogy a "táblában" gyorsan tudjunk értékeket tárolni és visszakeresni, erre például egy hash-tábla alkalmas. A tábla kulcsa c, az értéke pedig az x kitevő. Például Sage-ben használhatjuk a Python-féle szótár (`dict`) típust (üres szótár: `T = {}`, elem hozzáadása: `T[c] = x`, elem keresése: `c in T`, majd lekérdezése: `T[c]`).

Az algoritmus kb. $2\sqrt{p}$ szorzást igényel, amely a lineáris keresésnél sokkal jobb (pl. $p \approx 1000000$ esetén csak $\sim$2000 szorzás kell hozzá). De ha összehasonlítjuk a gyorshatványozással, akkor ez nagy p-kre még mindig lassú. Ráadásul a memóriaigénye is sokkal nagyobb (kb. $2\sqrt{p}$ számot kell tárolni).

5. Diffie–Hellman-kulcscsere

Tegyük fel, hogy két ember titkosítva szeretne kommunikálni egy olyan csatornán keresztül, amit bárki lehallgathat (pl. internet). Ehhez szükségük van egy *titkos kulcs*ra, amelynek ismeretében ők ketten el tudják olvasni egymás üzeneteit, mások viszont nem. A probléma az, hogy még sosem találkoztak, így a kulcsot is csak ezen a nyilvános csatornán keresztül tudják megbeszélni, ahol bármit mondanak, azt más is hallhatja. Hogyan tudják mégis egyeztetni a közös titkos kulcsot?

Erre az első ránézésre lehetetlennek tűnő kérdésre ad választ a **Diffie–Hellman-kulcscsere**. A módszer azon alapul, hogy a diszkrét logaritmus probléma nehéz, azaz ha van egy titkos $n \in \mathbb{N}$ kitevőnk, akkor a $g^n \bmod p$ hatványt nyugodtan nyilvánosságra hozhatjuk (g-vel és p-vel együtt), hiszen senki nem fogja tudni kiszámítani belőle a titkos n kitevőt.

A Diffie–Hellman-kulcscsere a következőképpen működik (minden hatványt modulo p értünk):

1. A két résztvevő, Alíz és Bob megegyeznek egy p prímszámban és egy g primitív gyökben modulo p (ezek nyilvános adatok, és akár előre rögzített konstansok is lehetnek).
2. Mindketten generálnak egy-egy véletlen, titkos kitevőt 1 és $p-1$ között: a és b, ahol a-t csak Alíz ismeri, és b-t csak Bob.
3. Alíz kiszámítja g^a-t, és elküldi Bobnak.
4. Bob kiszámítja g^b-t, és elküldi Alíznak.
5. Mindketten kiszámítják g^{ab}-t az általuk ismert adatokból:
   - Alíz ismeri a-t és g^b-t, ebből $g^{ab} = \left(g^b\right)^a$.
   - Bob ismeri b-t és g^a-t, ebből $g^{ab} = \left(g^a\right)^b$.

A közös titok tehát g^{ab}, amelyet a fenti műveletsor után mind Alíz, mind Bob ki tud számítani. Viszont bárki más, aki hallgatózik a csatornán, csak g-t, g^a-t és g^b-t láthatja, ezekből pedig nem fogja tudni kiszámítani g^{ab}-t, mert ahhoz ismernie kéne valamelyik titkos kitevőt (a-t vagy b-t), ahhoz pedig meg kéne oldania a diszkrét logaritmus problémát.

Példa.

1. Legyen a modulus $p = 23$ és a primitív gyök $g = 5$.
2. Alíz véletlen titkos kitevője $a = \mathbf{7}$, Bobé pedig $b = \mathbf{21}$.
3. Alíz számítása: $g^a \equiv 5^7 \equiv \mathbf{17} \pmod{23}$, amit elküld Bobnak.
4. Bob számítása: $g^b \equiv 5^{21} \equiv \mathbf{14} \pmod{23}$, amit elküld Alíznak.
5. Ebből mindketten ki tudják számítani g^{ab}-t:
   - Alíz: $g^{ab} \equiv \left(g^b\right)^a \equiv 14^7 \equiv \mathbf{19} \pmod{23}$.
   - Bob: $g^{ab} \equiv \left(g^a\right)^b \equiv 17^{21} \equiv \mathbf{19} \pmod{23}$.

A Diffie–Hellman-kulcscsere biztonságához szükséges feltétel, hogy a p prímszám elég nagy legyen. A gyakorlatban néhány ezer bites prímszámokat szoktak használni (pl. 2048-bitest, azaz p felírása kettes számrendszerben 2048 számjegy, ami tízes számrendszerben 617 számjegy), lásd pl.: [5].

Vigyázat: a fenti leírás az alapvető koncepciók ismertetésére szorítkozik, így egyszerűsítéseket tartalmaz, és önmagában nem elegendő a kulcscsere biztonságos implementációjához. Általában a kriptográfiai algoritmusok implementálását – játékpéldákat leszámítva – bízzuk a témában jártas szakemberekre, mert a legkisebb hibák is súlyos biztonsági réseket okozhatnak.

A Diffie–Hellman-kulcscserét számos helyen használják, különösen internetes rendszerekben, mint pl. HTTPS, SSH, VPN-ek stb. – általában más kriptográfiai algoritmusokkal együtt, pl. RSA és AES (amelyekről későbbi jegyzetekben lesz szó).

6. Véletlenszám-generátorok

A legtöbb programozási nyelvben van *véletlenszám-generátor* [*random number generator (RNG)*]. Ezek azonban legtöbbször nem "valódi" véletlen számokat állítanak elő (mint amit például kockadobással vagy egyéb fizikai eljárással kaphatnánk), hanem egy egyszerű algoritmussal számítják ki a – kellően véletlennek tűnő – értékeket. Az ilyeneket *pszeudovéletlen számok*nak nevezzük.

Egy nagyon egyszerű pszeudovéletlenszám-generátort kaphatunk moduláris hatványozással. Bár a hatványok periodikusan ismétlődnek, de egy perióduson belül kellően véletlenszerűnek tűnnek (például 5 hatványai modulo 7 a következők: $1, 5, 4, 6, 2, 3$). Például C++-ban az `std::minstd_rand` [6] generátor a következő paramétereket használja:

$$x_n := 48271^n x_0 \bmod 2147483647,$$

ahol x_0 tetszőleges kezdőérték [angolul *seed*] (ezt gyakran az aktuális időből hozzák létre, hogy mindig más legyen); a modulus $p = 2147483647 = 2^{31} - 1$ egy olyan prímszám, hogy a maradékok éppen beleférjenek egy előjeles 32-bites számba (és hogy hatékonyan lehessen vele maradékosan osztani); a szorzó, 48271 pedig egy primitív gyök modulo p.

Ez a véletlenszám-generátor nagyon egyszerű és nagyon hatékony, ezért jól használható például szimulációkban és játékokban. Nem alkalmas viszont a Diffie–Hellman-kulcscsere kitevőinek (a és b) generálására (sem bármely más olyan algoritmusban, ahol a biztonság fontos), mert az egyszerűsége miatt nagyon könnyű megjósolni a következő számokat, és ezáltal az egész eljárás értelmét veszti.

Ehelyett ún. *biztonságos véletlenszám-generátorokat* használnak [*cryptographically secure pseudo-random number generators (CSPRNG)*], amelyek bonyolultabbak, de kellő matematikai garanciával rendelkeznek arra, hogy ne lehessen megjósolni a következő számokat (még egyes számjegyeiket sem).

Számos programozási nyelvben, így Sage-ben (és Python-ban) is vannak külön biztonságos és nem biztonságos véletlenszám-generátorok:

```
sage: randrange(100)                # NEM biztonságos
94
sage: import secrets
sage: secrets.randbelow(100)        # biztonságos
67
sage: secrets.randbelow(int(100))  # ugyanaz régebbi Sage verziókban
56
```

7. Szerző

Ezt a jegyzetet Uray M. János írta, részben felhasználva az alábbi forrásokat:

[1] https://compalg.elte.gitlab-pages.hu/dimoa-web/tematika/dimoa
[2] https://en.wikipedia.org/wiki/Modular_exponentiation
[3] https://en.wikipedia.org/wiki/Discrete_logarithm
[4] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
[5] Prímek a Diffie–Hellman-kulcscseréhez: https://www.rfc-editor.org/rfc/rfc3526
[6] https://en.cppreference.com/w/cpp/numeric/random/linear_congruential_engine.html

Hibákat, észrevételeket itt lehet jelezni: uray.janos@inf.elte.hu.