Diszkrét logaritmus

Számelmélet

Diszkrét modellek alkalmazásai

1. Prímszám modulus

Folytatva az előző jegyzetet (Moduláris aritmetrika), tekintsük azt az esetet, amikor a modulus (m) prímszám. Láttuk, hogy ilyenkor bizonyos műveletek leegyszerűsödnek, például minden nemnulla elemmel lehet osztani.

Tekintsük tehát $\mathbb{Z}_p$-t, ahol p prímszám. Ezen belül is tekintsük csak a nemnulla elemeket:

1.1. Definíció. p prímszám esetén legyen $\mathbb{Z}_p^*$ a $\mathbb{Z}_p$ azon elemeinek halmaza, amely nem tartalmazza a nullát, azaz $\mathbb{Z}_p^* := \mathbb{Z}_p \setminus \{\overline{0}\}$.

Például $\mathbb{Z}_5^* = \{\overline{1}, \overline{2}, \overline{3}, \overline{4}\}$, általában $\mathbb{Z}_p^* = \{\overline{1}, \overline{2}, \ldots, \overline{p-1}\}$ (tehát $\mathbb{Z}_p^*$ elemszáma $p-1$). Miért jó ez?

1.2. Tétel. *Ha p prímszám, akkor* $\forall \overline{a}, \overline{b} \in \mathbb{Z}_p^* : \overline{a} \cdot \overline{b} \in \mathbb{Z}_p^* \ \wedge \ \overline{a}/\overline{b} \in \mathbb{Z}_p^*$.

Vagyis mind a szorzás, mind az osztás mindig elvégezhető $\mathbb{Z}_p^*$-on belül, és az eredmény is mindig ezen belül marad. Ez a szorzásnál annyiból újdonság, hogy a szorzat sosem lesz nulla. De ez csak prímszám modulus esetén igaz! Például ha a modulus $m = 10$, akkor $\overline{4} \cdot \overline{5} = \overline{20} = \overline{0}$. (Az ilyeneket fogjuk később *nullosztók*nak nevezni.) Viszont a fenti tétel nem terjed ki az összeadásra és a kivonásra. Nem is tud, még prímszámokra sem, például modulo $p = 5$ esetén $\overline{3} + \overline{2} = \overline{0}$.

2. Moduláris hatványozás

Az előző jegyzetben megismertük a moduláris aritmetika alapjait: az összeadást, a kivonást, a szorzást és az osztást. Most nézzük meg a hatványozást.

A hatványozás pozitív egész kitevőre nem más, mint ismételt szorzás, például egészek esetén $5^3 = 5 \cdot 5 \cdot 5 = 125$. Modulárisan is ugyenezt jelenti, például $\mathbb{Z}_7$-ben: $\overline{5}^3 = \overline{5} \cdot \overline{5} \cdot \overline{5} = \overline{4} \cdot \overline{5} = \overline{6}$.

Egész kitevőre (azaz pozitív, nulla és negatív egészekre) a szokásos módon lehet általánosítani:

2.1. Definíció. Legyen $m \in \mathbb{N}^+, \overline{a} \in \mathbb{Z}_m, k \in \mathbb{Z}$.

- Ha $k > 0$, akkor $\overline{a}^k := \overline{a} \cdot \overline{a} \cdot \ldots \cdot \overline{a}$, ahol pontosan k tényező van.
- Ha $k = 0$, akkor $\overline{a}^k := \overline{1}$.
- Ha $k < 0$, és $\overline{a}$ invertálható, akkor $\overline{a}^k := \left(\overline{a}^{-1}\right)^{-k}$ (ahol $-k$ pozitív).

Például $\mathbb{Z}_7$-ben, ha tudjuk, hogy $\overline{5}$ inverze $\overline{3}$, akkor $\overline{5}^{-2} = \left(\overline{5}^{-1}\right)^2 = \overline{3}^2 = \overline{3} \cdot \overline{3} = \overline{2}$.

Megjegyzés: az inverz jele, $\overline{a}^{-1}$ éppen egybeesik a negatív hatványozás jelentésével $k = -1$-re.

A következő tétel biztosítja, hogy a hatványozás valóban értelmes maradékosztályokra, azaz az eredmény maradéka nem függ attól, hogy a maradékosztály melyik elemét hatványozzuk:

2.2. Tétel. *Ha $a \equiv b \pmod{m}$, akkor $a^k \equiv b^k \pmod{m}$.*

Eszerint például $5^3 \equiv 12^3 \pmod 7$, azaz mindegy, hogy az $\overline{5}^3$ alakkal számolunk, vagy a $\overline{12}^3$ alakkal (ne feledjük: $\mathbb{Z}_7$-ben $\overline{5} = \overline{12}$), ugyanaz lesz az eredmény. (Vigyázat: ez csak a hatványalapra igaz, a kitevőre nem! Tehát például $5^{10} \not\equiv 5^3 \pmod 7$. Erről később még lesz szó.)
Sage-ben többféleképpen is lehet modulárisan hatványozni:

```
sage: 3^100 % 7              # LASSÚ! (kiszámítja 3^100-t egész számként)
4
sage: Z7 = Zmod(7)
sage: Z7(3)^100              # maradékosztályt ad vissza
4
sage: power_mod(3, 100, 7)  # egész számot ad vissza
4
```

Most állítsuk elő $\mathbb{Z}_7^*$-ban az elemek első néhány hatványát:

```
sage: matrix([[a^k for k in range(20)] for a in Zmod(7) if a != 0])
[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]
[1 3 2 6 4 5 1 3 2 6 4 5 1 3 2 6 4 5 1 3]
[1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4]
[1 5 4 6 2 3 1 5 4 6 2 3 1 5 4 6 2 3 1 5]
[1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6]
```

Először is vegyük észre, hogy a hatványok periodikusan ismétlődnek. Az, hogy ismétlődnek, nem meglepő, hiszen $\mathbb{Z}_7^*$-ban csak véges sok elem van, tehát nem lehet minden hatvány különböző. Az is látható, hogy legelőször mindig az $\overline{1} = \overline{a}^0$ ismétlődik. Miért van ez így? Mert ha bármilyen más ismétlés van, pl. $\overline{a}^8 = \overline{a}^2$, akkor ezt leoszthatjuk $\overline{a}^2$-nel, és kapunk egy korábbi ismétlést: $\overline{a}^6 = \overline{1}$. Most tegyük fel a kérdést, hogy mikor lesz a legelső ismétlés.

2.3. Definíció. Egy $\overline{a} \in \mathbb{Z}_m$ elem *rendje* az a legkisebb $k \in \mathbb{N}^+$ szám, amelyre $\overline{a}^k = \overline{1}$.
Jelölés: $o(\overline{a})$ vagy $\operatorname{ord}(\overline{a})$.
Vagyis egy elem rendje az a legkisebb kitevő, amelyre emelve $\overline{1}$-et kapunk. Vagyis: a rend nem más, mint az ismétlés periódusának hossza (hiszen $\overline{a}^{j+o(\overline{a})} = \overline{a}^j \overline{a}^{o(\overline{a})} = \overline{a}^j \overline{1} = \overline{a}^j$).
Például $\mathbb{Z}_7$-ben $\overline{4}$ rendje 3 ($o(\overline{4}) = 3$), mert $\overline{4}^3 = \overline{1}$, de $\overline{4}^1 = \overline{4} \neq \overline{1}$ és $\overline{4}^2 = \overline{2} \neq \overline{1}$. Vagy például azt is leolvashatjuk, hogy $o(\overline{5}) = 6$.
A táblázatból az is látható, hogy minden elem hatodik hatványa $\overline{1}$ (de némelyiknek ennél kisebb hatványa is $\overline{1}$). Ez általában is igaz:

2.4. Tétel (Kis Fermat-tétel)**.** *Ha p prím, $a \in \mathbb{Z}$ és $p \nmid a$, akkor $a^{p-1} \equiv 1 \pmod p$.*
Átfogalmazva maradékosztályokra: $\forall \overline{a} \in \mathbb{Z}_p^* : \overline{a}^{p-1} = \overline{1}$.
Ebből tehát látható, hogy minden elem rendje legfeljebb $p-1$ lehet ($o(\overline{a}) \leq p-1$). De ennél többet is tudunk mondani a rendről:

2.5. Tétel (Lagrange-tétel)**.** *Ha p prím és $\overline{a} \in \mathbb{Z}_p^*$, akkor $o(\overline{a}) \mid p-1$.*
Valóban, a fenti táblázatból látható, hogy $\mathbb{Z}_7^*$-ban minden elem rendje 1, 2, 3 vagy 6, amelyek a $7-1=6$ osztói.
Külön felhívjuk a figyelmet két speciális esetre: egyrészt természetesen bármely modulusra $o(\overline{1}) = 1$, hiszen $\overline{1}$ minden hatványa $\overline{1}$, másrészt pedig $\mathbb{Z}_7$-ben $o(\overline{6}) = 2$, és ez is igaz általában is: $\mathbb{Z}_p$-ben $o(\overline{p-1}) = 2$, mert $\overline{p-1}^2 = \overline{-1}^2 = \overline{1}$ (kivéve $\mathbb{Z}_2$-ben, ahol $\overline{-1} = \overline{1}$).

Vegyük észre azt is, hogy bármely elem moduláris hatványai között ott van az inverze. Miért? Mert azt tudjuk, hogy $\overline{a}^{p-1} = \overline{1}$, és ezt leosztva $\overline{a}$-val azt kapjuk, hogy $\overline{a}^{p-2} = \overline{a}^{-1}$. Az $\overline{a}^{-1}$ legelső előfordulása pedig $\overline{a}^{-1} = \overline{a}^{o(\overline{a})-1}$. (Például $\mathbb{Z}_7$-ben $\overline{4}$ inverze $\overline{2}$, rendje 3, és valóban: $\overline{4}^{3-1} = \overline{2}$.)

3. Gyorshatványozás

Most arra keressük a választ, hogy hogyan lehet hatékonyan kiszámítani az $\overline{a}^n$ hatványt egy adott $\mathbb{Z}_p$ struktúrában.

Először is, mivel a kis Fermat-tétel szerint $\overline{a}^{p-1} = \overline{1}$, ezért az n kitevő helyett elég annak a maradékát venni $p-1$ szerint: $\overline{a}^n = \overline{a}^{n \bmod p-1}$. Például $\mathbb{Z}_7$-ben $\overline{5}^6 = \overline{1}$, így bármilyen $\overline{5}^{6k}$ is $\overline{1}$, tehát pl. $\overline{5}^{63} = \overline{5}^{63 \bmod 6} = \overline{5}^3 = \overline{6}$. (Emlékezzünk arra a figyelmeztetésre, hogy a kivetővel nem lehet ugyanúgy modulárisan számolni, mint az alappal. Most látjuk, hogy ha *ugyanúgy* nem is, de egy másik modulussal ($p-1$) lehet, pl. $\mathbb{Z}_7$-ben 6-tal. Erről később részletesebben is lesz szó.)

De ez még nem elég, hiszen a redukált kitevő is lehet nagy, ha a modulus nagy. Ezért megismerjük a *gyorshatványozás* algoritmusát, amely nemcsak $\mathbb{Z}_p$-ben, hanem bármely olyan struktúrában működik, ahol van hatványozás (pl. egész számok, valós számok, mátrixok stb.), és azt szeretnénk minél kevesebb szorzással előállítani.

Például hogyan számítjuk ki a^8-t? A definíció szerint úgy, hogy $a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a$. Ez hét szorzást jelent. Vajon nem lehetne-e ugyanezt kevesebb szorzással kiszámítani? De: $a^8 = \left(a^4\right)^2 = \left(\left(a^2\right)^2\right)^2$, ehhez pedig mindössze három szorzás kell (három négyzetreemelés). És mi van, ha a szám nem kettőhatvány? Ha páros, akkor az első lépés ugyanaz: $a^{2n} = (a^n)^2$, és ezt addig folytatjuk, amíg a belső n is páros marad; ha pedig egyszer páratlan lesz, akkor 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 6 szorzással tudjuk kiszámítani.

Vegyük észre, hogy a fenti felírásnak szoros kapcsolata van a kitevő kettes számrendszerbeli alakjához. Ha megnézzük a műveletek sorát, akkor azt látjuk, hogy minden lépésben van egy négyzetreemelés, ami után vagy megszorozzuk az eredményt a-val, vagy 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 miatt), akkor éppen a kitevő kettes számrendszerbeli felírását kapjuk, például $22 = 10110_2$.

A gyorshatványozás teljes algoritmusa így néz ki:

1. Bemenet: a (tetszőleges hatványozható érték) és $n \in \mathbb{N}^+$ kitevő.
2. Írjuk fel n-et kettes számrendszerben: $n = (b_l \dots b_1 b_0)_2$, azaz $n = b_0 + b_1 2 + \dots b_l 2^l$ (ahol $b_l = 1$).
3. $r := a$.
4. Minden i-re $l-1$-től 0-ig:
   (a) $r := r \cdot r$.
   (b) Ha $b_i = 1$, akkor $r := r \cdot a$.
5. Kimenet: r.

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

4. Diszkrét logaritmus

Láttuk, hogy $\mathbb{Z}_p^*$-ban az elemek rendje vagy p–1, vagy annak valamely osztója. Most tekintsük azokat az elemeket, amelyek rendje a lehető legmagasabb érték:

4.1. Definíció. Egy p prím esetén $\overline{g} \in \mathbb{Z}_p^*$ *generátor* $\mathbb{Z}_p^*$-ban, ha $o(\overline{g}) = p-1$.

Például $\mathbb{Z}_7$-ben generátorok a $\overline{3}$ és az $\overline{5}$, mert azok rendje 6 (lásd a korábbi táblázatot).

Mire utal a "generátor" elnevezés? Írjuk fel megint $\overline{5}$-nek a hatványait:

```
sage: [mod(5, 7)^k for k in range(0, 20)]
[1, 5, 4, 6, 2, 3, 1, 5, 4, 6, 2, 3, 1, 5, 4, 6, 2, 3, 1, 5]
```

Vegyük észre, hogy ezen hatványok között szerepel az összes elem $\mathbb{Z}_7^*$-ban. Ez igaz általában is:

4.2. Tétel. *Ha $\overline{g} \in \mathbb{Z}_p^*$ generátor, akkor $\forall \overline{a} \in \mathbb{Z}_p^* : \exists k \in \mathbb{N} : \overline{g}^k = \overline{a}$.*

Más szóval: egy generátor a hatványaival előállít minden elemet $\mathbb{Z}_p^*$-ban.

(A bizonyítás egyszerű: a generátor definíciója alapján az első ismétlés p–1 elem után történik, tehát az első p–1 elemnek különbözőnek kell lennie, viszont $\mathbb{Z}_p^*$-ban pontosan p–1 elem van, tehát az összes elemének szerepelnie kell az első $p-1$ hatvány között.)

A továbbiakban ilyen generátorokkal fogunk foglalkozni. De felmerül a kérdés, hogy vajon minden $\mathbb{Z}_p^*$-ban találunk-e ilyet. A válasz igenlő (de nehéz bizonyítani, ezért azt mellőzzük):

4.3. Tétel. *Ha p prím, akkor $\mathbb{Z}_p^*$-ban létezik generátor.*

Generátor tehát mindig van (prím modulusra), és előállítja az összes nemnulla elemet. Ráadásul az első $p-1$ kitevőre mindegyiket pontosan egyszer. Érdemes tehát feltenni a következő kérdést: egy adott elem hányadik helyen szerepel ebben a listában? Vegyük észre, hogy ez a kérdés éppen a hatványozás inverze: most nem a hatvány értékére vagyunk kíváncsiak az alap és a kitevő ismeretében, hanem a kitevőt szeretnénk megkapni az alapból és a hatványból. Ez a kérdés nagyon hasonlít a valós számoknál ismert logaritmusra: az $b^x = a$ valós egyenlet megoldása $x = \log_b a$, például a $2^x = 4\sqrt{2}$ egyenlet megoldása $x = \log_2 4\sqrt{2} = 2{,}5$. Vezessük be ennek a moduláris analógját:

4.4. Definíció. Ha $\overline{a}, \overline{b} \in \mathbb{Z}_p^*$, akkor a legkisebb $x \in \mathbb{N}$-et, amelyre $\overline{b}^x = \overline{a}$, az $\overline{a}$-nak a $\overline{b}$-alapú *diszkrét logaritmus*ának (vagy *index*ének) nevezzük. Jelölés: $x = \log_{\overline{b}} \overline{a}$ (vagy $x = \mathrm{ind}_{\overline{b}} \overline{a}$).

Például $\mathbb{Z}_7$-ben $\overline{6}$-nak az $\overline{5}$-ös alapú diszkrét logaritmusa 3, mert $\overline{5}^3 = \overline{6}$ (lásd az $\overline{5}$ hatványait fent).

Ha $\overline{g} \in \mathbb{Z}_p^*$ generátor, akkor $\log_{\overline{g}} \overline{a}$ biztosan létezik bármilyen $\overline{a} \in \mathbb{Z}_p^*$-ra.

De hogyan tudjuk kiszámítani a diszkrét logaritmust? Például úgy, hogy kiszámoljuk $\overline{g}$ hatványait 0-tól $p-2$-ig, és amelyik megegyezik $\overline{a}$-val, annak a kitevője lesz a diszkrét logaritmus. Ez azonban nagy p-re nagyon lassú módszer, mert a hatványok kiszámításához összesen $p-1$ szorzás kell, tehát ha pl. $p \approx 1000000$, akkor ez kb. egymillió szorzást jelent.

Nézzünk egy valamelyest hatékonyabb módszert, amelyet angolul *baby-step giant-step* algoritmusnak hívnak. Az ötlet az, hogy vegyünk egy közbülső N számot, amelyre kb. $N^2 \approx p-1$, és keressük az $\overline{a} = \overline{g}^x$ ismeretlen x kitevőjét $x = iN + j$ alakban, ahol $0 \le i, j < N$ (például ha $N = 10$ és $x = 76$, akkor $x = 7N + 6$, azaz $i = 7$ és $j = 6$). Először számítsuk ki az első néhány $\overline{g}^j$ hatványt: $\overline{1}$, $\overline{g}$, $\overline{g}^2$, …, $\overline{g}^{N-1}$ ("baby-step"), és tároljuk el őket. Ezután vegyük a kérdéses $\overline{a} = \overline{g}^x$ számot, és kezdjük el szorozgatni $\overline{g}^{-N}$-nel: $\overline{a}$, $\overline{g}^{-N}\overline{a}$, $\overline{g}^{-2N}\overline{a}$, … ("giant-step"). Ez azért jó, mert a két sorozat (a "baby-step" és a "giant-step") előbb-utóbb összeér: amikor $\overline{a}$-t éppen $\overline{g}^{-iN}$-nel szoroztunk meg (ahol i az $x = iN + j$ felírásból van, tehát nem ismerjük), akkor $\overline{g}^{-iN}\overline{a} = \overline{g}^{-iN+x} = \overline{g}^j$, ami szerepel az eltárolt $\overline{g}^j$ értékek között. Ha ezt megtaláltuk, akkor megvan i és j, és így x is.

A teljes baby-step giant-step algoritmus tehát így néz ki:

1. Bemenet: p prímszám, $\overline{a}, \overline{g} \in \mathbb{Z}_p^*$, ahol $\overline{g}$ egy generátor.
2. $N := \left\lceil \sqrt{p-1} \right\rceil$.
3. $\overline{b} := \overline{1}$.
4. Minden j-re 0-tól $N-1$-ig:
   (a) Tároljuk el $(\overline{b}, j)$-t egy táblában. (Ahol $\overline{b} = \overline{g}^j$.)
   (b) $\overline{b} := \overline{b} \cdot \overline{g}$.
5. $\overline{c} := \overline{b}^{-1}$ $(= \overline{g}^{-N})$.
6. $\overline{b} := \overline{a}$.
7. Minden i-re 0-tól $N-1$-ig:
   (a) Ha $\overline{b}$ benne van a táblában valamely j-vel, akkor készen vagyunk. Eredmény: $x := iN + j$.
   (b) $\overline{b} := \overline{b} \cdot \overline{c}$.

Ahhoz, hogy ez hatékony legyen, az kell, hogy a "táblában" gyorsan tudjunk értékeket tárolni és visszakeresni. Erre például egy hash-tábla alkalmas, amely (jó esetben) minden műveletet konstans időben hajt végre (azaz az idő nem függ a már benne lévő elemek számától). A tábla kulcsa $\overline{b}$, az értéke pedig a j index. Például Sage-ben használhatjuk a Python-féle szótár (`dict`) típust (üres szótár: `T = {}`, elem hozzáadása: `T[b] = j`, elem keresése: `b in T`, majd lekérdezése: `T[b]`).
Ebben az esetben a fenti algoritmus kb. $2\sqrt{p}$ szorzást igényel, amely az előző naiv módszernél sokkal jobb (pl. $p \approx 1000000$ esetén ~ 2000 szorzás kell). De ha összehasonlítjuk a gyorshatványozás futási idejével, akkor ez nagy p-kre még mindig viszonylag lassú. Ráadásul sokkal nagyobb memóriaigényű is (kb. $2\sqrt{p}$ számot kell tárolni).
Akkor hát hogyan lehet a diszkrét logaritmust kiszámítani hatékonyan, mint a hatványozást? A válasz: nem tudjuk. Olyannyira nem, hogy a diszkrét logaritmus probléma mind a mai napig a számításelmélet nyitott problémái közé tartozik. De ez nem baj. Sőt, mint majd látni fogjuk, kifejezetten hasznos!
Sage-ben a valós számokra már ismert `log(a, b)` függvény diszkrét logaritmust is tud számítani, ha maradékosztályokat adunk neki (de természetesen nagyon nagy modulusokra ez sem fog működni):

```
sage: Z7 = Zmod(7)
sage: log(Z7(6), Z7(5))
3
sage: Z7(5)^3
6
```

5. Diffie–Hellman-kulcscsere

Tegyük fel, hogy két ember szeretne egy titkos információban megállapodni. A probléma az, hogy nem tudnak titokban beszélni, hanem bármit, amit mondanak egymásnak, más is hallhatja. Hogyan tudnak így mégis egy közös titkot megbeszélni?
A kérdés nem légbőlkapott. Gondoljunk az internetre, ahol a nem titkosított üzeneteket könnyű lehallgatni. De hogyan tud két fél titkosítva kommunikálni, ha még nem találkoztak, azaz csak a nyílt interneten tudják megbeszélni a titkosításhoz szükséges titkos adatokat (pl. kulcsot)?

Erre az első ránézésre lehetetlennek tűnő kérdésre ad választ a *Diffie–Hellman-kulcscsere*. A megoldás azon alapul, hogy a diszkrét logaritmus probléma nehéz, azaz ha van egy titkos $n \in \mathbb{N}$ kitevőnk, akkor egy $\overline{g} \in \mathbb{Z}_p^*$ generátor $\overline{g}^n$ hatványát nyugodtan nyilvánosságra hozhatjuk ($\overline{g}$-vel együtt), hiszen senki nem fogja tudni kiszámítani a titkos n kitevőt. (Legalábbis ha p elég nagy, és bizonyos tulajdonságoknak megfelel, amelyeket itt nem részletezünk.)

A Diffie–Hellman-kulcscsere a következőképpen működik:

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

A közös titkos adat tehát $\overline{g}^{ab}$, amelyet a fenti műveletsor után mind Alíz, mind Bob ki tud számítani. Viszont bárki más, aki hallgatózik a csatornán, csak $\overline{g}$-t, $\overline{g}^a$-t és $\overline{g}^b$-t láthatja, ezekből viszont nem fogja tudni kiszámítani $\overline{g}^{ab}$, mert ahhoz meg kéne oldania a diszkrét logaritmus problémát.

Megjegyzés: ahhoz, hogy az eljárás valóban biztonságos legyen, bizonyos feltételeknek teljesülnie kell. Egyrészt, mint már említettük, p-nek elég nagynak és megfelelőnek kell lennie. Másrészt, amikor Alíz és Bob generálják a véletlen kitevőket, a-t és b-t, akkor azoknak valóban *elég véletlennek* kell lenniük, mert ha egy külső támadó meg tudja őket tippelni (akár csak bizonyos valószínűséggel), akkor az egész algoritmus semmit nem ér.

A *biztonságos véletlenszám-generátorok* (cryptographically secure pseudorandom number generators) kellő matematikai garanciákkal rendelkeznek arra, hogy egy külső megfigyelő ne tudja megtippelni a következő számokat. A nem biztonságos véletlenszám-generátorok általában egyszerűbbek, talán hatékonyabbak is, de nem rendelkeznek ilyen garanciával, ezért ezeket ne használjuk olyan esetben, amikor a biztonság fontos. A legtöbb nyelvben, így Sage-ben is (a Python-ból) vannak biztonságos és nem biztonságos véletlenszám-generátorok is:

```
sage: randrange(100)            # NEM biztonságos (0 és 99 között)
94
sage: randrange(100, 200, 10)  # úgy paraméterezhető, mint a range()
180
sage: import secrets
sage: secrets.randbelow(100)   # biztonságos (0 és 99 között)
67
```

6. Szerző

Ezt a jegyzetet Uray M. János írta, részben felhasználva Nagy Ádám jegyzetét:
https://compalg.elte.gitlab-pages.hu/dimoa-web/tematika/dimoa
Hibákat, észrevételeket itt lehet jelezni: uray.janos@inf.elte.hu.