# Algoritmusok és adatszerkezetek II.

## Sztring,-és számelméleti algoritmusok

Szegedi Tudományegyetem

# Mintaillesztés problémája

$T = [t_1 \dots t_n]$ input tartalmazza-e a $P = [p_1 \dots p_m]$ mintát? Ha igen, mely inputpozíció(k)tól/eltolási érték(ek)től kezdődően? Tipikusan $n \gg m$

# Mintaillesztés problémája

$T = [t_1 \dots t_n]$ input tartalmazza-e a $P = [p_1 \dots p_m]$ mintát? Ha igen, mely inputpozíció(k)tól/eltolási érték(ek)től kezdődően? Tipikusan $n \gg m$

**Példa**

Hányszor szerepel az `'orosz'` szó a *Háború és béke* c. műben?

Nyers erőt használva legrosszabb esetben $O(mn)$ vizsgálat kell
Milyen hatékonyabb módszerek vannak?

- Mintaillesztés automatával
- Knuth-Morris-Prat
- **Rabin-Karp algoritmus**

# Jelölések

- $\mathrm{P}_i$ jelölje P-nek az $i$ hosszúságú prefixét (kezdőszeletét), azaz pl. $\mathrm{P}_3 = bab$, $\mathrm{P}_1 = b$, illetve $\mathrm{P}_0 = \epsilon$ ($\epsilon$ az üres szót jelöli)

# Jelölések

- $\mathtt{P}_i$ jelölje $\mathtt{P}$-nek az $i$ hosszúságú prefixét (kezdőszeletét), azaz pl. $\mathtt{P}_3 = bab$, $\mathtt{P}_1 = b$, illetve $\mathtt{P}_0 = \epsilon$ ($\epsilon$ az üres szót jelöli)
- $\mathtt{X} \sqsupset \mathtt{Y}$ jelölje azt, ha $\mathtt{X}$ sztring szuffixe $\mathtt{Y}$-nak (azaz $\mathtt{Y}$ végződése maga $\mathtt{X}$)

**Példa**

$\underline{aaba} \sqsupset cac\underline{aaba}$, ugyanakkor $\underline{aaba} \not\sqsupset cac\underline{aabb}$

Megjegyzés: Az $\mathtt{Y} \sqsupset \mathtt{Y}$, valamint az $\epsilon \sqsupset \mathtt{Y}$ relációk triviálisan teljesülnek minden $\mathtt{Y}$-ra.

# Véges állapotú automaták

Automata alatt egy $M = (Q, q_0, A, \Sigma, \delta)$ rendezett ötöst értünk, ahol

- $Q$ a lehetséges állapotok halmaza
- $q_0$ a kezdőállapot
- $A \subseteq Q$ a végállapotok halmaza
- $\Sigma$ egy véges ábécé
- $\delta : Q \times \Sigma \to Q$ az állapotátmenet-függvény

# Mintaillesztés véges állapotú automatákkal

- $Q$-t válasszuk $\{q_0, q_1, \ldots q_m\}$-nak
- $q_i$ állapot jelentése: az input aktuális pozíciójáig a minta első $i$ karaktere illeszkedik
- $q_m$ állapotba elérve elmondható, hogy megtaláltuk a $P$-nek egy $T$-beli előfordulását

# $\delta$ kiszámítása

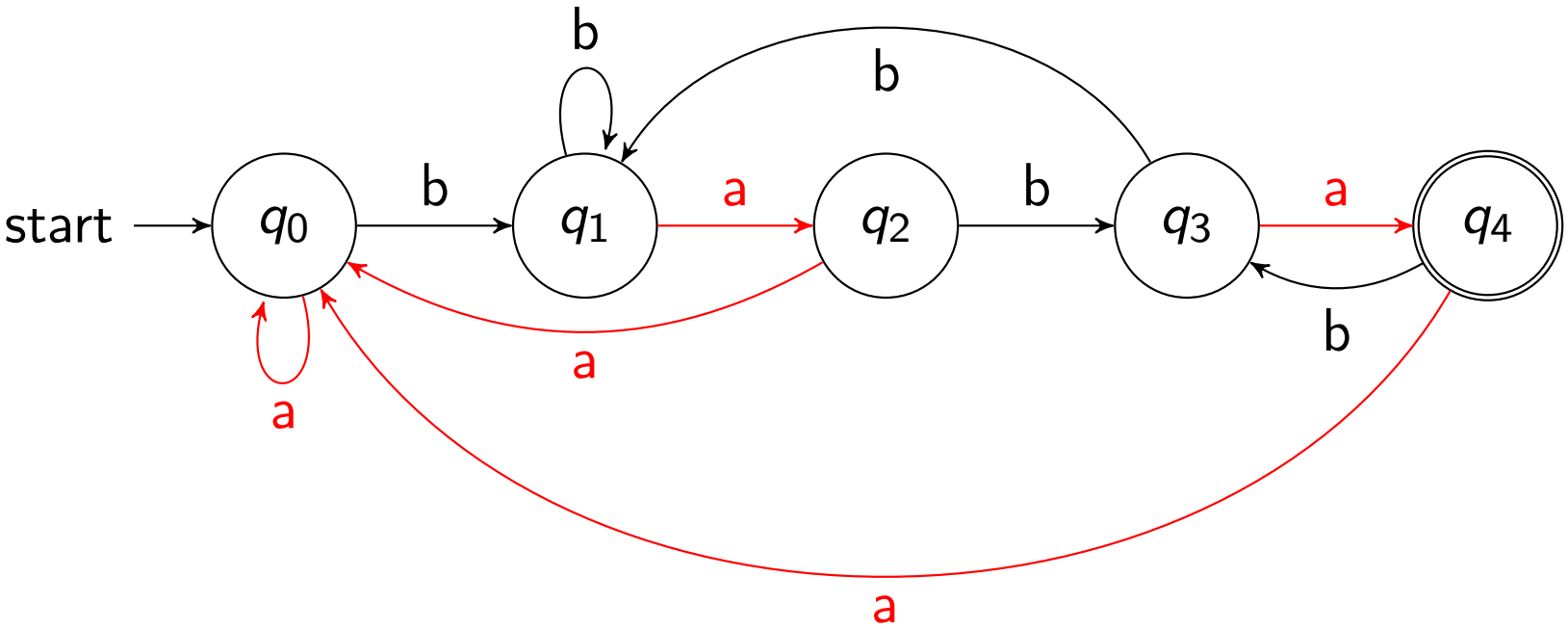
```
ÁtmenetFüggvénySzámítás(P, Σ) {
  m = P.length
  for (q = 0 to m) {
    for (a ∈ Σ) {
      k = min(m + 1, q + 2)
      repeat
        k = k - 1
      until P_k ⊐ P_q a
      δ(q, a) = k
    }
  }
  return δ
}
```

# $P = baba$ minta felismerését végző véges automata

## Az állapotátmenet-függvény táblázatos formában

| | $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ |
|---|---|---|---|---|---|
| a | $q_0$ | $\mathbf{q_2}$ | $q_0$ | $\mathbf{q_4}$ | $q_0$ |
| b | $\mathbf{q_1}$ | $q_1$ | $\mathbf{q_3}$ | $q_1$ | $q_3$ |

Kitöltése és tárolása egyaránt $O(m|\Sigma|)$ költségű



# Knuth-Morris-Pratt (KMP) algoritmus

## A véges automatával történő feldolgozás korlátai

Előfeldolgozás gyanánt ki kell tölteni egy $m|\Sigma|$ méretű táblázatot (és persze tárolni is kell azt)

# Knuth-Morris-Pratt (KMP) algoritmus

## A véges automatával történő feldolgozás korlátai

Előfeldolgozás gyanánt ki kell tölteni egy $m|\Sigma|$ méretű táblázatot (és persze tárolni is kell azt)

## Észrevételek

1. Ha $P = aab$ minta illeszkedett az input $i$-edik pozíciójára, akkor az $i + 1$ pozíciótól kezdődően biztos nem beszélhetünk illeszkedésről
2. Ha $P = aaa$ minta nem illeszkedett az input $i$-edik pozíciójára, akkor az $i + 1$ pozíciótól kezdődően biztos nem beszélhetünk illeszkedésről

# Prefixfüggvény

- $\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$ $P = [p_1 \ldots p_m]$ minta prefixfüggvénye, ha $\pi[q] = \max\{k : k < q \wedge \mathrm{P}_k \sqsupset \mathrm{P}_q\}$
- Azaz $\pi[q]$ megadja $\mathrm{P}$ azon leghosszabb ($q$-nál rövidebb) prefixének hosszát, ami valódi szuffixe $\mathrm{P}_q = [p_0 \ldots p_q]$-nak
- Értelmezése: ha **nem sikerül továbbillesszük** a mintát az inputra, az **legalább** mely állapotig vet minket vissza

# Prefixfüggvény

- $\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$ $P = [p_1 \ldots p_m]$ minta prefixfüggvénye, ha $\pi[q] = \max\{k : k < q \wedge \mathrm{P}_k \sqsupset \mathrm{P}_q\}$
- Azaz $\pi[q]$ megadja P azon leghosszabb ($q$-nál rövidebb) prefixének hosszát, ami valódi szuffixe $\mathrm{P}_q = [p_0 \ldots p_q]$-nak
- Értelmezése: ha **nem sikerül továbbillesszük** a mintát az inputra, az **legalább** mely állapotig vet minket vissza

## Példa

$T = abaababababaca$ inputban keressük a $P = abababababca$ mintát

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $P[i]$ | a | b | a | b | a | b | a | b | c | a |
| $\pi[i]$ | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 0 | 1 |

# A prefixfüggvény növekedése

Indirekt belátható, hogy $\pi[i+1] - \pi[i] \leq 1$ minden esetben teljesül

# A prefixfüggvény növekedése

Indirekt belátható, hogy $\pi[i+1] - \pi[i] \leq 1$ minden esetben teljesül

$$\overbrace{\underbrace{s_1\ s_2}_{\pi[i]=2}\ s_3\ s_4}^{\pi[i+1]=4} \cdots \overbrace{\underbrace{s_{i-2}}_{\pi[i]=2}\ s_{i-1}\ s_i\ s_{i+1}}^{\pi[i+1]=4}$$

Azaz $\pi[i+1] - \pi[i] > 1$ csak úgy teljesülhet, ha $\pi[i]$ nem $\mathtt{P}_i$ leghosszabb valódi szuffixének a hossza, ami ellentmondás.

# Prefixfüggvény viselkedése

- Folytatódó illeszkedés

$$\underbrace{\overbrace{s_1\ s_2\ s_3}^{\pi[i]}\ \overbrace{s_4}^{s_4=s_{i+1}}}_{\pi[i+1]=\pi[i]+1}\ \ldots\ \underbrace{\overbrace{s_{i-2}\ s_{i-1}\ s_i}^{\pi[i]}\ \overbrace{s_{i+1}}^{s_4=s_i+1}}_{\pi[i+1]=\pi[i]+1}$$

- Nem (közvetlen) folytatódó illeszkedés

$$\underbrace{\overbrace{s_1\ s_2}^{j}\ s_3\ s_4}^{\pi[i]}\ \ldots\ \overbrace{\underbrace{s_{i-3}\ s_{i-2}\ s_{i-1}\ s_i}_{j=\pi[\pi[i]]}}^{\pi[i]}\ s_{i+1}$$

# Prefixfüggvény kiszámítása

```
PrefixSzámít(P) {
  m = P.length
  π[1] = 0
  i = 0
  for (q=2 to m) {
    while i > 0 and P[i+1] ≠ P[q]
      i = π[i]
    if P[i+1] = P[q]
      i = i+1
    π[q]=i
  }
  return π
}
```

# KMP algoritmus

```
KMP-ILLESZTŐ(T, P) {
  n = T.length
  m = P.length
  π = PREFIXSZÁMÍT(P)
  q = 0
  for (i = 1 to n)
    while q > 0 and P[q + 1] ≠ T [i]
      q = π[q]
    if P[q + 1] = T [i]
      q = q + 1
    if q = m
      print("Illeszkedés a %d. pozícióval bezárólag" % i)
      q = π[m]
}
```

# Rabin-Karp algoritmus

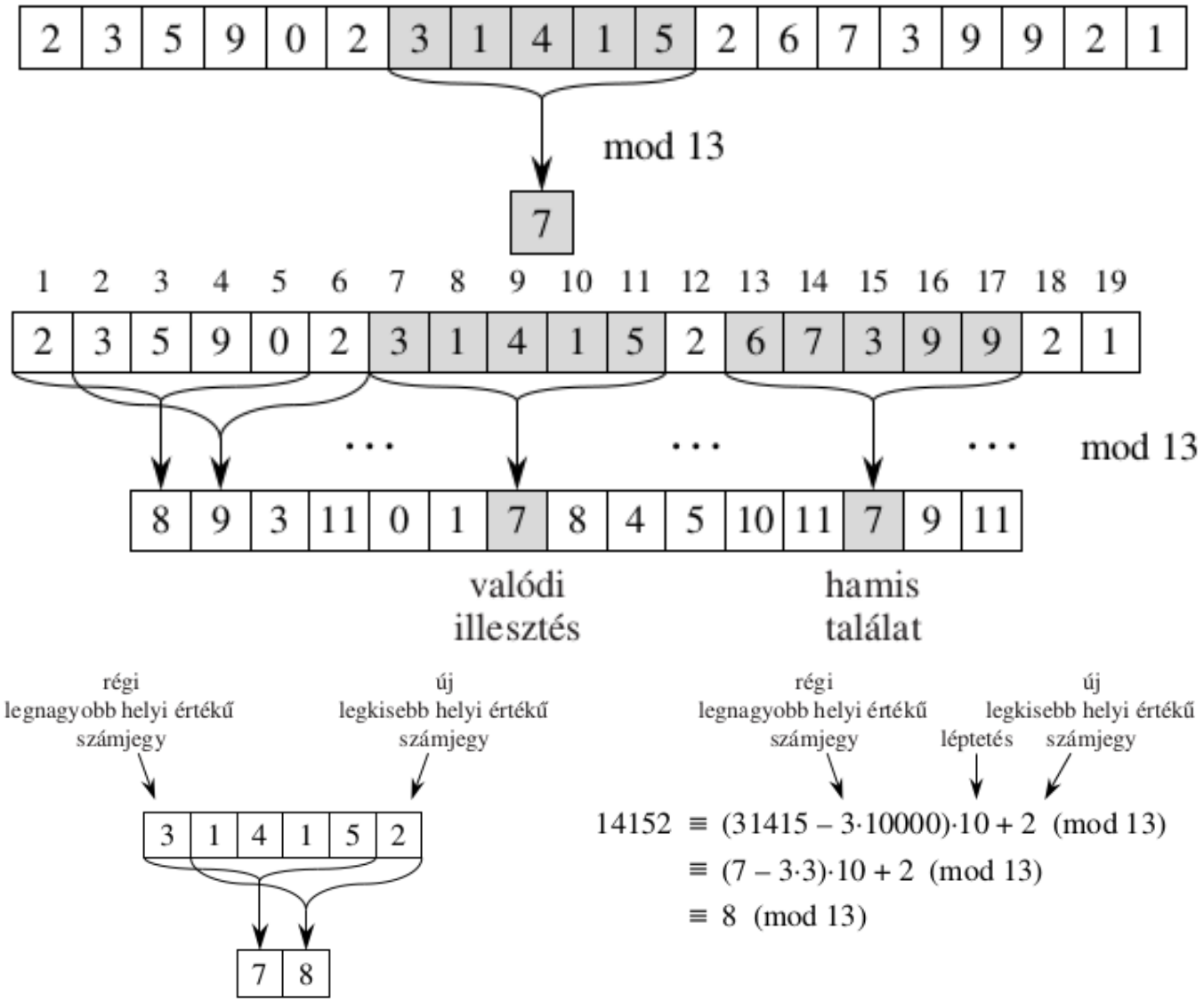
Az egyszerűség kedvéért a mintára és az inputra tekintsünk 10-es számrendszerbeli számokként

## Alapötlet

$P$ mintára alkalmazzunk egy $h_q(x) = x \mod q$ hasítófüggvényt.
$T$-nek csak azon $S = [t_j]_{j=i}^{i+m-1}$ részsztringjei egyezhetnek meg $P$-vel, melyekre $h_q(S) = h_q(P)$
$h_q(S) = h_q(P)$ természetesen csak szükségességi feltételt támaszt

# Rabin-Karp algoritmus illusztráció (Forrás: CLRS 32.5 ábra)

$P = 31415, h_q(x) = x \mod 13$



# Rabin-Karp – a hasítófüggvény csúsztatása

**Észrevétel**

$h_q$-nak az input minden indexén kezdődő kiszámítása költséges

# Rabin-Karp – a hasítófüggvény csúsztatása

## Észrevétel

$h_q$-nak az input minden indexén kezdődő kiszámítása költséges

## A hatékonyság záloga

Ahogy $abc$ és $bcd$ számpárral teljesül

$$bcd = 10 * (abc - 100 * a) + d$$

egyenlőség, hasonlóan igaz

$$h_q(bcd) = h_q(10 * (h_q(abc) - h_q(100) * a) + d)$$

összefüggés is.

# Kis kitérő: Bloom filterek

- Probabilisztikus adatszerkezet a Keres műveletre nézve
- Implementációja: (egyenletesen szóró) $h$ hasítófüggvény és egy (kezdetben csupa 0) bitvektor
  - $x$ elem beszúrásakor a bitvektor $h(x)$ indexét 1-re állítjuk
  - $x$ elem keresésnél ha a bitvektor $h(x)$ indexe 1, akkor azt mondjuk, hogy $x$-et tartalmazza a bloom filterünk
    - Hamis elutasítás soha nem teszünk, ugyanakkor hamis pozitív választ adhatunk
- Hatékonysága függ $h$ hasítófüggvénytől, valamint az eltárolt elemek számától (ami kihat a bitvektor kitöltöttségére)
- Linkek: Bloom filter demo és Guava API

# Bloom filterek tévesztési arányának elemzése

- Egy bloom filter olyan arányban ad egy elem tartalmazására nézve igenlő választ, mint amennyi bitje egyesre lett állítva a beszúrások hatására
  - Feltesszük, hogy az alkalmazott $h$ hasítófüggvény jól szór, azaz $m$ ekvivalenciaosztályba való sorolás esetén az objektumok közelítőleg $\frac{1}{m}$ részét sorolja az összes osztályba

**Hány bitje lesz egy $m$ hosszú bloom filternek 1-es $n$ beszúrás után?**

1 beszúrás során $\frac{1}{m}$ valószínűséggel töltünk föl egy bitet
1 beszúrás során $1 - \frac{1}{m}$ valószínűséggel *nem* töltünk föl egy bitet
$n$ beszúrás során $\left(1 - \frac{1}{m}\right)^n$ valószínűséggel *nem* töltünk föl egy bitet

# Bloom filterek tévesztési arányának elemzése

- Egy bloom filter olyan arányban ad egy elem tartalmazására nézve igenlő választ, mint amennyi bitje egyesre lett állítva a beszúrások hatására
  - Feltesszük, hogy az alkalmazott $h$ hasítófüggvény jól szór, azaz $m$ ekvivalenciaosztályba való sorolás esetén az objektumok közelítőleg $\frac{1}{m}$ részét sorolja az összes osztályba

**Hány bitje lesz egy $m$ hosszú bloom filternek 1-es $n$ beszúrás után?**

1 beszúrás során $\frac{1}{m}$ valószínűséggel töltünk föl egy bitet
1 beszúrás során $1 - \frac{1}{m}$ valószínűséggel *nem* töltünk föl egy bitet
$n$ beszúrás során $\left(1 - \frac{1}{m}\right)^n$ valószínűséggel *nem* töltünk föl egy bitet
$\left(1 - \frac{1}{m}\right)^n = \left[\left(1 - \frac{1}{m}\right)^m\right]^{\frac{n}{m}} \approx e^{-\frac{n}{m}}$ (nagy $m$ esetén)
$n$ beszúrást követően $1 - e^{-\frac{n}{m}}$ valószínűséggel lesz a bloom filter valamely bitje 1-esre állítva

# Mintaillesztés főbb megközelítései

| Algoritmus | Előfeldolgozás | Illesztés[1] |
|---|---|---|
| Nyers erő | 0 | $O(mn)$ |
| Rabin-Karp | $\Theta(m)$ | $O(mn)$ |
| Véges automata | $O(m\|\Sigma\|)$ | $\Theta(n)$ |
| Knuth-Morris-Prat | $\Theta(m)$ | $\Theta(n)$ |

- Rabin-Karp a nyers erő módszerének heurisztikus kiterjesztéseként tekinthető
- KMP pedig a véges automatákkal való illesztés egy hatékonyabb verziója

[1] legrosszabb eset

# A moduláris hatványozás

**Alapprobléma: Mi az $a^b \mod n$ kifejezés értéke?**

Gyakorlati jelentőség: titkosító eljárások (pl. RSA) használata során szükségünk lehet ilyen számítások elvégzésére

Már viszonylag kis $b$-re is rengeteg bitműveletet kell elvégezzünk

# A moduláris hatványozás

**Alapprobléma: Mi az $a^b \mod n$ kifejezés értéke?**

Gyakorlati jelentőség: titkosító eljárások (pl. RSA) használata során szükségünk lehet ilyen számítások elvégzésére

Már viszonylag kis $b$-re is rengeteg bitműveletet kell elvégezzünk

**Példa – Mi lesz $7^{560} \mod 561$ értéke? Avagy**

$$1798466729205729065772587222245603360838566081370073425616126361443960897695529556655491359956040756080966911621011841135459725266382550047840553113905985423059583570970823912250610774332816201171300138264486062817086659379316597967367552530749773664710631469233738652235015321857530762927108874018017743927794756795105563119669819528260254870572046992619139736642578579937440451434475311215405742159698029610038720861638609917020355065913128470296742603625091567459651360 01 \mod 561 = ?$$

# Moduláris hatványozás

```
MODULÁRIS-HATVÁNYOZÓ(a, b, n){
   c = 0
   d = 1
   B = [bk ... b1 b0]
   for (i=k; k >=0; --k) {
      c = 2*c
      d = (d*d) mod n
      if (B[i] == 1) {
         c = c+1
         d = (d*a) mod n
      }
   }
   return d
}
```

**Megjegyzések**

1. $b$-nek $k$ bites bináris alakja $B$
2. $c$ csak egy segédváltozó

# Moduláris hatványozás

```
MODULÁRIS-HATVÁNYOZÓ(a, b, n){
   c = 0
   d = 1
   B = [bk ... b1 b0]
   for (i=k; k >=0; --k) {
      c = 2*c
      d = (d*d) mod n
      if (B[i] == 1) {
         c = c+1
         d = (d*a) mod n
      }
   }
   return d
}
```

## Megjegyzések

1. $b$-nek $k$ bites bináris alakja $B$
2. $c$ csak egy segédváltozó

## Kérdés

- Mi lesz $d$ és $c$ értéke a for ciklus első végrehajtása után?

# Moduláris hatványozás

```
MODULÁRIS-HATVÁNYOZÓ(a, b, n){
    c = 0
    d = 1
    B = [bk ... b1 b0]
    for (i=k; k >=0; --k) {
        c = 2*c
        d = (d*d) mod n
        if (B[i] == 1) {
            c = c+1
            d = (d*a) mod n
        }
    }
    return d
}
```

## Megjegyzések

1. $b$-nek $k$ bites bináris alakja $B$
2. $c$ csak egy segédváltozó

## Kérdés

- Mi lesz $d$ és $c$ értéke a for ciklus első végrehajtása után?

## Műveletigény

Ha $a, b$ és $n$ $k$ biten elfér, mindez $O(k)$ aritmetikai műveletet és $O(k^3)$ bitműveletet jelent

# A moduláris hatványozás működése

- Az algoritmus for ciklusában minden egyes iteráció megkezdése előtt a $c$ és $d$ változók értékeire teljesül, hogy:
  1. $c$ változó értéke megegyezik a $[b_k \dots b_{i+1}]$ bináris értékkel
  2. $d = a^c \mod n$
- A fenti ciklusinvariáns teljesül, megmarad és befejeződik $\rightarrow$

# Moduláris hatványozás példa

## $7^{560} mod 561 = ?$

$a = 7, b = 560, n = 561$ $(B = [1000110000])$

| $i$ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $b_i$ | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $c$ | 1 | 2 | 4 | 8 | 17 | 35 | 70 | 140 | 280 | 560 |
| $d$ | 7 | 49 | 157 | 526 | 160 | 241 | 298 | 166 | 67 | **1** |