

Bevezetés a Számításelméletbe II. 13. előadás

Sali Attila

Budapest Műszaki és Gazdaságtudományi Egyetem

Számítástudományi és Információelméleti Tsz.

I. B. 137/b

`sali@cs.bme.hu`

2002 május 7.

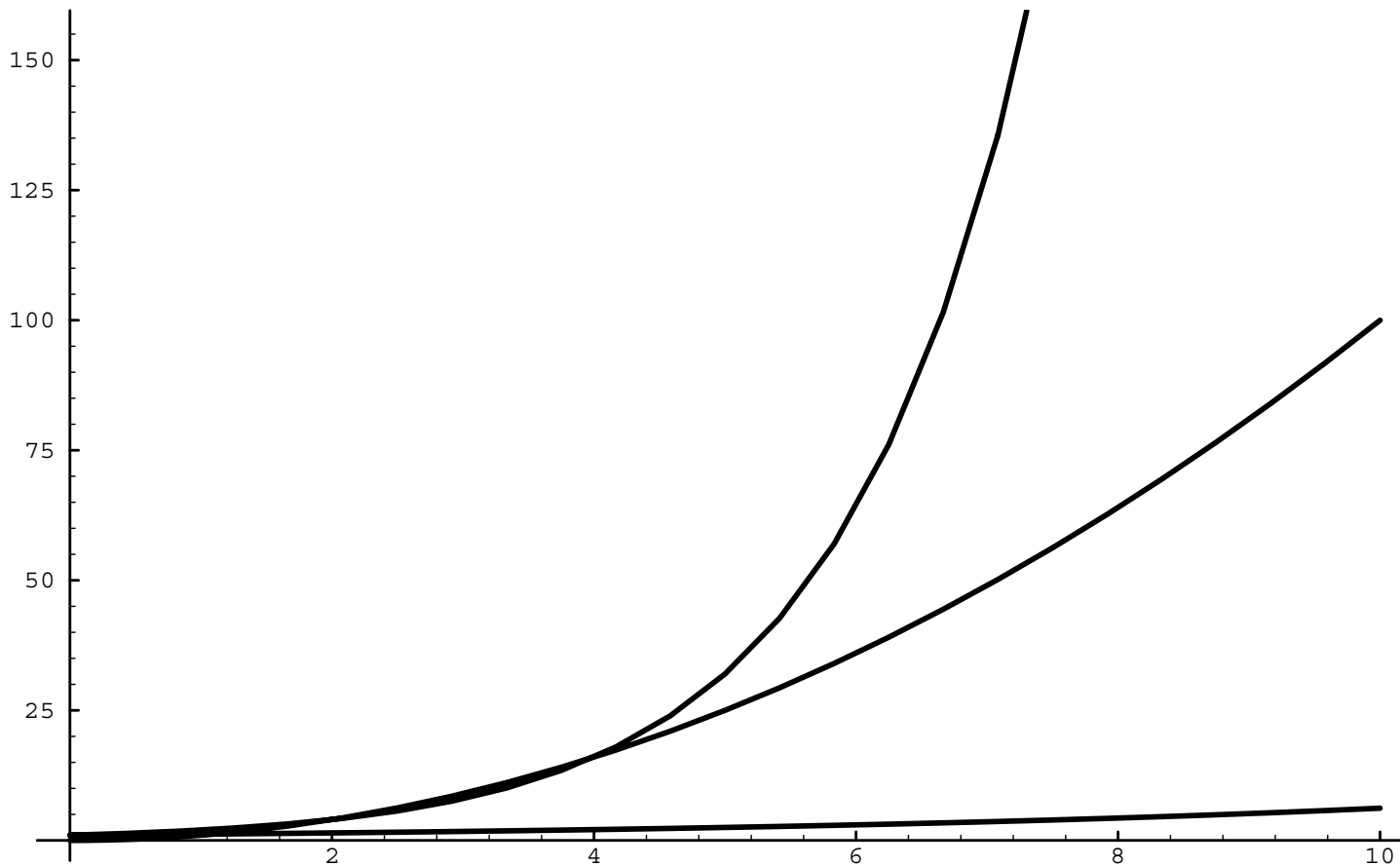
Miért a polinom idejű algoritmus jó?

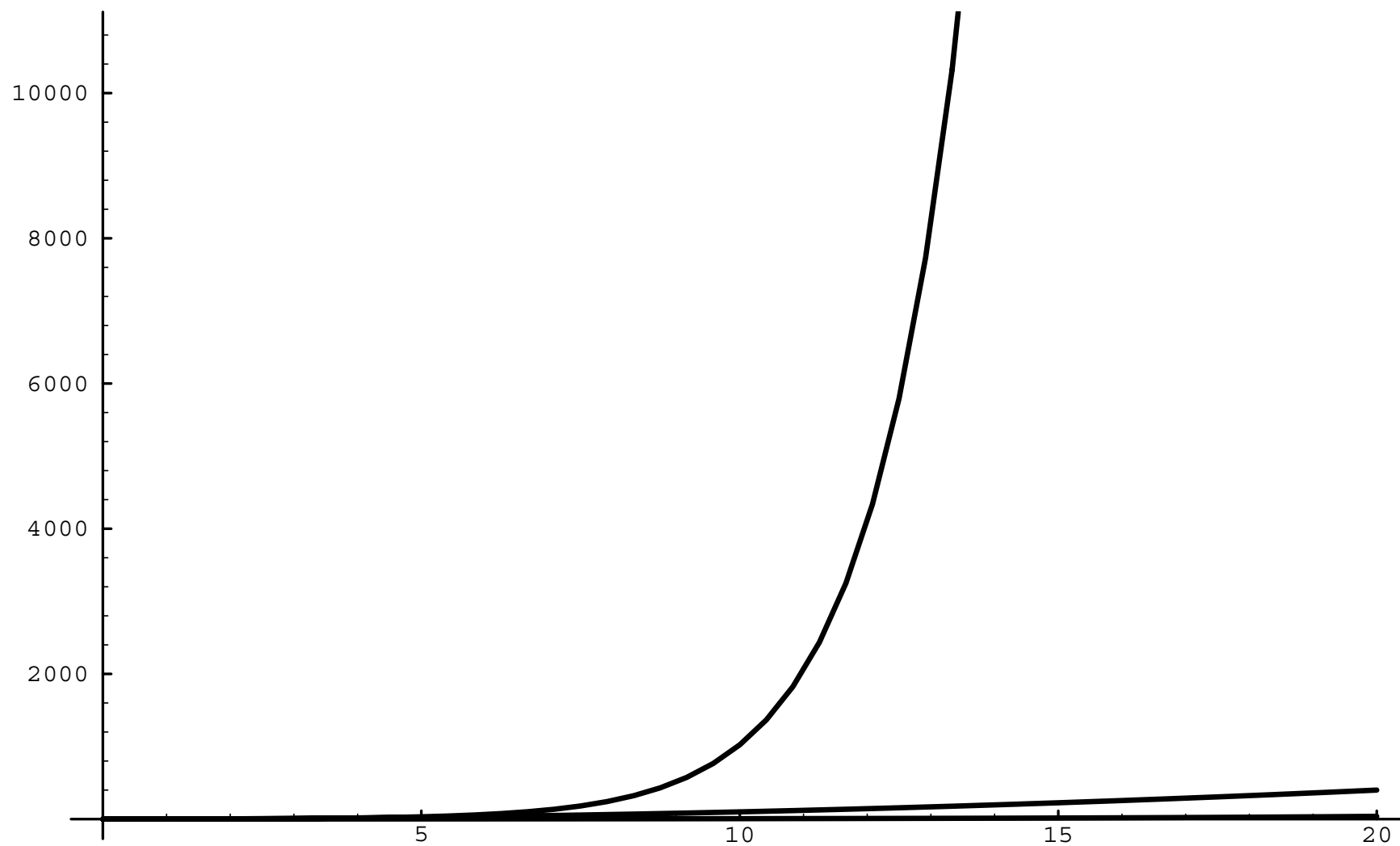
Miért a polinom idejű algoritmus jó?

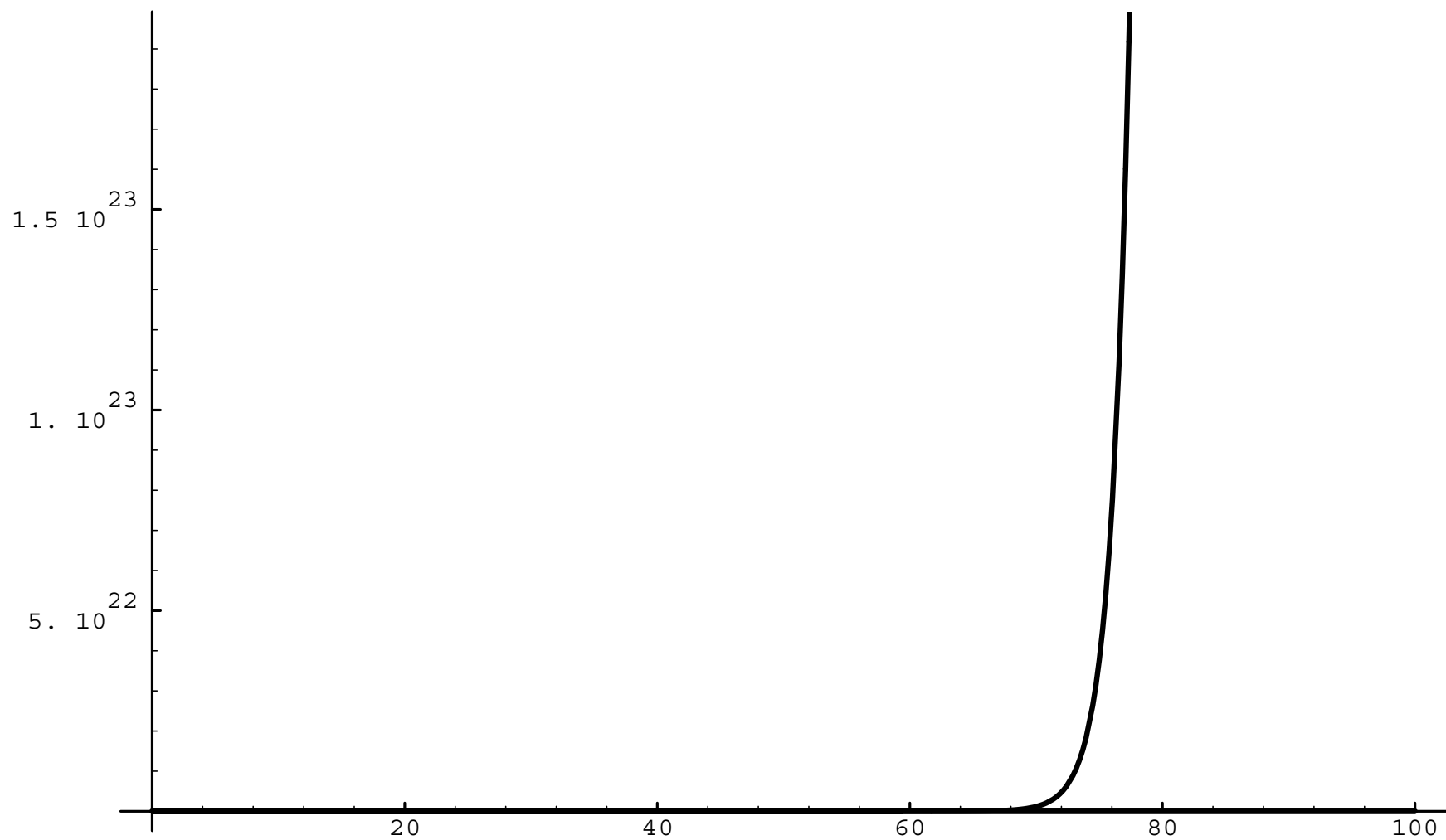
Hasonlítsuk össze az $f(x) = x^2$, $f(x) = 1.2^x$, $f(x) = 2^x$ függvények grafikonjait

Miért a polinom idejű algoritmus jó?

Hasonlítsuk össze az $f(x) = x^2$, $f(x) = 1.2^x$, $f(x) = 2^x$ függvények grafikonjait







Input mérete 1000

Ha az algoritmus lépésszáma n^2 , akkor $n = 1000$ esetén 10^6 lépést kell tenni.

Input mérete 1000

Ha az algoritmus lépésszáma n^2 , akkor $n = 1000$ esetén 10^6 lépést kell tenni. Ez egy mai PC-n kevesebb, mint egy másodperc.

Input mérete 1000

Ha az algoritmus lépésszáma n^2 , akkor $n = 1000$ esetén 10^6 lépést kell tenni. Ez egy mai PC-n kevesebb, mint egy másodperc.

Ha viszont a lépésszám 2^n , akkor $2^{1000} \approx 10^{300}$ lépés

Input mérete 1000

Ha az algoritmus lépésszáma n^2 , akkor $n = 1000$ esetén 10^6 lépést kell tenni. Ez egy mai PC-n kevesebb, mint egy másodperc.

Ha viszont a lépésszám 2^n , akkor $2^{1000} \approx 10^{300}$ lépés, ami egy másodpercenként 10^{100} műveletet végző géppel (mai PC $\approx 10^{10}$) 10^{200} másodperc,

Input mérete 1000

Ha az algoritmus lépésszáma n^2 , akkor $n = 1000$ esetén 10^6 lépést kell tenni. Ez egy mai PC-n kevesebb, mint egy másodperc.

Ha viszont a lépésszám 2^n , akkor $2^{1000} \approx 10^{300}$ lépés, ami egy másodpercenként 10^{100} műveletet végző géppel (mai PC $\approx 10^{10}$) 10^{200} másodperc, ami kb. $3 \cdot 10^{192}$ év!

Input mérete 1000

Ha az algoritmus lépésszáma n^2 , akkor $n = 1000$ esetén 10^6 lépést kell tenni. Ez egy mai PC-n kevesebb, mint egy másodperc.

Ha viszont a lépésszám 2^n , akkor $2^{1000} \approx 10^{300}$ lépés, ami egy másodpercenként 10^{100} műveletet végző géppel (mai PC $\approx 10^{10}$) 10^{200} másodperc, ami kb. $3 \cdot 10^{192}$ év!

Tehát azok az algoritmusok „jók”, amelyek lépésszáma polinomiális.

Input hossza

Input hossza

$x + y$ esetén mi az input hossza?

Input hossza

$x + y$ esetén mi az input hossza?

Egy k -jegyű x szám 10^{k-1} és $10^k - 1$ között van. Így leírásához $\lceil \log_{10} x \rceil$ számjegy kell.

Input hossza

$x + y$ esetén mi az input hossza?

Egy k -jegyű x szám 10^{k-1} és $10^k - 1$ között van. Így leírásához $\lceil \log_{10} x \rceil$ számjegy kell.

$\log_a x = c \cdot \log_b x$, ahol $c = \frac{\lg b}{\lg a}$ csak a -tól és b -től függ (vagyis x -től nem).

Input hossza

$x + y$ esetén mi az input hossza?

Egy k -jegyű x szám 10^{k-1} és $10^k - 1$ között van. Így leírásához $\lceil \log_{10} x \rceil$ számjegy kell.

$\log_a x = c \cdot \log_b x$, ahol $c = \frac{\lg b}{\lg a}$ csak a -tól és b -től függ (vagyis x -től nem). Így az a kijelentés, hogy egy mennyiség $c \cdot \log x$ -szel becsülhető, akkor is értelmes, ha a logaritmus alapját nem rögzítjük előre le (csak persze akkor c értékét sem tudjuk megmondani).

Input hossza

$x + y$ esetén mi az input hossza?

Egy k -jegyű x szám 10^{k-1} és $10^k - 1$ között van. Így leírásához $\lceil \log_{10} x \rceil$ számjegy kell.

$\log_a x = c \cdot \log_b x$, ahol $c = \frac{\lg b}{\lg a}$ csak a -tól és b -től függ (vagyis x -től nem). Így az a kijelentés, hogy egy mennyiség $c \cdot \log x$ -szel becsülhető, akkor is értelmes, ha a logaritmus alapját nem rögzítjük előre le (csak persze akkor c értékét sem tudjuk megmondani).

Az input hossza tehát $\lceil \log_{10} x \rceil + \lceil \log_{10} y \rceil \approx \log_{10} xy$ miatt xy logaritmusával arányos.

Elemi aritmetikaialgoritmusok

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 2 \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 7 \ 2 \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 0 \ 7 \ 2 \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

$$\begin{array}{r} 6 \ 5 \ 9 \\ - \ 4 \ 1 \ 3 \\ \hline \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

$$\begin{array}{r} 6 \ 5 \ 9 \\ - \ 4 \ 1 \ 3 \\ \hline \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

$$\begin{array}{r} 6 \ 5 \ 9 \\ - \ 4 \ 1 \ 3 \\ \hline 6 \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

$$\begin{array}{r} 6 \ 5 \ 9 \\ - \ 4 \ 1 \ 3 \\ \hline 4 \ 6 \end{array}$$

Elemi aritmetikaialgoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

$$\begin{array}{r} 6 \ 5 \ 9 \\ - \ 4 \ 1 \ 3 \\ \hline 2 \ 4 \ 6 \end{array}$$

Elemi aritmetikai algoritmusok

Összeadás és kivonás

$$\begin{array}{r} 4 \ 1 \ 3 \\ + \ 6 \ 5 \ 9 \\ \hline 1 \ 0 \ 7 \ 2 \end{array}$$

$$\begin{array}{r} 6 \ 5 \ 9 \\ - \ 4 \ 1 \ 3 \\ \hline 2 \ 4 \ 6 \end{array}$$

Lineáris lépésszámú

Szorzás

$$\underline{413} \cdot 659$$

Szorzás

$$\begin{array}{r} 413 \\ \hline 8 \end{array} \cdot 659$$

Szorzás

$$\begin{array}{r} 413 \\ \hline 78 \end{array} \cdot 659$$

Szorzás

$$\begin{array}{r} 413 \\ \times 659 \\ \hline \end{array}$$

Szorzás

$$\begin{array}{r} 413 \\ 2 \overline{) 478} \end{array} \cdot 659$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2478 \end{array}$$

5

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2478 \\ 65 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 065 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 7 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 17 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 717 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 7 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 67 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 167 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 2167 \end{array}$$

Szorzás

$$\begin{array}{r} 4 1 3 \cdot 6 5 9 \\ 2 \overline{) 4 7 8 0 6 5 3 7 1 7} \\ 4 7 8 \\ 2 0 6 5 \\ 3 7 1 7 \\ \hline 7 2 1 6 7 \end{array}$$

Szorzás

$$\begin{array}{ccccccc}
 & 4 & 1 & 3 & . & 6 & 5 & 9 \\
 2 & \overline{4} & 7 & 8 & & & & \\
 & 2 & 0 & 6 & 5 & & & \\
 & & 3 & 7 & 1 & 7 & & \\
 \hline
 2 & 7 & 2 & 1 & 6 & 7 & &
 \end{array}$$

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 272167 \end{array}$$

Négyzetes lépésszámú.

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 272167 \end{array}$$

Négyzetes lépésszámú. Lehet gyorsabban is!

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 272167 \end{array}$$

Négyzetes lépésszámú. Lehet gyorsabban is!

Az iskolai osztás nem igazi algoritmus!

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 272167 \end{array}$$

Négyzetes lépésszámú. Lehet gyorsabban is!

Az iskolai osztás nem igazi algoritmus! Ugyanis meg kell becsülni „hányszor van meg” az osztó az osztandó megfelelő darabjában

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 272167 \end{array}$$

Négyzetes lépésszámú. Lehet gyorsabban is!

Az iskolai osztás nem igazi algoritmus! Ugyanis meg kell becsülni „hányszor van meg” az osztó az osztandó megfelelő darabjában
Egy becsleés helyettesíthető 9 szorzással és kivonással

Szorzás

$$\begin{array}{r} 413 \cdot 659 \\ 2 \overline{) 478} \\ 2065 \\ 3717 \\ \hline 272167 \end{array}$$

Négyzetes lépésszámú. Lehet gyorsabban is!

Az iskolai osztás nem igazi algoritmus! Ugyanis meg kell becsülni „hányszor van meg” az osztó az osztandó megfelelő darabjában

Egy becsleés helyettesíthető 9 szorzással és kivonással \implies Polinomiális algoritmus.

Hatványozás

Hatványozás

2^x végeredményének *kiírásához* (tehát nem a kiszámításához) már $\log 2^x = x$ lépés kell,

Hatványozás

2^x végeredményének *kiírásához* (tehát nem a kiszámításához) már $\log 2^x = x$ lépés kell, ez pedig az input hosszának (vagyis $\log x$ -nek) exponenciális függvénye!

Euklideszi algoritmus

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$a = h_1 b + m_1 \quad (0 \leq m_1 < b)$$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{aligned} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \end{aligned}$$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{array}{ll} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \\ m_1 &= h_3 m_2 + m_3 & (0 \leq m_3 < m_2) \end{array}$$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{array}{ll} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \\ m_1 &= h_3 m_2 + m_3 & (0 \leq m_3 < m_2) \\ &\vdots & \vdots \end{array}$$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{array}{ll} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \\ m_1 &= h_3 m_2 + m_3 & (0 \leq m_3 < m_2) \\ &\vdots & \vdots \end{array}$$

Az eljárás akkor ér véget, ha nincs az osztásnak maradéka, vagyis

$$m_{n-2} = h_n m_{n-1}$$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{array}{ll} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \\ m_1 &= h_3 m_2 + m_3 & (0 \leq m_3 < m_2) \\ &\vdots & \vdots \end{array}$$

Az eljárás akkor ér véget, ha nincs az osztásnak maradéka, vagyis

$$m_{n-2} = h_n m_{n-1}$$

Vegyük észre: $m_{i+2} < \frac{1}{2}m_i$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{array}{ll} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \\ m_1 &= h_3 m_2 + m_3 & (0 \leq m_3 < m_2) \\ &\vdots & \vdots \end{array}$$

Az eljárás akkor ér véget, ha nincs az osztásnak maradéka, vagyis

$$m_{n-2} = h_n m_{n-1}$$

Vegyük észre: $m_{i+2} < \frac{1}{2}m_i \implies n \approx \log a$

Euklideszi algoritmus

Ha $a > b$, akkor az $a : b$ maradékos osztást elvégezzük, majd b -t osztjuk a maradékkal stb.:

$$\begin{array}{ll} a &= h_1 b + m_1 & (0 \leq m_1 < b) \\ b &= h_2 m_1 + m_2 & (0 \leq m_2 < m_1) \\ m_1 &= h_3 m_2 + m_3 & (0 \leq m_3 < m_2) \\ &\vdots & \vdots \end{array}$$

Az eljárás akkor ér véget, ha nincs az osztásnak maradéka, vagyis

$$m_{n-2} = h_n m_{n-1}$$

Vegyük észre: $m_{i+2} < \frac{1}{2}m_i \implies n \approx \log a \implies$ *polinomiális lépésszám*

$\text{mod } m$ **hatványozás**

$\text{mod } m$ hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

$\text{mod } m$ hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás

mod m hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás– exponenciális lépésszám!

mod m hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás– exponenciális lépésszám!

Írjuk fel n -et 2-es számrendszerben ($\log n$ darab osztás) $n = 2^{i_1} + 2^{i_2} + \dots + 2^{i_k}$ alakban ($i_1 < i_2 < \dots < i_k$).

mod m hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás– exponenciális lépésszám!

Írjuk fel n -et 2-es számrendszerben ($\log n$ darab osztás) $n = 2^{i_1} + 2^{i_2} + \dots + 2^{i_k}$ alakban ($i_1 < i_2 < \dots < i_k$). $a^n = a^{2^{i_1}} \cdot a^{2^{i_2}} \cdot \dots \cdot a^{2^{i_k}}$

mod m hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás– exponenciális lépésszám!

Írjuk fel n -et 2-es számrendszerben ($\log n$ darab osztás) $n = 2^{i_1} + 2^{i_2} + \dots + 2^{i_k}$ alakban ($i_1 < i_2 < \dots < i_k$). $a^n = a^{2^{i_1}} \cdot a^{2^{i_2}} \cdot \dots \cdot a^{2^{i_k}} \implies$ elég $a^{2^{i_j}}$ \pmod{m} számokat kiszámolni és összeszorozni.

mod m hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás– exponenciális lépésszám!

Írjuk fel n -et 2-es számrendszerben ($\log n$ darab osztás) $n = 2^{i_1} + 2^{i_2} + \dots + 2^{i_k}$ alakban ($i_1 < i_2 < \dots < i_k$). $a^n = a^{2^{i_1}} \cdot a^{2^{i_2}} \cdot \dots \cdot a^{2^{i_k}} \implies$ elég $a^{2^{i_j}} \pmod{m}$ számokat kiszámolni és összeszorozni. Ez megy $\log n$ darab négyzetre emeléssel és maradékos osztással.

mod m hatványozás

$a^n \pmod{m}$ eredménye 0 és $m - 1$ közé esik \implies kiírható gyorsan.

Az $\underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-szer}}$ kiszámítási mód $n - 1$ szorzás– exponenciális lépésszám!

Írjuk fel n -et 2-es számrendszerben ($\log n$ darab osztás) $n = 2^{i_1} + 2^{i_2} + \dots + 2^{i_k}$ alakban ($i_1 < i_2 < \dots < i_k$). $a^n = a^{2^{i_1}} \cdot a^{2^{i_2}} \cdot \dots \cdot a^{2^{i_k}} \implies$ elég $a^{2^{i_j}} \pmod{m}$ számokat kiszámolni és összeszorozni. Ez megy $\log n$ darab négyzetre emeléssel és maradékos osztással.

A mod m hatványozás polinomiális időben elvégezhető.

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t,

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t,

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb.

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb. Egy ilyen lépés után a megmaradtak közül a legkisebb prím, őt hagyjuk meg, de a többszöröseit húzzuk ki.

Prímtesztelés

Eratoszthenész „szita-algoritmusa”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb. Egy ilyen lépés után a megmaradtak közül a legkisebb prím, őt hagyjuk meg, de a többszöröseit húzzuk ki. Így elvileg bármilyen határig előbb-utóbb elő lehet állítani az összes prímet.

Prímtesztelés

Eratoszthenész „szita-algoritmusa”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb. Egy ilyen lépés után a megmaradtak közül a legkisebb prím, őt hagyjuk meg, de a többszöröseit húzzuk ki. Így elvileg bármilyen határig előbb-utóbb elő lehet állítani az összes prímet.

Naiv módszer: $2, 3, 4, \dots, \lfloor \sqrt{n} \rfloor$ -ig minden számról megnézni, nem osztója-e n -nek.

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb. Egy ilyen lépés után a megmaradtak közül a legkisebb prím, őt hagyjuk meg, de a többszöröseit húzzuk ki. Így elvileg bármilyen határig előbb-utóbb elő lehet állítani az összes prímet.

Naiv módszer: $2, 3, 4, \dots, \lfloor \sqrt{n} \rfloor$ -ig minden számról megnézni, nem osztója-e n -nek. Ha k osztója n -nek, akkor n/k is osztója, és $\min(k, n/k) \leq \lfloor \sqrt{n} \rfloor$.

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb. Egy ilyen lépés után a megmaradtak közül a legkisebb prím, őt hagyjuk meg, de a többszöröseit húzzuk ki. Így elvileg bármilyen határig előbb-utóbb elő lehet állítani az összes prímet.

Naiv módszer: $2, 3, 4, \dots, \lfloor \sqrt{n} \rfloor$ -ig minden számról megnézni, nem osztója-e n -nek. Ha k osztója n -nek, akkor n/k is osztója, és $\min(k, n/k) \leq \lfloor \sqrt{n} \rfloor$.

Lépésszám-igénye $\lfloor \sqrt{n} \rfloor$ -nel arányos, ami *nem* polinomiális $\log n$ -ben.

Prímtesztelés

Eratoszthenész „szita-algoritmus”: Írjuk fel az egész számokat 2-től n -ig, húzzuk ki (vagyis szitáljuk ki) a páros számokat, kivéve a 2-t, azután a maradékból a hárommal oszthatókat, kivéve a 3-t, azután az öttel oszthatókat, kivéve az 5-t stb. Egy ilyen lépés után a megmaradtak közül a legkisebb prím, őt hagyjuk meg, de a többszöröseit húzzuk ki. Így elvileg bármilyen határig előbb-utóbb elő lehet állítani az összes prímet.

Naiv módszer: $2, 3, 4, \dots, \lfloor \sqrt{n} \rfloor$ -ig minden számról megnézni, nem osztója-e n -nek. Ha k osztója n -nek, akkor n/k is osztója, és $\min(k, n/k) \leq \lfloor \sqrt{n} \rfloor$.

Lépésszám-igénye $\lfloor \sqrt{n} \rfloor$ -nel arányos, ami **nem** polinomiális $\log n$ -ben. Az eredeti Eratoszthenész–szitáról is belátható, hogy a lépésszám az input hosszának exponenciális függvénye. **Viszont, ha n nem prím, akkor megkapjuk egy osztóját!**

Polinomiális idejű prímtesztelés

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér,

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz,*

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel.

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Ha n összetett, akkor

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Ha n összetett, akkor

- $t^{n-1} \equiv 1 \pmod{n}$

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Ha n összetett, akkor

- $t^{n-1} \equiv 1 \pmod{n}$ azaz t a „cinkosa” n -nek.

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Ha n összetett, akkor

- $t^{n-1} \equiv 1 \pmod{n}$ azaz t a „cinkosa” n -nek.
- $t^{n-1} \not\equiv 1 \pmod{n}$

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Ha n összetett, akkor

- $t^{n-1} \equiv 1 \pmod{n}$ azaz t a „cinkosa” n -nek.
- $t^{n-1} \not\equiv 1 \pmod{n} \implies n$ *biztos nem* prím,

Polinomiális idejű prímtesztelés

Az algoritmus polinomrendben véget ér, de *az eredmény nem biztosan, hanem csak valószínűleg igaz*, illetve *ha összetett számnak tartja n -et, akkor általában nem találja meg egyetlen osztóját sem.*

Alapja: Euler-Fermat tétel. Ha n prím, akkor $t^{n-1} \equiv 1 \pmod{n}$ minden t -re, mely n -hez relatív prím (vagyis melyre $t \not\equiv 0 \pmod{n}$)

Ha n összetett, akkor

- $t^{n-1} \equiv 1 \pmod{n}$ azaz t a „cinkosa” n -nek.
- $t^{n-1} \not\equiv 1 \pmod{n} \implies n$ **biztos nem** prím, azaz t az n „árulója”

Vannak-e árulók?

Carmichael szám: ha nincs árulója

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS $\text{cinkos} \cdot \text{cinkos} = \text{cinkos}$.

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n}$

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.
cinkos·áruló=áruló.

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.
cinkos·áruló=áruló. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv a \pmod{n}$

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.

cinkos·áruló=áruló. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv a \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \cdot a \pmod{n}$.

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.

cinkos·áruló=áruló. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv a \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \cdot a \pmod{n}$.

Ha c_1, c_2, \dots, c_s az összes cinkos sorozata, és a egy áruló,

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.

cinkos·áruló=áruló. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv a \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \cdot a \pmod{n}$.

Ha c_1, c_2, \dots, c_s az összes cinkos sorozata, és a egy áruló, akkor az ac_1, ac_2, \dots, ac_s sorozat minden tagja áruló és mind különbözőek.

Vannak-e árulók?

Carmichael szám: ha nincs árulója kevés ilyen van.

1. Állítás. *Ha egy n számnak van árulója, akkor akkor legalább annyi az árulója, mint a cinkosa.*

BIZONYÍTÁS cinkos·cinkos=cinkos. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv 1 \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \pmod{n}$.

cinkos·áruló=áruló. $t_1^{n-1} \equiv 1 \pmod{n}$ és $t_2^{n-1} \equiv a \pmod{n} \implies (t_1 t_2)^{n-1} = t_1^{n-1} t_2^{n-1} \equiv 1 \cdot a \pmod{n}$.

Ha c_1, c_2, \dots, c_s az összes cinkos sorozata, és a egy áruló, akkor az ac_1, ac_2, \dots, ac_s sorozat minden tagja áruló és mind különbözőek. Így már $2s$ darab különböző maradékosztályt találtunk, és lehet, hogy további árulók is vannak.

Az algoritmus elve

Egymástól függetlenül véletlenszerűen választunk q darab maradékosztályt, és ha mindegyikre $m^{n-1} \equiv 1 \pmod{n}$ teljesül,

Az algoritmus elve

Egymástól függetlenül véletlenszerűen választunk q darab maradékosztályt, és ha mindegyikre $m^{n-1} \equiv 1 \pmod{n}$ teljesül, akkor n lehet ugyan összetett szám,

Az algoritmus elve

Egymástól függetlenül véletlenszerűen választunk q darab maradékosztályt, és ha mindegyikre $m^{n-1} \equiv 1 \pmod{n}$ teljesül, akkor n lehet ugyan összetett szám, de ennek a valószínűsége $(1/2)^q$ -nél kisebb.

0.: $i \leftarrow 1$

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

4.: Ha $i = 100 \rightarrow$ STOP 3

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

4.: Ha $i = 100 \rightarrow$ STOP 3

5.: $i \leftarrow i + 1$ és folytassuk az 1. lépésnél.

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

4.: Ha $i = 100 \rightarrow$ STOP 3

5.: $i \leftarrow i + 1$ és folytassuk az 1. lépésnél.

STOP 1: n összetett szám, $d(n, m)$ egy osztója

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

4.: Ha $i = 100 \rightarrow$ STOP 3

5.: $i \leftarrow i + 1$ és folytassuk az 1. lépésnél.

STOP 1: n összetett szám, $d(n, m)$ egy osztója

STOP 2: n összetett szám, de egyetlen osztóját sem találtuk meg

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

4.: Ha $i = 100 \rightarrow$ STOP 3

5.: $i \leftarrow i + 1$ és folytassuk az 1. lépésnél.

STOP 1: n összetett szám, $d(n, m)$ egy osztója

STOP 2: n összetett szám, de egyetlen osztóját sem találtuk meg

STOP 3: n valószínűleg prím (a hiba valószínűsége $< (1/2)^{100}$)

0.: $i \leftarrow 1$

1.: Válasszunk véletlenszerűen egy $1 < m < n$ számot

2.: Határozzuk meg m és n legnagyobb közös osztóját. Ha ez egynél nagyobb \rightarrow STOP 1

3.: Ha $m^{n-1} \not\equiv 1 \pmod{n} \rightarrow$ STOP 2

4.: Ha $i = 100 \rightarrow$ STOP 3

5.: $i \leftarrow i + 1$ és folytassuk az 1. lépésnél.

STOP 1: n összetett szám, $d(n, m)$ egy osztója

STOP 2: n összetett szám, de egyetlen osztóját sem találtuk meg

STOP 3: n valószínűleg prím (a hiba valószínűsége $< (1/2)^{100}$)

A 2. és 3. lépés is elvégezhető polinom időben, így az egész algoritmus lépésszám-igénye is az input hosszának polinomjával becsülhető.

Javítás – Carmichael kiküszöbölése

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk.

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2.

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Ha $n - 1 = 2^t q$ (ahol q már páratlan szám), akkor

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Ha $n - 1 = 2^t q$ (ahol q már páratlan szám), akkor

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{4}} + 1\right) \cdots \left(m^{\frac{n-1}{2^t}} + 1\right) \left(m^{\frac{n-1}{2^t}} - 1\right) \quad (1)$$

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Ha $n - 1 = 2^t q$ (ahol q már páratlan szám), akkor

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{4}} + 1\right) \cdots \left(m^{\frac{n-1}{2^t}} + 1\right) \left(m^{\frac{n-1}{2^t}} - 1\right) \quad (1)$$

Belátható, hogy a 3. lépés helyett az alábbi állhat:

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Ha $n - 1 = 2^t q$ (ahol q már páratlan szám), akkor

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{4}} + 1\right) \cdots \left(m^{\frac{n-1}{2^t}} + 1\right) \left(m^{\frac{n-1}{2^t}} - 1\right) \quad (1)$$

Belátható, hogy a 3. lépés helyett az alábbi állhat:

3': Ha az (1) jobboldalán látható $t + 1$ tényező egyike sem osztható n -nel, akkor STOP 2.

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Ha $n - 1 = 2^t q$ (ahol q már páratlan szám), akkor

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{4}} + 1\right) \cdots \left(m^{\frac{n-1}{2^t}} + 1\right) \left(m^{\frac{n-1}{2^t}} - 1\right) \quad (1)$$

Belátható, hogy a 3. lépés helyett az alábbi állhat:

3': Ha az (1) jobboldalán látható $t + 1$ tényező egyike sem osztható n -nel, akkor STOP 2.

Így nem egy, hanem $t + 1$ oszthatóságot kell ellenőriznünk, de $t = \log \frac{n-1}{q} < \log n$

Javítás – Carmichael kiküszöbölése

3. lépést megváltoztatjuk. Az algoritmus 3. lépése: ha $m^{n-1} - 1$ nem osztható n -nel, akkor STOP 2. Mivel n páratlan,

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{2}} - 1\right)$$

Ha $n - 1 = 2^t q$ (ahol q már páratlan szám), akkor

$$m^{n-1} - 1 = \left(m^{\frac{n-1}{2}} + 1\right) \left(m^{\frac{n-1}{4}} + 1\right) \cdots \left(m^{\frac{n-1}{2^t}} + 1\right) \left(m^{\frac{n-1}{2^t}} - 1\right) \quad (1)$$

Belátható, hogy a 3. lépés helyett az alábbi állhat:

3': Ha az (1) jobboldalán látható $t + 1$ tényező egyike sem osztható n -nel, akkor STOP 2.

Így nem egy, hanem $t + 1$ oszthatóságot kell ellenőriznünk, de $t = \log \frac{n-1}{q} < \log n \implies$ nem rontja el a polinomialitást.

Alkalmazás – nyilvános kulcsú titkosítás

Alkalmazás – nyilvános kulcsú titkosítás

Elképzeltető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Alkalmazás – nyilvános kulcsú titkosítás

Elképzelhető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek.

Alkalmazás – nyilvános kulcsú titkosítás

Elképzeltető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek. Annak adhatják ki az adatokat, aki n valamely osztóját megmondja.

Alkalmazás – nyilvános kulcsú titkosítás

Elképzelhető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek. Annak adhatják ki az adatokat, aki n valamely osztóját megmondja. Annak ellenőrzése, hogy az adatokért jelentkező személy által mondott k szám osztója-e n -nek (vagyis $k = p$ vagy $k = q$ teljesül-e), gyorsan elvégezhető,

Alkalmazás – nyilvános kulcsú titkosítás

Elképzelhető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek. Annak adhatják ki az adatokat, aki n valamely osztóját megmondja. Annak ellenőrzése, hogy az adatokért jelentkező személy által mondott k szám osztója-e n -nek (vagyis $k = p$ vagy $k = q$ teljesül-e), gyorsan elvégezhető, de n -ből p és q előállítása mai tudásunk szerint reménytelenül nehéz.

Alkalmazás – nyilvános kulcsú titkosítás

Elképzelhető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek. Annak adhatják ki az adatokat, aki n valamely osztóját megmondja. Annak ellenőrzése, hogy az adatokért jelentkező személy által mondott k szám osztója-e n -nek (vagyis $k = p$ vagy $k = q$ teljesül-e), gyorsan elvégezhető, de n -ből p és q előállítása mai tudásunk szerint reménytelenül nehéz.

Az n számot (a jelszónkat) nem kell titokban tartani

Alkalmazás – nyilvános kulcsú titkosítás

Elképzelhető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek. Annak adhatják ki az adatokat, aki n valamely osztóját megmondja. Annak ellenőrzése, hogy az adatokért jelentkező személy által mondott k szám osztója-e n -nek (vagyis $k = p$ vagy $k = q$ teljesül-e), gyorsan elvégezhető, de n -ből p és q előállítása mai tudásunk szerint reménytelenül nehéz.

Az n számot (a jelszónkat) nem kell titokban tartani – a konkurencia (akitől védjük az információt) éppúgy nem tud semmit sem kezdeni az n számmal, mint a számítógép programozója.

Alkalmazás – nyilvános kulcsú titkosítás

Elképzelhető-e olyan „jelszó”, amit a rendszer maga sem ismer, és mégis tudja ellenőrizni, hogy mi ismerjük-e?

Ötlet: Válasszunk ki két 200-jegyű p és q prímszámot és csak az $n = pq$ szorzatukat adjuk meg a gépnek. Annak adhatják ki az adatokat, aki n valamely osztóját megmondja. Annak ellenőrzése, hogy az adatokért jelentkező személy által mondott k szám osztója-e n -nek (vagyis $k = p$ vagy $k = q$ teljesül-e), gyorsan elvégezhető, de n -ből p és q előállítása mai tudásunk szerint reménytelenül nehéz.

Az n számot (a jelszónkat) nem kell titokban tartani – a konkurencia (akitől védjük az információt) éppúgy nem tud semmit sem kezdeni az n számmal, mint a számítógép programozója. Ha persze egyszer közöljük a számítógéppel p vagy q értékét, akkor attól kezdve nem lehetünk biztonságban.

Kódolás, dekódolás

Kódolás, dekódolás

Üzenet: számjegyek sorozata.

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.newline

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.
A kódolás egy $y = C(x)$ függvény, mely a 400-jegyű x számhoz egy másik 400-jegyű y számot rendel.

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.
A kódolás egy $y = C(x)$ függvény, mely a 400-jegyű x számhoz egy másik 400-jegyű y számot rendel. E függvény inverzét, az $x = D(y)$ függvényt dekódoló függvénynek nevezzük.

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.
A kódolás egy $y = C(x)$ függvény, mely a 400-jegyű x számhoz egy másik 400-jegyű y számot rendel. E függvény inverzét, az $x = D(y)$ függvényt dekódoló függvénynek nevezzük. Mindenki nyilvánosságra hozza a saját C kódoló függvényét, de titokban tartja a D dekódoló függvényt.

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.
A kódolás egy $y = C(x)$ függvény, mely a 400-jegyű x számhoz egy másik 400-jegyű y számot rendel. E függvény inverzét, az $x = D(y)$ függvényt dekódoló függvénynek nevezzük. Mindenki nyilvánosságra hozza a saját C kódoló függvényét, de titokban tartja a D dekódoló függvényt. Ekkor ha az i -ik személy (a feladó) el akarja küldeni az x üzenetet a j -ik személynek (a címzettnek), akkor az általa is hozzáférhető C_j kódolófüggvényt alkalmazva az $y = C_j(x)$ üzenetet küldi el.

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.
A kódolás egy $y = C(x)$ függvény, mely a 400-jegyű x számhoz egy másik 400-jegyű y számot rendel. E függvény inverzét, az $x = D(y)$ függvényt dekódoló függvénynek nevezzük. Mindenki nyilvánosságra hozza a saját C kódoló függvényét, de titokban tartja a D dekódoló függvényt. Ekkor ha az i -ik személy (a feladó) el akarja küldeni az x üzenetet a j -ik személynek (a címzettnek), akkor az általa is hozzáférhető C_j kódolófüggvényt alkalmazva az $y = C_j(x)$ üzenetet küldi el. A címzett alkalmazza a csak általa ismert D_j dekódoló függvényt és megkapja a $D_j(y) = D_j(C_j(x)) = x$ üzenetet.

Kódolás, dekódolás

Üzenet: számjegyek sorozata. Feltehetjük, hogy a titkosítandó majd továbbítandó üzenet mondjuk 400-jegyű számok sorozata.
A kódolás egy $y = C(x)$ függvény, mely a 400-jegyű x számhoz egy másik 400-jegyű y számot rendel. E függvény inverzét, az $x = D(y)$ függvényt dekódoló függvénynek nevezzük. Mindenki nyilvánosságra hozza a saját C kódoló függvényét, de titokban tartja a D dekódoló függvényt. Ekkor ha az i -ik személy (a feladó) el akarja küldeni az x üzenetet a j -ik személynek (a címzettnek), akkor az általa is hozzáférhető C_j kódolófüggvényt alkalmazva az $y = C_j(x)$ üzenetet küldi el. A címzett alkalmazza a csak általa ismert D_j dekódoló függvényt és megkapja a $D_j(y) = D_j(C_j(x)) = x$ üzenetet. A rendszerben részt vevő többi ember számára y dekódolhatatlan.

Hogy lehet olyan C_1, C_2, \dots kódoló és D_1, D_2, \dots dekódoló függvényeket készíteni, hogy bármely x -re $C_i(x)$ vagy $D_i(x)$ kiszámítása gyorsan elvégezhető legyen,

Hogy lehet olyan C_1, C_2, \dots kódoló és D_1, D_2, \dots dekódoló függvényeket készíteni, hogy bármely x -re $C_i(x)$ vagy $D_i(x)$ kiszámítása gyorsan elvégezhető legyen, *de a C_i ismeretében D_i -re ne lehessen következtetni?*

Hogy lehet olyan C_1, C_2, \dots kódoló és D_1, D_2, \dots dekódoló függvényeket készíteni, hogy bármely x -re $C_i(x)$ vagy $D_i(x)$ kiszámítása gyorsan elvégezhető legyen, *de a C_i ismeretében D_i -re ne lehessen következtetni?*

Az i -ik résztvevő választ két 200-jegyű prímszámot, p_i -t és q_i -t.

Hogy lehet olyan C_1, C_2, \dots kódoló és D_1, D_2, \dots dekódoló függvényeket készíteni, hogy bármely x -re $C_i(x)$ vagy $D_i(x)$ kiszámítása gyorsan elvégezhető legyen, *de a C_i ismeretében D_i -re ne lehessen következtetni?*

Az i -ik résztvevő választ két 200-jegyű prímszámot, p_i -t és q_i -t.
 $n_i = p_i q_i$. $\implies \varphi(n_i) = (p_i - 1)(q_i - 1)$, jelöljük ezt a mennyiséget m_i -vel.

Hogy lehet olyan C_1, C_2, \dots kódoló és D_1, D_2, \dots dekódoló függvényeket készíteni, hogy bármely x -re $C_i(x)$ vagy $D_i(x)$ kiszámítása gyorsan elvégezhető legyen, *de a C_i ismeretében D_i -re ne lehessen következtetni?*

Az i -ik résztvevő választ két 200-jegyű prímszámot, p_i -t és q_i -t.
 $n_i = p_i q_i$. $\implies \varphi(n_i) = (p_i - 1)(q_i - 1)$, jelöljük ezt a mennyiséget m_i -vel. A résztvevő ezen kívül kiválaszt egy olyan e_i számot is, melyre $1 \leq e_i \leq n_i$ és amely relatív prím $(p_i - 1)$ -hez is és $(q_i - 1)$ -hez is.

Hogy lehet olyan C_1, C_2, \dots kódoló és D_1, D_2, \dots dekódoló függvényeket készíteni, hogy bármely x -re $C_i(x)$ vagy $D_i(x)$ kiszámítása gyorsan elvégezhető legyen, *de a C_i ismeretében D_i -re ne lehessen következtetni?*

Az i -ik résztvevő választ két 200-jegyű prímszámot, p_i -t és q_i -t.
 $n_i = p_i q_i \implies \varphi(n_i) = (p_i - 1)(q_i - 1)$, jelöljük ezt a mennyiséget m_i -vel. A résztvevő ezen kívül kiválaszt egy olyan e_i számot is, melyre $1 \leq e_i \leq n_i$ és amely relatív prím $(p_i - 1)$ -hez is és $(q_i - 1)$ -hez is. Végül megoldva egy kongruenciát meghatározza azt a d_i számot, melyre $e_i d_i \equiv 1 \pmod{m_i}$.

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$.

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i}$$

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i} = x^{hm_i+1} =$$

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i} = x^{hm_i+1} = \left[x^{\varphi(n_i)} \right]^h \cdot x$$

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i} = x^{hm_i+1} = \left[x^{\varphi(n_i)} \right]^h \cdot x \equiv x \pmod{n_i}.$$

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i} = x^{hm_i+1} = \left[x^{\varphi(n_i)} \right]^h \cdot x \equiv x \pmod{n_i}.$$

Mindez csak akkor működik, ha x relatív prím n_i -hez.

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i} = x^{h m_i + 1} = \left[x^{\varphi(n_i)} \right]^h \cdot x \equiv x \pmod{n_i}.$$

Mindez csak akkor működik, ha x relatív prím n_i -hez. \implies Az üzenetet nem 400, hanem 399 jegyű számsorozatokra bontjuk, majd mindegyik sorozat utolsó elemét úgy választjuk meg, hogy e feltétel teljesüljön.

Az i -ik résztvevő nyilvánosságra hozza az n_i és e_i számokat, viszont titokban tartja a p_i, q_i, m_i és d_i számokat.

A C_i kódolófüggvény egy x üzenethez hozzárendeli azt az $y = C_i(x)$ számot, melyre

$$y \equiv x^{e_i} \pmod{n_i},$$

míg a D_i dekódolófüggvény az y -hoz annak d_i -ik hatványát rendeli $\pmod{n_i}$. Így

$$y^{d_i} \equiv x^{e_i d_i} = x^{hm_i+1} = \left[x^{\varphi(n_i)} \right]^h \cdot x \equiv x \pmod{n_i}.$$

Mindez csak akkor működik, ha x relatív prím n_i -hez. \implies Az üzenetet nem 400, hanem 399 jegyű számsorozatokra bontjuk, majd mindegyik sorozat utolsó elemét úgy választjuk meg, hogy e feltétel teljesüljön.

Dekódolás után egyszerűen elhagyjuk az utolsó számjegyet.

Valódi-e a feladó?

Valódi-e a feladó?

A klasszikus titkosírások gyenge pontjai:

Valódi-e a feladó?

A klasszikus titkosírások gyenge pontjai:

Ha két személy kódolt üzeneteket akart váltani egymással, akkor először meg kellett állapodniuk egymással a kódban – ez most nem kell

Valódi-e a feladó?

A klasszikus titkosírások gyenge pontjai:

Ha két személy kódolt üzeneteket akart váltani egymással, akkor először meg kellett állapotodniuk egymással a kódban – ez most nem kell

Ha t személy akar így levelezni, akkor nem kell $\binom{t}{2}$ féle titkos kódot kitalálni, és mégis bármely üzenet rejtve marad a többi $t - 2$ résztvevő előtt.

Valódi-e a feladó?

A klasszikus titkosírások gyenge pontjai:

Ha két személy kódolt üzeneteket akart váltani egymással, akkor először meg kellett állapotodniuk egymással a kódban – ez most nem kell

Ha t személy akar így levelezni, akkor nem kell $\binom{t}{2}$ féle titkos kódot kitalálni, és mégis bármely üzenet rejtve marad a többi $t - 2$ résztvevő előtt. a címzett soha nem tudhatta, hogy tényleg a feladó írt-e neki, vagy „az ellenség” kezébe került a kód, és hamisítványt kap.

Valódi-e a feladó?

A klasszikus titkosírások gyenge pontjai:

Ha két személy kódolt üzeneteket akart váltani egymással, akkor először meg kellett állapotodniuk egymással a kódban – ez most nem kell

Ha t személy akar így levelezni, akkor nem kell $\binom{t}{2}$ féle titkos kódot kitalálni, és mégis bármely üzenet rejtve marad a többi $t - 2$ résztvevő előtt. a címzett soha nem tudhatta, hogy tényleg a feladó írt-e neki, vagy „az ellenség” kezébe került a kód, és hamisítványt kap. Az új módszernek látszólag ugyanez a hátránya (hisz C_j -hez mindenki hozzáfér, nem csak az i -ik résztvevő).

Az i -ik résztvevő ne x -re, hanem $z = D_i(x)$ -re alkalmazza a $w = C_j(z)$ kódolást, majd ezt a w „üzenetet” küldi el.

Az i -ik résztvevő ne x -re, hanem $z = D_i(x)$ -re alkalmazza a $w = C_j(z)$ kódolást, majd ezt a w „üzenetet” küldi el. A címzett (tehát a j -ik résztvevő) először a csak általa ismert D_j , majd a nyilvánosság számára hozzáférhető C_i függvényt alkalmazza,

Az i -ik résztvevő ne x -re, hanem $z = D_i(x)$ -re alkalmazza a $w = C_j(z)$ kódolást, majd ezt a w „üzenetet” küldi el. A címzett (tehát a j -ik résztvevő) először a csak általa ismert D_j , majd a nyilvánosság számára hozzáférhető C_i függvényt alkalmazza,

$$C_i(D_j(w)) = C_i(D_j(C_j(z))) = C_i(z) = C_i(D_i(x)) = x.$$

Az i -ik résztvevő ne x -re, hanem $z = D_i(x)$ -re alkalmazza a $w = C_j(z)$ kódolást, majd ezt a w „üzenetet” küldi el. A címzett (tehát a j -ik résztvevő) először a csak általa ismert D_j , majd a nyilvánosság számára hozzáférhető C_i függvényt alkalmazza,

$$C_i(D_j(w)) = C_i(D_j(C_j(z))) = C_i(z) = C_i(D_i(x)) = x.$$

Így az üzenetet csak j tudja elolvasni, és biztos lehet benne, hogy csak i küldhette.

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést,

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést, tehát a kapott w üzenetet is, meg annak x jelentését is be kell mutatnia,

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést, tehát a kapott w üzenetet is, meg annak x jelentését is be kell mutatnia, de a bírónak sem akarja megmondani a saját D_j dekódoló eljárását és nyilván nem kényszerítheti az „ellenérdekelt” i -t saját D_i eljárásának felfedésére.

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést, tehát a kapott w üzenetet is, meg annak x jelentését is be kell mutatnia, de a bírónak sem akarja megmondani a saját D_j dekódoló eljárását és nyilván nem kényszerítheti az „ellenérdekelt” i -t saját D_i eljárásának felfedésére. A pert kezdeményező j nem csak w -t, hanem az $u = D_j(w)$ „félleg dekódolt” üzenetet is bemutatja a „bírónak”.

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést, tehát a kapott w üzenetet is, meg annak x jelentését is be kell mutatnia, de a bírónak sem akarja megmondani a saját D_j dekódoló eljárását és nyilván nem kényszerítheti az „ellenérdekelt” i -t saját D_i eljárásának felfedésére. A pert kezdeményező j nem csak w -t, hanem az $u = D_j(w)$ „félíg dekódolt” üzenetet is bemutatja a „bírónak”. A „bíró” kizárólag a nyilvánosság számára is hozzáférhető C_i , C_j kódolási eljárások segítségével ellenőrizheti, hogy

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést, tehát a kapott w üzenetet is, meg annak x jelentését is be kell mutatnia, de a bírónak sem akarja megmondani a saját D_j dekódoló eljárását és nyilván nem kényszerítheti az „ellenérdekelte” i -t saját D_i eljárásának felfedezésére. A pert kezdeményező j nem csak w -t, hanem az $u = D_j(w)$ „félleg dekódolt” üzenetet is bemutatja a „bírónak”. A „bíró” kizárólag a nyilvánosság számára is hozzáférhető C_i, C_j kódolási eljárások segítségével ellenőrizheti, hogy (1) $w = C_j(u)$, tehát tényleg j -nek jött az üzenet, és hogy

Bíróságon

Előfordulhat, hogy i rendel valamit j -től, majd nem fizet, így j -nek „bíróság” elé kell vinnie az ügyet: Be akarja bizonyítani, hogy i feladta a rendelést, tehát a kapott w üzenetet is, meg annak x jelentését is be kell mutatnia, de a bírónak sem akarja megmondani a saját D_j dekódoló eljárását és nyilván nem kényszerítheti az „ellenérdekelt” i -t saját D_i eljárásának felfedésére. A pert kezdeményező j nem csak w -t, hanem az $u = D_j(w)$ „félig dekódolt” üzenetet is bemutatja a „bírónak”. A „bíró” kizárólag a nyilvánosság számára is hozzáférhető C_i, C_j kódolási eljárások segítségével ellenőrizheti, hogy (1) $w = C_j(u)$, tehát tényleg j -nek jött az üzenet, és hogy (2) $x = C_i(u)$, tehát tényleg i -től jött az üzenet.