

# Rekurzió

Ionescu Klára

[clara@cs.ubbcluj.ro](mailto:clara@cs.ubbcluj.ro)

# 1. Mi a hatása?

**Algoritmus miEz(a):**

**Ha  $a = 0$  akkor  
térítsd a**

**különben**

**$c = a \bmod 10$**

**Ha  $c \bmod 2 \neq 0$  akkor**

**térítsd  $c + 10 * \text{miEz}(a \text{ DIV } 10)$**

**különben**

**térítsd  $\text{miEz}(a \text{ DIV } 10)$**

**vége(ha)**

**vége(ha)**

**Vége(algoritmus)**

Téríti az a szám páratlan számjegyeiből felépített számot.

# Hogyan keressük a választ?

- Megszámoljuk a változókat: **a**, **c** és **amit térít**
- Készítünk ellenőrző táblázatot:

a	c	Amit térít	A rekurzióból visszatérve
<b>12345</b>	<b>5</b> (páratlan)	<b><math>5 + 10 * \text{miEz}(1234)</math></b>	<b><math>5 + 10 * 13 = 135</math></b>
<b>1234</b>	<b>4</b> (páros)	<b><math>\text{miEz}(123)</math></b>	<b>13</b>
<b>123</b>	<b>3</b> (páratlan)	<b><math>3 + 10 * \text{miEz}(12)</math></b>	<b><math>3 + 10 * 1 = 13</math></b>
<b>12</b>	<b>2</b> (páros)	<b><math>\text{miEz}(1)</math></b>	<b>1</b>
<b>1</b>	<b>1</b> (páratlan)	<b><math>1 + 10 * \text{miEz}(0)</math></b>	<b><math>1 + 10 * 0 = 1</math></b>
<b>0</b>		<b>0</b>	<b>Lentről felfele ↑</b>

# Az algoritmus iteratív változata

**Algoritmus miEzIt(a):**

// 1. Kezdőérték b-nek és 10 hatványainak

// 2. A Ha utasításból Amíg lesz

// 3. Térítés helyett b-nek adunk értéket

    b = 0; p = 1

**Amíg** a > 0 **végezd el:**

        c = a MOD 10

**Ha** c MOD 2 ≠ 0 **akkor**

            b = b + p \* c

            p = p \* 10

**vége(ha)**

        a = a DIV 10

**vége(amíg)**

**térítsd b**

**Vége(algoritmus)**

## 2. Mi a hatása?

```
Algoritmus F_It(a, b):  
// iteratív algoritmus  
    c = 1  
    Amíg b > 0 végezd el:  
        Ha b MOD 2 = 1 akkor  
            c = c * a MOD 10  
        vége(ha)  
        a = a * a MOD 10  
        b = b DIV 2  
    vége(amíg)  
    térítsd c  
Vége(algoritmus)
```

# Mi a hatása?

```
Algoritmus F_It(a, b):  
  // írunk megjegyzéseket  
  c = 1 // kezdőérték (a c-t fogjuk téríteni)  
  Amíg b > 0 végezd el:  
    Ha b MOD 2 = 1 akkor // ha b páratlan  
      c = c * a MOD 10  
      // c régi értékét megszorozzuk az a utolsó számjegyével  
    vége(ha)  
    a = a * a MOD 10  
    // a új értéke: megszoroztuk az utolsó számjegyével  
    b = b DIV 2 // b-t felezzük  
  vége(amíg)  
  térítsd c  
Vége(algoritmus) // mihez hasonlít? Mire gondolunk?
```

# Hogyan keressük a választ?

- Megszámoljuk a változókat: **a**, **c** és **amit térít**
- Készítünk ellenőrző táblázatot:

a	b	c
<b>2</b>	<b>4</b>	<b>1</b>
<b><math>2 * 2 \% 10 = 4</math></b>	<b><math>4 / 2 = 2</math> (páros)</b>	<b>1</b>
<b><math>4 * 4 \% 10 = 16</math></b>	<b><math>2 / 2 = 1</math> (páratlan)</b>	<b><math>1 * (16 \% 10) = 1 * 6 = 6</math></b>
	<b>0</b>	

Meghatározza és téríti az  $a^b$  utolsó számjegyét.

# Rekurzív változat

**Algoritmus F\_Rek(a, b):**

Ha  $b = 0$  akkor

térítsd 1

különben

Ha  $b \bmod 2 = 1$  akkor

térítsd  $a \bmod 10 * F\_Rek(a * a \bmod 10, b \operatorname{DIV} 2) \bmod 10$

különben

térítsd  $F\_Rek(a * a \bmod 10, b \operatorname{DIV} 2) \bmod 10$

vége(ha)

vége(ha)

Vége(algoritmus)



# Mat-Info Verseny és Felvételi feladatok

## 1. Ackermann

- Legyenek az  $m$  és  $n$  természetes számok ( $0 \leq m \leq 10, 0 \leq n \leq 10$ ), valamint az  $Ack(m, n)$  algoritmus, amely kiszámítja az Ackermann-függvény értékét  $m$  és  $n$  esetében.
- Állapítsátok meg, hányszor hívja meg önmagát az  $Ack(m, n)$  algoritmus a következő utasítások végrehajtásának következtében.

$m = 1$ $n = 2$ $Ack(m, n)$	A. 7-szer B. 10-szer C. 5-ször D. ugyanannyiszor, mint a következő utasítások végrehajtásának következtében:	$m = 1$ $n = 3$ $Ack(m, n)$
-----------------------------------	-----------------------------------------------------------------------------------------------------------------------	-----------------------------------

**Algoritmus Ack(m, n):**

**Ha  $m = 0$  akkor**

**térítsd  $n + 1$**

**különben**

**Ha  $m > 0$  és  $n = 0$  akkor**

**térítsd Ack(m - 1, 1)**

**különben**

**térítsd Ack(m - 1, Ack(m, n - 1))**

**vége(ha)**

**vége(ha)**

**Vége(algoritmus)**

Követnünk kell a paraméterek értékeit a hívássorozaton belül, – esetleg lerajzoljuk a végrehajtási verem tartalmát és megszámloljuk a rekurzív hívásokat:

**Ack(1, 2)  $\Rightarrow$  (1. hívás):**

**Ack(0, Ack(1, 1))  $\Rightarrow$  (2. hívás):**

**Ack(1, Ack(1, 0))  $\Rightarrow$  (3. hívás):**

**Ack(0, 1)  $\Rightarrow$  (4. hívás):**

**Ack(0, 2)  $\Rightarrow$  (5. hívás)**

**Ack(0, 3): vége.**

# Megoldás

- Tehát, a rekurzív hívások száma 5, vagyis a **C.** válasz jó
- Így  $\Rightarrow$  **A.** és **B.** válaszok nem jók.
- Az **A.** esetben leírt módon megvizsgáljuk a **D.** esetet is
- $\Rightarrow$  a rekurzív hívások száma 7  $\Rightarrow$  a **D.** válasz sem jó

## 2. Mely értékek szükségesek?

Legyen a **különbség(a, n)** algoritmus, ahol **a** egy **n** elemű ( $0 < n < 100$ ) sorozat, amely egész számokat tárol:

**Algoritmus különbség(a, n)**

Ha  $n = 0$  akkor

térítsd 0

vége(ha)

Ha  $|a[n]| \bmod 2 = 0$  akkor

térítsd  $\text{különbség}(a, n - 1) + a[n]$

különben

térítsd  $\text{különbség}(a, n - 1) - a[n]$

vége(ha)

**Vége(algoritmus)**

Az  $n$  és  $a$  mely értékeire térít a fenti algoritmus  $0$ -át?

A.  $n = 4$  és  $a = (6, 4, 5, 5)$

B.  $n = 4$  és  $a = (-6, 5, 4, -7)$

C.  $n = 8$  és  $a = (-6, 5, -1, -4, 1, 4, -7, 6)$

D.  $n = 8$  és  $a = (-6, -3, 0, 1, 2, 3, -1, 4)$

### Megoldás

- az aktuális összeget nem tároljunk egy változóban
- az utolsó híváskor  $0$  kezdőértéket térítünk
- minden rekurzív hívásból való visszatéréskor az aktuális összeghez hozzáadjuk a soron következő számot, ha az páros,
- illetve kivonjuk, ha páratlan.
- Azokat a bemeneti adatokat választjuk ki, amelyeknek esetében a páros számok összege egyenlő a páratlanok összegével. Helyes válaszok: **A.**, **B.** és **D.**

### 3. Kiegészítés

Legyen a **kizárPáratlan( $n$ )** algoritmus, ahol  $n$  ( $1 \leq n \leq 100\,000$ ) természetes szám. Állapítsátok meg, melyik utasítást kellene a „...” helyére írni, ahhoz, hogy az algoritmus zárja ki az  $n$  számból a páratlan értékű számjegyeket.

**Algoritmus** **kizárPáratlan( $n$ )**

Ha  $n = 0$  akkor

térítsd 0

vége(ha)

Ha  $n \bmod 2 = 1$  akkor

térítsd **kizárPáratlan( $n \text{ DIV } 10$ )**

vége(ha)

térítsd ...

**Vége(algoritmus)**

- A.  $\text{kizárPáratlan}(n \bmod 10) * 10 + n \text{ DIV } 10 //$  nem jó
- B.  $\text{kizárPáratlan}(n) * 10 + n \bmod 10 //$  nem jó
- C.  $\text{kizárPáratlan}(n \text{ DIV } 10) * 10 + n \bmod 10 //$  Horner séma
- D.  $\text{kizárPáratlan}((n \text{ DIV } 10) \bmod 10) * 10 //$  nem jó

## Megoldás

- Egy új számot kell felépítenünk az adott  $n$  szám páros számjegyeiből.
- Amíg  $n$  páratlan szám, megtörténik a rekurzív hívás, anélkül, hogy módosítanánk az új számnak megfelelő térítendő értéket.
- A páros számjegyek a **C.** válaszban leírt utasítás eredményeként épülnek majd a térítendő értékbe.
- Az eddig kiszámolt értéket szorozzuk **10**-zel és hozzáadjuk a páros számjegy értékét. Az  $n$  paraméter új értéke  $n / 10$ .
- Tehát a helyes válasz: **C.**

## 4. Számjegyszorzat

- A **számJegyek(*n*, *d*)** algoritmus (*n* és *d* természetes számok,  $10 \leq n \leq 100\,000$ ,  $1 \leq d \leq 9$ ), meghatározza és visszatéríti azt a legkisebb természetes számot, amelynek *d*-nél kisebb vagy *d*-vel egyenlő, nem nulla számjegyei vannak, és amely számjegyeknek a szorzata egyenlő *n*-nel.
- Például, ha *n* = 108 és *d* = 9, az algoritmus 269-et térít vissza. Ha ilyen szám nem létezik, az algoritmus -1-et térít.
- Állapítsátok meg, hányszor hívja meg önmagát a **számJegyek(*n*, *d*)** algoritmus az alábbi programrészlet végrehajtásának következtében:



**Algoritmus számJegyek(n, d)**

**Ha  $d = 1$  akkor**

**Ha  $n = 1$  akkor térítsd 0**

**különben térítsd -1**

**vége(ha)**

**különben**

**Ha  $n \text{ MOD } d = 0$  akkor**

**érték  $\leftarrow$  számJegyek( $n \text{ DIV } d$ ,  $d$ )**

**Ha érték  $< 0$  akkor térítsd -1**

**különben térítsd érték \* 10 + d**

**vége(ha)**

**különben térítsd számJegyek( $n$ ,  $d - 1$ )**

**vége(ha)**

**vége(ha)**

**Vége(algoritmus)**

**beOlvas n**

**érték = számJegyek( $n$ , 9)**

**A. Ha  $n = 108$ , az algoritmus 11-szer hívja meg önmagát.**

**B. Ha  $n = 109$ , az algoritmus 8-szor hívja meg önmagát.**

**C. Ha  $n = 13$ , az algoritmus egyszer sem hívja meg önmagát.**

**D. Ha  $n = 100$ , az algoritmus 10-szer hívja meg önmagát.**

# Megoldás

számjegyek(108, 9)  $\Rightarrow$  (0. hívás):  
számjegyek(12, 9)  $\Rightarrow$  (1. hívás):  
számjegyek(12, 8)  $\Rightarrow$  (2. hívás):  
számjegyek(12, 7)  $\Rightarrow$  (3. hívás):  
számjegyek(12, 6)  $\Rightarrow$  (4. hívás):  
számjegyek(2, 6)  $\Rightarrow$  (5. hívás):  
számjegyek(2, 5)  $\Rightarrow$  (6. hívás):  
számjegyek(2, 4)  $\Rightarrow$  (7. hívás):  
számjegyek(2, 3)  $\Rightarrow$  (8. hívás):  
számjegyek(2, 2)  $\Rightarrow$  (9. hívás):  
számjegyek(1, 2)  $\Rightarrow$  (10. hívás):  
számjegyek(1, 1)  $\Rightarrow$  (11. hívás)

- A hívások száma **11**, tehát **A.** helyes.
- A **B.** esetben, **8** rekurzív hívás után áll le a hívások sorozata. Így a **B.** válasz is helyes.
- A **C.** válasz nem helyes, mivel, ha  **$n = 13$** , az algoritmusnak „nincs oka”, hogy egyszer se hívja meg önmagát.
- A **D.** esetben a rekurzív hívások száma **8**, tehát **D.** sem helyes.

## 5. Varázslat

- Egy számjegymágus olyan varázslatot végez, amelynek eredményeképpen egy  $x$  természetes szám ( $100 < x < 1\,000\,000$ , amelynek a 10-es számrendszerben van legkevesebb két 0-tól különböző számjegye) szétválik két pozitív természetes számra: a *bal* és *jobb* számokra, amelyek egymás után ragasztva megadják az  $x$  számot. Ugyanakkor a *bal* és *jobb* számok szorzata a lehető legnagyobb. Például, ha  $x = 1092$ , a varázslat szétválasztja a *bal* = 10 és *jobb* = 92 számokra.
- Az adott algoritmusok közül melyik alkalmazza a varázslatot az  $x$  természetes számra, amelynek 10-es számrendszerben van legkevesebb két 0-tól különböző számjegye ( $100 \leq x \leq 1\,000\,000$ )? Az algoritmus meghatározza a  $z$  természetes számban ( $0 \leq z \leq 1\,000\,000$ ) az  $x$  szám *jobb* részét. A következő algoritmusok léteznek:

# Varázslat

- **hatvány(*b*, *p*)** – meghatározza a  $b^p$  értéket (*b* a *p*. hatványon), *b*, *p* – természetes számok ( $1 \leq b \leq 20$ ,  $1 \leq p \leq 20$ );
- **szjSzáma(*sz*)** – meghatározza az *sz* szám ( $0 \leq sz \leq 1\,000\,000$ ) számjegyeinek darabszámát;

**A.**

**Algoritmus** varázslat( $x$ ,  $z$ )

maxSzorzat  $\leftarrow$  -1

eredmény  $\leftarrow$  0

**Amíg**  $x > 0$  **végezd el**

$z \leftarrow (x \bmod 10) * \text{hatvány}(10, \text{szjSzáma}(z)) + z$

$x \leftarrow x \text{ DIV } 10$

**Ha**  $x * z > \text{maxSzorzat}$  **akkor**

maxSzorzat  $\leftarrow x * z$

eredmény  $\leftarrow z$

**vége(ha)**

**vége(amíg)**

térítsd maxSzorzat

**Vége(algoritmus)**

## B.

**Algoritmus** varázslat(x, z)

    t  $\leftarrow$  0

    Ha x > 0 akkor

        y  $\leftarrow$  (x MOD 10) \* hatvány(10, szjSzáma(z)) + z

        t  $\leftarrow$  x DIV 10

        Ha x \* z < y \* t akkor

            térítsd varázslat(y, t)

        különben

            térítsd t

        vége(ha)

    különben

        térítsd t

    vége(ha)

**Vége(algoritmus)**

C.

**Algoritmus** varázslat(x, z)

maxSzorzat  $\leftarrow$  -1

eredmény  $\leftarrow$  0

**Amíg**  $x > 0$  **végezd el**

$z \leftarrow (x \bmod 10) * \text{hatvány}(10, \text{szjSzáma}(z)) + z$

$x \leftarrow x \text{ DIV } 10$

**Ha**  $x * z > \text{maxSzorzat}$  **akkor**

$\text{maxSzorzat} \leftarrow x * z$

        eredmény  $\leftarrow z$

**vége(ha)**

**vége(amíg)**

térítsd eredmény

**Vége(algoritmus)**

**D.**

**Algoritmus** varázslat( $x$ ,  $z$ )

Ha  $x > 0$  akkor

$y \leftarrow (x \bmod 10) * \text{hatvány}(10, \text{szjSzáma}(z)) + z$

$t \leftarrow x \text{ DIV } 10$

Ha  $x * z < y * t$  akkor

térítsd varázslat( $y$ ,  $t$ )

különben

térítsd  $z$

vége(ha)

különben

térítsd  $z$

vége(ha)

**Vége(algoritmus)**



# Megoldás

- Az **A.** algoritmus inicializálja az **eredmény** változó értékét, de sehol nem használja fel.
- Ez az észrevétel arra ösztönöz, hogy keressünk a változatok között egy másikat, amely szintén iteratív és nagyvonalakban hasonlít az **A.** algoritmushoz, de nem tartalmazza az előbb említett hibát, vagy ehhez hasonlót.
- Ez a **C.** algoritmus, amely nem a **maxSzorzat** változó értékét téríti, hanem az **eredmény** változóét. A két algoritmus közül egyértelmű, hogy az **A.** nem helyes, mivel a feladat nem a maximális szorzatot kéri, hanem az **x** szám **jobb** részét.
- A **C.** algoritmus a **z** változóban építi a szám **jobb** részét **x** számjegyeiből jobbról balra haladva, és minden lépésben aktualizálja, ha szükséges, a **maxSzorzat** értékét is.
- Ha ez megtörtént, megjegyzi az **eredmény** változóban azt a **z** értéket (a szám **jobb** részét), amely nagyobb **maxSzorzat** értéket eredményez. Végül a **maxSzorzat** legnagyobb értékéhez „tartozó” **eredmény** változó értékét téríti. Tehát a **C.** algoritmus helyes.

# Megoldás

- A **B.** algoritmus két paramétere, hasonlóan a **C.** algoritmus-hoz: **x** és **z**, ahol **x**-ben az **x** szám **bal** részét, **z**-ben a **jobb** részét szeretné felépíteni az algoritmus.
- Ez az algoritmus a **t** változó értékét téríti, vagy meghívja ön-magát az **y** és **t** aktuális paraméterekkel. De **t** az **x** változónak a **bal** része, hiszen 10-zel való osztások eredménye. Ebből következik, hogy a **varázslat(y, t)** rekurzív hívás helyett **varázslat(t, y)** lett volna a helyes, ezáltal **x** bal része **t**-től kapna értéket, jobb része **y**-től. A **térítsd t** utasítás is hibás, hiszen a feladat a szám **jobb** részét kéri, de **t** a **bal** része.
- Összehasonlítjuk a **B.** rekurzív megoldást a **D.**, szintén rekurzív megoldással és azonnal látjuk, hogy ugyanaz a gond a rekurzív hívás aktuális paramétereinek sorrendjével: **varázslat(y, t)** helyett **varázslat(t, y)** lett volna a helyes sorrend.
- Tehát csak egy helyes megoldást találtunk, a **C.**-t.