**BBTE – Matematika-Informatika Kar, tavaszi Verseny 2022**

**1.** Legyen a bűvös(x) algoritmus, ahol ***x*** természetes szám (1 ≤ ***x*** ≤ 32000):

**Algorithm** bűvös(x):

    bal ← 1

    jobb ← x

**While** bal ≤ jobb **execute**

        közép ← (bal + jobb) **DIV** 2

**If** közép \* közép = x **then**

**return** ***True***

**EndIf**

**If** közép \* közép < x **then**

            bal ← közép + 1

**else**

            jobb ← közép – 1

**EndIf**

**EndWhile**

**return** ***False***

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Az ***x*** bármely 10-nél szigorúan kisebb értékére az algoritmus *False*-t térít vissza.
2. Az algoritmus törzstényezőkre bontja az ***x*** számot.
3. Az algoritmus *True*-t térít vissza, ha az ***x*** szám négyzetszám.
4. Az algoritmus nem térít vissza *True*-t az ***x*** egyetlen megengedett értékére sem.

**Megoldás**

[Hogyan kezdünk a megoldáshoz.docx](Hogyan%20kezdünk%20a%20megoldáshoz.docx)

* A fenti példa esetében az **ACD** állítások a visszatérített logikai értékre hivatkoznak, miközben az algoritmus – valóban – *True*-t vagy *False*-t térít.
* A **B** állítás törzstényezőkről beszél, így ezt máris „eldobjuk”.
* Most már elemezhetjük az algoritmust. Mintha hasonlítana a bináris keresés algoritmusára...
* De itt nincs sorozat, ahol kereshetnénk egy adott számot…
* A ***közép*** értékének kiszámítása után az algoritmus akkor térít *True*-t, ha ***közép*** \* ***közép*** = ***x***.
* Máris gyanakszunk, hogy itt azt vizsgáljuk, hogy ***x*** négyzetszám-e?
* Tovább vizsgálódunk, amíg a meggyőződünk róla, hogy a feltételezésünk helyes.
* Tehát a **C** **változat helyes**.
* Mivel a **D** ellentmond **C**-nek (nem térít vissza *True*-t…), a **D**-t is elvetjük.
* Azt könnyű belátni, hogy az algoritmusnak nincs oka *False*-t téríteni vissza, ha ***x*** 10-nél szigorúan kisebb (**A**).

Zárójelben: eldönteni, hogy ***x*** négyzetszám-e, lehetett volna rendkívül egyszerűen: összehasonlítjuk ***x*** négyzetgyökét a négyzetgyök egész részével, de így (lehet), hogy nő az algoritmus lépéseinek száma…

**2.** Legyen a számol(a, b) algoritmus, ahol ***a*** és ***b*** természetes számok (1 ≤ ***a***, ***b*** ≤ 10000):

**Algorithm** számol(a, b):

    x ← 1

**For** i ← 1, b **execute**

        x ← (x **MOD** 10) \* a

**EndFor**

**return** x

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Ha ***a*** = 107 és ***b*** = 101, a visszatérített érték 107.
2. Ha ***a*** = 1001 és ***b*** = 101, a visszatérített érték 1001.
3. Az algoritmus minden 1 ≤ ***a*** ≤ 10000 és ***b*** = 101 értékre az ***a*** értékét téríti vissza.
4. Az algoritmus minden ***a*** = 1001 és 1 ≤ ***b*** ≤ 10000 értékre 1001-et térít vissza.

**Megoldás**

Számadatokkal dolgozunk, az állítás is egy számról szól, ami az algoritmus eredménye.

Vesszük rendre az állításokat és elkészítjük az első ellenőrző táblázatot.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***a*** | ***b*** | ***x*** | ***i*** | * Amikor ***i*** = 6 vagy 7, leállunk, hiszen ***i*** 101-ig nő; * Kiszámítjuk, hogy a (kékkel jelzett) „periódus” hányszor ismétlődik: 101 **DIV** 4 = 25. * De minket a 101 **MOD** 4 kifejezés értéke érdekel. * Mivel ez **1**, következik, hogy amikor ***i*** = 101, ***x*** értéke 107. * Tehát **A** **helyes**. |
| 107 | 101 | 1 | 1 |
|  |  | 1 \* 107 | 2 |
|  |  | 7 \* 107 | 3 |
|  |  | 9 \* 107 | 4 |
|  |  | 3 \* 107 | 5 |
|  |  | 1 \* 107 | 6 |

A **B** állítás szerint: ha ***a*** = 1001 és ***b*** = 101, az algoritmus 1001-et térít vissza.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***a*** | ***b*** | ***x*** | ***i*** | * Látható, hogy minden lépésben ***x*** értéke 1001; * Igy a visszatérített érték is 1001. * Tehát **B** **is helyes**. |
| 1001 | 101 | 1 | 1 |
|  |  | 1 \* 1001 | 2 |
|  |  | 1 \* 1001 | 3 |
|  |  | … | 4 |

A **C** feltételezés szerint, az algoritmus minden lehetséges 1 ≤ ***a*** ≤ 104 és ***b*** = 101 értékpár esetében az ***a*** értékét téríti vissza. Legyen ***a*** = 345:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***a*** | ***b*** | ***x*** | ***i*** | * Az előző két állításból kiindulva, gondolhatnánk, hogy ez a feltételezés is helyes, hiszen az **A** és **B** esetekben is ***a*** értékét térítette az algoritmus. * De itt most egy általánosabb esetet tárgyalunk, hiszen ***a*** értéke tetszőleges. * Itt a visszatérített érték 5 \* 345 = 1725, vagyis nem ***a*** értéke. * Tehát **C** nem helyes. |
| 345 | 101 | 1 | 1 |
|  |  | 1 \* 345 | 2 |
|  |  | 5 \* 345 | 3 |
|  |  | 5 \* 345 | 4 |
|  |  | … | 5 |

A **D** állítás szerint, ha ***a*** = 1001 és ***b*** tetszőleges (1 ≤ ***b*** ≤ 104), az algoritmus 1001-et térít vissza.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***a*** | ***b*** | ***x*** | ***i*** | * Legyen ***b*** = 345. Hasonlóan, mint a **B** esetben, ***x*** értéke minden lépésben 1001. Észrevesszük, hogy ***b*** értéke nincs hatással a visszatérített értékre. * Tehát **D is helyes**. |
| 1001 | 345 | 1 | 1 |
|  |  | 1 \* 1001 | 2 |
|  |  | 1 \* 1001 | 3 |
|  |  | 1 \* 1001 | 4 |
|  |  | … | 5 |

**4**. Legyen a keres(n) algoritmus, ahol ***n*** és ***b*** természetes számok (0 ≤ ***n*** ≤ 106, 2 ≤ ***b*** < 10):

**Algorithm** keres(n, b):

    v ← 0

**If** n = 0 **then**

**return** 1

**else**

        m ← n

**While** m > 0 **execute**

**If** m **MOD** b = 0 **then**

                v ← v + 1

**EndIf**

            m ← m **DIV** b

**EndWhile**

**return** v

  **EndIf**

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Az algoritmus meghatározza és visszatéríti az ***n*** szám számjegyeinek darabszámát.
2. Az algoritmus 1-et térít vissza, ha az ***n*** szám ***b***-nek valamely hatványa és 0-át különben.
3. Az algoritmus meghatározza és visszatéríti az ***n*** szám ***b*** számrendszerbeli ábrázolásában található 0 számjegyek darabszámát.
4. Az algoritmus 1-et térít vissza, ha az ***n*** szám utolsó számjegye egyenlő ***b***-vel és 0-át különben.

**Megoldás**

* A négy feltételezett hatás két csoportra osztható:
  + **B** és **D** úgy néz ki, mintha tulajdonképpen *True*-t és *False*-t térítene vissza az algoritmus,
  + **A** és **C** valamilyen darabszám visszatérítését említi.
* Megnézzük az algoritmust, és látjuk, hogy ***v***-t téríti vissza, és hogy ez a ***v*** egy **számláló**.
  + Így **B**-t és **D**-t elvetjük.
* Viszonylag hamar azt is belátjuk, hogy nem minden számjegyet számlál a ***v***,
* csak azokat a nullás számjegyeket, amelyek az ***n*** szám ***b*** számrendszerbeli ábrázolásában találhatók.
* Tehát **C helyes**.

**8.** Az alábbialgoritmusnak bemeneti paraméterei az ***n*** egész szám (1 ≤ ***n*** ≤ 10000) és az ***n*** elemű ***v*** sorozat, amelynek elemei természetes számok (***v***[1], ***v***[2], …, ***v***[***n***]).

**Algorithm** fn(v, n):

    a ← 0

**For** i ← 1, n **execute**

        ok ← ***True***

        b ← v[i]

**While** (b ≠ 0) **AND** (ok = ***True***) **execute**

**If** b **MOD** 2 = 1 **then**

                ok ← ***False***

**EndIf**

            b ← b **DIV** 10

**EndWhile**

**If** ok = ***True*** **then**

            a ← a + 1

**EndIf**

**EndFor**

**return** a

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Az algoritmus visszatéríti a ***v*** sorozat páratlan értékű elemeinek darabszámát.
2. Az algoritmus visszatéríti a ***v*** sorozatban található kettőhatványok darabszámát.
3. Az algoritmus visszatéríti a ***v*** sorozatban található azon elemek darabszámát, amelyek csak páros számjegyeket tartalmaznak.
4. Az algoritmus visszatéríti a ***v*** sorozatban található azon elemek darabszámát, amelyek csak páratlan számjegyeket tartalmaznak.

**Megoldás**

* Itt minden állítás *darabszámok* visszatérítéséről szól, tehát azt kell megvizsgálnunk az algoritmusban, hogy *minek a darabszámát*?
* A **For** ciklus a ***v*** tömb minden elemét feldolgozza.
* A feldolgozás során ismételten megvizsgálja, hogy a ***b***-be másolt elem párosszám-e, és osztja ***b***-t 10-zel.
* Így tulajdonképpen az elem minden számjegyét megvizsgálja, de a számláló csak akkor nő, ha minden számjegy páros volt.
* Tehát a **C** **helyes**.

**11.** A mitCsinál(n) algoritmus bemeneti paramétere az ***n*** természetes szám (0 ≤ ***n*** ≤ 10000):

**Algorithm** mitCsinál(n):

    s ← 0

**While** n > 0 **execute**

       c ← n **MOD** 10

**If** c **MOD** 2 = 0 **then**

           s ← s + c

**EndIf**

       n ← n **DIV** 10

**EndWhile**

**return** s

**EndAlgorithm**

Mit fog visszatéríteni a mitCsinál(9876) hívás?

1. 16 B. 48 C. 14 D. 63

**Megoldás**

Sajnos ☹, számolnunk kell.

|  |  |  |
| --- | --- | --- |
| ***n*** | ***c*** | ***s*** |
| 9876 |  | 0 |
|  | 6 | 6 |
|  | 7 |  |
|  | 8 | 14 |
|  | 9 |  |

**12.** A generálás(n) algoritmus az ***n*** természetes számot dolgozza fel (0 < ***n*** < 100):

**Algorithm** generálás(n):

    nr ← 0

**For** i ← 1, 1801 **execute**

        used[i] ← ***False***

**EndFor**

**While not** used[n] **execute**

        sum ← 0

        used[n] ← ***True***

**While** n ≠ 0 **execute**

            digit ← n **MOD** 10

            n ← n **DIV** 10

            sum ← sum + digit \* digit \* digit

**EndWhile**

        n ← sum

        nr ← nr + 1

**EndWhile**

**return** nr

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Ha ***n*** = 10, az algoritmus 2-t térít vissza.
2. Ha ***n*** = 10, az algoritmus 1-et térít vissza.
3. Ha ***n*** = 3, az algoritmus 4-et térít vissza.
4. A generálás(3) és a generálás(30) hívás ugyanazt az értéket téríti vissza.

**Megoldás**

* A ***used*** tömb neve ad kis információt arra vonatkozóan, hogy mi a szerepe: „használt”.
* Vagyis a **While** addig fog dolgozni, amíg egy olyan ***n*** értékhez jutunk, amelyet „használtunk már”.
* Látható, hogy az aktuális ***n*** értékkel egyenlő indexű elem *True* értéket kap.
  + Az algoritmus belső **While** ciklusa kiszámol egy új ***n*** értéket, amely egyenlő a régi ***n*** számjegyei köbének összegével.
  + A feldolgozás során ***nr***-ben megszámoljuk az új ***n*** értékek darabszámát (ez a feldolgozások száma is), ezt fogja visszatéríteni az algoritmus.
* De a kérdések nem az algoritmus logikájára vonatkoznak, hanem konkrét értékekre.
* A táblázatot nagyon figyelmesen kell kitöltenünk!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***n*** | ***nr*** | ***sum*** | ***digit*** | ***used*** |
| 10 | 0 |  |  | (F, F, F, F, F, F, F, F, F, F) |
| 1 |  | 0 | 0 | (F, F, F, F, F, F, F, F, F, T) |
| 0 |  | 1 | 1 | (F, F, F, F, F, F, F, F, F, T) |
| 1 | 1 | 0 |  | (T, F, F, F, F, F, F, F, F, T) |
| 1 |  | 1 | 1 |  |
|  | 2 |  |  | Vége, mivel ***used*1** = *True* |

* Hasonlóan járunk el a **B**, **C**, **D** esetekben is.
* Kiderül, hogy **A** **és C helyes**.

**14.** Állapítsátok meg, hogy az alábbi kifejezések közül melyiknek van akkor és csakis akkor *igaz* értéke, ha az ***n*** természetes szám osztható 3-mal és utolsó számjegye 4 vagy 6:

1. (n **MOD** 3 = 0) **OR** ((n **MOD** 10 = 4) **AND** (n **MOD** 10 = 6))
2. (n **MOD** 6 = 0) **AND** ((n **MOD** 10 = 4) **OR** (n **MOD** 10 = 6))
3. ((n **MOD** 9 = 0) **AND (**n **MOD** 10 = 4)) **OR** ((n **MOD** 3 = 0) **AND (**n **MOD** 10 = 6))
4. (n **MOD** 3 = 0) **AND** (((n **MOD** 2 = 0) **AND** (n **MOD** 5 = 0)) **OR** ((n **MOD** 2 = 0) **AND** (n **MOD** 5 = 1)))

**Megoldás**

* Nagyon figyelünk, amikor egy kijelentésben megjelenik az „*akkor és csakis akkor*”, mivel ez azt jelenti, hogy ha egy kifejezés értéke *igaz* *a megadott feltételek mellett, nem lehet igaz más körülmények között is*.
* Az **A** esetet hamar kizárhatjuk, mivel az ***n*** szám utolsó számjegye nem lehet egyidőben 4 is és 6 is.
* A **B** tartalmaz egy kis „csapdát”, mert ***n*** 6-tal való oszthatóságához köti az utolsó számjegy értékének vizsgálatát.
  + Igen, de az utolsó számjegynek 4-nek vagy 6-nak kell lennie, vagyis a számunk páros.
  + Ha még teljesül a 3-mal történő oszthatóság, máris kiderül, hogy helyes a 6-tal való oszthatóság is.
* Tehát **B** **helyes**.
* A **C**-ben ott a 9-cel való oszthatóság, és mivel az **AND** operátor következik, ezt elvetjük.
* A **D**-ben zavaró az 5-tel való oszthatóság vizsgálata, de mégis megvizsgáljuk ezt az esetet is.
* Ha például ***n*** = 30, az (n **MOD** 2 = 0) **AND** (n **MOD** 5 = 0) részkifejezés értéke *True*, és 30 osztható 3-mal, de az utolsó számjegy nem 4 vagy 6.
* Ha ***n*** = 30, az (n **MOD** 2 = 0) **AND** (n **MOD** 5 = 1) részkifejezés értéke *False*, így az (((n **MOD** 2 = 0) **AND** (n **MOD** 5 = 0)) **OR** ((n **MOD** 2 = 0) **AND** (n **MOD** 5 = 1))) részkifejezés értéke *True* (az **OR** miatt), de *False* kellene legyen.
* Következik, hogy a **D** kifejezés értéke lehet *True* olyan ***n*** érték esetében is, amely nem felel meg a követelményeknek, így nem csak *akkor és csakis akkor* *True*, amikor ***n*** rendelkezik a megadott tulajdonságokkal.
* Tehát **D** nem helyes.

**24.** Adott a következő kifejezés: ***E*(*x*) *= a*0*+ a*1 *\* x + a*2 *\* x*2 *+ a*3 *\* x*3 *+ a*4 *\* x*4**, ahol ***a*0**, ***a*1**,***a*2**,***a*3**, ***a*4** és ***x*** nullától különböző valós számok. Az ***E*(*x*)** kifejezés értékének kiszámítása érdekében elvégzendő szorzások minimális darabszáma:

1. 4
2. 5
3. 7
4. 3

**Megoldás**

* Az ***E*(*x*)** kifejezés tulajdonképpen egy *polinom*.
* Az értékét a *Horner* sémával számítjuk ki legeredményesebben (minimális darabszámú szorzással).
* A polinomot Horner a következőképpen írta fel:

***Pn***(***x***) = ***a*0*+ a*1 *\* x + a*2 *\* x*2 *+ a*3 *\* x*3 *+ … + an* *\* xn*** *=* (felírva ***x*** csökkenő hatványai szerint)

= ***an* *\* xn*** + ***an* - 1 *\* xn* - 1 *+ … + a*2 *\* x*2 *+ a*1 *\* x + a*0** =

= ((…(***an***) ***\* x*** + ***an* – 1**) \* ***x*** + … + ***a*2) *\* x + a*1) *\* x + a*0**

* Az így felírt polinom értékét a következő algoritmussal számítjuk ki:

P = an

**For** i = n - 1, 0 **execute**

P = p \* x + ai

* A szorzások száma ***n.***
* Az ***E***(***x***) kifejezés egy 4-ed fokú polinom, tehát a helyes válasz az **A**.

**BBTE – Matematika-Informatika Kar, nyári felvételi vizsga, 2022 július 14**

**1.** Adott a mitCsinál(a, b) algoritmus, ahol ***a*** és ***b*** természetes számok (a hívás pillanatában 1 ≤ ***a***, ***b*** ≤ 10000).

**Algorithm** mitCsinál(a, b):

**While** ((a **MOD** 10) = (b **MOD** 10)) **AND** (a ≠ 0) **AND** (b ≠ 0) **execute**

a ← a **DIV** 10

b ← b **DIV** 10

**EndWhile**

**If** ((a = 0) **AND** (b = 0)) **then**

**return *True***

**else**

**return *False***

**EndIf**

**EndAlgorithm**

A mitCsinál(a, b) algoritmus *akkor és csakis akkor* térít vissza *True*-t, ha:

1. ***a*** és ***b*** számjegyeinek darabszáma azonos
2. ***a*** egyenlő ***b***-vel
3. ***a*** és ***b*** ugyanazokat a számjegyeket tartalmazza, de más sorrendben
4. ***a*** utolsó számjegye egyenlő ***b*** utolsó számjegyével

**Megoldás**

* Különös figyelemmel vizsgálódunk, mivel ott van a szigorú követelmény: „*akkor és csakis akkor*”.
* A **While** ciklus addig dolgozik, jobbról balra haladva párhuzamosan a két szám számjegyein, amíg ezek *egyenlőek* és egyik számból sem „fogytak el” a számjegyek.
* A **While** után az algoritmus akkor térít *True*-t, ha mindkét szám számjegyei elfogytak.
* Mivel az algoritmus a számjegyeket előfordulásuk sorrendjében hasonlítja össze, következik, hogy a számjegyek csak akkor „fogyhatnak el” egyidőben, ha a két szám egyenlő.
* Tehát a **B** a helyes.

**3.** Az alábbi algoritmusok közül melyik téríti vissza az ***n*** természetes szám prímtényezős felbontásában szereplő különböző prímszámok darabszámát (a hívás pillanatában 5 < ***n*** < 105).

|  |  |
| --- | --- |
| // A prime vektor hossza n  // prime[i] értéke *True*, ha az i szám  // prímszám és *False* különben  **Algorithm** prímtényezőkSzáma\_A(n, prime):  d ← 2  db ← 0  p ← 0  **While** n > 0 **execute**  **While** n **MOD** d = 0 **execute**  p ← p + 1  n ← n **DIV** d  **EndWhile**  **If** p ≠ 0 **then**  db ← db + 1  **EndIf**  d ← d + 1  **While** prime[d]= ***False* execute**  d ← d + 1  **EndWhile**  p ← 0  **EndWhile**  **return** db  **EndAlgorithm**  **Algorithm** prímtényezőkSzáma\_C(n):  db ← 0  **For** d ← 2, n **execute**  **If** n **MOD** d = 0 **then**  db ← db + 1  **EndIf**  **While** n **MOD** d = 0 **execute**  n ← n **DIV** d  **EndWhile**  **EndFor**  **return** db  **EndAlgorithm** | **Algorithm** prímtényezőkSzáma\_B(n):  d ← 2  db ← 0  **While** n > 1 **execute**  p ← 0  **While** n **MOD** d = 0 **execute**  p ← p + 1  n ← n **DIV** d  **EndWhile**  **If** p > 0 **then**  db ← db + 1  **EndIf**  **If** d = 2 **then**  d ← d + 1  **else**  d ← d + 2  **EndIf**  **EndWhile**  **return** db  **EndAlgorithm**  **Algorithm** prímtényezőkSzáma\_D(n):  db ← 0  d ← 2  **While** d \* d ≤ n **execute**  **If** n **MOD** d= 0 **then**  db← db + 1  **EndIf**  **While** n **MOD d** = 0 **execute**  n ← n **DIV** d  **EndWhile**  d ← d + 1  **EndWhile**  **return** db  **EndAlgorithm** |

* Azok a rácstesztek, amelyek azt kérik, hogy döntsük el, melyik algoritmus dolgozik helyesen az adott négy közül azért okozhatnak gondot, mert, ha nem arra koncentrálunk, hogy „belássuk” a helyességet, illetve megtaláljuk az esetleges hibát, és megpróbáljuk kézzel „futtatni”, nagyon sok időt felemészthetnek.
* Tehát, megjegyezzük, hogy az algoritmus „az ***n*** természetes szám prímtényezős felbontásában szereplő különböző prímszámok darabszámát” kellene visszatérítse.
* Nagy előny, ha ismerjük a törzstényezőkre bontó algoritmust…
* A különbség abban áll, hogy nem kell elmenteni, esetleg kiírni ezeket, csak *megszámolni*… a *különbözőket*!
* Fontos rájönni a változók jelentésére:
  + ***db*** a különböző prímtényezők darabszáma (ezt téríti mindegyik algoritmus),
  + ***d*** az aktuális osztó
  + ***p*** tartja számon, hogy ***d*** hányszor osztotta ***n***-t maradék nélkül
* Most követjük az **A** változat logikáját:
  + Minden jónak tűnik, egy kivétellel: az algoritmus addig kellene dolgozzon, amíg ***n*** 1-gyé nem válik.
  + Ha ***n*** = 1-re is engedjük, hogy az algoritmus tovább dolgozzon, nem fog találni osztót, és csak akkor fog leállni, amikor eléri a prímek sorozatának végét.
* Tehát **A** nem helyes.
* Vizsgáljuk a ***B*** változatot:
  + Ez hasonlít az előző algoritmushoz, nem a prímszámok sorozatát használja, de (legalább) nem próbál osztani páros számokkal.
  + Látjuk, hogy a **While** feltétele ***n*** > 1, tehát **B** **helyes**.
* **C**:
  + A **C is helyes**, dacára annak, hogy kevésbé hatékony, hiszen 2-től ***n***-ig minden számmal oszt.
* **D**:
  + „Messziről” nézve ez az algoritmus is helyesnek tűnik, de szerencsés ötletnek bizonyul, ha megvizsgáljuk alaposan a külső **While** feltételét: d \* d ≤ n
  + Mivel ***n*** értéke a lépések során csökken, ez a feltétel leállítja az algoritmust, mielőtt megtalálná az összes prímtényezőt. Tehát **D** nem helyes.

**9.** A következő kifejezések közül melyekekvivalensek az x **MOD** y kifejezéssel minden ***x*** és ***y*** (0 < ***x***,***y*** ≤ 10000) szigorúan pozitív természetes szám esetében?

1. x **DIV** y
2. x – (y \* (x **DIV** y))
3. x – (x \* (x **DIV** y))
4. x **DIV** y + y **DIV** x

**Megoldás**

* Az **A** biztos nem helyes.
* A **D** kifejezésről már ***x*** = 1 és ***y*** = 1 esetében el tudjuk dönteni, hogy nem felel meg, hiszen az érték = 2 (0 helyett).
* A **C** esetében, ha ***x*** nagyobb, mint ***y***, a kifejezés értéke negatív vagy 0, tehát nem lehet egyenlő az eredeti kifejezés értékével. Például, ha ***x*** = 3 és ***y*** = 2,

x – (x \* (x **DIV** y)) = 3 – (3 \* (3 **DIV** 2)) = 3 – 3 = 0.

* Tehát **C** nem helyes.
* Maradt a **B** 😊

**10a.** Legyen az ***n*** változó, amely egy természetes számot tárol. Az alábbi kifejezések közül melyeknek van akkor és csakis akkor *True* értékük, ha ***n*** osztható 2-vel és 3-mal?

1. (n **DIV** 2 = 0) **OR** (n **DIV** 3 ≠ 0)
2. (n **MOD** 3 = 2) **OR** (n **MOD** 2 = 3)
3. (n **MOD** 2 ≠ 1) **AND** (n **MOD** 3 = 0)
4. (n **MOD** 2 = 0) **AND** (n **MOD** 3 ≠ 1)

**Megoldás**

* Az **A** biztos nem helyes, hiszen oszt, ahelyett, hogy egész osztási maradékot számolna.
* A **B** esetében előbb kiszámítjuk a kifejezés értékét, ha ***n*** = 8: mivel a 3-mal történő egészosztási maradék 2, az operátor **OR**, a kifejezés értéke *True*, de 8 nem osztható 2-vel és 3-mal is.
* Tehát **B** nem helyes.
* A **C** kifejezést egy kicsit átalakítjuk: (n **MOD** 2 = 0) **AND** (n **MOD** 3 = 0)

és így egyértelmű, hogy ez **helyes**.

* A **D** kifejezést is átalakítanánk, de rájövünk, hogy fölösleges, hiszen, ha a 3-mal való osztási maradék nem 1, lehetne 0 vagy 2, és innen nem következik, hogy ***n*** osztható 3-mal:

**10b.** Az alábbi kifejezések közül melyeknek van akkor és csakis akkor *True* értékük, ha ***n*** osztható 2-vel és 3-mal?

1. (n **MOD** 2) - (n **MOD** 3) = 0
2. (n **MOD** 2) - (n **MOD** 3) < 0
3. (n **MOD** 2) + (n **MOD** 3) > 0
4. (n **MOD** 2) + (n **MOD** 3) = 0

**Megoldás**

* A **D** a legszembetűnőbb, mivel az összeg csak a 6-tal osztható számok esetében lesz 0, és 6 = 2\*3, tehát **helyes**.
* Ebből következik, hogy a másik három változat nem lehet helyes 😊

**13.** Mi lesz az100101100111 bináris szám értéke a 10-es számrendszerben?

A. 2407 B. 2408 C. 1203 D. Az A., B., C. válaszok egyike sem

**Megoldás**

Megszámoljuk a számjegyeket. Ez 12, majd kiszámítjuk a szám értékét jobbról balra haladva az adott számjegysorozatban (azért jobbról balra, hogy „vihessük tovább” a részeredményeket.

1 \* 20 + 1 \* 21 + 1 \* 22 + 0 \* 23 + 0 \* 24 + 1 \* 25 + 1 \* 26 + 0 \* 27 + 1 \* 28 + 0 \* 29 + 0 \* 210 + 1 \* 211 =

1 + 2 + 4 + 32 + 64 + 256 + 256 \* 2 \* 2 \* 2 = 2407

Tehát **A** **a helyes**.

**21.** Adott az ***n*** természetes szám, amely felírható három természetes szám szorzataként (***n*** = ***a*** \* ***b*** \* ***c***). A következő kifejezések közül melyiknek egyenlő az értéke ***n***-nek a ***d*** természetes számmal való osztási maradékával (1 ≤ ***n***, ***a***, ***b***, ***c***, ***d*** ≤ 10000)?

1. (a **MOD** d) \* b \* c
2. ((a **MOD** d) \* (b **MOD** d) \* (c **MOD** d)) **MOD** d
3. (a **MOD** d) \* (b **MOD** d) \* (c **MOD** d)
4. (a **DIV** d) \* (b **DIV** d) \* (c **DIV** d)

**Megoldás**

* Az **A** nem lehet jó, mivel a ***b***-t és ***c***-t nem osztja.
* A **B** kifejezés „érdekes”, de egy kissé meglepő, hogy a szorzat esetében is kiszámítja a ***d***-vel történő osztás maradékét.
* Valóban szükség van erre, hiszen előfordulhat, hogy a maradékok szorzata nagyobb, mint ***d***.
* Ha bizonytalanok vagyunk, veszünk egy egyszerű példát.
* Legyen ***a*** = 1, ***b*** = 2, ***c*** = 3 és ***d*** = 4. Ekkor ***n*** = 6 és 6 **MOD** 4 = 2.
* (a **MOD** d) \* (b **MOD** d) \* (c **MOD** d) = (1 **MOD** 4) \* (2 **MOD** 4) \* (3 **MOD** 4) = 1 \* 2 \* 3 = 6
* Tehát a **B helyes.**
* A **C** pontosan az utolsó maradékszámítás nélküli előző kifejezés, tehát nem helyes.
* Egyértelmű, hogy a **D** nem lehet jó.