**Mat–Infó verseny 2019**

1. **Ackermann**

Legyenek az ***m*** és ***n*** természetes számok (0 ≤ ***m*** ≤ 10, 0 ≤ ***n*** ≤ 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) |

|  |  |  |
| --- | --- | --- |
| **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)** | 1. 7-szer 2. 10-szer 3. 5-ször 4. ugyanannyiszor, mint az alábbi utasítások végrehajtásának következtében:  |  | | --- | | m ← 1, n ← 3  Ack(m, n) | |

**Megoldás**

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékére hányszor hívja meg önmagát. A megoldáshoz követnünk kell a paraméterek értékeit a hívássorozaton belül, – esetleg leraj­zoljuk a végrehajtási verem tartalmát és megszámoljuk a rekurzív hívásokat:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ack(1, 2) ⇒ (1. hívás):  Ack(0, Ack(1, 1)) ⇒ (2. hívás):  Ack(1, Ack(1, 0)) ⇒ (3. hívás):  Ack(0, 1) ⇒ (4. hívás):  Ack(0, 2) ⇒ (5. hívás)  Ack(0, 3): vége. | |  |  |  | | --- | --- | --- | |  |  | Hívás sorszáma | | 3 | ***n*** | 5 | | 0 | ***m*** | | 2 | ***n*** | 4 | | 0 | ***m*** | | 1 | ***n*** | 3 | | 0 | ***m*** | | 0 | ***n*** | 2 | | 1 | ***m*** | | 1 | ***n*** | 1 | | 1 | ***m*** | | 2 | ***n*** | 0 (első hívás) | | 1 | ***m*** | |

Tehát, a rekurzív hívások száma 5, vagyis a **C.** válasz jó, így azt is tudjuk bizonyosan, hogy az **A.** és **B.** válaszok nem jók. De maradt még egy válasz, amelyről még nem tudjuk, hogy helyes vagy sem. A fent leírt módon megvizsgáljuk ezt az esetet is, és megállapítjuk, hogy a rekurzív hívások száma 7, ami különbözik 5-től. Tehát a **D.** válasz sem jó.

1. **Mely értékek szükségesek?**

Legyen a különbség(a, n) algoritmus, ahol ***a*** egy ***n*** elemű (0 < ***n*** < 100) sorozat, amely egész számokat tárol. Az ***n*** és ***a*** mely értékeire térít az algoritmus 0-át?

|  |  |
| --- | --- |
| **Algoritmus** különbség(a, n)  **Ha** n = 0 **akkor** **térítsd** 0  **vége(ha)**  **Ha** |a[n]| **MOD** 2 = 0 **akkor**  **térítsd** különbség(a, n - 1) + a[n]  **különben**  **térítsd** különbség(a, n - 1) - a[n]  **vége(ha)**  **Vége(algoritmus)** | **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**

Könnyű belátni, hogy a páros számokat összeadja az algoritmus, míg a páratlanokat kivonja az aktuális összegből, amit nem tárolunk egy változóban, de – mivel 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. Tehát azokat a bemeneti adatokat választjuk ki, amelyeknek esetében a páros számok összege egyenlő a páratlanok összegével. Helyes válaszok: **A.**, **B.** és **D**.

1. **Kiegészítés (6 pont)**

Legyen a kizárPáratlan(n) algoritmus, ahol ***n*** (1 ≤ ***n*** ≤ 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 **MOD** 2 = 1 **akkor**  **térítsd** kizárPáratlan(n **DIV** 10)  **vége(ha)**  **térítsd** ...  **Vége(algoritmus)** | 1. kizárPáratlan(n **MOD** 10) \* 10 + n **DIV** 10 2. kizárPáratlan(n) \* 10 + n **MOD** 10 3. kizárPáratlan(n **DIV** 10) \* 10 + n **MOD** 10 4. kizárPáratlan((n **DIV** 10) **MOD** 10) \* 10 |

**Megoldás**

Tulajdonképpen egy új számot kell felépítenünk az adott ***n*** szám páros számjegyeiből. Látjuk, hogy 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**.

**Felvételi verseny 2019**

1. **Számjegyszorzat**

A számJegyek(n, d) algoritmus (***n*** és ***d*** természetes számok, 10 ≤ ***n*** ≤ 100 000, 1 ≤ ***d*** ≤ 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 **MOD** d = 0 **akkor**  érték ← számJegyek(n **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) |  1. Ha ***n*** = 108, az algoritmus 11-szer hívja meg önmagát. 2. Ha ***n*** = 109, az algoritmus 8-szor hívja meg önmagát. 3. Ha ***n*** = 13, az algoritmus egyszer sem hívja meg önmagát. 4. Ha ***n*** = 100, az algoritmus 10-szer hívja meg önmagát. |

**Megoldás**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A rekurzív hívások számát megszámolhatjuk, ha hívásso­rozatot készítünk vagy, ha követjük a paraméterek értékeit a végrehajtási veremben:  számjegyek(108, 9) ⇒ (0. hívás):  számjegyek(12, 9) ⇒ (1. hívás):  számjegyek(12, 8) ⇒ (2. hívás):  számjegyek(12, 7) ⇒ (3. hívás):  számjegyek(12, 6) ⇒ (4. hívás):  számjegyek(2, 6) ⇒ (5. hívás):  számjegyek(2, 5) ⇒ (6. hívás):  számjegyek(2, 4) ⇒ (7. hívás):  számjegyek(2, 3) ⇒ (8. hívás):  számjegyek(2, 2) ⇒ (9. hívás):  számjegyek(1, 2) ⇒ (10. hívás):  számjegyek(1, 1) ⇒ (11. hívás) | |  |  |  | | --- | --- | --- | |  |  | Hívás sorszáma | | 1 | ***d*** | 11 | | 1 | ***n*** | | 2 | ***d*** | 10 | | 1 | ***n*** | | 2 | ***d*** | 9 | | 2 | ***n*** | | 3 | ***d*** | 8 | | 2 | ***n*** | | 4 | ***d*** | 7 | | 2 | ***n*** | | 5 | ***d*** | 6 | | 2 | ***n*** | | 6 | ***d*** | 5 | | 2 | ***n*** | | 6 | ***d*** | 4 | | 12 | ***n*** | | 7 | ***d*** | 3 | | 12 | ***n*** | | 8 | ***d*** | 2 | | 12 | ***n*** | | 9 | ***d*** | 1 | | 12 | ***n*** | | 9 | ***d*** | 0 (első hívás) | | 108 | ***n*** | |

A hívások száma 11, tehát **A.** helyes.

A **B.** válasz esetében, valóban 8 rekurzív hívás után áll le a hívások sorozata. Az ***n*** paraméter értéke nem változik, miközben ***d*** csökken 9-től 1-ig, amikor -1-et térít, mivel nem talál megoldást. 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. Ha készítünk hívássorozatot, látni fogjuk, hogy nem talál megoldást, de a **B.** esethez hasonlóan szintén 8-szor hívja meg önmagát, miközben ***d*** csökken 9-től 1-ig, amikor leáll, tehát **C.** nem helyes.

A **D.** válasz esetében a rekurzív hívások száma 8, tehát **D.** sem helyes.

1. **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*** = 1 092, a varázslat szétválasztja a ***bal*** = 10 és ***jobb*** = 92 számokra.

Az alábbi algoritmusok közül melyik alkalmazza a varázslatot az ***x*** természetes számra, amelynek 10-es szám­rendszerben van legkevesebb két 0-tól különböző számjegye (100 ≤ ***x*** ≤ 1 000 000)? Az algoritmus meghatározza a ***z*** természetes számban (0 ≤ ***z*** ≤ 1 000 000) az ***x*** szám ***jobb*** részét. Az alábbi algoritmusok léteznek:

* hatvány(b, p) – meghatározza a ***bp*** értéket (***b*** a ***p***. hatványon), ***b***, ***p***– természetes számok (1 ≤ ***b*** ≤ 20, 1 ≤ ***p*** ≤ 20);
* szjSzáma(sz) – meghatározza az ***sz*** szám (0 ≤ ***sz*** ≤ 1 000 000) számjegyeinek darabszámát;

|  |  |
| --- | --- |
|  | **Algoritmus** varázslat(x, z)  maxSzorzat ← -1  eredmény ← 0  **Amíg** x > 0 **végezd el**  z ← (x **MOD** 10) \* hatvány(10, szjSzáma(z)) + z  x ← x **DIV** 10  **Ha** x \* z > maxSzorzat **akkor**  maxSzorzat ← x \* z  eredmény ← z  **vége(ha)**  **vége(amíg)**  **térítsd** maxSzorzat  **Vége(algoritmus)** |
| 1. a | **Algoritmus varázslat**(x, z)  t ← 0  **Ha** x > 0 **akkor**  y ← (x **MOD** 10) \* hatvány(10, szjSzáma(z)) + z  t ← 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)** |
|  | **Algoritmus varázslat**(x, z)  maxSzorzat ← -1  eredmény ← 0  **Amíg** x > 0 **végezd el**  z ← (x **MOD** 10) \* hatvány(10, szjSzáma(z)) + z  x ← x **DIV** 10  **Ha** x \* z > maxSzorzat **akkor**  maxSzorzat ← x \* z  eredmény ← z  **vége(ha)**  **vége(amíg)**  **térítsd** eredmény  **Vége(algoritmus)** |
|  | **Algoritmus varázslat**(x, z)  **Ha** x > 0 **akkor**  y ← (x **MOD** 10) \* hatvány(10, szjSzáma(z)) + z  t ← x **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.

A **B.** algoritmus két paramétere, hasonlóan a **C.** algoritmushoz: ***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 önmagát 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***-tól. A **térítsd** t utasítás is hibás, hiszen a feladat a szám ***jobb*** részét kéri, de ***t*** a ***bal*** része.

A továbbiakban 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.

**Matek-Infó verseny 2018**

1. **Számolás**

Legyen a számol(a, b) algoritmus, amelynek bemeneti paraméterei az ***a*** és ***b*** pozitív természetes számok, ahol 1 ≤ ***a*** ≤ 100, 1 ≤ ***b*** ≤ 100.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6. | **Algoritmus** számol(a, b):  **Ha** a ≠ 0 **akkor**  **térítsd** számol(a **DIV** 2, b + b) + b \* (a **MOD** 2)  **vége(ha)**  **térítsd** 0  **Vége(algoritmus)** |

Az alábbi válaszok közül melyek hamisak?

1. ha ***a*** és ***b*** egyenlők, az algoritmus ***a*** értékét téríti
2. ha ***a*** = 1000 és ***b*** = 2, az algoritmus 10-szer hívja meg önmagát
3. az algoritmus által kiszámított és térített érték egyenlő (***a*** / 2 + 2 \* ***b***)-vel
4. az 5. sorban található utasítás egyszer sem hajtódik végre
5. az 5. sorban található utasítás egyszer hajtódik végre

**Megoldás**

Megjegyezzük, hogy a *hamis* állításokat kell megadnunk!

Remélhetően sok versenyző felismeri az algoritmust, hiszen ez az úgynevezett „orosz szorzás”. Az sem kizárt, hogy van olyan versenyző is, aki tudja, hogy ez egy logaritmikus bonyolultságú algoritmus, tehát a **B.** válasz helyes, hiszen log21000 ≈ 10 (mivel 210 = 1024). Tehát a **B.**-t nem jelöljük meg.

Ha bárki hiányolná a szorzandó vizsgálatát (páros vagy páratlan?), ajánlatos figyelmesen tanulmányoznia, például az algoritmus 3. sorát:

**térítsd** számol(a **DIV** 2, b + b) + b \* (a **MOD** 2).

A rekurzív hívás paraméterei – a várakozásnak megfelelően – a szorzandó fele és a szorzó kétszerese. Az utasítást a következőképpen értékelhetjük ki: ha ***a*** páratlan szám, a **MOD** 2 = 1, tehát a térített értékhez hozzá­adjuk ***b*** \* 1 = ***b***-t, különben a ***b***-t 0-val szorozza az algoritmus, tehát nem adja hozzá.

Ha valaki nem ismeri az algoritmust, az sem okoz gondot, hiszen, például az **A.** választ illetően, azonnal észre lehet venni, hogy az algoritmus sehol nem vizsgálja (még „burkoltan” sem) ***a*** és ***b*** egyenlőségét. Másfelől, az eredmény bizonyos aktuálisan érvényes ***b*** értékek összege, így az algoritmus soha nem térítheti ***a***-t. Megtaláltuk az első hamis állítást.

A **C.** pontban olyan állítás található, amely szintén nem lehet igaz; a fent leírt érveléshez hasonlóan rájö­vünk, hogy az összeg, amit térít az algoritmus nem lehet egyenlő (***a*** / 2 + 2 \* ***b***)-vel. Ezek szerint a **C.** is hamis.

A **D**. és **E.** állításokat együttesen vizsgáljuk. Hamar belátjuk, hogy **E.** helyes, hiszen a rekurzív hívások addig ismétlődnek, amíg ***a*** > 0, de ezután végrehajtódik az 5. sorban található utasítás. Tulajdonképpen ekkor kap kezdőértéket az összeg, ami végül ***a*** és ***b*** szorzata. Tehát a hamis állítások: **A**., **C.** és **D.**

1. **Törzstényezők**

Legyen a törzsTényezők(n, d, k, x) algoritmus, amely meghatározza az ***n*** természetes szám ***k*** darab törzstényezőjét, a törzstényezők keresését a ***d*** értéktől kezdve. Bemeneti paraméterek az ***n***, ***d*** és ***k*** számok, kimeneti paraméterek az ***x*** sorozat, amely a ***k*** törzstényezőt tartalmazza (1 ≤ ***n*** ≤ 10000, 2 ≤ ***d*** ≤ 10000, 0 ≤ ***k*** ≤ 10000).

|  |
| --- |
| **Algoritmus** törzsTényezők(n, d, k, x):  **Ha** n **MOD** d = 0 **akkor**  k ← k + 1  x[k] ← d  **vége(ha)**  **Amíg** n **MOD** d = 0 **végezd el**  n ← n **DIV** d  **vége(amíg)**  **Ha** n > 1 **akkor**  törzsTényezők(n, d + 1, k, x)  **vége(ha)**  **Vége(algoritmus)** |

Határozzátok meg, hányszor hívja meg önmagát a törzsTényezők(n, d, k, x) algoritmus a következő prog­ramrészlet végrehajtásának következtében:

|  |
| --- |
| n ← 120  d ← 2  k ← 0  törzsTényezők(n, d, k, x) |

**A.** 3-szor **B.** 5-ször **C.** 9-szer **D.** 6-szor

**E.** ugyanannyiszor, mint a következő programrészlet esetében:

|  |
| --- |
| n ← 750  d ← 2  k ← 0  törzsTényezők(n, d, k, x) |

**Megoldás**

A rekurzív algoritmus akkor hívja meg önmagát, ha ***n*** (még) nagyobb, mint 1. A folytatási feltételt magába foglaló **Ha** utasítás előtt az ***n*** szám aktuális ***d*** osztóját elhelyezi az ***x*** sorozatba, de előbb aktualizálja a sorozat ***k*** hosszát, majd addig osztja az ***n*** számot ***d***-vel, amíg ez maradék nélkül lehetséges. Így eléri azt, hogy egy törzstényező csak egyszer kerüljön az ***x*** sorozatba. Könnyen belátható, hogy a rekurzív hívások száma egyenlő azoknak a ***d*** értékeknek a számával, amelyek nagyobbak, mint 2 és kisebbek vagy egyenlők ***n*** legnagyobb prímosztójával. Az **A.** és **E.** pontoknak megfelelően az újrahívások száma 5 – 2 = 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 120 | **2** | xxxxxxxxx | 750 | **2** |
| 60 | 2 |  | 375 | **3** |
| 30 | 2 |  | 125 | **5** |
| 15 | **3** |  | 25 | 5 |
| 5 | **5** |  | 5 | 5 |
| 1 |  |  | 1 |  |

Látjuk, hogy ha ***n*** = 120, a rekurzív hívások száma 3, tehát nem lehet 5 vagy 9, esetleg 6. Ha ***n*** = 750, a rekurzív hívások száma szintén 3, tehát a helyes válaszok: **A.** és **E.**

**B.1. Konverzió**

Legyen a konverzió(s, hossz) algoritmus, amely átalakítja az ***s*** karakterláncot, amely egy 16-os szám­rendszerben ábrázolt szám, a megfelelő 10-es számrendszerben érvényes alakjára. Az ***s*** karakterlánc ***hossz*** darab karaktert tartalmaz, ahol a karakter értéke lehet egy '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' számjegy, vagy egy 'A', 'B', 'C', 'D', 'E', 'F' nagybetű (a ***hossz*** természetes szám, 1 ≤ ***hossz*** ≤ 10).

Írjátok le a konverzió(s, hossz) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos az alábbi algoritmuséval. Az adatok garantáltan helyesek.

|  |
| --- |
| **Algoritmus** konverzió(s, hossz):  szám ← 0  **Minden** i = 1, hossz **végezd el**  **Ha** s[i]≥ 'A' **akkor**  szám ← szám \* 16 + s[i] - 'A' + 10  **különben**  szám ← szám \* 16 + s[i] - '0'  **vége(ha)**  **vége(minden)**  **térítsd** szám  **Vége(algoritmus)** |

**Megoldás**

Az értékelés a következő szempontok szerint történt:

* A rekurzív algoritmus fejléce azonos-e az iteratív fejlécével? (2 pont)
* A rekurzív hívások leállási feltétele helyes-e? (1 pont)
* A leálláskor térített érték helyes-e? (1 pont)
* Helyesen vizsgálja, hogy a karakter nem számjegy? (2 pont)
* Helyesen határozza meg a nem számjegy karakter értékét? (2 pont)
* Helyesen határozza meg a számjegy karakter értékét? (2 pont)

Egy helyes rekurzív algoritmus:

|  |
| --- |
| **Algoritmus** konverzió(s, hossz):  **Ha** hossz > 0 **akkor**  **Ha** s[hossz] ≥ 'A' **akkor**  **térítsd** konverzió(s, hossz - 1) \* 16 + s[hossz] - 'A' + 10  **különben**  **térítsd** konverzió(s, hossz - 1) \* 16 + s[hossz] - '0'  **vége(ha)**  **különben**  **térítsd** 0  **vége(ha)**  **Vége(algoritmus)** |

**Felvételi 2018, július**

1. **Vajon mit csinál? (5p)**

Adott a kifejezés(n) algoritmus, ahol ***n*** (1 ≤ ***n*** ≤ 10000) természetes szám.

|  |
| --- |
| **Algoritmus** kifejezés(n):  **Ha** n > 0 **akkor**  **Ha** n **MOD** 2 = 0 **akkor**  **térítsd** -n \* (n + 1) + kifejezés(n - 1)  **különben**  **térítsd** n \* (n + 1) + kifejezés(n - 1)  **vége(ha)**  **különben**  **térítsd** 0  **vége(ha)**  **Vége(algoritmus)** |

Állapítsátok meg az *E*(*n*) kifejezésnek azt a matematikai alakját, amelyet a fenti algoritmus számít ki:

* 1. E(*n*) = 1\* 2 - 2 \* 3 + 3 \* 4 + ... + (-1)*n*+1 \* *n* \* (*n* + 1)
  2. E(*n*) = 1\* 2 - 2 \* 3 + 3 \* 4 + ... + (-1)*n* \* *n* \* (*n* + 1)
  3. E(*n*) = 1\* 2 + 2 \* 3 + 3 \* 4 + ... + (-1)*n*+1 \* *n* \* (*n* + 1)
  4. E(*n*) = 1\* 2 + 2 \* 3 + 3 \* 4 + ... + (-1)*n* \* *n* \* (*n* + 1)
  5. E(*n*) = 1\* 2 - 2 \* 3 - 3 \* 4 - ... - (-1)*n* \* *n* \* (*n* + 1)

**Megoldás**

Az algoritmus nem hívja meg önmagát, ha ***n*** értéke 0-ra csökkent, ekkor kap az összeg, amit kiszámít 0 kezdőértéket. Látjuk, hogy az ***n*** aktuális értékének párosságától függően az rekurzív hívás különböző értékeket térít. Ha ***n*** páros, az aktuális tagot kivonja az összeg aktuális értékéből, ha ***n*** páratlan, hozzáadja. Ekkor már kizárhatjuk a lehetséges válaszok közül a **C.**-t és a **D.**-t, mivel ezekben a kifejezésekben nincs kivonás. A továbbiakban vizsgáljuk a kifejezések utolsó tagját. Az **A.** kifejezésben a hatványkitevő (***n*** + 1), amiből következik, hogy ha ***n*** páros (tehát (***n*** + 1) páratlan), ezt a tagot kivonjuk a kifejezés eddigi értékéből. Ez megfelel az algoritmus előbbi elemzésének, tehát az **A.** válasz helyes. A **B.** kifejezésben a (-1)-et az ***n.*** hat­ványra emeljük, de így a tagok előjele, nem az algoritmusnak megfelelően váltakozik, tehát a **B.** válasz sem helyes. Az **E.** kifejezésben túl sok tag előjele „–”, tehát nem ezt a kifejezést értékeli az algoritmus.

1. **Tegyél 'b' betűket**

Legyen az ***n*** × ***n*** méretű négyzetes ***mat*** tömb (***n*** – páratlan természetes szám, 3 ≤ ***n*** ≤ 100). A tegyélB(mat, n, i, j) algoritmus 'b' betűket tesz a ***mat*** tömb bizonyos pozícióira. Az ***i*** és ***j*** paraméterek természetes számok (1 ≤ ***i*** ≤ ***n***, 1 ≤ ***j*** ≤ ***n***).

|  |  |  |
| --- | --- | --- |
| **Algoritmus** tegyélB(mat, n, i, j):  **Ha** i ≤ n **DIV** 2 **akkor**  **Ha** j ≤ n - i **akkor**  mat[i][j] ← 'b'  tegyélB(mat, n, i, j + 1)  **különben**  tegyélB(mat, n, i + 1, i + 2)  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** | ­ | Határozzátok meg, hányszor hívja meg önmagát a tegyélB al­go­ritmus a következő programrészlet végrehajtásának követ­kez­tében: |
|  | n ← 7  i ← 2  j ← 4  tegyélB(mat, n, i, j) |

**A.** 5-ször

**B.** ugyanannyiszor, mint a következő programrészlet esetében:

|  |
| --- |
| n ← 9, i ← 3, j ← 5  tegyélB(mat, n, i, j) |

**C.** 10-szer

**D.** 0-szor

**E.** végtelenszer

**Megoldás**

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékére hányszor hívja meg önmagát. A megoldáshoz követnünk kell a paraméterek értékeit a hívássorozaton belül.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| tegyélB(mat, 7, 2, 4) ⇒ (1. hívás):  tegyélB(mat, 7, 2, 5) ⇒ (2. hívás):  tegyélB(mat, 7, 2, 6) ⇒ (3. hívás):  tegyélB(mat, 7, 3, 4) ⇒ (4. hívás):  tegyélB(mat, 7, 3, 5) ⇒ (5. hívás)  tegyélB(mat, 7, 4, 5): vége. | |  |  |  | | --- | --- | --- | |  |  | Hívás sorszáma | | 5 | ***j*** | 5 | | 4 | ***i*** | | 5 | ***j*** | 4 | | 3 | ***i*** | | 4 | ***j*** | 3 | | 3 | ***i*** | | 6 | ***j*** | 2 | | 2 | ***i*** | | 5 | ***j*** | 1 | | 2 | ***i*** | | 4 | ***j*** | 0 (első hívás) | | 2 | ***i*** | |

Tehát, a rekurzív hívások száma 5, vagyis az **A.** válasz jó, így azt is tudjuk bizonyosan, hogy a **C.**, **D.** és **E.** válaszok nem jók. Megvizsgáljuk fent leírt módon a **B.** esetet is, és megállapítjuk, hogy a rekurzív hívások száma szintén 5, tehát a **B.** válasz is jó.

1. **Számolás - karakterekkel**

Legyen a számolásKarakterekkel(s, n, p, q, szám) algoritmus, ahol ***s*** egy ***n*** karakterből álló sorozat (***n*** természetes szám, 1 ≤ ***n*** ≤ 9), ***p***, ***q*** és ***szám*** természetes számok (1 ≤ ***p*** ≤ ***n***, 1 ≤ ***q*** ≤ ***n***, ***p*** ≤ ***q***).

|  |  |
| --- | --- |
| **Algoritmus** számolásKarakterekkel(s, n, p, q, szám):  eredmény ← 0  i ← p  **Amíg** i ≤ q **végezd el**  **Amíg** i ≤ q **és** s[i] ≥ '0' **és** s[i] ≤ '9' **végezd el**  szám ← szám \* 10 + s[i] - '0'  i ← i + 1  **vége(amíg)**  eredmény ← eredmény + szám  szám ← 0  i ← i + 1  **vége(amíg)**  **térítsd** eredmény  **Vége(algoritmus)** | Írjátok le a számolásKarakterekkel(s, n, p, q, szám) algoritmus *rekurzív* változatát úgy, hogy a fej­léce és a hatása legyen azonos a fenti algoritmuséval. Az alábbi programrészletből hívjuk meg:  **Beolvas:** n, s, p, q  **KiÍr:** számolásKarakterekkel(s, n, p, q, 0) |

**Megoldás**

Az értékelés szempontjai:

A rekurzív algoritmus fejléce azonos-e az iteratív fejlécével? (2 pont), a rekurzív hívások leállási feltétele helyes-e? (1 pont), a leálláskor térített érték helyes-e? (1 pont), helyesen vizsgálja, hogy a karakter nem szám­jegy? (2 pont), helyesen határozza meg a nem számjegy karakter értékét? (2 pont), helyesen határozza meg a számjegy karakter értékét? (2 pont)

Egy helyes rekurzív algoritmus:

|  |
| --- |
| **Algoritmus** számolásKarakterekkel(s, n, p, q, szám):  **Ha** p > q **akkor**  **térítsd** szám  **különben**  **Ha** s[p] < '0' **vagy** s[p] > '9' **akkor**  **térítsd** szám + számolásKarakterekkel(s, n, p + 1, q, 0)  **különben**  **térítsd** számolásKarakterekkel(s, n, p + 1, q, 10 \* szám + s[p] - '0')  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** |

**Felvételi verseny 2018, szeptember**

1. **Logikai kifejezés kiértékelése**

Adott a ***k*** elemű ***s*** sorozat, amelynek elemei logikai (boolean) típusúak és a kiértékelés(s, k, i) algo­ritmus, ahol ***k*** és ***i*** (0 ≤ ***i*** ≤ ***k*** ≤ 100) természetes számok.

|  |  |
| --- | --- |
| **Algoritmus** kiértékelés(s, k, i)  **Ha** i ≤ k **akkor**  **Ha** s[i] **akkor**  **térítsd** s[i]  **különben**  **térítsd** (s[i] **vagy** kiértékelés(s, k, i + 1))  **vége(ha)**  **különben**  **térítsd** *hamis*  **vége(ha)**  **Vége(algoritmus)** | Határozzátok meg, hányszor hívja meg önmagát a kiértékelés(s, k, i) algoritmus a következő programrészlet végrehajtásának következtében: |

|  |
| --- |
| s ← (*hamis*, *hamis*, *hamis*, *hamis*, *hamis*, *hamis*, *igaz*, *hamis*, *hamis*, *hamis*)  k ← 10  i ← 3  kiértékelés(s, k, i) |

1. 3-szor
2. ugyanannyiszor, mint a következő programrészlet esetében

|  |
| --- |
| s ← (*hamis*, *hamis*, *hamis*, *hamis*, *hamis*, *hamis*, *hamis*, *igaz*)  k ← 8  i ← 4  kiértékelés(s, k, i) |

1. 6-szor
2. egyszer sem
3. végtelenszer

**Megoldás**

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékére hányszor hívja meg önmagát. A megoldáshoz követnünk kell a paraméterek értékeit a hívássorozaton belül, – esetleg leraj­zoljuk a végrehajtási verem tartalmát és megszámoljuk a rekurzív hívásokat:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| kiértékelés(s, 10, 3) ⇒ (1. hívás):  kiértékelés(s, 10, 4) ⇒ (2. hívás):  kiértékelés(s, 10, 5) ⇒ (3. hívás):  kiértékelés(s, 10, 6) ⇒ (4. hívás): kiértékelés(s, 10, 7): vége. | |  |  |  | | --- | --- | --- | |  |  | Hívás sorszáma | | 7 | ***i*** | 4 | | 10 | ***k*** | | 6 | ***i*** | 3 | | 10 | ***k*** | | 5 | ***i*** | 2 | | 10 | ***k*** | | 4 | ***i*** | 1 | | 10 | ***k*** | | 3 | ***i*** | 0 (első hívás) | | 10 | ***k*** | |

Tehát, a rekurzív hívások száma 4, vagyis nem 3, tehát az **A.**, **C.**, **D.** és **E.** válaszok nem jók. De maradt még egy válasz, amelyről még nem tudjuk, hogy helyes vagy sem. Megvizsgáljuk a fent leírt módon a **B.** esetet is, és megállapítjuk, hogy a rekurzív hívások száma 4, tehát a **B.** válasz jó.

1. **Egyesítés**

Adottnak tekintjük az eleme(x, a, n) algoritmust, amely eldönti, hogy az ***x*** természetes szám eleme-e az ***n*** elemű ***a*** halmaznak; ***a*** egy ***n*** elemű sorozat, amely egy természetes számokat tartalmazó halmazt ábrázol (1 ≤ ***n*** ≤ 200, 1 ≤ ***x*** ≤ 1000).

Legyen az alább megadott egyesítés(a, n, b, m, c, p) algoritmus, ahol ***a***, ***b*** és ***c*** sorozatok, amelyek természetes számokat tároló és rendre ***n***, ***m*** és ***p*** elemű halmazokat ábrázolnak (1 ≤ ***n*** ≤ 200, 1 ≤ ***m*** ≤ 200, 1 ≤ ***p*** ≤ 400). A bemeneti paraméterek ***a***, ***n***, ***b*** és ***m***, kimeneti paraméterek pedig ***c*** és ***p***.

|  |
| --- |
| 1. **Algoritmus** egyesítés(a, n, b, m, c, p): 2. **Ha** n = 0 **akkor** 3. **Minden** i = 1, m **végezd el** 4. p ← p + 1 5. c[p] ← b[i] 6. **vége(minden)** 7. **különben** 8. **Ha nem** eleme(a[n], b, m) **akkor** 9. p ← p + 1 10. c[p] ← a[n] 11. **vége(ha)** 12. egyesítés(a, n - 1, b, m, c, p) 13. **vége(ha)** 14. **Vége(algoritmus)** |

A következő állítások közül melyek bizonyulnak mindig igaznak?

1. ha az ***a*** halmaz egy elemet tartalmaz, az egyesítés(a, n, b, m, c, p) algoritmus végtelen ciklusba kerül
2. ha az ***a*** halmaznak négy eleme van, az egyesítés(a, n, b, m, c, p) algoritmus első meghívása maga után vonja a 12. sorban található utasítás végrehajtását négyszer
3. ha az ***a*** halmaznak öt eleme van, az egyesítés(a, n, b, m, c, p) algoritmus első meghívása maga után vonja a második sorban található utasítás végrehajtását ötször
4. ha az ***a*** halmaznak ugyanannyi eleme van, mint a ***b*** halmaznak, az egyesítés(a, n, b, m, c, p) algoritmus végrehajtása után a ***c*** halmaznak ugyanannyi eleme lesz, mint az ***a*** halmaznak
5. ha az ***a*** és ***b*** halmazok elemei azonosak, az egyesítés(a, n, b, m, c, p) algoritmus végrehajtása után a ***c*** halmaznak ugyanannyi eleme lesz, mint az ***a*** halmaznak

**Megoldás**

Az **A.** válasz nem lehet helyes. Ha az ***a*** halmaznak egy eleme van, ***n*** = 1 (vagyis nem 0) az algoritmus előbb eldönti, hogy ez az elem megtalálható-e a ***b*** halmazban. Ha ***a*1** értéke nem eleme a ***b*** halmaznak, elhelyezi ***a*1**-et a ***c*** halmazba, majd meghívja önmagát ***n*** = 0-ra, különben csak a rekurzív hívást hajtja végre. A következő végrehajtáskor megtörténik a kezdőértékadás (az algoritmus bemásolja a ***b*** halmaz elemeit a ***c*** halmazba), kilép az aktuális hívásból és visszatér a hívás helyére. Ezúttal kilép az algoritmusból és visszatér abba a program­egységbe, amely meghívta eredetileg. Tehát nem kerül végtelen ciklusba.

A **B.** állítást egyszerű ellenőrizni, mivel a 12. sorban a rekurzív hívás található, amely pontosan annyiszor kerül végrehajtásra, ahány eleme van az ***a*** halmaznak. Tehát a **B.** válasz helyes.

A **C.** pontban leírtaknak megfelelően az ***n*** változó értékének 0-val történő összehasonlításainak száma egyenlő ***n***-nel. Ez hibás állítás, mivel az összehasonlítást előbb az eredeti ***n***-re, majd (***n*** – 1)-re, végül 0-ra végzi az algoritmus (ha ***n*** = 5, a második sorban található utasítást 6-szor hajtja végre az algoritmus).

A **D.** feltételezés sem lehet helyes. Ha például, az ***a*** halmaznak és a ***b***-nek is két eleme van: ***a*** = (1, 2) és a ***b*** = (3, 4), az egyesítés eredménye ***c*** = (1, 2, 3, 4).

Az **E.** feltételezés helyes, hiszen egy érték csak egyszer szerepelhet az egyesített halmazban. Ha például, ***a*** = (1, 2) és a ***b*** = (1, 2), az egyesítés eredménye ***c*** = (1, 2).

1. **Hatványra emelés**

Melyik alábbi algoritmus számítja ki helyesen ***ab*** értékét, ahol ***a*** és ***b*** természetes számok (1 ≤ ***a*** ≤ 11, 0 ≤ ***b*** ≤ 11)?

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Algoritmus** hatvány(a, b):  eredmény ← 1  **Amíg** b > 0 **végezd el**  **Ha** b **MOD** 2 = 1 **akkor**  eredmény ← eredmény \* a  **vége(ha)**  b ← b **DIV** 2  a ← a \* a  **vége(amíg)**  **térítsd** eredmény  **Vége(algoritmus)** |  | **Algoritmus** hatvány(a, b):  **Ha** b ≠ 0 **akkor**  **Ha** b **MOD** 2 = 1 **akkor**  **térítsd** hatvány(a \* a, b / 2) \* a  **különben**  **térítsd** hatvány(a \* a, b / 2)  **vége(ha)**  **különben**  **térítsd** 1  **vége(ha)**  **Vége(algoritmus)** |
|  | **Algoritmus** hatvány(a, b):  eredmény ← 1  **Amíg** b > 0 **végezd el**  eredmény ← eredmény \* a  b ← b - 1  **vége(amíg)**  **térítsd** eredmény  **Vége(algoritmus)** |  | **Algoritmus** hatvány(a, b):  **Ha** b = 0 **akkor**  **térítsd** 1  **vége(ha)**  aux ← hatvány(a, b **DIV** 2)  **Ha** b **MOD** 2 = 0 **akkor**  **térítsd** aux \* aux  **különben**  **térítsd** a \* aux \* aux  **vége(ha)**  **Vége(algoritmus)** |
|  | **Algoritmus** hatvány(a, b):  **Ha** b = 0 **akkor**  **térítsd** 1  **vége(ha)**  **térítsd** a \* hatvány(a, b - 1)  **Vége(algoritmus)** |  |  |

**Megoldás**

Nagy előnyt jelent, ha ismerjük *al-Kwarîzmi*, „gyorshatvány” néven ismert algoritmusát. Ennek iteratív változatát az **A.** pontban látjuk, a rekurzívat pedig a **B.** pontban. Hasonlóan párba állítható a két – hatványt számoló – naiv algoritmus: A **C.** az iteratív, az **E.** a rekurzív változat. Ezek mind helyesek. A **D.** algoritmust összehasonlítjuk a **B.** – szintén rekurzív – algoritmussal és belátjuk, hogy ugyanaz a hatásuk, csak **D.** felhasznál egy segédváltozót, amelybe ideiglenesen elmenti az aktuális hívás által térített értéket. Tehát, mind az öt algo­ritmus helyes.

**B.1. Polinom értéke**

Adott a kiértékelés(n, egyh, x) algoritmus, ahol ***egyh*** egy ***n* + 1** elemű, valós számokat tároló sorozat, amelynek értékei a [-100, 100] intervallumhoz tartoznak és amelyek az ***n*** fokú ***P*(*x*)** = ***egyh*1 \* *xn* + *egyh*2 \* *xn* - 1 + .... + *egyhn* \* *x* + *egyhn*+ 1** polinom együtthatói, ***x*** csökkenő hatványainak sorrendjében (***n*** természetes szám, 1 ≤ ***n*** ≤ 10). Az algoritmus meghatározza a polinom értékét egy adott ***x*** pontban (***x*** valós szám, amely a [-10, 10] intervallumhoz tartozik.

|  |
| --- |
| **Algoritmus** kiértékelés(n, egyh, x):  érték ← 0.0  **Minden** i ← 1, n + 1 **végezd el**  érték ← érték \* x + egyh[i]  **vége(minden)**  **térítsd** érték  **Vége(algoritmus)** |

Írjátok le a kiértékelés(n, egyh, x) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval.

**Megoldás**

Az értékelés a következő szempontok szerint történt:

* A rekurzív algoritmus fejléce azonos az iteratív fejlécével? (2 pont)
* Rekurzív hívás (2 pont)
  + A rekurzív hívás a megfelelő helyen található? (1 pont)
  + A paraméterezés helyes-e? (1 pont)
* Helyes értéket térít leálláskor? (2 pont)
* A folytatáskor térített érték helyes-e? (2 pont)

Egy lehetséges algoritmus:

|  |
| --- |
| **Algoritmus** kiértékelésRek(n, egyh, x):  **Ha** n = 0 **akkor**  **térítsd** egyh[1]  **különben**  **térítsd** egyh[n + 1] + x \* kiértékelésRek(n - 1, egyh, x)  **vége(ha)**  **Vége(algoritmus)** |

**Mat-Infó verseny, 2017**

Adott a következő algoritmus, amelynek három, nem nulla természetes szám bemeneti paramétere van: ***a****,* ***b*** és ***sz***, amelyeknek értékei kisebbek, mint 10 000:

|  |
| --- |
| **Algoritmus** f(a, b, sz):  k ← 0  **Amíg** b < sz **végezd el:**  k ← k + 1  b ← a + b  a ← b - a  **vége(amíg)**  **térítsd** k  **Vége(algoritmus)** |

* 1. Adjátok meg a feladat szövegét, amelyet ez az algoritmus old meg, ha ***a*** = 1 és ***b*** = 0 értékekre hívjuk meg.
  2. Mit térít az f(1,0,10) hívás?
  3. Írjátok le a fenti algoritmusnak egy *rekurzív* változatát, megőrizve az iteratív (nem rekurzív) változat fej­lécét.

**Megoldás**

Adva van egy algoritmus és meg kell fogalmaznunk a feladat kijelentését, amelyet ezzel az algoritmussal oldunk meg. Ha a versenyző találkozott ezzel az algoritmussal, azonnal tudni fogja, hogy az adott ***sz*** számnál kisebb *Fibonacci* számokat generálja, megszámolja ezeket ***k***-ban és téríti ezt a számot. Ha nem ismeri fel az algoritmus funkcióját, ellenőrző táblázatot készít:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***a*** | ***b*** | ***sz*** | ***k*** |  | A táblázatban látható, hogy az ***a***, illetve a ***b*** értékei, rendre *Fibonacci* számok.A ***k*** értéke a ***b*** -ben generált, ***sz***-nél kisebb számok darabszáma. Így a válaszok:   1. Az algoritmus az ***sz***-nél szigorúan kisebb *Fibonacci* számok darabszámát határozza meg. 2. Ha az algoritmust az f(1,0,10) utasítással hívjuk meg, a térített érték 7. 3. Egy helyes rekurzív algoritmus: |
| 1 | 0 | 10 | 0 |  |
| 0 | 1 |  | 1 |  |
| 1 | 1 |  | 2 |  |
| 1 | 2 |  | 3 |  |
| 2 | 3 |  | 4 |  |
| 3 | 5 |  | 5 |  |
| 5 | 8 |  | 6 |  |

|  |
| --- |
| **int** f\_Rek(**int** a, **int** b, **int** szam){  **if** (b < szam)  **return** f\_Rek(b, a + b, szam) + 1;  **else**  **return** 0;  } |

**Felvételi verseny, 2017**

Legyen a következő alprogram, ahol ***n*** bemeneti paraméter, ***p*** és ***i*** kimeneti paraméterek (***n***, ***p***, ***i*** – termé­szetes számok, 1 ≤ ***n*** ≤ 1 000 000, (0 ≤ ***p*** ≤ 1 000 000, 0 ≤ ***i*** ≤ 1 000 000):

|  |
| --- |
| **Algoritmus** f(n, p, i):  **Ha** n ≤ 9 **akkor**  **Ha** n **mod** 2 = 0 **akkor**  p ← n  i ← 0  **különben**  p ← 0  i ← n  **vége(ha)**  **különben**  f(n **div** 10, p, i)  **Ha** n **mod** 2 = 0 **akkor**  p ← p \* 10 + n **mod** 10  **különben**  i ← i \* 10 + n **mod** 10  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** |

1. Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
2. Mi lesz ***p*** és ***i*** értéke az f(205609, p, i) hívás után?
3. Írjátok le egy iteratív változatát az adott algoritmusnak, megőrizve a rekurzív változat fejlécét.

**Megoldás**

Ahhoz, hogy megírjuk az algoritmus iteratív változatát, előbb meg kell értenünk az adott algoritmus funk­cióját. Azt, hogy számjegyekre bontja az adott ***n*** számot, hamar belátjuk, de azt is meg kell értenünk, hogy mit ábrázol a ***p*** és az ***i*** változó.

A rekurzív hívások leállnak, amikor az ***n*** szám egyszámjegyű lett. Ekkor adunk kezdőértékeket ***p***-nek és ***i***-nek. Ha ***n*** páros, ***p*** értéke ***n***, ***i*** értéke pedig 0 lesz. Más szóval, ***p*** kezdőértéke az a páros értékű számjegy, amely az ***n*** szám ismételt osztásai után maradt belőle. Ha ***n*** páratlan, a két kezdőértékadás máris segít a feladat funk­ciójának megállapításában, hiszen most ***p*** kezdőértéke 0, és ***i*** kezdőértéke ***n***. Így megállapítjuk, hogy ***p*** az ***n*** szám páros számjegyeiből, ***i*** a páratlanokból álló számok értéke lesz. A számjegyek az eredeti sorrendben épülnek a ***p***-be és az ***i***-be.

Válaszolunk a kérdésekre:

1. Az algoritmus meghatározza az ***n*** szám páros, illetve páratlan számjegyeit – az eredeti sorrendben – tartal­mazó számokat.
2. Ha meghívjuk az algoritmust az f(205609, p, i), ***p*** = 2060, ***i*** = 59 lesz.
3. Egy lehetséges helyes iteratív algoritmus, amelynek a fejléce azonos a rekurzív algoritmus fejlécével:

|  |
| --- |
| **void** fIterativ\_1(**int** n, **int** & paros, **int** & paratlan){  paros = 0; paratlan = 0;  **int** hatvanyParatlan = 1;  **int** hatvanyParos = 1;  **while** (n){  **if** (n & 1){  paratlan = paratlan + hatvanyParatlan \* (n % 10);  n /= 10;  hatvanyParatlan \*= 10;  }  **else**{  paros = paros + hatvanyParos \* (n % 10);  n /= 10;  hatvanyParos \*= 10;  }  }  } |

A következő, kicsit hosszabb algoritmus is helyes:

|  |
| --- |
| **void** fIterativ\_2(**int** n, **int** & paros, **int** & paratlan){  paros = 0; paratlan = 0;  **int** hatvanyParos = 0;  **int** hatvanyParatlan = 0;  **while** (n > 0){  **int** utolsoSzj = n % 10;  **if** (utolsoSzj % 2 == 0){  **if** (paros == 0 && hatvanyParos == 0){  paros = utolsoSzj;  hatvanyParos = 10;  }  **else** {  paros = paros + utolsoSzj \* hatvanyParos;  hatvanyParos \*= 10;  }  }  **else** {  **if** (paratlan == 0){  paratlan = utolsoSzj;  hatvanyParatlan = 10;  }  **else** {  paratlan = paratlan + utolsoSzj \* hatvanyParatlan;  hatvanyParatlan \*= 10;  }  }  n /= 10;  }  } |

**Matek-Infó verseny, 2016**

Legyen a következő alprogram, ahol az ***a*** bemeneti paraméter természetes szám (0 < ***a*** ≤ 30 000):

|  |
| --- |
| **Algoritmus** F(a)**:**  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)** |

**a.** Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.

**b.** Mit térít az F(2103) hívás?

**c.** Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív algoritmus fejlécével.

**Megoldás**

Előbb nézzük meg, mi a hatása az algoritmusnak? Látható, hogy a ***b*** változó értékét téríti, amely az algo­ritmus elején 0 kezdőértéket kap. A feldolgozás során ***b*** értékébe rendre beépülnek az ***a*** szám páratlan szám­jegyei balról jobbra, az eredeti sorrendjükben. Ennek érdekében az algoritmus felhasználja a ***p*** változót, amely a 10-nek különböző hatványait tárolja, a hatványkitevő növekvő sorrendjében. Így a ***b*** első számjegye az ***a*** szám első páratlan számjegye lesz. Ha az ***a*** szám nem tartalmaz egyetlen páratlan számjegyet sem, ***b*** értéke 0 marad.

Válaszolunk a kérdésekre:

**a.** Az algoritmus az ***a*** szám páratlan számjegyeiből felépíti a ***b*** számot, megőrizve a számjegyek eredeti sor­rendjét. Ha az ***a*** számnak nincs páratlan számjegye, ***b*** értéke 0 lesz.

**b.** F(2103) = 13

**c.** Egy lehetséges, helyes rekurzív algoritmus, amelynek fejléce azonos az iteratív algoritmus fejlécével:

|  |
| --- |
| **int** FRek(**int** a){  **if** (a < 1)  **return** a;  **else**{  **int** c = a % 10;  **if** (c % 2 != 0)// páratlan számjegy esete  **return** c + 10 \* FRek(a / 10); // beépítjük a térítendő értékbe  **else**  **return** FRek(a / 10); // páros számjegy, figyelmen kívül hagyjuk  }  } |

**Felvételi verseny, 2016**

Adott a következő alprogram, ahol az ***a*** és ***b*** (0 < ***a*** ≤ 10 000, 0 ≤ ***b*** ≤ 10 000) természetes számok bemeneti paraméterek.

|  |
| --- |
| **Algoritmus** F(a, b):  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)**  **térít** c  **Vége(algoritmus)** |

* 1. Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
  2. Mit térít az F(1002,6) hívás?
  3. Írjátok le egy *rekurzív* változatát a fenti iteratív (nem rekurzív) algoritmusnak. A fejléce legyen azonos a fenti algoritmus fejlécével.

**Megoldás**

**a.** Az algoritmus ismerősnek tűnhet sok feladatmegoldó számára, mivel hasonlít *al-Kwarîzmi* módszeréhez, amely hatékonyan számítja ki ***ab*** értékét. De van egy furcsa különbség: a térített érték, amely itt a ***c*** válto­zóban jön létre, a gyorshatvány néven ismert algoritmusban, a **Ha** utasítás akkor ágán a ***c*** \* ***a*** új értéket kapja. Itt viszont (c \* a) **mod** 10 található. A másik különbség a **Ha** utasítás után látható, ahol ***a*** \* ***a*** helyett (a \* a) **mod** 10 található. Ha nem jövünk rá, hogy az algoritmus mit számol ki, végrehajthatjuk egy-két egyszerű példára, amíg világossá nem válik, hogy a térített érték az ***ab*** érték utolsó számjegye. (5 pont)

**b.** F(1002,6) = 4 (3 pont)

**c.** Az algoritmusnak egy lehetséges, helyes rekurzív változata, amelynek a fejléce azonos az iteratív algoritmu­séval:

|  |
| --- |
| **Algoritmus** FRek(a, b):  **Ha** b = 0 **akkor**  **térítsd** 1  **különben**  **Ha** b **mod** 2 = 1 **akkor**  **térítsd** a **mod** 10 \* FRek(a\*a **mod** 10, b **div** 2) **mod** 10  **különben**  **térítsd** FRek(a\*a **mod** 10, b **div** 2) **mod** 10  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** |

**Kitűzött feladatok**

1. **Hatvány**

A következő rekurzív algoritmus minimális számú szorzással számítja ki az ***xn*** hatvány értékét (***x*** – valós szám, 0 < ***x*** ≤ 10, ***n*** – természetes szám, 1 ≤ ***n*** ≤ 40).

Állapítsátok meg, hány szorzást végez el az algoritmus, ha ***x*** és ***n*** alább megadott értékeire hívjuk meg.

|  |  |
| --- | --- |
| **Algoritmus** f(x, n)  **Ha** n = 0 **akkor**  **térítsd** 1  **különben**  **Ha** n *páratlan szám* **akkor**  **térítsd** f(x \* x, n **DIV** 2) \* x  **különben**  **térítsd** f(x \* x, n **DIV** 2)  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** | 1. Ha ***x*** = 1.5 és ***n*** = 10, akkor a szorzások száma= 6 2. Ha ***x*** = 2.55 és ***n*** = 1, akkor a szorzások száma= 1 3. Ha ***x*** = 3.14 és ***n*** = 32, akkor a szorzások száma = 7 4. Ha ***x*** = 0.5 és ***n*** = 1, akkor a szorzások száma= 2 |

1. **Számolás – karakterekkel**

Legyen a számolásKarakterekkel(s, n, i, szám) algoritmus, ahol ***s*** egy ***n*** karakterből álló sorozat (***n*** természetes szám, 1 ≤ ***n*** ≤ 200), ***i*** és ***szám*** természetes számok (1 ≤ ***i*** ≤ ***n***).

|  |
| --- |
| **Algoritmus** számolásKarakterekkel(s, n, i, szám):  eredmény ← 0  **Amíg** i ≤ n **végezd el**  **Amíg** i ≤ n **és** s[i] ≥ '0' **és** s[i] ≤ '9' **végezd el**  szám ← szám \* 10 + s[i] - '0'  i ← i + 1  **vége(amíg)**  eredmény ← eredmény + szám  szám ← 0  i ← i + 1  **vége(amíg)**  **térítsd** eredmény  **Vége(algoritmus)** |

Írjátok le a számolásKarakterekkel(s, n, i, szám) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval. Az alábbi programrészletből hívjuk meg:

|  |
| --- |
| **Beolvas:** n, s  **KiÍr:** számolásKarakterekkel(s, n, 1, 0) |

1. **Iteratívból rekurzív (1)**

Legyen a következő alprogram, ahol az ***a*** és ***b*** bemeneti paraméterek természetes számok (0 < ***a*** ≤ 1 000 és 0 < ***b*** ≤ 1 000):

|  |  |
| --- | --- |
| **Algoritmus** f(a, b):  eredmény ← 0  **Amíg** a > 0 **végezd el:**  **Ha** a **mod** 2 = 1 **akkor**  eredmény ← eredmény + b  **vége(ha)**  a ← a **div** 2  b ← b + b  **vége(amíg)**  **térít** eredmény  **Vége(algoritmus)** | * 1. Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.   2. Mit térít az f(5, 15) hívás?   3. Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív (nem rekurzív) algoritmus fejlécével. |

1. **Iteratívból rekurzív (2)**

Adott a következő alprogram, ahol az ***sz*** tetszőleges természetes szám bemeneti paraméter (1 < ***sz*** ≤ 10 000):

|  |  |
| --- | --- |
| **Algoritmus** F(sz)**:**  a ← 1  b ← 0  **Amíg** sz > a + b **végezd el:**  a ← a + b  b ← a - b  **vége(amíg)**  **Ha** sz = a + b **akkor**  **térítsd true**  **különben**  **térítsd false**  **vége(ha)**  **Vége(algoritmus)** | 1. Adjátok meg annak a feladatnak a szövegét, amelyet ez az algorit­mus old meg. 2. Mit térít az F(17) hívás? 3. Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív (nem rekurzív) algoritmus fejlécével. |

1. **Iteratívból rekurzív (3)**

Adott a következő alprogram, ahol az ***n*** és ***k*** bemeneti paraméterek (1 < ***n*** ≤ 30 000 és 1 < ***k*** ≤ 30 000) és természetes számok:

|  |  |
| --- | --- |
| **Algoritmus** F(n, k)**:**  érték ← 0  **Amíg** n ≥ k **végezd el:**  n ← n **div** k  érték ← érték + 1  **vége(amíg)**  **térít** érték  **Vége(algoritmus)** | 1. Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg. 2. Mit térít az F(98, 2) hívás? 3. Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív (nem rekurzív) algoritmus fejlécével. |

1. **Előszelet (2)**

*Sors-számjegynek* hívjuk azt a természetes számot, amelyet adott természetes számra a következőképpen számítunk ki: összeadjuk a szám számjegyeit, majd a kapott összeg számjegyeit, és így tovább, amíg a kapott összeg nem válik egyszámjegyű számmá. Például, a 182 *sors-számjegye* 2 (1 + 8 + 2 = 11, 1 + 1 = 2).

Egy pontosan ***k*** számjegyű ***p*** számot egy legkevesebb ***k*** számjegyű ***q*** szám *előszeleté*nek nevezünk, ha a ***q*** szám első ***k*** számjegyéből alkotott szám (balról jobbra tekintve) egyenlő ***p***-vel. Például, 17 előszelete 174-nek, és 1713 előszelete 1 713 242-nek.

Legyen az ***sz*** természetes szám (0 < ***sz*** ≤ 30 000) és egy ***m*** soros és ***n*** oszlopos (0 < ***m*** ≤ 100, 0 < ***n*** ≤ 100) ***A*** mátrix (kétdimenziós tömb), amelynek elemei 30 000-nél kisebb természetes számok. Adva van a sorsSzámjegy(x) algoritmus, amely meghatározza az ***x*** számhoz rendelt sors-számjegyet:

|  |
| --- |
| **Algoritmus** sorsSzámjegy(x):  s ← 0  **Amíg** x **>** 0 **végezd el**  s ← s + x **MOD** 10  x ← x **DIV** 10  **Ha** x = 0 **akkor**  **Ha** s < 10 **akkor**  **térítsd** s  **különben**  x ← s  s ← 0  **vége(ha)**  **vége(ha)**  **vége(amíg)**  **térítsd s**  **Vége(algoritmus)** |

**Követelmények:**

1. Írjátok le egy *rekurzív* változatát (ismétlő struktúrák nélkül) a sorsSzámjegy(x) algoritmusnak. A fejléce és a hatása legyen azonos a fenti algoritmus fejlécével és hatásával.
2. Írjátok le a sorsSzámjegy(x) algoritmus rekurzív változatának (amelyet kidolgoztatok az **a.** pontnál) a ma­tematikai modelljét (vagyis, írjátok le a rekurzív függvényt matematikai képlet formájában).
3. Írjatok alprogramot, amely – felhasználva a sorsSzámjegy(x) alprogramot – meghatározza az ***sz*** szám leg­hosszabb előszeletét(***prefix***), amelyet az adott tömb elemeinek megfelelő *sors-számjegyeiből* fel lehet építeni. Egy ilyen sors-számjegyet akárhányszor fel lehet használni. Ha nem építhető fel előszelet, ***prefix*** = -1. Az alprogram bemeneti paraméterei: ***sz***, ***m***, ***n*** és az ***A*** mátrix, kimeneti paramétere: ***prefix***.

***Példa*:** ha ***sz*** = 12319, ***m*** *=* 3, ***n*** *=* 4 és a mátrix: , akkor a leghosszabb előszelet ***prefix*** = 1231, a megfelelő sors-számjegyek pedig:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mátrixelem értéke | 182 | 12 | 274 | 22 | 1 | 98 | 56 | 5 | 301 | 51 | 94 |
| Sors-számjegy | 2 | 3 | 4 | 4 | 1 | 8 | 2 | 5 | 4 | 6 | 4 |

1. **Bűvös számok (2)**

Legyen két természetes szám ***p*** és ***q*** (2 ≤ ***p*** ≤ 10*,*2 ≤ ***q*** ≤ 10). Egy természetes számot *bűvös*neknevezünk, ha a ***p*** számrendszerben felírt alakjában szereplő számjegyek halmaza azonos a ***q*** számrendszerben felírt alak­jában szereplő számjegyek halmazával. Például. ha ***p*** = 9 és ***q*** = 7, (31)10 *bűvös szám*, mivel (34)9 = (43)7; ha ***p*** = 3 és ***q*** = 9, (9)10 *bűvös szám*, mivel (100)3 = (10)9. Adott még a számjegyek(x, b, c) alprogram, amely meghatározza az ***x*** szám számjegyeit a ***b*** számrendszerben (a ***c*** sorozatban):

|  |
| --- |
| **Algoritmus** számjegyek(x, b, c):  **Amíg** x **>** 0 **végezd el**  c[x **MOD** b] ← 1  x ← x **DIV** b  **vége(amíg)**  **Vége(algoritmus)** |

**Követelmények:**

1. Írjátok le egy *rekurzív* változatát (ismétlő struktúrák nélkül) a számjegyek(x, b, c) algoritmusnak. A fej­léce és a hatása legyen azonos a fenti algoritmus fejlécével és hatásával.
2. Írjátok le a számjegyek(x, b, c) algoritmus rekurzív változatának (amelyet kidolgoztatok az **a.** pontnál) matematikai modelljét, (vagyis, írjátok le a rekurzív függvényt matematikai képlet formájában).
3. Írjatok alprogramot, amely – felhasználva a számjegyek(x, b, c) alprogramot – adott ***p*** és ***q*** számrendsze­rek ismeretében, meghatározza azt a *bűvös* számokból álló ***a*** sorozatot, amely minden 0-nál szigorúan na­gyobb és adott ***n*** (1 < ***n*** ≤ 10 000) természetes számnál szigorúan kisebb számot tárol. Az alprogram beme­neti paraméterei ***p*** és ***q*** (a két alap) és az ***n*** szám. Kimeneti paraméter az ***a*** sorozat és ennek ***k*** hossza.

***Példa:*** ha ***p*** = 9, ***q*** = 7 és ***n*** = 500, az ***a*** sorozatnak ***k*** = 11 eleme lesz: (1, 2, 3, 4, 5, 6, 31, 99, 198, 248, 297).

1. **Törzstényezők**

Írjatok rekurzív algoritmust, amely kiír egy ***n*** természetes számot (1 ≤ ***n*** ≤ 29) törzstényezőkre bontva!

1. **Tökéletes szám**

Írjatok rekurzív algoritmust, amely ellenőrzi, hogy az ***n*** természetes szám (1 ≤ ***n*** ≤ 29) tökéletes szám-e (egyenlő-e a nála kisebb osztóinak összegével)!

1. ***n* szám legnagyobb közös osztója**

Írjatok rekurzív algoritmust, amely kiszámítja ***n*** (1 ≤ ***n*** ≤ 10 000) darab szám legnagyobb közös osztóját! A számok természetes számok és mind kisebbek, mint 29.

1. **Fordított szám**

Írjatok rekurzív algoritmust, amely megfordítja az ***n*** természetes szám számjegyeinek sorrendjét és visszatéríti a megfordított számot! Az alprogram bemeneti és kimeneti paramétere egyben az ***n*** (1 ≤ ***n*** ≤ 29).

1. **Palindromszó**

Írjatok rekurzív algoritmust, amely ellenőrzi, hogy egy beolvasott karakterlánc palindrom tulajdonságú-e (tükörszó?)! A karakterlánc hossza kisebb, mint 250.