1. **Mit ír ki? (6 pont)**

Adott az alábbi program. Állapítsátok meg, mit ír ki a program a végrehajtás eredményeként.

|  |  |
| --- | --- |
| 1. P(a, b) = 253;   a = 11; b = 11; | 1. P(a, b) = 132;   a = 11; b = 121; |
| 1. P(a, b) = 22;   a = 11; b = 11; | 1. P(a, b) = 253;   a = 11; b = 121; |

|  |  |
| --- | --- |
| **C++** | **C** |
| **#include<iostream>**  **using namespacestd;**  **int** P(**int** x, **int** &y){  y = y \* x;  x = x + y;  **return** x + y;  }  **int** main(){  **int** a = 11; **int** b = a;  cout << "P(a, b) = " << P(a, b);  cout << endl << "; a = " << a;  cout << "; b = " << b << endl;  **return** 0;  } | **#include<stdio.h>**  **int** P(**int** x, **int** \*y){  \*y = (\*y) \* x;  x = x + (\*y);  **return** x + (\*y);  }  **int** main(){  **int** a = 11; **int** b = a;  printf("P(a, b) = %d", P(a, &b));  printf(";\na = %d", a);  printf("; b = %d\n", b);  **return** 0;  } |
| **Pascal** | |
| **function** P(x : Integer; **var** y : Integer) : Integer;  **begin**  y := y \* x;  x := x + y;  P := x + y;  **end**;  **var** a, b : Integer;  **Begin**  a := 11; b := a;  Write('P(a, b) = ', P(a, b))  WriteLn('; ');  Write('a = ', a);  WriteLn('; b = ', b);  **End**. | |

**Megoldás**

Hasonlóan járunk el, mint a Mat–Infó verseny **A.1.** feladatának megoldásakor.

A program itt is rögzíti a bemenet értékeit: ***a*** = 11, ***b*** = ***a***, tehát ***b*** = 11. Fontos észrevennünk, hogy ***x*** *érték* szerint átadott, míg ***y*** *cím* (illetve *referencia*) szerint átadott paraméter. Következik, hogy ***x*** értéke a függvényen belül 132 lesz, de a hívás helyén megőrzi értékét (11). Az ***y*** paraméter értéke a függvényen belül 121 lesz, ez a hívás helyén is érvényes lesz, a függvény által térített érték pedig ***x*** + ***y*** = 132 + 121 = 253. Tehát a helyes válasz: **D**.

1. **Kiértékelés (6 pont)**

Adott az ***n*** elemű (1 ≤ ***n*** ≤ 10 000), 30 000-nél kisebb természetes számokat tároló ***v*** sorozat. Állapítsátok meg, hogy a következő programrészletek közül me­lyiket kellene a „…” helyére írni, ahhoz, hogy a feldolgoz(v, n, er, m) algo­ritmus ismétlő struktúrájának végrehajtása után az ***er*** sorozat, a ***v*** sorozat azon elemeit tárolja, amelyek 5-nek azon többszörösei, amelyek páros indexű helyeken találhatók. Az ***er*** sorozat hosszát az ***m*** változó tárolja.

|  |
| --- |
| **Algoritmus** feldolgoz(v, n, er, m)  m ← 0  **Minden** i = 1, n **végezd el**  ...  **vége(minden)**  **Vége(algoritmus)** |

|  |  |  |  |
| --- | --- | --- | --- |
| **A.** | **Ha** i **MOD** 2 = 0 **és** v[i] **MOD** 5 = 0 **akkor**  m ← m + 1  er[m] ← v[i]  **vége(ha)** | **B.** | **Ha** i **MOD** 2 = 0 **akkor**  **Ha** v[i] **MOD** 10 = 5 **akkor**  m ← m + 1  er[m] ← v[i]  **vége(ha)**  **Ha** v[i] **MOD** 10 = 0 **akkor**  m ← m + 1;  er[m] ← v[i]  **vége(ha)**  **vége(ha)** |
| **C.** | **Ha** v[i] **MOD** **10** = 0 **akkor**  **Ha** i **MOD** 2 = 0 **akkor**  m ← m + 1  er[m] ← v[i]  **különben**  **Ha** v[i] **MOD** 10 = 5 **akkor**  m ← m + 1;  er[m] ← v[i]  **vége(ha)**  **vége(ha)**  **vége(ha)** | **D.** | **Ha** i **MOD** 2 = 0 **és** v[i] **MOD** 10 = 5  **és** v[i] **MOD** 10 = 0 **akkor**  m ← m + 1  er[m] ← v[i]  **vége(ha)** |

**Megoldás**

A **Ha** utasítások feltételeit (a logikai kifejezéseket) fogjuk vizsgálni. Az **A.** változatban az (i **MOD** 2 = 0 **és** v[i] **MOD** 5 = 0) kifejezés akkor térít *igaz*-at, ha az aktuális elem indexe páros és az elem 5-nek többszöröse. Tehát **A.** helyes.

A **B**. algoritmusrészlet csak akkor foglalkozik egy adott elemmel, ha az indexe páros, majd két egymás után írt **Ha** utasításban foglalkozik az elem értékével. Mindkét logikai kifejezés helyes és szükséges, hiszen 5 többszöröseinek utolsó számjegye vagy 0 vagy 5. Így **B**. is helyes.

A **C.** algoritmus csak 10 többszöröseit dolgozza fel, tehát hibás.

A **D.** szintén hibás, mivel a logikai kifejezés második és harmadik részkifeje­zése nem lehet *igaz* egyszerre (v[i] **MOD** 10 = 5 **és** v[i] **MOD** 10 = 0).

1. **Mely értékek szükségesek? (6 pont)**

Legyen a számol(n, v) algoritmus, ahol ***v*** egy ***n*** egész számot tároló sorozat (***n*** – természetes szám, 1 ≤ ***n*** ≤ 1 000).

Állapítsátok meg, hogy a paraméterek mely értékeire térít az algoritmus 0-t.

|  |  |
| --- | --- |
| **Algoritmus** számol(n, v)  a ← 0  b ← 0  **Minden** i = 1, n **végezd el**  a ← a + v[i]  **vége(minden)**  **Minden** i = 1, n **végezd el**  a ← a – v[i]  **Ha** a = b **akkor**  **térítsd** v[i]  **vége(ha)**  b b + v[i]  **vége(minden)**  **térítsd** -1  **Vége(algoritmus)** | 1. ***n*** = 5, ***v*** = (4, 5, 7, 3, 6) 2. ***n*** = 7, ***v*** = (-3, 1, 2, 0 , 5,-2, -3) 3. ***n*** = 4, ***v*** = (-2, 2, 5, -5) 4. ***n*** = 8, v = (1, -7, 3, 0, -2, 1, -2, 0) |

**Megoldás**

Az algoritmus leáll, és téríti az aktuális elem értékét, ha a két lokális változó (***a*** és ***b***) értéke egyenlő. Mivel csak a **B.** és **D.** sorozatokban található 0 értékű elem, ezért csak ezeket fogjuk vizsgálni, hiszen az algoritmus egy sorozatelemet térít, amelynek a követelmény szerint 0-nak kell lennie. A kezdőértékadások után (***a*** = 0 és ***b*** = 0), az algoritmus előbb összeadja a sorozat elemeit ***a***-ban. A **B.**-ben megadott sorozat esetében ez az összeg 0. A sorozat második bejárásakor, ebből az összegből (vagyis ***a***-ból) rendre kivonjuk az aktuális elem értékét és ugyanezt az elemet hozzáadjuk ***b***-hez. Ezeket a műveleteket követjük a következő táblázatban.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Elem sorszáma*** | ***a*** | ***b*** | ***vi*** | ***a***-t ***b***-vel a kivonás után hasonlítja össze az algoritmus |
|  | 0 | 0 |  |  |
| 1 | 0 – (-3) = 3 | 0 + (-3) = -3 | -3 | ***a*** = 3, ***b*** = 0 |
| 2 | 3 – 1 = 2 | -3 + 1 = -2 | 1 | ***a*** = 2, ***b*** = -3 |
| 3 | 2 – 2 = 0 | -2 + 2 = 0 | 2 | ***a*** = 0, ***b*** = -2 |
| 4 | 0 – 0 = 0 |  | 0 | ***a*** = ***b*** = 0, és ***vi*** = 0 |

Következik a táblázatból, hogy a **B.** sorozat esetében, az algoritmus 0-t térít.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Elem sorszáma*** | ***a*** | ***b*** | ***vi*** | ***a***-t ***b***-vel a kivonás után hasonlítja össze az algoritmus |
|  | -6 | 0 |  |  |
| 1 | -6 – 1 = -7 | 0 + 1 = 1 | 1 | ***a*** = 3, ***b*** = 0 |
| 2 | -7 – (-7) = 0 | 1 + (-7) = -6 | -7 | ***a*** = 2, ***b*** = -3 |
| 3 | 0 – 3 = -3 | -6 + 3 = -3 | 3 | ***a*** = 0, ***b*** = -2 |
| 4 | -3 – 0 = -3 |  | 0 | ***a*** = ***b*** = 0, és ***vi*** = 0 |

Tehát, a **D.** pontban megadott sorozat esetében szintén 0-t térít az algoritmus.

1. **Vajon mit csinál? (6 pont)**

Legyen a találdKi(n) algoritmus, ahol ***n*** természetes szám (1 ≤ ***n*** ≤ 10000). Állapítsátok meg a találdKi(n) algoritmus hatását.

|  |  |
| --- | --- |
| **Algoritmus** találdKi(n)  f ← 0  p ← -1  **Minden** c = 0, 9 **végezd el**  x ← n  k ← 0  **Amíg** x > 0 **végezd el**  **Ha** x **MOD** 10 = c **akkor**  k ← k + 1  **vége(ha)**  x ← x **DIV** 10  **vége(amíg)**  **Ha** k > f **akkor**  p ← c  f ← k  **vége(ha)**  **vége(minden)**  **térítsd** p  **Vége(algoritmus)** | 1. Kiszámítja és visszatéríti a 10-es számrend­szerben megadott ***n*** szám számjegyeinek da­rabszámát. 2. Kiszámítja és visszatéríti a 10-es számrend­szerben megadott ***n*** szám legnagyobb szám­jegyének darabszámát. 3. Kiszámítja és visszatéríti a 10-es számrend­szerben megadott ***n*** szám egyik olyan szám­jegyét, amelynek előfordulási száma maxi­mális. 4. Kiszámítja és visszatéríti a 10-es számrend­szerben megadott ***n*** szám, ***c***-vel egyenlő számjegyeinek darabszámát. |

**Megoldás**

Az algoritmus generálja ***c***-ben a számjegyeket 0-tól 9-ig, majd megszámolja, minden számjegy esetében, hogy ezek hányszor fordulnak elő az ***n*** számban. Az előfordulások ***k*** számát felhasználja ahhoz, hogy aktualizálja ***f***-ben az előfordulá­sok maximumát és ***p***-ben az ehhez az előforduláshoz tartozó számjegyet. Tehát a helyes válasz **C**. Itt fölösleges a többi válasszal foglalkozni, a tartalmuk nem lehet helyes, tekintve, hogy a kiválasztott helyes választól eltérő hatást jelölnek meg.

1. **Sajátos sorozat generálása (6 pont)**

Ismert, hogy egy ***k*** elemű ***x*** sorozat (***x*** = (***x***1, ***x***2, ***x***3, ..., ***xk***)) ***p*** hosszúságú tömb­szakasza az ***x*** sorozat ***p*** darab eleméből áll, amelyek egymás utáni helyeket fog­lalnak el. Például, ***y*** = (***x***3, ***x***4, ***x***5, ***x***6) egy ***p*** = 4 hosszúságú tömbszakasz.

Legyen az ***M*** ={1, 2, ..., ***n***} számjegyeket tároló halmaz, és az ***M*** halmaz ***n***! darab permutációja (***n*** – természetes szám, ***n*** ≤ 9). Létre lehet hozni azt a legrö­videbb ***ss*** számjegysorozatot, amely ***M*** számjegyeit tárolja és az ***M*** halmaz összes ***n*!** permutációja megtalálható az ***ss***-ben, ***n*** hosszúságú tömbszakaszok formájá­ban.

Például, ha ***n*** = 3, a permutációk száma 6, az ***ss*** sorozatot 9 számjegy alkotja és lehetne, például ***ss*** = (1, 2, 3, 1, 2, 1, 3, 2, 1). A ***perm*1** = (1, 2, 3) permutációnak felhasználjuk az utolsó két számjegyét és hozzáfűzünk egy harmadik számjegyet, ahhoz, hogy egy másik permutációt kapjunk, vagyis a ***perm*2** = (2, 3, 1) permutá­ciót; majd a ***perm*2** két utolsó számjegye után helyezhetjük a 2-es számjegyet és megkapjuk a ***perm*3** = (3, 1, 2) permutációt. Ha az (1, 2) után írnánk a 3-as szám­jegyet, megkapnánk a ***perm*1** permutációt, amely már létezik az eddig generált számjegysorozatban. Ekkor ***perm*3**-nak csak az utolsó számjegyét használjuk fel és egy olyan permutációt keresünk, amely a 2-es számjeggyel kezdődik és még nincs benne a sorozatban stb. Így, a létrehozott sorozatban egyetlen olyan három számjegyből álló tömbszakasz található, amely nem permutáció: (1, 2, 1); a többi tömbszakasz, (vagyis (2, 1, 3), (1, 3, 2) és (3, 2, 1)) helyesek.

Állapítsátok meg, hogy az ***M*** = {1, 2, 3, 4} halmaz esetében legkevesebb hány számjegy típusú eleme lesz az ***ss*** sorozatnak.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ***perm*1** | | |  | | | | | | | 1 | 2 | 3 | 1 | 2 | 1 | 3 | 2 | 1 | |  | ***perm*2** | | |  | | | | | |

**A.** 55 **B.** 16 **C.** 33 **D.** 37

**Megoldás**

Generáljuk a kért ***ss*** sorozatot. Ahhoz, hogy ne veszítsünk el egyetlen permu­tációt sem, kénytelenek vagyunk leírni ezeket. Amint egy permutációt elhelyez­tünk ***ss***-be, kihúzzuk, hogy ne használjuk fel még egyszer. A feladat szövege tar­talmazza a számjegysorozat létrehozatalának stratégiáját, tehát most csak arra kell vigyáznunk, hogy figyelmesen dolgozzunk és ne tévedjünk.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***Permutáció sorszáma*** | ***Perm.*** | ***Sorsz.*** | ***Perm.*** | ***Sorsz.*** | ***Perm.*** | ***Sorsz.*** | ***Perm.*** |
| **1.** | 1234 | **7.** | 2134 | **13.** | 3124 | **19.** | 4123 |
| **2.** | 1243 | **8.** | 2143 | **14.** | 3142 | **20.** | 4132 |
| **3.** | 1324 | **9.** | 2314 | **15.** | 3214 | **21.** | 4213 |
| **4.** | 1342 | **10.** | 2341 | **16.** | 3241 | **22.** | 4231 |
| **5.** | 1423 | **11.** | 2413 | **17.** | 3412 | **23.** | 4312 |
| **6.** | 1432 | **12.** | 2431 | **18.** | 3421 | **24.** | 4321 |

Elkezdjük az „építkezést”. Minden lépésben keresünk egy olyan permutációt, amely az aktuális permutációnak utolsó három számjegyével kezdődik. Ez csak a 19. permutáció beszúrásáig sikerül. Ekkor egy olyan permutációt keresünk, amely az eddig generált számjegysorozat utolsó két számjegyével kezdődik. Ez a 9. Folytatjuk a számjegysorozat építését az eddig alkalmazott stratégia alapján.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Sorsz.*** | **1.** | **10.** | **17.** | **19.** | **9.** | **14.** | **5.** | **22.** |
| ***Perm.*** | 1234 | 2341 | 3412 | 4123 | 2314 | 3142 | 1423 | 4231 |
| ***Sorsz.*** | **13.** | **2.** | **12.** | **23.** | **7.** | **4.** | **18.** | **21.** |
| ***Perm.*** | 3124 | 1243 | 2431 | 4312 | 2134 | 1342 | 3421 | 4213 |
| ***Sorsz.*** | **3.** | **16.** | **11.** | **20.** | **15.** | **8.** | **6.** | **24.** | |
| ***Perm.*** | 1324 | 3241 | 2413 | 4132 | 3214 | 2143 | 1432 | 4321 | |

Ha most megszámoljuk a táblázatban fehéren maradt számjegyeket (vagyis azokat, amelyekre nem csúszott rá más számjegy az ideragasztott permutációból) megkapjuk a számjegysorozat hosszát. Ha generáltuk a számjegysorozatot, – ter­mészetesen – megszámolhatjuk az elemeit. A számjegyek száma 33, tehát a **C.** válasz a helyes.

1. **Számjegyszorzat (6 pont)**

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. **Sorozat feldolgozása (6 pont)**

Adott a feldolgoz(n, x) algoritmus, ahol ***x*** egy ***n* –** 1 elemű, különböző ter­mészetes számokat tároló sorozat, amely számok az {1, 2, …, ***n***} halmazhoz tar­toznak. Állapítsátok meg az algoritmus által visszatérített érték jelentését.

|  |
| --- |
| **Algoritmus** feldolgoz(n, x)  s ← 0  **Minden** i = 1, n - 1 **végezd el**  s ←s + x[i]  **vége(minden)**  **térítsd** n \* (n + 1) / 2 - s  **Vége(algoritmus)** |

1. Az algoritmus az első ***n*** nem nulla természetes szám összegének és az ***x*** soro­zat elemei összegének különbségét téríti vissza.
2. Az algoritmus az első ***n*** nem nulla természetes szám összegének és az ***x*** soro­zat elemei összegének (kivéve az utolsó elemet) különbségét téríti vissza.
3. Az algoritmus annak az {1,2, …, ***n***} halmazhoz tartozó természetes számnak az értékét téríti, amely nem szerepel az ***x*** sorozatban.
4. Az algoritmus 0-t térít vissza.

**Megoldás**

Az **A.** kijelentés egyszerű és könnyen ellenőrizhető, hogy helyes.

A **B.** esetében azonnal észrevesszük, hogy az algoritmus nem a feltételezett funkciót teljesíti. Egyrészt az ***x*** sorozat elemeinek darabszáma ***n*** – 1, így a záró­jelben található kitételből („kivéve az utolsó elemet”) az következik, hogy csak ***n*** – 2 elem összegét számolja ki az algoritmus. De ez nem igaz, hiszen a **Minden** ciklus ***i*** ciklusváltozójának értékei: 1, 2, …, ***n*** – 1, vagyis ***n*** – 1 elemet adunk össze (az ***x*** sorozat minden elemét).

A **C.** kijelentés meglepő, de miután észrevesszük, hogy az első ***n*** természetes szám összegéből ***n*** – 1 *különböző* természetes szám összegét vonjuk ki, amely számok az {1, 2, …, ***n***} halmazhoz tartoznak, egyértelműen következik, hogy a sorozatból hiányzik ennek a halmaznak egy eleme. Az algoritmus pontosan ennek az értékét számolja ki. Például, ha ***n*** = 4, a halmaz = {1, 2, 3, 4}. Legyen ***x*** = (1, 2, 4). A halmaz elemeinek összege 10, a sorozat elemeinek összege pedig ***s*** = 7. A kettő különbsége 3, amely pontosan az ***x***-ből hiányzó elem értéke. Tehát, **C.** helyes.

A **D.** válasz szintén megtévesztő. Az algoritmus által térített érték csak akkor lehetne 0, ha az ***x*** sorozat elemeinek összege egyenlő lenne az első ***n*** természetes szám összegével. De ez lehetetlen, mivel az ***x*** sorozatnak csak ***n*** – 1 eleme van az {1, 2, …, ***n***} halmazból és ezek az elemek különbözők.

1. **Varázslat (6 pont)**

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, de más eltérést is tartalmaz. 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 eredményezett. 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 a 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.

1. **Egészítsétek ki (6 pont)**

Adott az ***n*** elemű (3 ≤ ***n*** ≤ 100), növekvően rendezett ***x*** sorozat, amely 30 000-nél kisebb különböző természetes számokat tartalmaz. A legközelebbi(x, bal, jobb, ér) algoritmus meghatározza az ***x*** sorozat legnagyobb értékű elemének pozícióját, amely a ***bal*** és ***jobb*** pozíciók között helyezkedik el (1 ≤ ***bal*** < ***jobb*** ≤ ***n***) és, amelynek az értéke kisebb, vagy egyenlő ***ér***-rel. Ha nem létezik ilyen elem, a legközelebbi(x, bal, jobb, ér) algoritmus 0-t térít vissza.

A modulusz(a) algoritmus az ***a*** egész szám abszolút-értékét téríti vissza.

A számol(n, x, adottSz) algoritmus meghatározza azt az elemét az ***x*** soro­zatnak, amely a legközelebb áll ***adottSz***-hoz. Ha két elem azonosan közel van ***adottSz*** értékéhez, az algoritmus a nagyobb számot határozza meg.

Legyen ***n*** = 5, ***x*** = (5, 9, 11, 15, 99) és ***adottSz*** = 12. Állapítsátok meg melyik kifejezéssel helyettesíthető a „…” a legközelebbi(x, bal, jobb, ér) algoritmus­ban, ahhoz, hogy a számol(n, x, adottSz) algoritmus 11-et térítsen vissza.

|  |
| --- |
| **Algoritmus** legközelebbi(x, bal, jobb, ér)  **Ha** ér > x[jobb] **akkor**  **térítsd** jobb  **vége(ha)**  **Ha** ér < x[bal] **akkor**  **térítsd** bal - 1  **vége(ha)**  közép ← (bal + jobb) **DIV** 2  **Ha** ... **akkor**  **térítsd** közép - 1  **különben**  **Ha** ér < x[közép] **akkor**  **térítsd** legközelebbi(x, bal, közép - 1, ér)  **különben**  **térítsd** legközelebbi(x, közép + 1, jobb, ér)  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** |
| **Algoritmus** számol(n, x, adottSz)  i ← legközelebbi(x, 1, n, adottSz)  **Ha** i = 0 **akkor**  **térítsd** x[i + 1]  **különben**  **Ha** modulusz(x[i]- adottSz) < modulusz(x[i + 1] - adottSz) **akkor**  **térítsd** x[i]  **különben**  **térítsd** x[i + 1]  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** |

1. x[közép - 1] <= ér **és** ér < x[közép]
2. x[közép - 1] <= ér **vagy** ér < x[közép]
3. x[közép - 1] < ér **és** ér <= x[közép]
4. x[közép] <= ér **és** ér < x[közép - 1]

**Megoldás**

Az algoritmusban felismerjük a bináris keresést, azzal a különbséggel, hogy itt nem egy adott értékű elemet keresünk, hanem egy másikat, amelynek az értéke a legközelebb áll az adott számhoz. A számol(n, x, adottSz) algoritmus meg­hívja a legközelebbi(x, bal, jobb, ér) algoritmust, amely téríti az adott szám valamelyik szomszédos értékű elemének indexét. A továbbiakban, a térített indexnek megfelelő elem és az adott szám különbsége alapján eldönti, hogy me­lyik érték a legközelebbi, így ezzel a gonddal a legközelebbi(x, bal, jobb, ér) algoritmus nem foglalkozik. Következik, hogy azt az indexet térítjük, amelytől balra egy olyan elem található, amely kisebb, mint az adott szám, miközben jobbra egy nagyobb.

Ugyanakkor, a hiányzó feltételt tartalmazó utasítás fölöslegesnek tűnik, mivel sejtjük, hogy az algoritmus működése során, vagy valamelyik önmeghívás, vagy a két leállási feltétel közül valamelyik hajtódik majd végre. Megvizsgáljuk, mi történne, ha a ***közép*** kiszámítása utáni **Ha** (a felkínált feltételektől függetlenül) és ennek a **Ha**-nak az **akkor** ága hiányozna. Lássuk, hogyan hajtódik végre az algo­ritmus a fenti példa esetében (***n*** = 5, ***x*** = (5, 9, 11, 15, 99) és ***adottSz*** = 12):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***bal*** | ***jobb*** | ***közép*** | ***ér*** | *Végrehajtott utasítás* | *Magyarázat* |
| 1 | 5 | 3 | 12 | **Ha** ér > x[jobb] **akkor** | Mivel 12 < 99 a **Ha**-nak vége |
|  |  |  |  | **Ha** ér < x[bal] **akkor** | Mivel 12 > 5 a **Ha**-nak vége |
|  |  |  |  | **Ha** ér < x[közép] **akkor** | Mivel 12 > 11, a **különben**  ág következik (újra hívás) |
| 4 | 5 | 4 |  | **Ha** ér > x[jobb] **akkor** | Mivel 12 < 99 a **Ha**-nak vége |
|  |  |  |  | **Ha** ér < x[bal] **akkor** | Mivel 12 < 15, téríti ***bal*** – 1-et  vagyis 3-at (***x*3** = 11), vége |
|  |  |  |  |  |

Az egyetlen eset, amelynek megfelelően a tárgyalt **Ha** utasítást végrehajtja a program, a **B.** eset. Ekkor a térített érték 2, vagyis ***x*2** = 9. De mégsem ez lesz a végeredmény, erről gondoskodik a számol(n, x, adottSz) alprogram.

Mivel beláttuk, hogy a hiányzó feltételt tartalmazó utasítás fölösleges, válasz lehetne **A.**, **B.**, **C.**, **D.**, tehát, mindegyiket helyesnek nyilvánítjuk.

1. **Lovagok és hazugok (6 pont)**

Egy szigeten csak lovagok élnek, akik mindig igazat mondanak és hazugok, akik mindig hazudnak. Egy látogató, aki a szigetre érkezett szeretné eldönteni két helybéli lakos természetét, akikkel találkozna a szigeten. Így, amikor találkozik két lakossal, A-val és B-vel, felteszi a ***Q*1** kérdést A-nak: „Mindketten lovagok vagytok?” de a kapott válasz (***V*1**) alapján nem tudja eldönteni, hogy milyen ter­mészetű a két lakos. Ezért, a látogató feltesz egy újabb ***Q*2** kérdést A-nak: „Egyformák vagytok, mindketten lovagok vagy mindketten hazugok?”; ezúttal, a kapott válasz (***V*2**) alapján a látogató már megvilágosodik (vagyis, most már tudja, hogy melyik lakos lovag és melyik hazug).

Az alábbi változatok közül, melyik felel meg az A lakos válaszainak, tudva azt, hogy a látogató pontosan meg tudta állapítani a két helybéli lakos természetét.

1. ***V*1**: Igen, ***V*2**: Igen
2. ***V*1**: Igen, ***V*2**: Nem
3. ***V*1**: Nem
4. A **B.** és **C.** válaszok helyesek

**Megoldás**

Mivel a **C.** csak a ***V*1** választ adja meg, ellentmondásba kerültünk a kijelentés­sel, miszerint az első válasz a látogatónak nem volt elegendő. A **D.** pont szerint a **B.** és **C.** válaszok helyesek. Ez megint nem lehetséges, hiszen a **C.**-ről már eldön­töttük, hogy nem helyes. Ahhoz, hogy az **A.** és **B.** válaszokat vizsgálhassuk, ké­szítünk egy táblázatot. A fejlécben feltüntetünk minden lehetséges esetet, ami a két lakos természetét illeti. A következő sor A válaszait tartalmazza a megfelelő esetekben:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A / B** | **L / L** | **L / H** | **H / L** | **H / H** |
| ***V*1** | igen | nem | igen | igen |
| ***V*2** | igen | nem | igen | nem |

Ha feltételezzük, hogy A és B is lovag, a válaszok „igen” + „igen”. Sajnos, ugyanezt a választ kapjuk akkor is, ha A hazug és B lovag. Tehát *nem* tudjuk, hogy az A lakos lovag vagy hazug.

A „nem” + „nem” válaszokat nem értékeljük, mivel nem szerepelnek a lehet­séges válaszok között a feladatban.

Marad az A lakos „igen” + „nem” válasza. Ha A lovag lenne, az első válaszból az derülne ki, hogy lovagok mindketten, de a második válasz ennek ellentmond. Marad a feltételezés, hogy A hazug, de akkor mindkét válasza hamis, vagyis *nem* mindketten lovagok, de *egyformák*, vagyis hazugok. Így a **B.** válasz a helyes.

**B Rész (30 pont)**

**Banánok**

Egy hajótörés után a tengerészeknek sikerült a mentőcsóna­kokkal megmenekülni. Minden csónakban volt három tenge­rész és minden csónak más-más szigeten vetődött partra. Éle­lem után kellett nézniük és minden ***i***. szigeten a tengerészek összegyűjtöttek ***bi*** banánt, amelyeket elraktározták a csónakjuk­ba, mivel úgy döntöttek, hogy csak másnap osztják szét egymás között. Bármely csónakban legtöbb ***k*** banán fér el. Az éjszaka folyamán, az egyik szigeten, az egyik tengerész felkelt és szétosztotta a csónakban levő banánokat három részre, minden halomba ugyanannyi banánt tett, de maradt még egy banán, amit nem tehetett egyik halomba sem, így ezt megette. Ezután, az egyik részt elrejtette, a másik két részt visszatette a csónakba és lefeküdt aludni. Reggelig, minden ten­gerész, mindegyik szigeten elvégezte ugyanezt a titkos beavatkozást (elosztotta a banánokat három egyenlő részre és megette az egyetlen megmaradt banánt). Reggel, mindegyik szigeten, a megmaradt banánokat három egyenlő (nem üres) részre osztották és megint maradt egy banán, amit a tengerészek egy majomnak adtak. Igen, mindegyik szigeten élt egy-egy majom!☺

**Követelmények**:

1. Ha az egyik szigeten, az éjszaka beállta előtt, a tengerészek összegyűjtöttek 241 banánt, hány banán maradt egy-egy halomban az osztozkodás végén (ezen a szigeten)? (2 pont)
2. Ha az egyik szigeten, az osztozkodás végén minden halom 15 banánt tar­talmazott, és egy másik szigeten minden halom 31 banánt, hány banánt gyűjtöttek összesen a két szigeten? (2 pont)
3. Írjatok alprogramot, amely egy adott ***k*** számra kiszámítja azt a lehetséges legnagyobb ***bmax*** értéket, amely azoknak a banánoknak a száma, amelye­ket a tengerészek egy bizonyos szigeten összegyűjtöttek. Bemeneti para­méter: ***k***. Kimeneti paraméter: ***bmax***. Keressetek képletet, amely kiszámítja ***bmax*** legnagyobb értékét ***k*** függvényében. Magyarázzátok el az alkalma­zott gondolatmenetet. (***bmax***, ***k*** – természetes számok, 1 ≤ ***bmax*** ≤ ***k*,** 100 ≤ ***k*** ≤ 10 000 000). (14 pont)

***Példa:*** ha ***k*** = 200, akkor ***bmax*** = 160.

1. Írjatok alprogramot, amely egy adott ***k*** számra kiszámítja az összes szigeten összegyűjtött maximális banánszámok összegét (***összegMax***). Tudjuk, hogy minden szigeten a tengerészek különböző ***bi*** számú banánt gyűjtöttek (***bi*** – természetes szám, ***i*** = 1, 2, ..., ***sz*,** 1 ≤ ***bi*** ≤ ***k*,** 100 ≤ ***k*** ≤ 10 000 000). Bemeneti paraméterek: ***k*** és ***sz*** – a szigetek száma (***sz*** – természetes szám, 2 ≤ ***sz*** ≤10). Kimeneti paraméter: ***összegMax*** – az összes szigeten össze­gyűjtött maximális banánszámok összege. A ***k*** és ***sz*** változók értékei úgy vannak megadva, hogy a feladatnak legyen megoldása. (12 pont)

***Példa:*** ha ***k*** = 400, ***sz*** = 3, akkor a három szigeten rendre 322, 241 és 160 banánt gyűjtöttek. Következik, hogy az összes szigeten összegyűjtött ma­ximális banánszámok összege ***összegMax =*** 322 + 241 + 160 = 723.

**Megoldás**

Bevezetünk néhány jelölést, ahol ***n*** az összegyűjtött banánok száma.

* ***n*** = 3 \* ***p*** + 1 (eredetileg összegyűjtött banánok)
* 2 \* ***p*** = 3***t*** + 1 (az első éjszaka után maradt banánok)
* 2 \* ***t*** = 3***q*** + 1, (a második éjszaka után)
* 2 \* ***q*** = 3***r*** + 1 (a harmadik éjszaka után)
* reggel 3 \* ***r*** + 1 banánt osztanak szét, (egy végleges halomban ***r*** banán van)

Elvégezzük a behelyettesítéseket és megkapjuk, hogy ***r*** = (8 \* ***n*** – 65) / 81 (\*), ami az osztozkodás reggelén található banánok száma (***n***, vagyis az eredetileg összegyűjtött banánok számának függvényében). Innen következik, hogy 81-esé­vel haladunk ahhoz, hogy végeredményhez jussunk.

Mivel egész számok halmazán dolgozunk, következik, hogy ***r*** páratlan szám és arra is rájövünk (az alábbiak alapján), hogy ***r*** legkisebb lehetséges értéke 7, és ***n*** legkisebb lehetséges értéke 79:

* ha ***r*** = 1, 2 \* ***q*** = 4, 2 \* ***t*** = 7 – lehetetlen,
* ha ***r*** = 3, 2 \* ***q*** = 10, 2 \* ***t*** = 16, 2 \* ***p*** = 25 – lehetetlen
* ha ***r*** = 5, 2 \* ***q*** = 16, 2 \* ***t*** = 25 – lehetetlen
* ha ***r*** = 7, 2 \* ***q*** = 22, 2 \* ***t*** = 34, 2 \* ***p*** = 52, ***n*** = 79

Erre az eredményre jutunk a következőképpen is:

A fent kiszámolt (\*) reláció alapján ⇒ ***n*** = (81 \* ***r*** + 65) / 8 (\*\*). Ezt fel lehet írni így is: ***n*** = (80 \* ***r*** + 64) / 8 + (***r*** + 1) / 8. Ebből következik, hogy ***r*** + 1 a 8 többszöröse, így ***r*** legkisebb lehetséges értéke 7.

Válaszolunk a feladatban megfogalmazott kérdésekre:

1. (\*) ⇒***r*** = (8 \* ***n*** – 65) / 81. Behelyettesítve ***n***-et: (241 \* 8 – 65) / 81 = 23.
2. (\*\*) ⇒ ***n*** = (81 \* ***r*** + 65) / 8. Behelyettesítve a két szigeten összegyűjtött banánok számát: (81 \* 15 + 65) / 8 + (81 \* 31 + 65) / 8 = 482 banán.
3. Az algoritmust is megírjuk a képleteinkre támaszkodva: megkeressük azt a 79-nél nagyobb számot, 81-esével haladva, amelyre: ***n\_max*** = 79 + ***d*** \* 81, ahol ***n\_max***-ot helyettesítjük ***k***-val (lehetséges legtöbb banán), és meghatároz­zuk ***d***-t: ***d*** = (***k*** – 79) / 81.

Innen következik, hogy: ***bmax*** = 79 + [(***k*** – 79) / 81] \* 81.

|  |
| --- |
| **int** bananokV1(**int** k){  **return** 79 + (k - 79) / 81 \* 81;  } |

Az algoritmus dolgozhat egy „okos” ismétlő struktúrával (felhasználva a (\*) relációt):

|  |
| --- |
| **int** bananokV2(**int** k){  **while** ((8 \* k - 65) % 81 != 0){  k--;  }  **return** k;  } |

Ha nincs jobb ötletünk, olyan algoritmust írunk, amely szimulálja az esemé­nyeket:

|  |
| --- |
| **int** bananokV3(**int** k){  **int** n = k;  **while** (n >= 4){  **if** (n % 3 == 1){  **int** n1 = 2 \* n / 3;  **if** (n1 % 3 == 1){  **int** n2 = 2 \* n1 / 3;  **if** (n2 % 3 == 1){  **int** n3 = 2 \* n2 / 3;  **if** (n3 % 3 == 1)  **return** n;  }  }  }  n--;  }  } |

1. Fel kell dolgoznunk minden sziget esetében az eseményeket, tehát az algo­ritmusnak tartalmaznia kell egy ismétlő struktúrát a szigetek feldolgozására és ki kell számolnia a különböző szigeteken összegyűjtött banánok számát, amit összegeznie kell. Vigyáznunk kell arra is, hogy az egyes szigeteken be­gyűjtött banánok összege különböző legyen:

|  |
| --- |
| **int** bananokMindenSzigetV1(**int** k, **int** sz){  **int** osszegMax = 0;  **for**(**int** i = 1; i <= sz; i++){ // vesszük sorra a szigeteket  **int** ni = bananokV2(k); // az i. szigeten összegyűjtött banánok száma  osszegMax += ni; // az összes szigeten összegyűjtött banánok  k = ni - 1; // a következő szigeten már kisebb a k  } // így különböző ni értékeket generálunk  **return** osszegMax;  } |

Ha a két követelményt összefűzzük:

|  |
| --- |
| **int** bananokV4(**int** k, **int** b[], **int** & nb){  **int** n = 1;  nb = 0;  **while** (n <= k){  **if** ((8 \* n - 65) % 81 == 0){  nb++;  b[nb] = n;  }  n++;  }  **return** b[nb];  } |
| **int** mindenSzigetV2(**int** k, **int** sz){  **int** nb = 0;  **int** b[100000];  bananokV4(k, b, nb);  **int** s = 0;  **for**(**int** i = nb; (i >= 1 && sz > 0); i--){  s += b[i];  sz--;  }  **return** s;  } |