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

**9.** A bűvös(s, n) algoritmusnak bemeneti paraméterei az ***n*** egész szám (1 ≤ ***n*** ≤ 10000) és az ***n*** elemű ***s*** karakterlánc (***s***[1], ***s***[2], …, ***s***[***n***]).

**Algorithm** bűvös(s, n):

    i ← n

**While** 1 ≤ i **execute**

  **If** s[i] ≠ s[n - i + 1] **then**

**return** 0

**EndIf**

        i ← i - 1

**EndWhile**

**return** 1

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Ha ***s*** páros darabszámú karakterből áll, az algoritmus 1-et térít vissza.
2. Az algoritmus 1-et térít vissza, ha ***s*** palindrom.
3. Az algoritmusban van egy hiba, mivel az ***n*** – ***i*** + 1 kifejezés negatív értéket is felvehet a végrehajtás során.
4. Ha ***s*** csak alfanumerikus karaktereket tartalmaz, az algoritmus 1-et térít vissza.

**Megoldás**

A válaszok közül a leghamarabb a **C**-t szűrjük ki, mint hibás állítást, hiszen az ***i*** kezdőértéke ***n***, tehát ***n*** – ***i*** + 1 értéke legelőször 1 (pozitív), majd ***i*** csökken, tehát a kifejezés értéke minden lépésben nő 1-gyel.

A **D** állítás is hibás, hiszen sehol nem vizsgáljuk, hogy a karakterek alfanumerikusak-e?

Az **A** szintén hibás, hiszen – egyrészt sehol nem számoljuk a karakterlánc hosszának paritását, másrészt az algoritmus nem a darabszám alapján térít 1-et vagy 0-át, hanem a karaktereket hasonlítja össze páronként. Ha a karakterek száma páros, attól még előfordulhat az összehasonlított karakterpárban különböző, amikor 0-át fog téríteni.

Marad a **B**, és ha odafigyelünk az algoritmus logikájára hamar belátjuk, hogy akkor térít 1-et, ha az **If** utasításnak soha nem hajtja végre a **then** ágát, vagyis minden karakterpár megegyezik. Ha balról jobbra haladva ugyanazokat a karaktereket találjuk, mint jobbról balra haladva, a karakterlánc *palindrom.*

Megzavarhat valakit, hogy miután az esetlegesen létező középső elemet is összehasonlította önmagával, fölöslegesen tovább dolgozik: (az (***i***, ***n*** – ***i*** + 1) indexű karakterpárt már összehasonlította, de most veszi az (***n*** – ***i*** + 1, ***i***) indexű karakterpárt. Így az első karakterpár indexei (***n***, 1), az utolsóé pedig (1, ***n***).

Tehát **csak a B helyes**.

**17.** Ki szeretnénk írni egy négyzetet és az átlóit\* (csillag) és . (pont) karakterek segítségével (pont karaktereket a négyzet belső részének kitöltéséhez használunk, eltekintve az átlóktól). Az alábbi példában egy olyan négyzet látható, amelynek az oldalát ***n*** = 6 csillag alkotja. Ehhez a kiíráshoz összesen 28 csillagot és 8 pontot használtunk fel.

\* \* \* \* \* \*

\* \* . . \* \*

\* . \* \* . \*

\* . \* \* . \*

\* \* . . \* \*

\* \* \* \* \* \*

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Ha ***n*** = 5, fel fogunk használni 22 csillagot és 4 pontot.
2. Ha ***n*** = 7, fel fogunk használni 34 csillagot és 15 pontot.
3. Ha ***n*** = 7, fel fogunk használni 33 csillagot és 16 pontot.
4. Ha ***n*** = 18, fel fogunk használni 100 csillagot és 224 pontot.

**Megoldás**

Ahhoz, hogy az **A**-t tudjuk eldönteni, nem árt, ha rajzolunk, hiszen azt is észrevettük, hogy a rajz különbözni fog, ha ***n*** páratlan szám:

\* \* \* \* \*

\* \* . \* \*

\* . \* . \*

\* \* . \* \*

\* \* \* \* \*

Most van 2 \* 5 + 2 \* 3 + 4 + 1 = 21 csillag, tehát **A** nem helyes (a pontok száma jó).

Mivel **B** és **C** közül csak az egyik lehet helyes, vizsgáljuk, hogy mi történik, ha ***n*** = 7. Először is: nő a pontok száma. A fenti ábra lila csillagja fölötti egy pontot tartalmazó sor fölött lesz 3 pont stb. Tehát az ***n*** = 5 esetében látható pontok számához, hozzáadunk 12 pontot, lesz összesen 16 pont. A csillagok száma 7 \* 7 – 16 = 49 – 16 = 33, tehát **helyes a C.**

Ha az ***n*** = 6 esetében látjuk, hogy a kereten kívül, befele haladva (***n*** / 2 – 1)-szer van 4-4 csillag, ez az ***n*** = 18 esetében 8 \* 4 = 32. A kerethez használunk 2 \* 18 + 2 \* 16 csillagot. Ez összesen 32 + 36 + 32 = 100 csillag. A pontok száma: 18 \* 18 – 100 = 324 – 100 = 224. Így **D helyes.**

**18.** Legyen a mitCsinál(T, n, e) algoritmus, amelynek bemeneti paraméterei az ***n*** természetes számot tároló ***T***, növekvően rendezett sorozat (***T***[1], ***T***[2], …, ***T***[***n***]), valamint az ***n*** és ***e*** természetes szám (1 ≤ ***n***, ***e*** ≤ 10000):

**Algorithm** mitCsinál(T, n, e):

**If** e **MOD** 2 = 0 **then**

        a ← 1

        b ← n

**While** a ≤ b **execute**

            m ← (a + b) **DIV** 2

**If** e < T[m] **then**

                b ← m - 1

**else**

**If** e > T[m] **then**

                    a ← m + 1

**else**

**return** m

**EndIf**

**EndIf**

**EndWhile**

**return** 0

**else**

        c ← 1

        g ← 0

**While** (c ≤ n) **AND** (g = 0) **execute**

**If** e = T[c] **then**

                g = c

**EndIf**

            c ← c + 1

**EndWhile**

**return** g

**EndIf**

**EndAlgorithm**

Állapítsátok meg, hogy az alábbi állítások közül melyek igazak!

1. Az algoritmus 0-át térít vissza, ha az ***e*** szám nem található meg a ***T*** sorozatban.
2. Ha az ***e*** szám páratlan és megtalálható a ***T*** sorozatban, az algoritmus visszatéríti azt a pozíciót a ***T*** sorozatban, ahol az ***e*** megtalálható, felhasználva a bináris keresés algoritmusát.
3. Ha az ***e*** szám páratlan és megtalálható a ***T*** sorozatban, az algoritmus visszatéríti azt a pozíciót a ***T*** sorozatban, ahol az ***e*** megtalálható, felhasználva a szekvenciális keresés algoritmusát.
4. Az algoritmus visszatéríti az ***e*** szám pozícióját a ***T*** sorozatban.

**Megoldás**

Látható, hogy az algoritmus szándéka kétféle kereső algoritmust alkalmazni annak függvényében, hogy a keresett ***e*** szám páros vagy páratlan. Ha ***e*** páros a bináris keresés pszeudokódja következik. A ***B*** hibás, mivel páratlan ***e*** érték keresése szekvenciális kereséssel történik.

Mivel, ha ***e*** páratlan szám, a szekvenciális keresést alkalmazza az algoritmus, a **C helyes**.

A **D** nem helyes, mivel nem tárgyalja a sikertelen keresés esetét.

Az **A** helyes, mivel mindkét algoritmus 0-át térít vissza, ha az ***e*** szám nem található meg a ***T*** sorozatban.

**22.** Legyen az ***s***, természetes számokat tároló sorozat, ahol ***si*** = , (***i*** = 1, 2, ...).

A művelet a bal és a jobb operandus számjegyeinek (10-es számrendszerben) konkatenálását (egymás után ragasztását) jelöli, ebben a sorrendben, az ***x*** pedig egy természetes szám (1 ≤ ***x*** ≤ 99). Például, ha ***x*** = 3, az ***s*** sorozat elemei a következők: 3, 4, 43, 434, 43443, ... . Adott ***k*** természetes szám (1 ≤ ***k*** ≤ 30) esetében állapítsátok meg az ***s*** sorozat azon tagja számjegyeinek darabszámát, amely a sorozat ***k*1** számjegyből álló tagja előtt található, ahol ***k*1** a legkisebb olyan szám, amelyre ***k*** ≤ ***k*1** ≤ 30 és létezik ***k*1** számjegyű tag.

1. ha ***x*** = 15 és ***k*** = 8, a keresett tag számjegyeinek darabszáma 6.
2. ha ***x*** = 2 és ***k*** = 6, a keresett tag számjegyeinek darabszáma 6.
3. ha ***x*** = 14 és ***k*** = 27, a keresett tag számjegyeinek darabszáma 26.
4. ha ***x*** = 5 és ***k*** = 12, a keresett tag számjegyeinek darabszáma 8.

**Megoldás**

Generáljuk a sorozatot az **A** esetben:

15, 16, 1615, 161516, 1615161615, így ***k1*** = 10, és az előtte található tag számjegyeinek darabszáma 6. Tehát **A helyes.**

Generáljuk a sorozatot a **B** esetben:

2, 3, 32, 323, 32323, 32323323. Megint túlléptük a megadott k számjegydarabszámot, tehát ***k1*** = 8, előtte a tag 5 számjegyből áll, tehát **B** nem helyes.

Nem folytatjuk a generálást, mivel sok időt felemésztene, hanem gondolkozunk: a sorozat tagjainak darabszámai a Fibonacci sorozat értékeihez hasonlóan „nőnek”: Az előbbi példa esetében: 1, 1, 2, 3, 5, 8 stb. De ha az **A**-ban található példát nézzük: 2, 2, 4, 6, 10 stb., vagyis minden tagnak értéke nagyobb, mint az előző esetben. De, ha észrevesszük, hogy három esetünk van, akkor máris könnyű a számolás.

* + - 1. ***x*** < 9, az első és második tag is egyszámjegyű
      2. ***x*** = 9, az első tag egy-, a második kétszámjegyű
      3. ***x*** > 9, az első és második tag is kétszámjegyű

Ezután jöhet a Fibonacci modell: alkalmazzuk az ***x*** = 17 és ***k*** = 27 esetre és kiszámoljuk a tagok számjegyeinek darabszámát: 2, 2, 4, 6, 10, 16, 26, 42, … Mivel ***k*** = 27, következik, hogy ***k1*** = 42. Az előtte levő tag számjegyeinek darabszáma 26, tehát **C helyes**.

D: ha ***x*** = 5, ***k*** = 12, a sorozatban levő tagok számjegyeinek darabszáma: 1, 1, 2, 3, 5, 8, 13, … tehát k1 = 13, az előtte levő tag számjegyeinek darabszáma 8. Tehát **D is helyes.**

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

**2.** Adott az f(a, n) algoritmus, ahol ***n*** egy nullától különböző természetes szám (2 ≤ ***n*** ≤ 10000) és ***a*** egy egész számokból álló ***n*** elemű vektor (***a***[1], ***a***[2], ..., ***a***[***n***], -100 ≤ ***a***[***i***] ≤ 100, ***i*** = 1, 2, ..., ***n***). A ***b*** lokális változó egy vektor.

**Algorithm** f(a, n):

i ← 2

b[1] ← a[1]

**While** i ≤ n **execute**

b[i] ← b[i - 1] + a[i]

i ← i + 1

**EndWhile**

**return** b[n]

**EndAlgorithm**

A következő állítások közül melyek igazak?

1. Az algoritmus az ***a*** vektorösszes elemének összegét téríti vissza.
2. Az algoritmus az ***a*** vektorutolsó két elemének összegét téríti vissza.
3. Az algoritmus az ***a*** vektorutolsó elemét téríti vissza.
4. Az algoritmus az ***a*** vektorutolsó ***n*** – 1 elemének összegét téríti vissza.

**Megoldás**

A ***b*** sorozat aktuális eleme „viszi tovább” az ***a*** sorozat elemeinek összegét. Tehát **A helyes**. Ebből következik, hogy **B**-t és **C**-t elvetjük. A **D** esetben megvizsgáljuk, hogy az ***a*** sorozat utolsó eleme bekerül-e az összegbe. Mivel a **While** struktúra ***i*** = ***n*** értékre dolgozik utoljára, látható, hogy a

b[i] ← b[i - 1] + a[i] utasítás értelmében „bekerül” az összegbe a[n] is, tehát **D** nem helyes.

**5.** Adott az f(x, n) algoritmus, ahol ***n*** természetes szám (2 ≤ ***n*** ≤ 10000) és ***x*** egy természetes számokból álló ***n*** elemű sorozat (***x***[1], ***x***[2], ..., ***x***[***n***], 1 ≤ ***x***[***i***] ≤ 10000, ***i*** = 1, 2, ..., ***n***).

**Algorithm** f(x, n):

**For** i = 1, n – 1 **execute**

**If** x[i] = x[i + 1] **then**

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

**EndIf**

**EndFor**

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

**EndAlgorithm**

A következő állítások közül melyek igazak?

1. Az algoritmus *False*-t térít vissza, ha az ***x*** sorozat két tetszőleges eleme különbözik.
2. Az algoritmus *False*-t térít vissza, ha az ***x*** sorozat két tetszőleges eleme egyenlő.
3. Az algoritmus *False*-t térít vissza, ha az ***x*** sorozat két egymást követő eleme egyenlő.
4. Az algoritmus *False*-t térít vissza, ha az ***x*** sorozat első két eleme egyenlő.

**Megoldás**

Ha figyelünk, **A**-t és **B**-t azonnal elvetjük, mivel nem tetszőleges elemeket hasonlít össze az algoritmus, hanem egymást követőket. Így a **C-t helyesnek** ítéljük meg. Még el kell eldöntenünk, hogy mi legyen a **D** állítással? A **For** ciklus első lépésében pontosan eaz első lét elemet hasonlítja össze az algoritmus, és ha ez a két elem egyenlő, *False*-t térít, tehát **D is helyes**.

**9.** Melyek lehetnek egy olyan vektor elemei, amelyre, ha alkalmazzuk a bináris keresés algoritmusát és a 36-os értéket keressük, az összehasonlítások egymás után rendre a 12, 24, 36 értékekkel történnek?

1. [2, 4, 7, 12, 24, 36, 50]
2. [2, 4, 8, 9, 12, 16, 20, 24, 36, 67]
3. [4, 8, 9, 12, 16, 24, 36]
4. [12, 24, 36, 42, 54, 66]

**Megoldás**

Természetesen ismernünk kell a bináris keresés algoritmusát!

**A**: [2, 4, 7, 12, 24, 36, 50], a középső elem = 12; 12 < 36, a jobb részsorozatban folytatjuk, most a középső érték pontosan 36, így vége. Tehát **A** nem helyes.

**B.** [2, 4, 8, 9, 12, 16, 20, 24, 36, 67], a középső elem = 12; 12 < 36, a jobb részsorozatban folytatjuk, most a középső elem 24; 24 < 36, a jobb részsorozatban folytatjuk, most az összehasonlítandó érték pontosan a 36, így vége. Tehát **B** **helyes**.

**C.** [4, 8, 9, 12, 16, 24, 36], a középső elem = 12; 12 < 36, a jobb részsorozatban folytatjuk, most a középső elem 24; 24 < 36, a jobb részsorozatban folytatjuk, most az összehasonlítandó érték pontosan a 36, így vége. Tehát **C** **helyes**.

**D**: [12, 24, 36, 42, 54, 66], az első összehasonlítandó elem pontosan 36, így vége. Tehát **D** nem helyes.

**14.** Adott a feldolgozás(s1, hossz1, s2, hossz2) algoritmus, ahol ***s*1** és ***s*2** két karakterlánc, amelyeknek a hossza ***hossz*1** és ***hossz*2** (1 ≤ ***hossz*1**, ***hossz*2** ≤ 1000). A két karakterlánc csak olyan karaktereket tárol, amelyeknek ASCII kódja az [1, 125] intervallumhoz tartozik. Az ***x*** lokális változó vektor. Az ascii(s, i)algoritmus az ***s*** karakterlánc ***i.*** karakterének ASCII kódját téríti vissza.

**Algorithm** feldolgozás(s1, hossz1, s2, hossz2):

**For** i = 1, 125 **execute**

x[i] ← 0

**EndFor**

**For** i = 1, hossz1 **execute**

x[ascii(s1, i)] ← x[ascii(s1, i)] + 1

**EndFor**

**For** i = 1, hossz2 **execute**

x[ascii(s2, i)] ← x[ascii(s2, i)] - 1

**EndFor**

ok ← ***True***

**For** i = 1, 125 **execute**

**If** x[i] ≠ 0 **then**

ok ← ***False***

**EndIf**

**EndFor**

**return** ok

**EndAlgorithm**

Adjátok meg az algoritmus hatását.

1. Az algoritmus *True*-t térít vissza, ha az ***s*1** és ***s*2** karakterlánc hossza azonos, különben *False*-t.
2. Az algoritmus *True*-t térít vissza, ha az ***s*1** és ***s*2** karakterláncok ugyanazokat a karaktereket tartalmazzák, minden karakter esetében ugyanazzal a gyakorisággal, különben *False*-t.
3. Az algoritmus *True*-t térít vissza, ha egyenként mindkét ***s*1** és ***s*2** karakterláncban előfordul minden karakter, amelyeknek ASCII kódja az [1, 125] intervallumhoz tartozik, különben *False*-t.
4. Az algoritmus *True*-t térít vissza, ha az ***s*1** és ***s*2** karakterláncok különböző karakterekből állnak, különben *False*-t.

**Megoldás**

Ki kell hámoznunk, hogy mit csinál az algoritmus. Előbb létrehoz a két karakterlánc alapján egy olyan gyakoriságtömböt, amelyben előbb nyilvántartásba veszi az ***s*1** karakterlánc elemeinek ASCII kódját, majd „kiveszi” az ***s*2** karakterlánc elemeinek ASCII kódját. A továbbiakban bejárja 1-től 125-ig az ASCII kódokat és vizsgálja, hogy a kódnak megfelelő elem nulla-e? Ha nem, *False*-t térít, tehát tulajdonképpen azt vizsgálja, hogy az ***x*** gyakoriságtömb tömb elemei nullák-e mind vagy sem. Ez a következővel egyenértékű: az ***s*1** karakterlánc elemei, sorrendtől függetlenül azonosak-e az ***s*2** karakterlánc elemeivel? Tehát, nem a halmazaikat hasonlítja, hanem az elemeket a gyakoriságaikkal. Ezt megtaláljuk a **B** állításban, amely, tehát **helyes**.

Visszatérünk az **A**-hoz: itt csak a hosszúságok egyenlőségéről van szó, ami nem elég, mint feltétel, hiszen abc-nek ugyanaz a hossza, mint xyz-nek. Így **A** nem helyes.

A **C** állítás azért nem helyes, mert az ***x*** tömb elemeit 0-val hasonlítja és akkor térít *False*-t ha talál nem nulla elemet.

A **D** állítás a **B** állítással ellentétes és pontatlan, mivel nem beszél a karakterek darabszámáról (gyakoriságáról), tehát nem helyes.

**16.** Adott az ***n*** elemű, természetes számokat tároló ***a*** vektor (***a***[1], ***a***[2], …, ***a***[***n***]), valamint az ***n*** és ***x*** természetes számok (1 ≤ ***n*** ≤ 10000). A következő programrészletek közül melyik írja ki azt a legkisebb pozíciót, ahol az ***a*** vektorban megtalálható az ***x*** érték, vagy -1-et ha ***x*** nem található meg az ***a*** vektorban?

|  |  |
| --- | --- |
| i ← 1  **While** (i ≤ n) **AND** (a[i] = x) **execute**  i ← i + 1  **EndWhile**  **If** i ≤ n **then**  **Write** i  **else**  **Write** –1  **EndIf**  i ← 1  **While** (i ≤ n) **AND** (a[i] = x) **execute**  i ← i + 1  **EndWhile**  **If** i = n + 1 **then**  **Write** i  **else**  **Write** –1  **EndIf** | i ← 1  **While** (i ≤ n) **AND** (a[i] ≠ x) **execute**  i ← i + 1  **EndWhile**  **If** i = n + 1 **then**  **Write** i  **else**  **Write** –1  **EndIf**  i ← 1  **While** (i ≤ n) **AND** (a[i] ≠ x) **execute**  i ← i + 1  **EndWhile**  **If** i ≤ n **then**  **Write** i  **else**  **Write** –1  **EndIf** |

**Megoldás**

Keresőalgoritmusok pszeudokódját kell megvizsgálnunk. Azt fogjuk keresni, hogy melyikben van hiba?

Az **A**-ban ott a hiba a **While** feltételében, mivel addig szeretné bejárni a sorozatot, amíg az elemek egyenlők ***x***-szel. Ha nem kellett bejárni végig a sorozatot, ***i*** értékét írja ki, vagyis az ***a*** sorozat azon első elemének indexét, amely nem egyenlő-e ***x***-szel.

A **B** algoritmusban csak az **If** hibás. Fel van cserélve a **then** ág az **else** ággal, tehát hibás.

A **C** algoritmusban is ugyanaz a **While**, mint A-ban, de különbözik az **If**. Ha sikerült bejárni végig a sorozatot, ***n*** + 1 értékét írja ki. Így azt vizsgálja, hogy az ***a*** sorozat minden eleme egyenlő-e ***x***-szel.

A **D** algoritmusban nincs hiba, tehát **helyes**.

**24.** Adott a det(a, n, m) algoritmus, ahol ***a*** egy ***n*** elemű, természetes számokból álló sorozat (***a***[1], ***a***[2], ..., ***a***[***n***], ha ***n*** ≥ 1) vagy az üres sorozat, ha ***n*** = 0. ***n*** és ***m*** természetes számok (0 ≤ ***n*** ≤ 100, 0 ≤ ***m*** ≤ 106).

**Algorithm** det(a, n, m):

**For** i ← 1, n – 1 **execute**

**For** j ← i + 1, n **execute**

**If** a[i] > a[j] **then**

tmp ← a[i]

a[i] ← a[j]

a[j] ← tmp

**EndIf**

**EndFor**

**EndFor**

i ← 1; j ← n; b ← ***False***

**While** i < j **execute**

**If** a[i] + a[j] = m **then**

b ← ***True***

**EndIf**

**If a**[i] + a[j] < m **then**

i ← i + 1

**else**

j ← j - 1

**EndIf**

**EndWhile**

**return** b

**EndAlgorithm**

A következő állítások közül melyek igazak?

1. Az algoritmus *True*-t térít vissza, ha az ***a*** sorozatban létezik egy számpár, amelynek összege ***m***.
2. Az algoritmus mindig *False*-t térít vissza.
3. Az algoritmus *False*-t térít vissza, ha ***n*** = 0.
4. Az algoritmus növekvő sorrendbe rendezi a sorozatot, majd más feldolgozásokat hajt végre.

**Megoldás**

Hosszú algoritmus, előbb megpróbáljuk kiszűrni a hibás feltételezéseket. Gyanús a ***B***, tehát megnézzük, hogy lehetséges-e, hogy az algoritmus mindig *False*-t térítsen. Valahol a közepénél ott az inicializálás: ***b*** (ezt téríti az algoritmus) *False* értéket kap, de következik egy feldolgozás, amikor lehetséges, hogy ***i*** < ***j*** indexértékekre az a[i] + a[j] = m feltétel igaz legyen, amikor ***b*** *True* értéket kap. Tehát **B** nem helyes.

A következő „furcsa” állítás a **C**. Az ***n*** = 0, tulajdonképpen azt jelenti, hogy az ***a*** sorozat üres, Követjük az algoritmus lépéseit: Mivel ***n*** = 0, az első **For**-ba nem lép be, következik az értékadás, amely ***i***-t 1-re állítja ***j***-t pedig ***n***-re. Így ***j*** = 0, tehát nem nagyobb, mint ***i***. Ebből következik, hogy az algoritmus egyszer sem hajtja végre a **While**-t és téríti ***b***-t, amely nem változtatta meg a kezdőértékét, ami *False* volt. Tehát **C helyes**.

Visszatérünk az **A**-hoz: „Az algoritmus *True*-t térít vissza, ha az ***a*** sorozatban létezik egy számpár, amelynek összege ***m***”. A **B** elemzése során láttuk, hogy ***b*** akkor kap *True* értéket, amikor az a[i] + a[j] = m feltétel igaz, és ezt a *True*-t sehol nem írja felül a későbbiekben *False*-szal. Tehát **A helyes**.

A **D** banális. Látható, hogy – valóban – rendezi előbb az ***a*** sorozatot, majd azon dolgozik, hogy értéket adjon a térítendő ***b***-nek. Tehát **D is helyes**.

**25.** Adott a bűvös(n, a) algoritmus, ahol ***a*** egy ***n*** elemű, természetes számokból álló vektor (***a***[1], ***a***[2], ..., ***a***[***n***], 1 ≤ ***n*** ≤ 10000).

**Algorithm** bűvös(n, a):

**If** n < 2 **then**

**return *False***

**EndIf**

**For** i ← 2, n **execute**

**If** a[i – 1] = a[i] **then**

**return *True***

**EndIf**

**EndFor**

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

**EndAlgorithm**

A következő állítások közül melyek igazak?

1. A bűvös(5, [2, 5, 4, 5, 4]) hívás eredményeként az algoritmus *False*-t térít vissza.
2. Az algoritmus akkor és csakis akkor dönti el, hogy az ***a*** vektorban léteznek duplák, ha a vektor növekvően/csökkenően rendezett.
3. A bűvös(9, [1, 2, 3, 4, 4, 5, 6, 7, 9]) hívás eredményeként az algoritmus *True*-t térít vissza.
4. A bűvös(5, [9, 5, 5, 2, 4]) hívás eredményeként az algoritmus *True*-t térít vissza.

**Megoldás**

Viszonylag könnyű belátni, hogy az algoritmus akkor térít *True*-t, amikor talált olyan két egymásutáni elemet, amelyeknek az értéke egyenlő. Ha nincs egymás után két azonos érték, *False*-t térít. Az **A**-ban megadott sorozatban nincs egymásutáni két azonos elem, tehát *False*-t fog téríteni. **A helyes**.

A **C**-ben megadott sorozatban van egymásutáni két azonos, tehát *True*-t fog téríteni. **C is helyes**.

A **D**-ben megadott sorozatban is van egymásutáni két azonos, tehát *True*-t fog téríteni. **D is helyes**.

Maradt a **B**, ahol a duplák keresését a sorozat rendezettségéhez köti. De ezt nem vizsgálja sehol, tehát **B** nem helyes.

**28.** Adott az AlexB(value, n, k, p) algoritmus, ahol ***value*** egy természetes számokból álló, ***n*** elemű sorozat (***value***[1], ***value***[2], ..., ***value***[***n***]), valamint ***n***, ***k*** és ***p*** természetes számok. Kezdetben a ***value*** sorozat ***n*** eleme 0-val egyenlő. A kiírás(value, n) algoritmus kiírja a ***value*** sorozatot egy sorba.

|  |  |
| --- | --- |
| **Algorithm** AlexB(value, n, k, p):  p ← p + 1  value[k] ← p  **If** p = n **then**  kiírás(value, n)  **else**  **For** i ← 1, n **execute**  **If** value[i] = 0 **then**  AlexB(value, n, i, p)  **EndIf**  **EndFor**  **EndIf**  p ← p - 1  value[k] ← 0  **EndAlgorithm** | Mit ír ki az algoritmus a 10. sorba, ha ***n*** = 5 és AlexB(value, 5, 1, 0) alakban hívjuk meg?   1. 1 5 2 3 4 2. 1 5 4 0 4 3. 5 5 5 5 5 4. 1 2 5 4 3 |

**Megoldás**

Megpróbáljuk megoldani a feladatot anélkül, hogy végrehajtanánk „kézzel” az algoritmust, amely létrehozza a ***value*** sorozatot. Talán a legkönnyebb belátni, hogy 0 értéket nem fog tartalmazni a sorozat, tehát **B** nem helyes. Az is látható, hogy nem generál olyan sorozatot, amely azonos értékeket tárolna, tehát **C** sem helyes.

Látjuk, hogy ***p*** 0 kezdőértékről indul és minden újrahíváskor nő 1-gyel. Sajnos, a kilépés előtt csökken 1-gyel, így nehezebb belátni, hogy mikor fog teljesülni a feltétel, miszerint ***n***-nel egyenlővé vált.

Mindenképpen, belátható, hogy az {1, 2, 3, 4, 5} halmaz olyan permutációit generálja és írja ki a program, amelyeknél az első elem mindig 1. Így 24 sorozatot fog kiírni.

Megpróbáljuk követni:

AlexB(value, 5, 1, 0) [0, 0, 0, 0, 0] =>

AlexB(value, 5, 1, 0) [1, 0, 0, 0, 0] =>

AlexB(value, 5, 2, 1) [1, 2, 0, 0, 0] =>

AlexB(value, 5, 3, 2) [1, 2, 3, 0, 0] =>

AlexB(value, 5, 4, 3) [1, 2, 3, 4, 0] =>

AlexB(value, 5, 5, 4) [1, 2, 3, 4, 5] => nő a ***p****,* ***n***-nel egyenlő, kiírja a ***value*** sorozatot (1)

AlexB(value, 5, 5, 3) [1, 2, 3, 0, 4] =>

AlexB(value, 5, 4, 4) [1, 2, 3, 5, 4] => kiírja a ***value*** sorozatot (2)

AlexB(value, 5, 4, 2) [1, 2, 0, 3, 0] =>

AlexB(value, 5, 3, 3) [1, 2, 4, 3, 0] =>

AlexB(value, 5, 5, 4) [1, 2, 4, 3, 5] => kiírja a ***value*** sorozatot (3)

AlexB(value, 5, 5, 3) [1, 2, 0, 3, 4] =>

AlexB(value, 5, 3, 4) [1, 2, 5, 3, 4] => kiírja a ***value*** sorozatot (4)

AlexB(value, 5, 5, 2) [1, 2, 0, 0, 3] =>

AlexB(value, 5, 3, 3) [1, 2, 0, 0, 3] =>

AlexB(value, 5, 4, 4) [1, 2, 4, 5, 3] => kiírja a ***value*** sorozatot (5)

AlexB(value, 5, 4, 3) [1, 2, 0, 4, 3] =>

AlexB(value, 5, 3, 4) [1, 2, 5, 4, 3] => kiírja a ***value*** sorozatot (6)

AlexB(value, 5, 3, 1) [1, 0, 2, 0, 0] =>

AlexB(value, 5, 2, 2) [1, 3, 2, 0, 0] =>

AlexB(value, 5, 4, 3) [1, 3, 2, 4, 0] =>

AlexB(value, 5, 5, 4) [1, 3, 2, 4, 5] => kiírja a ***value*** sorozatot (7)

AlexB(value, 5, 5, 3) [1, 3, 2, 0, 4] =>

AlexB(value, 5, 4, 4) [1, 3, 2, 5, 4] => kiírja a ***value*** sorozatot (8)

AlexB(value, 5, 4, 2) [1, 0, 2, 3, 0] =>

AlexB(value, 5, 2, 3) [1, 4, 2, 3, 0] =>

AlexB(value, 5, 5, 4) [1, 4, 2, 3, 5] => kiírja a ***value*** sorozatot (9)

AlexB(value, 5, 5, 3) [1, 0, 2, 3, 4] =>

AlexB(value, 5, 2, 4) [1, 5, 2, 3, 4] => kiírja a ***value*** sorozatot (10)

Megvan a sorozat, amely **A**-nál van feltüntetve. Tehát **A helyes.**