**Felvételi verseny (alapképzés) – 2016, július**

**Informatika írásbeli**

**I. TÉTELSOR**

**A versenyzők figyelmébe:**

1. A megoldásokat *pszeudokódban* vagy *programozási nyelvben* (*Pascal/C/C*++) írjátok!
2. A megoldásokra kapható pontszám első sorban az algoritmusok ***helyességétől*** függ, majd az algorit­musok ***hatékonyságától***, ami a *végrehajtási időt* és a *felhasznált memória méretét* illeti.
3. Feltétlenül írjatok ***megjegyzéseket*** (kommenteket) amelyek segítik az adott megoldás megértését (írjátok le a változók jelentését és a ***megoldás során alkalmazott ötleteiteket***).
4. Ne használjatok sajátos fejállományokban definiált függvényeket (például: *STL*, karakterláncokat fel­dol­gozó függvények stb.).

**I. Tétel (50 pont)**

1. **Tornyok (10 pont)**

Legyen megfelelő darabszámú azonos méretű érme, amelyekből tornyok építendők a következő szabályok alapján:

1. a legmagasabb torony magassága ***n*** (0 < ***n*** ≤ 13), a legkisebbnek a magassága 1;
2. a tornyok úgy kerülnek egymás mellé, hogy bármely két, azonos magasságú torony között létezik legalább egy magasabb torony, mint ez a kettő.

Írjatok alprogramot, amely kiszámítja azt a *legnagyobb toronyszámot* (***tornyokSzáma***), amelyek felépíthetők az adott szabályok alapján és az építkezéshez szükséges *érmék számát* (***érmékSzáma***). A legnagyobb torony­ma­gasság ***n*** értéke az alprogram bemeneti paramétere, a ***tornyokSzáma*** és az ***érmékSzáma*** kimeneti paraméterek.

***Példa:*** ha ***n*** = 3, ***tornyokSzáma*** = 7 és ***érmékSzáma*** = 11.

**2. Bűvös számok (20 pont)**

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. Példák: ha ***p*** = 9 és ***q*** = 7, (31)10 *bűvös zám*, mivel (34)9 = (43)7; ha ***p*** = 3 és ***q*** = 9, (9)10 *bűvös szám*, mivel (100)3 = (10)9.

Írjatok alprogramot, amely adott ***p*** és ***q*** számrendszerek ismeretében, meghatározza azt a *bűvös* számokból álló ***x*** 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 ***x*** sorozat és ennek ***k*** hossza.

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

**3. Beszúrás (20 pont)**

Adott az ***n*** elemű (0 < ***n*** ≤ 10 000) ***a*** sorozat, amelynek elemei 30 000-nél kisebb szigorúan pozitív termé­sze­tes szá­mo­k.

Írjatok alprogramot, amely a sorozat minden eleme után beszúr egy új számot, amely 2-nek az a legnagyobb hatványa, amely kisebb vagy egyenlő az adott elem értékével. Az ***a*** sorozat és az ***n*** értéke az alprogram beme­ne­ti és egyben kimeneti paraméterei.

***Példa:*** ha ***n*** = 4 és ***a*** = (3, 1, 24, 9), akkor az új ***a*** sorozat: (3, 2, 1, 1, 24, 16, 9, 8), valamint ***n*** = 8.

**II. Tétel (15 pont)**

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** { **MOD** *kiszámítja a* b *szám* 2**-***vel való egész osztási maradékát* }

c **←** (c \* a) **MOD** 10

**vége(ha)**

a **←** (a \* a) **MOD** 10

b **←** b **DIV** 2 { **DIV** *kiszámítja a* b *szám* 2**-***vel való egész osztási hányadosát* }

**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.

**III. Tétel (25 pont)**

**Szigetek**

Egy légitársaság az utasok rendelkezésére bocsátotta azoknak a földrajzi pontoknak a magasságait tartal­mazó sorozatot, amelyek fölött a repülőgép száll majd Kolozsvár és New York között. Az ***a*** sorozatnak ***n*** darab (3 ≤ ***n*** ≤ 10 000), 30 000-nél szigorúan kisebb természetes szám eleme van. A szárazföldnek megfele­lő pontok magas­ságai 0-tól különböznek, míg az óceánnak megfelelő pontok magasságai egyenlők 0-val. *Sziget*-nek olyan egymás utáni szárazföldnek megfelelő pontok sorozatát nevezzük, amely előtt és után víz található.

Írjatok programot amely:

1. Meghatározza és kiírja a leghosszabb sziget kezdetét, valamint a végét jelző pont sorszámát. Ha több megoldás létezik, csak egyet kell meghatároznotok. Ha nem létezik egyetlen sziget sem, akkor kiírja „*Nem létezik sziget*”.
2. Eldönti, hogy a leghosszabb sziget *hegy* típusú-e vagy sem és kiírja a „*Hegy*”, illetve a „*Nem hegy*” üzenetet. Egy szi­get *hegy* típusú, ha a felszínén található magasságok egy adott elemig szigorúan nö­vekvő – nem üres – sorozatot alkotnak, majd szigorúan csökkenőt, amely nem üres.
3. Eldönti, hogy a szárazföldnek megfelelő pontok magasságai páronként *különböző értékűek* vagy sem és kiírja a „*A magasságok különbözők*”, illetve a „*A magasságok nem különbözők*” üzenetet.
4. Ha a 3. pontban feltett kérdésre a válasz „*nem*”, meghatározza és kiírja a leggyakoribb magasság értékét és az előfordulásainak számát. Ha több ilyen magasság létezik, csak egyet kell megadnotok.

**1. *Példa:*** ha ***n*** = 15 és ***a*** = (10, 2, 1, 0, 7, 0, 1, 2, 13, 5, 0, 0, 8, 5, 2), összesen 2 sziget van, a leghosszabb sziget a 7. és 10. sorszámú magasságok között található. A legnagyobb sziget *hegy* típusú. A magasságok értékei nem különbözők, és a leggyakoribb magasság értéke 2, amely 3-szor fordul elő.

**2. *Példa:*** ha ***n*** = 10 és ***a*** = (1, 2, 0, 1, 2, 13, 0, 0, 1, 2), egyetlen sziget létezik, amely a 4. és 6. sorszámú pontok között található és *nem* *hegy* típusú. A magasságok értékei nem különbözők, az egyik leggyakoribb magasság értéke 1 és 3-szor fordul elő.

*Megjegyzés*: A példákban, az ***a*** sorozatot 1-től kezdve indexeltük.

Írjatok egy-egy alprogramot, amely:

1. beolvassa az ***a*** sorozat hosszát,ésaz ***a*** sorozatota billentyűzetről;
2. meghatározza a leghosszabb sziget kezdetének, valamint a végének megfelelő pont sorszámát;
3. eldönti, hogy egy sziget *hegy* típusú vagy sem;
4. eldönti, hogy a szárazföldnek megfelelő pontok magasságai páronként *különböző értékűek* vagy sem;
5. meghatározza a szárazföldön található leggyakoribb magasságot és ennek a magasságnak megfelelő elő­for­dulások számát.

**Megjegyzések:**

1. Minden tétel kidolgozása kötelező.
2. A megoldásokat a vizsgalapokra írjátok, (a piszkozatokat nem vesszük figyelembe).
3. Hivatalból jár 10 pont.
4. Rendelkezésetekre áll 3 óra.

**RÉSZLETES JAVÍTÁSI ÚTMUTATÓ – I. TÉTELSOR**

**HIVATALBÓL 10 pont**

**I. TÉTEL 50 pont**

**1. Tornyok** **10 pont**

* a *tornyokSzáma* és az *érmékSzáma* értékeinek helyes kiszámítása (***egyetlen*** ismétlő struktúrával)

6+4 pont

* a *tornyokSzáma* és az *érmékSzáma* értékeinek helyes kiszámítása (***két*** egymás után írt ismétlő struktúrával – ugyanabban az alprogramban vagy különböző alprogramokban) 4+2 pont

**2. Bűvös számok 20 pont**

* az ***x*** sorozat meghatározása, amely az ***n***-nél kisebb bűvös számokat tárolja (a *bűvös* tulajdonság meghatározása a számjegyhalmazok azonossága alapján, ha a számot a ***p*** és a ***q*** számrendszerben ábrázoljuk) 20 pont
* ha minden számjegyet tárol 10 pont

**3. Beszúrás 20 pont**

* helyes beszúrás az ***a*** sorozat egyetlen feldolgozásával, más sorozat felhasználása *nélkül* 20 pont
* helyes beszúrás, és felszabadítja az új elem helyét a sorozat elemeinek *jobbra tolásával*

***n***-szer (*ismételt bejárások*), más sorozat felhasználása *nélkül* 15 pont

* helyes beszúrás, *egyetlen* *bejárással*, és egy ***b*** *segédsorozat* létrehozásával; majd a ***b*** sorozatot

rámásolja az ***a*** sorozatra 10 pont

* helyes beszúrás, *egyetlen* *bejárással*, *két segédsorozat* felhasználásával: a ***b*** sorozatban tárolja

2-nek hatványait, amelyekre az ***a*** sorozat elemeinek megfelelően szükség lesz, a ***c*** sorozatban

* létrehozza a kért sorozatot; majd a ***c*** sorozatot rámásolja a régi ***a*** sorozatra 8 pont
* **más megoldások**, ha helyesek 5-10 pont

**II. TÉTEL 15 pont**

**Az *ab* érték utolsó számjegye**

**a. Követelmény**

* feladat kijelentése 5 pont

**b. Követelmény**

* helyes eredmény 3 pont

**c. Követelmény**

* az iteratív algoritmus fejlécével azonos fejlécű algoritmus: **F(a,b)**
  + algoritmus
    - az adott algoritmussal megegyező (vagy jobb) hatékonyságú algoritmus 3 pont
    - más, helyes algoritmus 1 pont
  + helyes önmeghívás 2 pont
  + rekurzív hívásokat leállító feltétel 2 pont
* más, eltérő fejléccel rendelkező algoritmus
  + algoritmus
    - az adott algoritmussal megegyező (vagy jobb) hatékonyságú algoritmus 3 pont
    - más, helyes algoritmus 1 pont
  + helyes önmeghívás 1 pont
  + rekurzív hívásokat leállító feltétel 1 pont

**III. TÉTEL 25 pont**

**Szigetek**

**Alprogramok:**

* adatok beolvasása **1 pont**
* eredmények kiírása **1 pont**
* a szigetek számának és a legnagyobb szigetnek meghatározása
  + *ha a sorozatot csak egyszer járja be* **5 pont**
  + más, helyes algoritmus, amely bizonyos részeredményeket eltárol (például a szigetek hosszát) 3 pont
* annak eldöntése, hogy a sziget *hegy* típusú-e**4 pont**
* annak eldöntése, hogy a szárazföldi magasságok értékei *különbözők* vagy sem **2 pont**
* a leggyakoribb magasság meghatározása és előfordulásainak száma **3 pont**

**Főprogram:** **1 pont**

* paraméterezés:

(helyes fejlécek, helyes hívások, 1-1 pont minden algoritmus esetében) **6 pont**

* olvashatóság:
  + megjegyzések **1 pont**
  + indentálás **1 pont**
  + beszédes azonosítók **1 pont**

**MEGOLDÁSOK**

**I.1. Tornyok 10 pont**

Az ***n*** magasságú toronyból egyetlen van. Ha lenne belőle kettő, a kettő között lennie kellene egy ***n*** + 1 magasságúnak, így magasabb lenne, mint a legmagasabb. Továbbá, legtöbb két darab ***n*** – 1 magasságú torony lesz, amelyek között ott van az egyetlen ***n*** magasságú. Ha kettőnél több ***n*** – 1 magas torony lenne, akkor ***n*** magasságú is több lenne.

Az ***n*** magasságú bal oldalán elhelyezhető további maximális számú torony a szabályoknak megfelelően. Így a középső legmagasabb most az ***n*** – 1 magasságú. Hasonlóan járunk el a jobb oldalon is.

Így a tornyokat tulajdonképpen rekurzívan helyezzük az „építménybe”: a tornyok sora egy középső (***n*** magasságú) toronnyal kezdődik, amelynek mindkét oldalán egy-egy – az eredeti toronysorral azonos szerkezetű toronysor található, amelynek közepén egy ***n*** – 1 magasságú található. A rekurzív összefüggések:

***tornyokSzáma***(***n***) = 2 \* ***tornyokSzáma***(***n*** – 1) + 1

***érmékSzáma***(***n***) = 2 \* ***érmékSzáma***(***n*** – 1) + ***n***

A rekurzió leáll, amikor ***n*** = 1, amikor ***tornyokSzáma***(1) = 1 és ***érmékSzáma***(1) = 1.

**Algoritmus** tornyokSzáma(n):

**Ha** n = 1 **akkor**

**térítsd** 1

**vége(ha)**

**térítsd** 2 \* tornyokSzáma(n - 1) + 1

**Vége(algoritmus)**

**Algoritmus** érmékSzáma(n):

**Ha** n = 1 **akkor**

**térítsd** 1

**vége(ha)**

**térítsd** 2 \* érmékSzáma(n - 1) + n

**Vége(algoritmus)**

**Algorimus** tornyokRekurzív(n, nrT, nrM):

nrT = tornyokSzáma(n)

nrM = érmékSzáma(n)

**Vége(algoritmus)**

**Iteratív megközelítés**

Mivel, tulajdonképpen 2 hatványait generáljuk (**2*n*-1**-ig), amelyeket *összeadunk*, az alábbi algoritmusok is helyesek:

A ***tornyokSzáma*** és ***érmékSzáma*** értékeket egyetlen ismétlő struktúrával számítjuk ki:

**Algoritmus** tornyok(n, tornyokSzáma, érmékSzáma):

{ *bemeneti paraméterek: n = a legmagasabb torony magassága* }

{ *kimeneti paraméterek: tornyokSzáma = a tornyok száma és érmékSzáma = az összes érme száma* }

érmékSzáma ← 0 { *egyelőre* *nem használtunk egyetlen érmét sem* }

tornyokSzáma ← 0 { *egyelőre* *nincs* *egy tornyunk sem* }

hatvány ← 1 { 2 *hatványait fogjuk generálni* }

**Minden** magasság = n, 1, -1 **végezd el:** { *a magasság csökken n-től* 1-*ig* }

tornyokSzáma ← tornyokSzáma + hatvány{ *a tornyok száma* 2*-hatványok összege* }

érmékSzáma ← érmékSzáma + hatvány \* magasság { *minden toronyban „magasság” darab érménk van* }

hatvány ← hatvány \* 2 { *egy bizonyos magasságú torony darabszáma* 2-*hatvány* }

**vége(minden)**

**Vége(algoritmus)**

De alkalmazhatunk egy-egy képletet is:

***tornyokSzáma***(***n***) = 1 + 2 + 22 + ... + 2***n*** – 1 = 2***n*** – 1,

***érmékSzáma***(***n***) = ***n*** + (***n*** – 1) \* 2 + (***n*** – 2) \* 22 + ... + 1 \* 2***n***-1

**Algoritmus** tornyokKéplet(n, tornyokSzáma, érmékSzáma):

hatvány = 1

**Minden** i = 1, n **végezd el:**

hatvány = hatvány \* 2

**vége(minden)**

tornyokSzáma = hatvány - 1

érmékSzáma = hatvány \* 2 - n - 2

**Vége(algoritmus)**

**I.2. Bűvös számok 20 pont**

Felépítjük az ***x*** szám ***p*** számrendszerben történő ábrázolásában található számjegyek tömbjét, majd – rendre – a ***q*** számrendszerben történő ábrázolás számjegyeit (anélkül, hogy tárolnánk egy új sorozatban). Ha az aktuális számjegy nem jelenik meg a ***p*** számrendszerben történt ábrázolásban, akkor ***x*** nem bűvös. Növeljük a számjegy előfordulásainak számát a számjegyek sorozatában. Azok a számok, amelyeknek megfelelően az előfordulások száma 1, nem bűvösek.

**Algoritmus** bűvösSzámok(x, p, q):

számjegyek[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

másolat = x

**Amíg** másolat ≠ 0 **végezd el:** { *x számjegyei a p számrendszerben* }

szj = másolat **MOD** p { *szj = az utolsó számjegy (a p számrendszerben)* }

számjegyek[szj] = 1

másolat = másolat **DIV** p

**vége(amíg)**

másolat = x

**Amíg** másolat ≠ 0 **végezd el:** { *x számjegyei a q számrendszerben* }

szj = másolat **MOD** q

**Ha** másolat[szj] = 0 **akkor** { *ha az aktuális számjegy (a q számrendszerben) nem szerepelt a p -ben* }

**térítsd** hamis

**vége(ha)**

számjegyek[szj] = számjegyek[szj] + 1

másolat = másolat **DIV** q

vége(amíg)

**Minden** i = 0, 10 **végezd el:**

**Ha** számjegyek[i] = 1 **akkor** { *ha i megjelent a p számrendszerben, de nem jelent meg a q-ban* }

**térítsd** *hamis*

**vége(ha)**

**vége(minden)**

**térítsd** *igaz*

**Vége(algoritmus)**

**Algoritmus** bűvösSzámokSorozata(p, q, n, k, t):

k = 0

**Minden** i = 1, n **végezd el:**

**Ha** bűvösSzám(i, p, q) **akkor**

k = k + 1

t = i

**vége(ha)**

**vége(minden)**

**Vége(algoritmus)**

**I.3. Beszúrás 20 pont**

Több megoldási stratégia is létezik:

**1. Változat:** Egy hatékony algoritmus egyszer járja be a sorozatot és nem használ segéd adatszerkezeteket. Megjegyezzük, hogy 2-nek hatványai kiszámíthatók szorzásokkal vagy bitműveletekkel  **20 pont**

**Algoritmus** beszúrHatványokat(n, a):

{ *Bemeneti és kimeneti paraméterek: n = a sorozat hossza, a = a sorozat* }

**Minden** i = n, 1, -1 **végezd el:** { *bejárjuk a sorozatot jobbról balra az n*-*dik és az* 1. *pozíciók között* }

a[2 \* i - 1] ← a[i] { *az adott elemet elhelyezzük a végső helyére, amelynek indexe* 2\**i* -1 }

aux1 ← a[i]{ *elmentjük ai-t, hogy ne kelljen indexelt elemet ismételten megkeresni az adatszegmensben* }

aux2 ← 1 { *aux2-ben számoljuk ki azt a 2-hatványt, amit a*2\**i-be fogunk helyezni* }

**Amíg** aux1 > 1 **végezd el:**

aux1 ← aux1 **DIV** 2

aux2 ← aux2 \* 2

**vége(amíg)**

a[2 \* i] ← aux2

{ *a 2\*i*-*dik pozícióba helyezzük azt a legnagyobb 2-hatványt, amely kisebb vagy egyenlő ai-vel* }

**vége(minden)**

n ← 2 \* n

**Vége(algoritmus)**

**2. Változat:** az új elemek beszúrása lehetségesúgy is, hogy a sorozatot eltoljuk ***n***-szer jobbra, így rendre felszabadítjuk a számukra szükséges helyet **1**5 **pont**

**3. Változat:** a beszúrást egy bejárással végezzük, de felhasználunk egy ***b*** segédtömböt, amelybe elmentjük az ***a*** sorozat soron levő elemét és a kért 2-hatvényt; majd a ***b*** sorozatot rámásoljuk ***a***-ra **10 pont**

**4. Változat:** helyes beszúrás, egyetlen bejárással, de két segédtömb felhasználásával: a ***b*** sorozatban tároljuk az ***a*** sorozat elemeinek megfelelő 2-hatványokat, a ***c*** sorozatban pedig felépítjük a kért sorozatot, majd rámásoljuk az ***a*** sorozatra **8 pont**

**II. Az *ab* érték utolsó számjegye 15 pont**

**a. Követelmény:**

* A függvény az ***ab*** érték utolsó számjegyét téríti.**5 pont**

**b. Követelmény:**

* F(1002,6) = 4 (minden vagy semmi) **3 pont**

(elég, ha megadják az értéket, **nem** kötelező megindokolni).

**c. Követelmény:**

* ha a rekurzív algoritmusnak ugyanaz a fejléce, mint az iteratívnak: F(a,b)
  + algoritmus
    - azonos hatékonyságú algoritmus az adott iteratív algoritmuséval 3 pont
    - más, helyes algoritmus 1 pont
  + helyes önmeghívás 2 pont
  + rekurzív hívásokat leállító feltétel 2 pont
* más, eltérő fejléccel rendelkező algoritmus
  + algoritmus
    - az adott algoritmussal megegyező hatékonyságú algoritmus 3 pont
    - más, helyes algoritmus 1 pont
  + helyes önmeghívás 1 pont
  + rekurzív hívásokat leállító feltétel 1 pont

**Algoritmus** FRek(a, b):

**Ha** b = 0 **akkor**

**térít** 1

**különben**

**Ha** b **mod** 2 = 1 **akkor**

**térít** a **mod** 10 \* FRek(a\*a **mod** 10, b **div** 2) **mod** 10

**különben**

**térít** FRek(a\*a **mod** 10, b **div** 2) **mod** 10

**vége(ha)**

**vége(ha)**

**Vége(algoritmus)**

**III. Szigetek 25 pont**

**Lásd a december 15.-én tartott konzultáció anyagában (is)**

* adatok beolvasása **1 pont**
* eredmények kiírása **1 pont**
* a szigetek számának és a legnagyobb szigetnek meghatározása
  + *ha a sorozatot csak egyszer járja be* **5 pont**

A szigetek számának és a legnagyobb szigetnek meghatározására tervezett algoritmus alapötlete arra az algoritmusra támaszkodik, amellyel azt a leghosszabb tömbszakaszt határozzuk meg, amely csak szigorúan pozitív számokat tartalmaz. Minden elemet csak egyszer dolgozunk fel, és nem használunk segéd adatszerkezeteket:

**Algoritmus** repülésSzimulálása\_v1(n, a, szigetekSzáma, maxEleje, maxVége):

{ *n: az a sorozat hossza, a: a magasságok tömbje, szigetekSzáma: a repülés során beazonosítható szigetek száma,* }

{ *maxEleje: a legnagyobb sziget kezdőpontjának sorszáma, maxVége: az utolsó pontjának sorszáma* }

maxEleje ← 0

maxVége ← 0

szigetekSzáma ← 0 { *még nem azonosítottunk be egyetlen szigetet sem* }

i ← 1 { *a magasságok sorozatot* 1*-től n-ig indexeljük* }

**Amíg** ai > 0 **végezd el:** { *repülünk Európa fölött* }

i ← i + 1

**vége(amíg)**

**Amíg** i ≤ n **végezd el:** { *amíg nincs vége a repülésnek* }

**Amíg** (ai = 0) **és** (i ≤ n) **végezd el:** { *repülünk az óceán fölött* }

i ← i + 1

**vége(amíg)**

kezd ← i { *itt egy sziget kezdődik, ha valóban szigetnek bizonyul, és nem Amerika* }

**Amíg** ai > 0 **végezd el:** { *repülünk a sziget fölött* }

i ← i + 1

**vége(amíg)**

**Ha** i ≤ n **akkor** { *ha nem ért véget az utazás (vagyis legutoljára nem Amerika fölött repültünk)* }

szigetekSzáma ← szigetekSzáma + 1 { *növeljük a szigetek számát* }

aktVég ← i – 1 { *az előző ponton van vége az aktuális szigetnek* }

**Ha** maxVége-maxEleje < aktVég–kezd **akkor** { *aktualizáljuk a legnagyobb szigetet* }

maxEleje ← kezd

maxVége ← aktVég

**vége(ha)**

**vége(ha)**

**vége(amíg)**

**Vége(algoritmus)**

* + más, helyes algoritmus, amely bizonyos részeredményeket eltárol (például a szigetek hosszát) 3 pont

**Algoritmus** repülésSzimulálása\_v2(n, a, szigetekSzáma, maxEleje, maxVége):

{ *n: az a sorozat hossza, a: a magasságok tömbje, szigetekSzáma: a repülés során beazonosítható szigetek száma,* }

{ *maxEleje: a legnagyobb sziget kezdőpontjának sorszáma, maxVége: az utolsó pontjának sorszáma* }

szigetekSzáma ← 0 { *még nem azonosítottunk be egyetlen szigetet sem* }

i ← 1

**Amíg** ai > 0 **végezd el:** { *repülünk Európa fölött* }

i ← i + 1

**vége(amíg)**

**Amíg** i ≤ n **végezd el:** { *amíg nincs vége a repülésnek* }

**Amíg** (ai = 0) **és** (i ≤ n) **végezd el:** { *repülünk az óceán fölött* }

i ← i + 1

**vége(amíg)**

szigetekSzáma = szigetekSzáma + 1 { *feltételezzük, hogy szigetet találtunk* }

kezdetekszigetekSzáma = i { *elmentjük a sziget kezdőpontjának sorszámát* }

**Amíg** (ai > 0) **és** (i ≤ n) **végezd el:** { *repülünk a sziget fölött* }

i ← i + 1

**vége(amíg)**

végekszigetekSzáma ← i – 1 { *elmentjük a sziget végpontjának sorszámát* }

**vége(amíg)**

szigetekSzáma ← szigetekSzáma - 1{ *az utolsó szárazföld Amerika volt* }

**Ha** szigetekSzáma ≠ 0 **akkor**

maxEleje ← kezdetek1

maxVége ← végek1

**Minden** j = 2, szigetekSzáma **végezd el:** { *bejárjuk a szigeteket* }

**Ha** maxVége - maxEleje < végekj - kezdetekj **akkor** { *aktualizáljuk a legnagyobb szigetet* }

maxEleje ← kezdetekj

maxVége ← végekj

**vége(ha)**

**vége(minden)**

**vége(ha)**

**Vége(algoritmus)**

* annak eldöntése, hogy a sziget *hegy* típusú-e**4 pont**

Ahhoz, hogy egy sorozat *hegy* típusú legyen, két olyan tömbszakaszból kell állnia, ahol az első szigorúan nö­vekvő, a második szigorúan csökkenő. Mivel csak a legnagyobb szigetet kell megvizsgálnunk, a magas­sá­gok ***a*** sorozatában csak az 1. követelménynek megfelelően meghatározott ***eleje*** és ***vége*** sorszámú pontok közé eső tömbszakaszt vizsgáljuk.

**Algoritmus** hegy(eleje, vége, a):

{ *megvizsgáljuk, hogy a legnagyobb sziget hegy típusú-e; ez az „eleje” és „vége” sorszámú elemek között található* }

i ← eleje

**Amíg** (i < vége) **és** (ai < ai+1) **végezd el:** { *vizsgáljuk, hogy a tömbszakasz eleje növekvő-e, és meddig* }

i ← i + 1

**vége(amíg)**

hegy ← i < vége { *a sziget csak akkor hegy típusú, ha nincs vége a növekvő tömbszakasz után* }

**Ha** hegy **akkor**

**Amíg** (i < vége) **és** (ai > ai+1) **végezd el:**

i ← i + 1 { *vizsgáljuk, hogy a tömbszakasz második része csökkenő-e, és meddig* }

**vége(amíg)**

hegy ← hegy **és** (i = vége)

{ *ha az első rész növekvő, a második rész csökkenő, és véget ért, akkor a sziget hegy típusú* }

**térít** hegy

**Vége(algoritmus)**

* annak eldöntése, hogy a szárazföldi magasságok értékei *különbözők* vagy sem **2 pont**

**Algoritmus** különbözők(n, a): { *eldöntjük, hogy az n elemű a sorozat különböző elemeket tartalmaz* }

különbözők ← **true**{ *feltételezzük, hogy a magasságok különbözők* }

i ← 1

**Amíg** különbözők **és** (i < n) **végezd el:**

**Amíg** ai = 0 **végezd el:** { *csak szárazföldet vizsgálunk* }

i ← i + 1

**vége(amíg)**

j ← i + 1 { *az i-edikről már tudjuk, hogy szárazföld, összehasonlítjuk a többi elemmel* }

**Amíg** (j ≤ n) **és** (ai ≠ aj) **végezd el:** { *amíg a magasságok különböznek, haladunk* }

j ← j + 1

**vége(amíg)**

különbözők ← j > n{ *ha az ai magasságot nem találtuk meg még egyszer "különbözők" értéke true* }

i ← i + 1{ *haladunk a sorozatban* }

**vége(amíg)**

**térít** különbözők

**Vége(algoritmus)**

* a leggyakoribb magasság meghatározása és előfordulásainak száma **2 pont**

Ha a szárazföldön található magasságok ***nem*** *különbözők*, meghatározzuk a leggyakoribbat és előfordulásainak számát. Ha több eredmény létezik, csak az elsőt adjuk meg.

**Algoritmus** legGyakoribb(n, a, magasság, k):

{ *magasság = a leggyakoribb magasság; előfordulásainak száma = k* }

**Minden** i = 1, n **végezd el:**  { *kezdőértékek* }

előfordulásokSzámai ← 0

**vége(minden)**

**Minden** i = 1, n **végezd el:**  { *meghatározzuk az előfordulások számát* }

előfordulásokSzámaa[i] ← előfordulásokSzámaa[i] + 1

**vége(minden)**

max ← 1 { *meghatározzuk a legnagyobb előfordulási szám helyét* }

**Minden** i = 1, n **végezd el:**

**Ha** előfordulásokSzámai > előfordulásokSzámamax **akkor**

max ← i

**vége(ha)**

**vége(minden)**

magasság ← amax { *"magasság" értéke a maximális gyakoriságú elem értéke* }

k ← előfordulásokSzámamax { *"k" értéke a maximális előfordulási szám* }

**Vége(algoritmus)**

**Főprogram:** **1 pont**

* paraméterezés:

(helyes fejlécek, helyes hívások, 1-1 pont minden algoritmus esetében) **6 pont**

* olvashatóság:
  + megjegyzések **1 pont**
  + indentálás **1 pont**
  + beszédes azonosítók **1 pont**