**BABEŞ-BOLYAI TUDOMÁNYEGYETEM**

**MATEMATIKA-INFORMATIKA KAR**

**Felvételi verseny - szeptember**

**Informatika írásbeli**

**A versenyzők figyelmébe:**

1. Minden tömböt 1-től kezdődően indexelünk.
2. A rácstesztekre (A rész) egy vagy több helyes válasz lehetséges. A válaszokat a vizsgadolgozatba írjátok (nem a feladatlapra). Ahhoz, hogy a feltüntetett pontszámot megkapjátok, elengedhetetlenül szükséges, hogy minden helyes választ megadjatok, és kizárólag csak ezeket.
3. A B részben szereplő feladatok megoldásait részletesen kidolgozva a vizsgadolgozatba írjátok.
   1. A feladatok megoldásait *pszeudokódban* vagy egy *programozási nyelvben* (*Pascal/C/C*++) kell megadnotok.
   2. A megoldások értékelésekor az első szempont az algoritmus ***helyessége***, majd a ***hatékonysága***, ami a *végrehajtási időt* és a *felhasznált memória méretét* illeti.
   3. A tulajdonképpeni megoldások előtt, ***kötelezően leírjátok szavakkal az alprogramokat (algoritmusokat), és megindokoljátok a megoldásotok lépéseit***. Feltétlenül írjatok ***megjegyzéseket*** (kommenteket), amelyek segítik az adott megoldás technikai részleteinek megértését. Adjátok meg az azonosítok jelentését és a fölhasznált adatszerkezeteket stb. Ha ez hiányzik, a tételre kapható pontszámotok 10%-kal csökken.
   4. Ne használjatok különleges fejállományokat, előre definiált függvényeket (például *STL*, karakterláncokat feldolgozó sajátos függvények stb.).

**A rész (30 pont)**

1. **Vajon mit csinál? (5p)**

A generál(n) algoritmus egy ***n*** természetes számot dolgoz fel (0 < ***n*** < 100).

|  |
| --- |
| **Algoritmus** generál(n):  szám ← 0  **Minden** i ← 1, 1801 **végezd el**  használti ← *hamis*  **vége(minden)**  **Amíg nem** használtn **végezd el**  összeg ← 0; használtn ← *igaz*  **Amíg** n ≠ 0 **végezd el**  számjegy ← n **MOD** 10  összeg ← összeg + számjegy \* számjegy \* számjegy  n ← n **DIV** 10  **vége(amíg)**  n ← összeg; szám ← szám + 1  **vége(amíg)**  **térítsd** szám  **Vége(algoritmus)** |

Állapítsátok meg a fenti algoritmus hatását.

1. ismételten kiszámítja az ***n*** szám számjegyei köbének összegét ameddig az összeg egyenlővé válik az ***n*** számmal és visszatéríti a végrehajtott ismétlések számát
2. kiszámítja az ***n*** szám számjegyei köbének összegét és visszatéríti ezt az összeget
3. kiszámítja az ***n*** szám számjegyei köbének összegét, felülírja ***n*** értékét ezzel az összeggel, és visszatéríti ezt az összeget
4. ismételten kiszámítja az ***n*** szám számjegyei köbének összegét ameddig egy összeget másodszorra kap meg, és visszatéríti a végrehajtott ismétlések számát
5. meghatározza, hogy hányszor lesz felülírva az ***n*** szám a számjegyei köbének összegével, ameddig egy már kiszámolt értéket vagy magát a számot kapja és visszatéríti ezt a számot
6. **Mely értékekre van szükség? (5p)**

Adott a feldolgoz(v, k) algoritmus, ahol ***v*** egy ***k*** elemű, természetes számokat tároló sorozat (1 ≤ ***k*** ≤ 1 000).

|  |
| --- |
| **Algoritmus** feldolgoz(v, k)  i ← 1; n ← 0  **Amíg** i ≤ k **és** vi ≠ 0 **végezd el**  y ← vi;c ← 0  **Amíg** y > 0 **végezd el**  **Ha** y **MOD** 10 > c **akkor**  c ← y **MOD** 10  **vége(ha)**  y ← y **DIV** 10  **vége(amíg)**  n ← n \* 10 + c; i ← i + 1  **vége(amíg)**  **térítsd** n  **Vége(algoritmus)** |

Állapítsátok meg, ***v*** és ***k*** mely értékeire térít vissza az algoritmus 928-at.

1. ***v*** = (194, 121, 782, 0) és ***k*** = 4
2. ***v*** = (928) és ***k*** = 1
3. ***v*** = (9, 2, 8, 0) és ***k*** = 4
4. ***v*** = (8, 2, 9) és ***k*** = 3
5. ***v*** = (912, 0, 120, 8, 0) és ***k*** = 5
6. **Logikai kifejezés kiértékelése (5p)**

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

|  |
| --- |
| **Algoritmus** kiértékelés(s, k, i)  **Ha** i ≤ k **akkor**  **Ha** si **akkor**  **térítsd** si  **különben**  **térítsd** (si **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
4. **Egyesítés (5p)**

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).

Legyenek az alább megadott egyesítés(a, n, b, m, c, p) és számol(a, n, b, m, c, p) algoritmusok, 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, m*** és ***p***, 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. cp ← bi 6. **vége(minden)** 7. **különben** 8. **Ha nem** eleme(an, b, m) **akkor** 9. p ← p + 1 10. cp ← an 11. **vége(ha)** 12. egyesítés(a, n - 1, b, m, c, p) 13. **vége(ha)** 14. **Vége(algoritmus)** | 1. **Subalgoritm** számol(a, n, b, m, c, p): 2. p ← 0 3. egyesítés(a, n, b, m, c, p) 4. **Vége(algoritmus)** |

A következő állítások közül melyek bizonyulnak mindig igaznak?

1. ha az ***a*** halmaz egyetlen elemet tartalmaz, a számol(a, n, b, m, c, p) algoritmus meghívása végtelen ciklust okoz
2. ha az ***a*** halmaznak négy eleme van, a számol(a, n, b, m, c, p) algoritmus meghívása maga után vonja az egyesítés(a, n, b, m, c, p) algoritmus 12. sorában található utasítás végrehajtását négyszer
3. ha az ***a*** halmaznak öt eleme van, a számol(a, n, b, m, c, p) algoritmus meghívása maga után vonja az egyesítés(a, n, b, m, c, p) algoritmus második sorában található utasítás végrehajtását ötször
4. ha az ***a*** halmaznak ugyanannyi eleme van, mint a ***b*** halmaznak, a számol(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, a számol(a, n, b, m, c, p) algoritmus végrehajtása után a ***c*** halmaznak ugyanannyi eleme lesz, mint az ***a*** halmaznak
6. **Hatványra emelés (5p)**

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)** |  |  |

1. **Legnagyobb többszörös (5p)**

Melyik alábbi algoritmus téríti az ***a*** természetes számnak azt a legnagyobb többszörösét, amely kisebb vagy egyenlő a ***b*** természetes számmal (0 < ***a*** < 10 000, 0 < ***b*** < 10 000, ***a*** < ***b***)?

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Algoritmus** f(a, b):  c ← b  **Amíg** c **MOD** a = 0 **végezd el**  c ← c – 1  **vége(amíg)**  **térítsd** c  **Vége(algoritmus)** |  | **Algoritmus** f(a, b):  **Ha** a < b **akkor**  **térítsd** f(2 \* a, b)  **különben**  **Ha** a = b **akkor**  **térítsd** a  **különben**  **térítsd** b  **vége(ha)**  **vége(ha)**  **Vége(algoritmus)** |
|  | **Algoritmus** f(a, b):  **térítsd** (b **DIV** a) \* a  **Vége(algoritmus)** |
|  | **Algoritmus** f(a, b):  **Ha** b **MOD** a = 0 **akkor**  **térítsd** b  **vége(ha)**  **térítsd** f(a, b-1)  **Vége(algoritmus)** |  | **Algoritmus** f(a, b):  c ← a  **Amíg** c < b **végezd el**  c ← c + a  **vége(amíg)**  **Ha** c = b **akkor**  **térítsd** c  **különben**  **térítsd** c – a  **vége(ha)**  **Vége(algoritmus)** |

**B rész (60 pont)**

1. **Polinom értéke (10 pont)**

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 (ismétlő struktúrák használata nélkül) úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval.

1. **Keresztmetszet (25 pont)**

Adott két sorozat, amelyek ‑30 000 és 30 000 közöttiegész számokat tárolnak, amelyek az egyes sorozatokon belül *különbözők*. Az ***a*** sorozatnak ***n*** (0 < ***n*** ≤ 10 000) eleme, a ***b*** sorozatnak ***m*** (0 < ***m*** ≤ 10 000) eleme van, és *növekvően rendezett*.

Írjatok algoritmust, amely meghatározza azt a ***k*** elemű (0 ≤ ***k*** ≤ 10 000) ***c*** sorozatot, amely az adott két sorozat közös elemeit tartalmazza, minden elemet csak egyszer, tetszőleges sorrendben. Az algoritmus bemeneti paraméterei a két sorozat (***a*** és ***b***) valamint a hosszúságaik (***n*** és ***m***). Kimeneti paraméterek a ***c*** sorozat és ennek ***k*** hosszúsága. Ha nincsenek közös elemek, ***k*** értéke 0.

***Példa:*** ha ***n*** = 4, ***a*** = (5, -7,-2, 3), ***m*** = 5 és ***b*** = (-2, 3, 5, 7, 8), a ***c*** sorozatnak ***k*** = 3 eleme van és ***c*** = (5, -2, 3).

1. **Tömbszakasz – testvérszámok nélkül (25 pont)**

Adott az ***n*** elemű (0 < ***n*** ≤ 10 000) ***a*** sorozat, amely 30 000-nél kisebb, nem nulla *különböző* természetes számokat tárol. Két számot *testvéreknek* nevezünk, ha *nem azonosak* és *van legkevesebb két különböző közös számjegyük*. Például, 5867 és 17526 *testvérek*, miközben 5867 és 152 nem *testvérek*. Hasonlóképpen, 131 és 114 nem testvérek.

**Követelmények:**

1. Írjatok algoritmust, amely eldönti, hogy az ***a*** természetes szám testvére-e a ***b*** természetes számnak (0 < ***a*** ≤ 30 000, 0 < ***b*** ≤ 30 000). Az algoritmus bemeneti paraméterei az ***a*** és ***b*** számok. Kimeneti paraméter a ***testvér*** logikai változó, amelynek értéke *igaz* ha ***a*** testvére ***b***-nek, különben *hamis*.
2. Írjatok algoritmust, amely meghatározza az ***a*** sorozatnak azt a leghosszabb tömbszakaszát, amely az ***a*** sorozat azon elemeit tartalmazza, amelyeknek *nincs egyetlen testvérük sem* az ***a*** sorozatban. Egy sorozat tömbszakasza a sorozat egymás utáni pozícióin található elemeit tartalmazza. Az algoritmus bemeneti paraméterei az ***a*** sorozat és az ***n*** hosszúsága. Kimeneti paraméterek a leghosszabb tömbszakasz kezdeti pozíciója (***start***) és a tömbszakasz ***k*** hosszúsága. Ha több azonos hosszúságú leghosszabb tömbszakasz létezik, a legutolsót kell megadnotok. Ha nem létezik egyetlen, adott tulajdonságú tömbszakasz sem, ***start*** értéke -1 és ***k*** értéke 0.

***Példa:*** Legyen ***n*** = 11 és ***a*** = (12345, 9, 100, 567, 5678, 345, 123, 8989, 222, 11, 78). Az ***x*** sorozat elemei, amelyeknek nincs testvérük: 9, 100, 8989, 222, 11, a keresett tömbszakasz pedig: (8989, 222, 11), tehát ***start*** = 8 és ***k*** = 3.

**Megjegyzések:**

1. Minden tétel kidolgozása kötelező.
2. A piszkozatokat nem vesszük figyelembe.
3. Hivatalból jár 10 pont.
4. Rendelkezésetekre áll 3 óra.

**MEGOLDÁS**

**A rész 30 pont**

**A. 1.** Vajon mit csinál? Válasz: E 5 pont

**A. 2.** Mely értékekre van szükség? Válaszok: A, C 5 pont

**A. 3.** Logikai kifejezés kiértékelése. Válasz: B 5 pont

**A. 4**. Egyesítés. Válaszok: B, E 5 pont

**A. 5.** Hatványra emelés. Válaszok: A, B, C, D, E 5 pont

**A. 6.** Legnagyobb többszörös. Válaszok: C, D, E 5 pont

**B rész 60 pont**

**B. 1. Polinom értéke 10 pont**

* Bemeneti és kimeneti paraméterek 2 pont
* Leállási feltétel 2 pont
* Újra hívás 2 pont
* Térített érték a rekurzióból történő kilépéskor 2 pont
* Térített érték, amikor még nincs kilépés 2 pont

Algoritmus: mivel az algoritmus iteratív változata a *Horner* sémát alkalmazza, a rekurzívba is ezt ültetjük át.

**Algoritmus** kiértékelésRek(n, egyh, x):

**Ha** n = 0 **akkor**

**térítsd** egyh[1]

**különben**

**térítsd** kiértékelésRek(n - 1, egyh, x) \* x + egyh[n + 1] { *Horner séma* }

**vége(ha)**

**Vége(algoritmus)**

**B. 2. Keresztmetszet (lásd a március 9.-ei konzultáció anyagát) 25 pont**

* Bemeneti és kimeneti paraméterek 2 pont
* Több megoldási stratégia is létezik:

bináris keresés kereséssel 23 pont

bináris keresés nélkül legtöbb 18 pont

**1. Változat:** A *hatékony* algoritmus felhasználja, hogy a második sorozat *rendezett*. Így, az első sorozat elemeit a *bináris keresés* algoritmusával megkeressük a másodikban. A megtalált elemek alkotják a közös elemek sorozatát **20 pont**

**Algoritmus** közösElemek(n, a, m, b, k, c):

k ← 0 { *egyelőre a c sorozatnak nincs egyetlen eleme sem* }

**Minden** i = 1, n **végezd el:** { *bejárjuk az a sorozatot és keressük az elemeit rendre a b sorozatban* }

megvan ← *hamis*

bal ← 1 { *mivel a b sorozat rendezett, lehetséges a bináris keresés* }

jobb ← m

**Amíg nem** megvan **és** bal ≤ jobb **végezd el:**

közép ← (bal + jobb) **DIV** 2

**Ha** a[i] = b[közép] **akkor** { *megtaláltuk, tehát a[i] közös elem* }

k ← k + 1

c[k] ← a[i] { *elhelyezzük a c sorozatban* }

megvan ← *igaz* { *a keresés le fog állni* }

**különben**

**Ha** a[i] < b[közép])

jobb ← közép - 1 { *tovább keresünk a b sorozat bal felében* }

**különben**

bal ← közép + 1 { *tovább keresünk a b sorozat jobb felében* }

**vége(ha)**

**vége(ha)**

**vége(amíg)**

**vége(minden)**

**Vége(algoritmus)**

*Megjegyzések*:

1. Megvalósítható két alprogrammal is (egy alprogram a keresésre és egy másik a ***c*** sorozat felépítésére).
2. A bináris keresés megírható külön alprogramban rekurzívan is.

**2. Változat:** A *naiv* algoritmus nem veszi figyelembe, hogy a második sorozat *rendezett.* Így, a megoldásban a „keresztmetszet” (klasszikus) programozási tételt alkalmazza: **14 pont**

**3. Változat:** ha előbb *rendezi* az a sorozatot és az „összefésülés”-hez hasonló feldolgozást végez: **12 pont**

**4. Változat:** A keresést két egymásba ágyazott **Minden** ciklussal végzi **10 pont**

**B. 3. Tömbszakasz – testvérszámok nélkül 25 pont**

* bemeneti és kimeneti paraméterek 2 pont
* a „testvér” tulajdonság ellenőrzése 10 pont
* egy tömbszakasz meghatározása 9 pont
* a leghosszabb tömbszakasz meghatározása 4 pont

**Algoritmus** testvérSzámok(a, b): { *meghatározzuk az a számjegyeinek statisztika-tömbjét* }

{ *majd rendre vizsgáljuk, hogy a b aktuális számjegye szerepel-e a-ban* }

**Minden** i = 0, 9 **végezd el:**

számjegyek\_A[i] ← *hamis*

**vége(minden)**

**Amíg** a > 0 **végezd el:** { *az a szám számjegyeinek statisztika-tömbje* }

számjegyek\_A[a **MOD** 10] ← *igaz*

a ← a **DIV** 10

**vége(amíg)**

közösSzj ← 0 { *a közös számjegyek darabszáma* }

**Amíg** b > 0 **végezd el:**

**Ha** számjegyek\_A[b **MOD** 10] **akkor** { *ha b aktuális számjegye szerepel* *az a számjegyei között* }

közösSzj ← közösSzj + 1

számjegyek\_A[b **MOD** 10] ← *hamis* { *töröljük ezt a számjegyet ahhoz, hogy ne számoljuk meg még egyszer* }

**vége(ha)**

b ← b **DIV** 10

**vége(amíg)**

**térítsd** közösSzj ≥ 2 { *ha a közös számjegyek darabszáma legalább 2* }

**Vége(algoritmus)**

**Algoritmus** testvérNélkül(n, a, testvér): { *felépítjük a sorozatot amelyben megjegyezzük, hogy van vagy nincs testvérük a-ban* }

**Minden** i = 1, n **végezd el:** { *bejárjuk az adott sorozatot* }

j← 1

testvér[i] ← *hamis* { *egyelőre nem tudjuk, hogy van-e testvére* }

**Amíg** j ≤ n **és** **nem** testvér[i] **végezd el:**

**Ha** i ≠ j **akkor**

testvér[i] ← testvérSzamok(a[i], a[j]) { *eldől, hogy van-e testvére a-ban* }

**vége(ha)**

j ← j + 1

**vége(amíg)**

**vége(minden)**

**Vége(algoritmus)**

**Algoritmus** leghosszabbTömbszakasz(n, a, testvér, m, b, startMax):

{ *a leghosszabb tömbszakasz, amely nem tartalmaz testvérszámokat* }

startMax ← 1 { *a leghosszabb tömbszakasz első elemének indexe* }

végeMax ← 1 { *a leghosszabb tömbszakasz utolsó elemének indexe* }

startAkt ← 1 { *az aktuális tömbszakasz első elemének indexe* }

végeAkt ← 1 { *az aktuális tömbszakasz utolsó elemének indexe* }

i ← 1

**Amíg** i ≤ n **végezd el:** { *bejárjuk az adott sorozatot* }

**Ha nem** testvér[i] **akkor** { *ha nincs testvére* }

startAkt ← i { *itt kezdődik az aktuális tömbszakasz* }

**Amíg nem** testvér[i] **és** i ≤ n **végezd el:** { *egymás után több testvér nélküli elem* }

i ← i + 1

**vége(amíg)**

végeAkt ← i - 1 { *itt ér véget az aktuális tömbszakasz* }

**Ha** végeMax - startMax < végeAkt – startAkt **akkor** { *aktualizáljuk, ha szükséges, a leghosszabb tömbszakaszt* }

startMax ← startAkt

végeMax ← végeAkt

**vége(ha)**

**vége(ha)**

i ← i + 1

**vége(amíg)**

m ← 0

**Minden** i = startMax, végeMax **végezd el:** { *átmásoljuk b-be a leghosszabb tömbszakaszt* }

m ← m + 1

b[m] ← a[i]

**vége(minden)**

**Vége(algoritmus)**