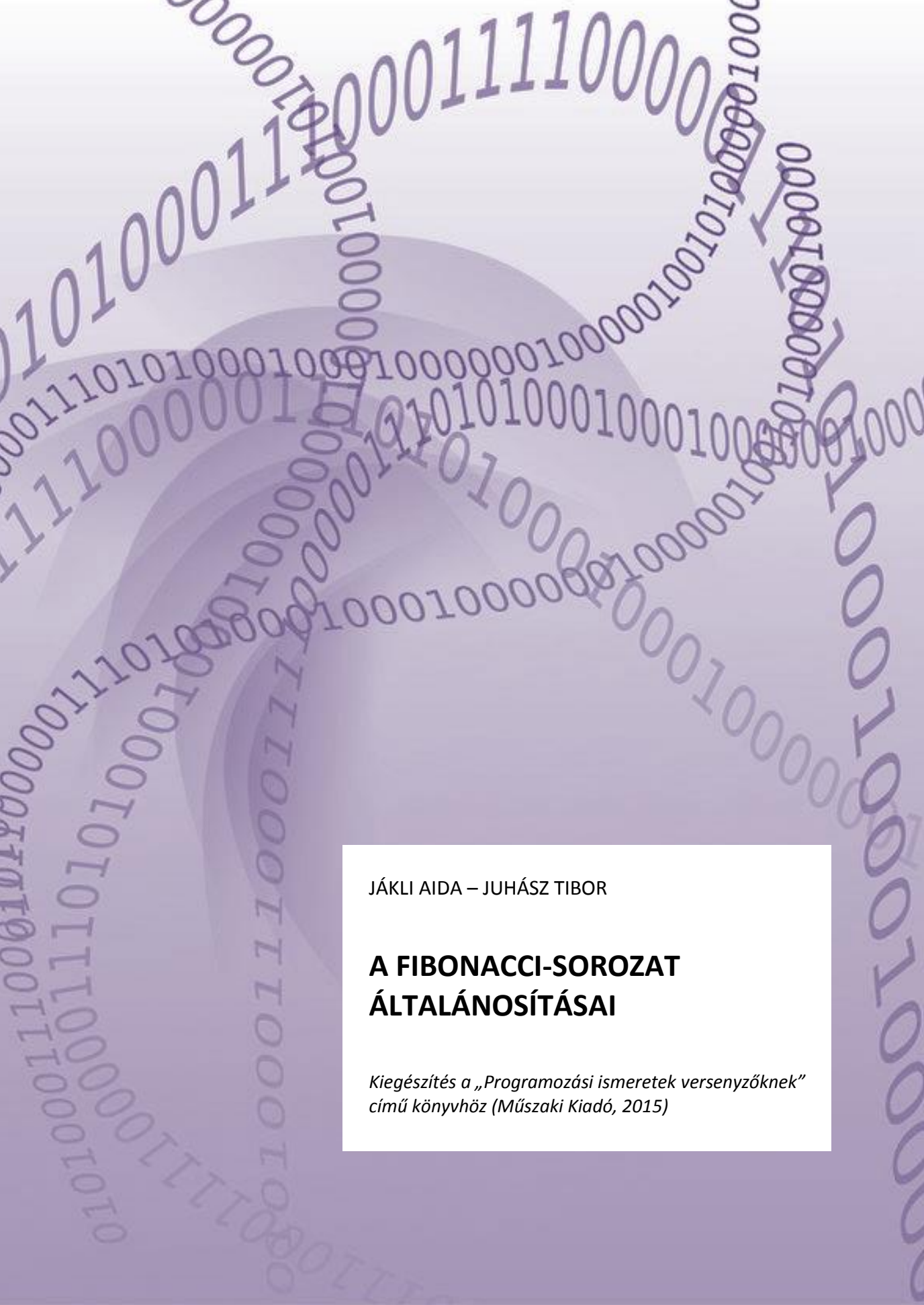


JÁKLI AIDA – JUHÁSZ TIBOR

A FIBONACCI-SOROZAT ÁLTALÁNOSÍTÁSAI

Kiegészítés a „Programozási ismeretek versenyzőknek” című könyvhöz (Műszaki Kiadó, 2015)

Jákli Aida – Juhász Tibor:

A Fibonacci-sorozat általánosításai

Kiegészítés a „Programozási ismeretek versenyzőknek” című könyvhöz (Műszaki Kiadó, 2015)

– 2014 –

Lektorálta: Tóth Bertalan

Tartalom

Bevezetés

Összeállításunkban áttekintjük a Nemes Tihamér országos informatikai tanulmányi verseny (OITV) és az informatika országos középiskolai verseny (OKTV) programozás kategóriájának azon feladatait, melyek a Fibonacci-sorozat alkalmazására vonatkoznak.

A felhasznált irodalom jegyzéke és a megoldásokra vonatkozó tudnivalók a *Programozási ismeretek versenyzőknek* (Műszaki Kiadó, 2015) bevezetésében olvashatók. A könyv részletes ismertetését lásd a programozás tankönyvsorozat webhelyén:

www.zmgzeg.sulinet.hu/programozas

A fenti webhelyen a bemutatott feladatok Visual Basic, illetve C++ nyelvű megoldása is megtalálható.

A versenyfeladatok forrása:

http://nemes.inf.elte.hu/nemes_archivum.html

Köszönet illeti dr. Zsakó Lászlót, amiért engedélyezte a versenyfeladatok szövegének közlését.

A szerzők

A Fibonacci-sorozat és általánosításai

A Fibonacci-sorozat tipikus példáját képezi a rekurzív módon megadott számsorozatoknak. Közvetlen kódolása bemutatja, hogy már kis elemszám esetén is kivárhatatlanul hosszú a futásidő. Ennek oka az azonos elemek többszöri (sokszori!) kiszámítása,[1] amit **Fibonacci-csapdának** nevezünk.

A Fibonacci-csapdát általában iterációval kerüljük el. A rekurzív képletet „alulról felfelé", a kisebb indexű tagoktól a nagyobbak felé haladva értékeljük ki.

Először az eredeti Fibonacci-sorozattal és általánosításaival kapcsolatos ismereteket, majd a versenyfeladatokat vesszük sorra.

A Fibonacci-sorozat

Fibonacci, azaz Leonardo Pisano Bigollo (kb. 1170 – kb. 1250) 1202-ben megjelent Liber Abaci (Könyv az abakuszról) című művében szerepelt a következő példa:

Egy [újszülött] *nyúlpár, mely először kéthónapos korában lesz szaporulatképes, havonta egy új nyúlpárnak ad életet. Az utódok első szaporulatára szintén két hónapot kell várni, de azután azok is hasonló ütemben hoznak létre újabb párokat. Hogyan alakul a nyúlpárok száma, ha mindegyikük életben marad?*[2]

A nyúlpárok számát az *n*-edik hónap elején jelöljük $f(n)$-nel! A feladat szövege alapján $f(1) = 1$, $f(2) = 1$ (csak a második hónap végén válnak ivaréretté). Az *n*-edik hónap elején meglévő nyúlpárokat két részre oszthatjuk. Egy részüket azok a párok alkotják, melyek már az előző hónapban is éltek. Ezek száma $f(n–1)$. Másik részüket azok a párok alkotják, melyek a hónap elejére ivaréretté vált pároktól születtek. Mivel a nyulak kéthónapos korukban válnak ivaréretté, ezek száma: $f(n–2)$. Így a nyúlpárok száma összesen:

$f(1) = 1$, $f(2) = 1$ és
$f(n) = f(n–1) + f(n–2)$ ha $f > 2$

A rekurzív képlettel meghatározott sorozatot Fibonacci-sorozatnak nevezzük.

A képlet alapján könnyen felírhatjuk a sorozat tagjait előállító rekurzív függvényt:

```
FÜGGVÉNY F(N MINT Egész) MINT Egész
   HA N == 1 VAGY N == 2 AKKOR
      F = 1
   EGYÉBKÉNT
      F = F(N–1) + F(N–2)
   ELÁGAZÁS VÉGE
FÜGGVÉNY VÉGE
```

A *fibonacci-1* program az *N* különböző értékeire meghatározza a Fibonacci-sorozat elemeit. Mivel a sorozat elemei nagyon gyorsan nőnek, használjunk elegendően nagy értelmezési tartománnyal rendelkező típust!

A Fibonacci-sorozat rekurzív algoritmusában ugyanazt a tagot sokszorosan kiszámítjuk (Fibonacci-csapda). Ezért a futásidő már viszonylag kis *N*-ekre is elviselhetet-

[1] Lásd például *Juhász T.–Kiss Zs.: Programozási ismeretek haladóknak*, 174. old.

[2] A feladat ezen megfogalmazásának forrása *Hámori Miklós: Arányok és talányok* (Typotex Elektronikus Kiadó Kft., 1994, 76. old.)

lenül hosszú lesz. Futtassuk a programot a konstansként deklarált *N* módosításával! Legfeljebb hány tag számítható ki például 1 perc futásidő alatt?

A futásidő csökkentéséhez készítsünk iteratív algoritmust!

Megtehetnénk, hogy a sorozat tagjait tömbben (vagy listában) tároljuk, *N*=1-től indítva a számítást. A nagyméretű kollekció helyett azonban elegendő két segédváltozót bevezetni, és értéküket léptetéssel módosítani:

```
FÜGGVÉNY F(N MINT Egész) MINT Egész
    VÁLTOZÓ F1, F2 MINT Egész
    F2 = 1
    F = 1
    CIKLUS I=3-tól N-ig
        F1 = F2
        F2 = F
        F = F1 + F2
    CIKLUS VÉGE
FÜGGVÉNY VÉGE
```

Vegyük észre, hogy az algoritmus nem tartalmaz feltételes elágazást!

Az iteratív algoritmust a *fibonacci-2* program mutatja be. A konstansként megadott *N* növelésével határozzuk meg, mely értékénél következik be túlcsordulás! A használt programozási nyelv ismer-e olyan egész adattípust, amely nagyobb *N*-ekre is lehetővé teszi a tagok meghatározását? Miért nem használhatunk valós típusokat a számításnál?

A Fibonacci-sorozat zárt alakja

Sok más rekurzív sorozathoz hasonlóan az *f*(*n*) értéke szintén megadható zárt alakban[3] (Binet-formula):

$$f(n) = \frac{\left(1+\sqrt{5}\right)^n - \left(1-\sqrt{5}\right)^n}{\sqrt{5}\cdot 2^n}$$

Érdekességként megjegyezzük, hogy bármely $n \geq 0$ esetén $\left|\frac{\left(1-\sqrt{5}\right)^n}{\sqrt{5}\cdot 2^n}\right| < \frac{1}{2}$, így ez a tag elhagyható a Binet-formulából. Az első tagot egészre kerekítve kapjuk meg az *f*(*n*)-et:

$$f(n) = \text{Kerekít}\left(\frac{\left(1+\sqrt{5}\right)^n}{\sqrt{5}\cdot 2^n}\right)$$

A fenti képlet átalakítható a következő formára:

$$f(n) = \text{Kerekít}\left(\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n}{\sqrt{5}}\right) = \text{Kerekít}\left(\frac{\Phi^n}{\sqrt{5}}\right)$$

[3] Explicit, azaz nem rekurzív képlettel.

ahol $\Phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618\ldots$, az aranymetszés aránya.

A *fibonacci-3* program kiírja a Fibonacci-sorozat fenti módszerekkel meghatározott elemeit.

A Binet-formula alkalmazásának lehetősége természetesen függ a valós típusok tárolásának pontosságától. Nagy *n*-ekre előfordulhat, hogy a fenti képletek pontatlanul adják meg az *f*(*n*) értékét. A *fibonacci-3* program segítségével határozzuk meg, hogy az alkalmazott programozási nyelv (fejlesztői környezet) esetén hány tagot kapunk meg pontosan!

A Fibonacci-sorozat alkalmazásai

A *hu.wikipedia.org* és az *en.wikipedia.org* webhelyek, illetve az irodalomjegyzékben feltüntetett források alapján felsorolunk néhány olyan feladatot, amely visszavezethető a Fibonacci-sorozatra.

1. Egy *n* hosszúságú szakaszt *f*(*n*+1)-féleképpen lehet kirakni 1 és 2 egység hosszúságú szakaszokból. Másképpen fogalmazva: az *n* számot *f*(*n*+1)-féleképpen írhatjuk fel 1-esek és 2-esek összegeként, ha a tagok sorrendje számít. Például: 3 = 1 + 1 + 1 = 1 + 2 = 2 + 1.
2. Az 1, 2, …, *n* számokból *f*(*n*+2)-féleképpen lehet kiválasztani olyan részhalmazokat, melyek nem tartalmaznak szomszédos számokat (ha az 1-et és az *n*-et is szomszédnak tekintjük).
3. Azon *n* elemű, bináris sorozatok száma, melyek
   - nem tartalmaznak szomszédos 1-eseket (vagy szomszédos 0-kat): *f*(*n*+2). Például: 0000, 0100, 0010, 0001, 0101, 1000, 1010 1001.
   - nem tartalmaznak páratlan darab szomszédos 1-est: *f*(*n*+1). Például: 0000, 0011, 0110, 1100, 1111.
   - nem tartalmaznak páros darab szomszédos 1-est vagy szomszédos 0-t: $2 \cdot f(n)$. Például: 0001, 1000, 1110, 0111, 0101, 1010.

Vegyük észre, hogy a bináris sorozatokra vonatkozó példák bitek helyett bármely jelpárra alkalmazhatók (például fej–írás sorozatok a pénzfeldobásnál stb.)! Fogalmazzuk át az 1. tételt is bitsorozatokra!

A fenti állítások közül az elsőt bizonyítjuk be. Konstruktív bizonyítást végzünk, azaz a bizonyítás során alkotjuk meg a bizonyítandó összefüggést.

Jelöljük *D*(*n*)-nel a keresett számot! Nyilván *D*(1) = 1 = *f*(2) és *D*(2) = 2 = *f*(3).

Az *n*-et eredményező összegeket két csoportba soroljuk. Az első csoportba kerüljenek azok az összegek, melyek első tagja 1, a második csoportba pedig azok, melyek első tagja 2. A két csoport között nincs átfedés, mert a sorrend számít. A keresett *D*(*n*) tehát:

D(*n*) = *D*(1-essel kezdődő összegek száma) + *D*(2-essel kezdődő összegek száma)

1-essel kezdődő összeg azonban ugyanannyi van, mint ahányféleképpen az *n*–1-et lehet felírni 1-esek és 2-esek összegeként. 2-essel kezdődő összeg pedig annyi, ahányféleképpen az *n*–2-t lehet így felírni. Azaz:

D(*n*) = *D*(*n*–1) + *D*(*n*–2)

Mint látjuk, a rekurzív formula pontosan megfelel a Fibonacci-sorozat képletének. A kiinduló elemek is ugyanazok, csak a 2. elemtől kezdődnek. Így:

$D(n) = f(n+1)$

Éppen ezt akartuk bizonyítani.

A fenti bizonyításhoz hasonló gondolatmenetek gyakran segítenek a programozási versenyfeladatok megoldásában. A versenyeken általában nem kell elvégeznünk a bizonyítást, csak „megsejteni" a helyes képletet, majd megírni a programot.

Javasoljuk az Olvasónak, hogy próbálja meg igazolni a további állításokat (akadnak köztük nehezen bizonyíthatók). Jussanak eszébe, ha hasonló feladatokkal találkozik a programozás versenyeken! ☺

A Fibonacci-sorozatnak más fontos szerep is jut a programozásban. Felhasználják a véletlenszám-generálásnál, a rendezési algoritmusoknál, az úgynevezett Fibonacci-kupac adatszerkezetnél, a tömörítési eljárásoknál, különböző optimumszámításoknál, a fájlban történő adatrendezésnél stb.

Érdekességként megemlítjuk a Zeckendorf-tételt, mely szerint minden pozitív egész szám előállítható egymástól különböző Fibonacci-számok összegeként. A felbontás egyértelmű, ha nem használunk fel egymást követő Fibonacci-számokat. Például:

$100 = 89 + 8 + 3 = 89 + 8 + 2 + 1 = 55 + 34 + 8 + 3$

A két utóbbi felbontásban szerepelnek egymást követő Fibonacci-számok (az 1 és a 2, illetve a 34 és az 55).

A felbontás meghatározható mohó algoritmussal, mindig a lehetséges legnagyobb Fibonacci-számot választva az összeadás következő tagjaként.[4]

Az egyértelmű felbontás következményeként bevezethetjük az úgynevezett Fibonacci-számrendszert. A 0, 1 számjegyekkel azt jelöljük, hogy az adott Fibonacci-szám szerepel-e a partícióban:

$100 = 1\cdot89 + 0\cdot55 + 0\cdot34 + 0\cdot21 + 0\cdot13 + 1\cdot8 + 0\cdot5 + 1\cdot3 + 0\cdot2 + 0\cdot1$
100 (*10-es számrendszerben*) = 1000010100 (*Fibonacci-számrendszerben*)

Végezetül megjegyezzük, hogy a legnagyobb közös osztó eukleidészi algoritmusa[5] akkor a leglassúbb, hogy ha két szomszédos Fibonacci-számra kell alkalmazni.

[4] Részletesebben lásd például: http://en.wikipedia.org/wiki/Zeckendorf's_theorem
[5] Lásd például *Juhász T.–Kiss Zsolt: Programozási ismeretek*, 149. oldal.

Versenyfeladatok

A Fibonacci-számok (és általánosításaik) alkalmazását egy többször is előforduló versenyfeladat változataival mutatjuk be.

Járdakövezés – 1.

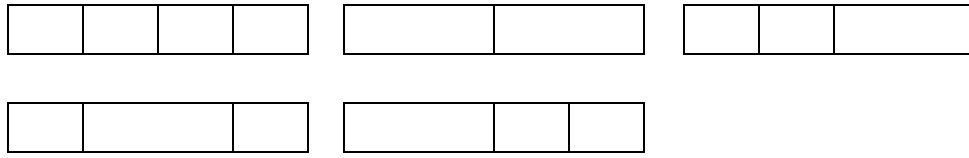
Nemes Tihamér OITV 2010. 2. forduló 1. korcsoport 3. feladat

Egy N ($1 \le N \le 80$) egység hosszú járdát 1 és 2 méretű lapokkal szeretnénk kikövezni. Hányféleképpen lehet ezt megtenni?

Készíts programot *lapok* néven, amely kiszámítja, hogy egy N egység hosszú járdát hányféleképpen lehet kikövezni 1 és 2 méretű lapokkal!

Példa:

N=4 egység hosszú járdát 5-féleképpen lehet kikövezni, a kikövezési lehetőségei:



Bemenet:
```
A járda hossza? 4
```

Kimenet:
```
A kikövezések száma: 5
```

Megoldás

n egység hosszúságú járdát a következőképpen hozhatunk létre:

a) egy n–1 egység hosszúságú járdát kiegészítünk egy 1 egység méretű lappal. Ez ugyanannyi eset, mint ahányféleképpen n–1 hosszúságú járdát készíthetünk.

b) egy n–2 egység hosszúságú járdát kiegészítünk egy 2 egység méretű lappal. Ez ugyanannyi eset, mint ahányféleképpen n–2 hosszúságú járdát készíthetünk.

A két eset nyilván nem fedi át egymást, hiszen az elsőben 1, a másodikban pedig 2 egység hosszúságú lappal végződik a járda.

Jelöljük $f(i)$-vel az i hosszúságú járda esetén a lehetőségek számát! Így:

$$f(n) = f(n-1) + f(n-2) \qquad \text{ha } n \ge 3$$

A rekurzív összefüggés alkalmazásához szükségünk van még a báziskritériumokra. Könnyen belátható, hogy $f(1) = 1$ és $f(2) = 2$.

Vegyük észre, hogy ez éppen a Fibonacci-sorozat, csak elhagytuk belőle az első tagot! Ezt visszavehetjük, ha kiegészítjük a sorozatot egy nulladik taggal. A kombinatorikában szokásos gondolatmenet alapján 0 hosszúságú járdát egyféleképpen készíthetünk, mégpedig úgy, hogy nem csinálunk semmit. Így $f(0) = 1$.

A megoldás tehát:

$f(0) = 1$, $f(1) = 1$ és

$$f(n) = f(n-1) + f(n-2) \qquad \text{ha } n \ge 2$$

A programban rekurzió helyett ciklust használunk. Ezzel elkerüljük a Fibonacci-csapdát. A sorozat tagjait tömb helyett változókban tároljuk, melyeket minden ismétlésnél módosítunk:

```
FÜGGVÉNY F(N MINT Egész) MINT Egész
    HA N == 0 VAGY N == 1 AKKOR VISSZATÉR 1-gyel
    ' N > 1 esetén:
    VÁLTOZÓ F1, F2, F3 MINT Egész
    F1 = 1 ' N = 0 esetén
    F2 = 1 ' N = 1 esetén
    CIKLUS I=2-től N-ig
        F3 = F1 + F2
        F1 = F2
        F2 = F3
    CIKLUS VÉGE
    VISSZATÉR F3-mal
FÜGGVÉNY VÉGE
```

Az algoritmusban kihasználtuk, hogy a visszatérési érték megadása egyben kilép a függvényből. A ciklusban ügyeljünk az értékadó utasítások sorrendjére! Mivel a sorozat tagjai nagyon gyorsan nőnek, tárolásukhoz válasszunk megfelelően nagy értékkészlettel rendelkező adattípust!

A teljes megoldást a *lapok* program tartalmazza. Az *n* bekérése helyett 0-tól 80-ig kiírjuk a képernyőre a függvényértékeket. Kiküszöbölhetjük az algoritmusból a feltételes elágazást (lásd a *fibonacci-2* programot)? Ha igen, alakítsuk át így a megoldást!

Megjegyezzük, hogy az 1. korcsoport 1. fordulójában ugyanez a feladat szerepelt, de program írása nélkül (*Járda*).

Járdakövezés – 2.

Nemes Tihamér OITV 2010. 2. forduló 2. korcsoport 4. feladat

Egy *N* egység hosszú járdát 1, 2 és 3 méretű lapokkal szeretnénk kikövezni. Hányféleképpen lehet ezt megtenni?

Készíts programot *kovezes* néven amely kiszámítja, hogy egy *N* egység hosszú járdát hányféleképpen lehet kikövezni 1, 2 és 3 méretű lapokkal!

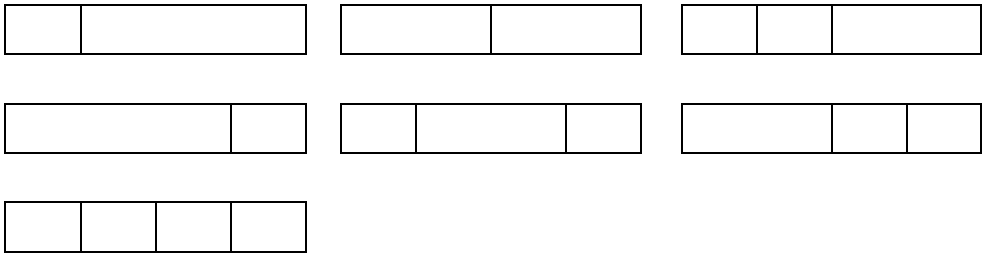
A *kovezes.be* szöveges állomány egyetlen sorában a járda hossza ($1 \le N \le 70$) van.

A *kovezes.ki* szöveges állomány egyetlen sora a lehetséges kikövezések számát tartalmazza!

Példa:

```
kovezes.be
4
```

```
kovezes.ki
7
```



Megoldás

Az előző feladat gondolatmenete alapján n egység hosszúságú járdát a következőképpen hozhatunk létre:

a) egy n–1 egység hosszúságú járdát kiegészítünk egy 1 egység méretű lappal. Ez ugyanannyi eset, mint ahányféleképpen n–1 hosszúságú járdát készíthetünk.
b) egy n–2 egység hosszúságú járdát kiegészítünk egy 2 egység méretű lappal. Ez ugyanannyi eset, mint ahányféleképpen n–2 hosszúságú járdát készíthetünk.
c) egy n–3 egység hosszúságú járdát kiegészítünk egy 3 egység méretű lappal. Ez ugyanannyi eset, mint ahányféleképpen n–3 hosszúságú járdát készíthetünk.

Jelöljük $f(i)$-vel az i hosszúságú járda esetén a lehetőségek számát! Így:

$$f(n) = f(n–1) + f(n–2) + f(n–3) \qquad \text{ha } n \geq 3$$

A báziskritériumok felírásához könnyen belátható, hogy $f(1) = 1$ és $f(2) = 2$. Az $f(3)$ értéke helyett most is az $f(0)$-t adjuk meg. 0 hosszúságú járdát egyféleképpen készíthetünk, mégpedig úgy, hogy nem csinálunk semmit. Így $f(0) = 1$.

A programban rekurzió helyett ciklust használunk. A ciklusban elkerülhetjük az ideiglenes változók értékének léptetését, ha egy listához adjuk hozzá az újabb elemet, illetve töröljük belőle a legrégebbit.

```
FÜGGVÉNY F(N MINT Egész) MINT Egész
    HA N == 0 AKKOR VISSZATÉR 1-gyel
    HA N == 1 AKKOR VISSZATÉR 1-gyel
    HA N == 2 AKKOR VISSZATÉR 2-vel
    ' N > 2 esetén:
    VÁLTOZÓ Lista MINT Lista(Elemtípus: Egész)
    VÁLTOZÓ Összeg MINT Egész
    Lista.Add(1) ' N = 0
    Lista.Add(1) ' N = 1
    Lista.Add(2) ' N = 2
    CIKLUS I=3-tól N-ig
        Összeg = Lista.Összegez()
        Lista.Kivesz(0) ' törli a 0 indexű elemet a listából
        Lista.Add(Összeg)
    CIKLUS VÉGE
    VISSZATÉR Lista(2)-vel
FÜGGVÉNY VÉGE
```

Az algoritmusban kihasználtuk, hogy a visszatérési érték megadása egyben kilép a függvényből. Mivel a sorozat tagjai nagyon gyorsan nőnek, tárolásukhoz válasszunk megfelelően nagy értékkészlettel rendelkező adattípust!

A teljes megoldást a *kovezes* program tartalmazza. A fájlból olvasás, illetve fájlba írás helyett N = 0-tól 70-ig kiírjuk a képernyőre a függvényértékeket.

A 2. korcsoport első fordulójában ugyanez a feladat szerepelt, de program írása nélkül (*Járda*).

Vegyük észre, hogy a feladat megoldását szolgáltató rekurzív képlet a Fibonacci-sorozat általánosítása három tagra! A sorozat elemeit gyakran tribonacci számoknak nevezik, a sorozatot pedig harmadrendű Fibonacci-sorozatnak (az eredeti tehát a má-

sodrendű Fibonacci-sorozat). Az összefüggést könnyen általánosíthatjuk, így beszélhetünk *n*-ed rendű Fibonacci-sorozatról.[6]

Járdakövezés – 3.

Informatika OKTV 2010. 2. forduló 3. korcsoport 5. feladat

Írj programot *jarda* néven, amely kiszámítja, hogy hányféleképpen lehet kikövezni egy 2**N* egység méretű járdát 1*1 és 1*2 méretű lapokkal!

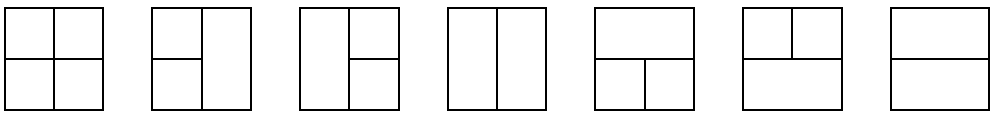
A *jarda.be* szöveges állomány első sorában a járda méretét megadó *N* egész szám van (1≤*N*≤36)

A *jarda.ki* szöveges állomány első és egyetlen sorába azt a számot kell írni, ahányféleképpen kikövezhető a 2**N* méretű járda!

Példa:

```
jarda.be
2
```

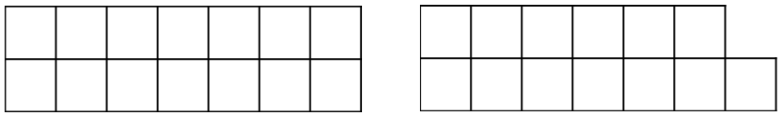
```
jarda.ki
7
```



Megoldás

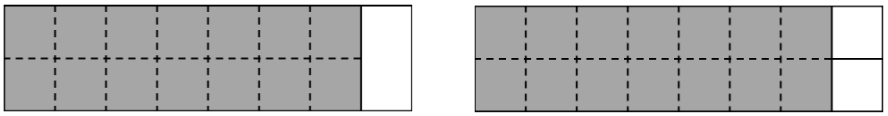
Az *n* hosszúságú járda kialakítását most is visszavezetjük rövidebb járdákra. Ügyelnünk kell arra, hogy ne számoljuk többször ugyanazt az esetet.

Nevezzük szabályosan lezárt járdának azt az *n* hosszúságú járdát, amelynek mindkét sora ugyanolyan hosszú (együtt kezdődnek és együtt végződnek, lásd az ábrát)! Vizsgáljuk meg, hányféleképpen alakítható ki ilyen járda a rövidebb, szabályosan lezárt járdákból! A lehetőségek számát jelöljük *f*(*n*)-nel! Feladatunk éppen az *f*(*n*) függvény értékének a meghatározása.



Szabályosan és nem szabályosan lezárt járdák

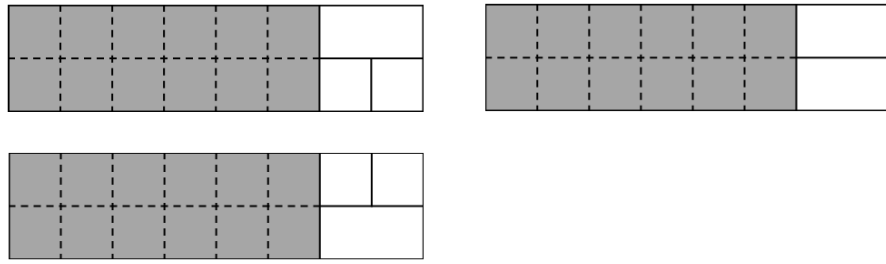
Egy szabályosan lezárt, *n*–1 hosszúságú járdából kétféleképpen képezhetünk *n* hosszúságú járdát. Ezen a módon tehát 2·*f*(*n*–1) féle járdához jutunk.



n hosszúságú járda kialakítása n–1 hosszúságú járdából

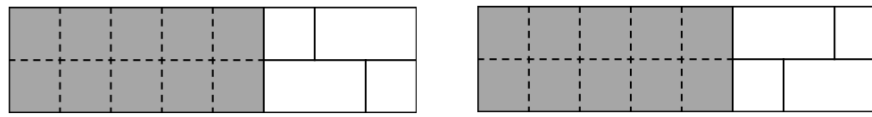
Egy szabályosan lezárt, *n*–2 hosszúságú járdából háromféleképpen képezhetünk *n* hosszúságú járdát úgy, hogy közben ne keletkezzen szabályosan lezárt, *n*–1 hosszúságú járda (azaz ne legyen átfedés az előző esettel). Ezen a módon tehát 3·*f*(*n*–2) féle járdát alakíthatunk ki.

[6] Lásd például: http://en.wikipedia.org/wiki/Generalizations_of_fibonacci_numbers



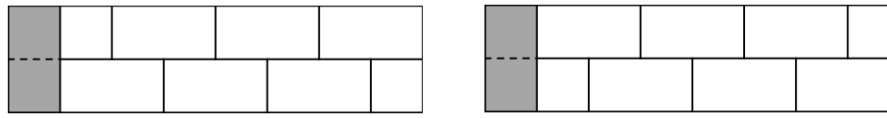
n hosszúságú járda kialakítása n–2 hosszúságú járdából

Egy szabályosan lezárt, *n*–3 hosszúságú járdából kétféleképpen képezhetünk *n* hosszúságú járdát úgy, hogy közben ne keletkezzen sem szabályosan lezárt, *n*–1 hosszúságú, sem pedig *n*–2 hosszúságú járda (azaz ne legyen átfedés az előző esetekkel). Ezen a módon tehát $2 \cdot f(n-3)$ féle járdát kapunk.

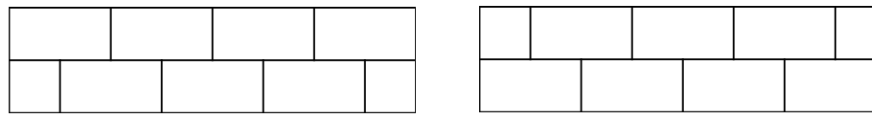


n hosszúságú járda kialakítása n–3 hosszúságú járdából

A továbbiakban csak a fenti ábrán látható két elrendezés folytatható visszafelé úgy, hogy ne legyen átfedés az előző esetekkel (lássuk be!). Azaz, átfedés nélkül egy *i* hosszúságú, szabályosan lezárt járdából $2 \cdot f(i)$ módon hozhatunk létre *n* hosszúságú járdát, bármely $1 \le i \le n-3$ esetén.



n hosszúságú járda kialakítása n–3-nál rövidebb, szabályosan lezárt járdából



Szabályosan lezárt járdákra nem visszavezethető járdák

Van azonban további kétféle olyan járda is, amely egyáltalán nem vezethető vissza semmilyen rövidebb, szabályosan lezárt járdára (lásd az előző ábrát). Tekintsük ezeket a 0 hosszúságú, szabályosan lezárt járda bővítéseinek. A lehetőségek számát $2 \cdot f(0)$-lal növelik.

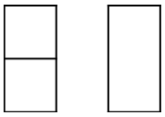
Így:

$$f(n) = 2 \cdot f(n-1) + 3 \cdot f(n-2) + 2 \cdot f(n-3) + \ldots + 2 \cdot f(2) + 2 \cdot f(1) + 2 \cdot f(0) \qquad \text{ha } n \ge 2$$

vagy:

$$f(n) = 2 \cdot [f(n-1) + f(n-2) + f(n-3) + \ldots + f(2) + f(1) + f(0)] + f(n-2) \qquad \text{ha } n \ge 2$$

Az $f(n)$ meghatározásához meg kell még adnunk $f(0)$ és $f(1)$ értékét. 1 egység hosszúságú járda kétféleképpen rajzolható fel, azaz $f(1) = 2$.



1 *egység hosszúságú járdák*

$f(0)$ értékét pedig tekintsük 1-nek: $f(0) = 1$, mivel 0 hosszúságú járda egyféleképpen alakítható ki (azaz nem készítjük el). Így:

$f(0) = 1, \quad f(1) = 2$ és

$$f(n) = 2 \cdot \sum_{i=0}^{n-1} f(i) + f(n-2) \quad \text{ha } n \geq 2$$

Az összefüggés alapján már megírhatjuk a programot. Közben ügyeljünk arra, hogy az *f* függvény értékeinek egymás után történő kiszámításával kerüljük el a rekurziót! A feladat szövege szerint az *n* legnagyobb értéke 36, amire a függvényérték 19 jegyű szám ($\sim 1{,}2 \cdot 10^{18}$). Válasszunk olyan adattípust, amely ekkora értéket pontosan (egész típusként) tud tárolni!

A *jarda-1* program fájlkezelés helyett (a feladat szövegével ellentétben) kilistázza a képernyőre az $f(n)$ értékeit n=2-től n=36-ig.

A fenti rekurzív formulát egyébként a következő módon egyszerűsíthetjük. Írjuk fel $f(n)$-et és $f(n–1)$-et:

$f(n) \quad = 2 \cdot f(n–1) + 2 \cdot f(n–2) + 2 \cdot f(n–3) + \ldots + 2 \cdot f(1) + 2 \cdot f(0) + f(n–2)$

$f(n–1) = \qquad 2 \cdot f(n–2) + 2 \cdot f(n–3) + \ldots + 2 \cdot f(1) + 2 \cdot f(0) + f(n–3)$

Vonjuk ki egymásból a két egyenlőséget:

$f(n) – f(n–1) = 2 \cdot f(n–1) + f(n–2) – f(n–3)$

Így

$f(n) = 3 \cdot f(n–1) + f(n–2) – f(n–3) \quad$ ha $n > 2$

Ennek a képletnek alkalmazásához azonban szükségünk van $f(2)$-re. A feladat szövegében szereplő példa alapján $f(2) = 7$. Írjuk meg így is a programot (lásd: *jarda-2*)!

A 3. korcsoport első fordulójában ugyanez a feladat szerepelt, de program írása nélkül (*Járdakövezés*).

Járdakövezés – 4.

Nemes Tihamér OITV 2013. 2. forduló 2. korcsoport 4. feladat

Egy 2 egység széles és *N* egység hosszú járdát kell kikövezni. A kövezésre 1x2 és 1x3 egység méretű lapokat lehet használni.

Készíts programot *ketsoros* néven[7], amely kiszámítja, hogy hányféleképpen lehet kikövezni a járdát!

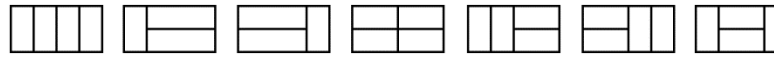
A *jarda.be* szöveges állomány első sorában egy pozitív egész szám van, a járda *N* hosszmérete ($1 \le N \le 50$).

A *jarda.ki* szöveges állomány egyetlen sora egy egész számot tartalmazzon, ahányféleképpen ki lehet kövezni a járdát!

Figyelem: a kiszámítandó érték nagy szám is lehet, ezért 64-bites egész számokat használj (Pascal esetén Int64, C/C++ esetén long long típus[8]).

Példa:

```
jarda.be          jarda.ki
4                 7
```



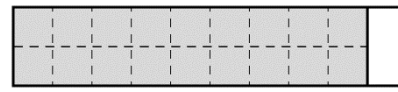
Megoldás az elemi esetek megszámolásával

Az előző feladat megoldásához hasonlóan nevezzük el szabályosan lezárt járdának azt az *n* hosszúságú járdát, amelyből nem lóg ki lap! Vizsgáljuk meg, hányféleképpen alakítható ki ilyen járda! A lehetőségek számát jelöljük $f(n)$-nel! Feladatunk az $f(n)$ függvény értékének a kiszámítása.

Felrajzolva az eseteket, könnyű belátni, hogy $f(1) = 1$, $f(2) = 2$ és $f(3) = 4$.

A további értékeket visszavezetjük az előző értékek meghatározására.

Egy szabályosan lezárt, $n-1$ hosszúságú járdából csak egyetlen módon készíthetünk n hosszúságú járdát, mégpedig úgy, hogy hozzáillesztjük az 1×2-es lapot. Így tehát $f(n-1)$ féle járdához jutunk.



n hosszúságú járda kialakítása n–1 hosszúságú járdából

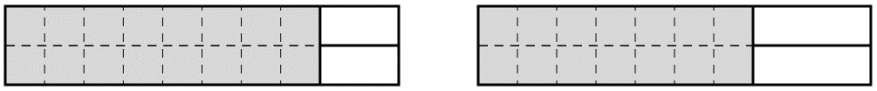
Egy szabályosan lezárt, $n-2$ hosszúságú járdákból ugyancsak egyféleképpen készíthetünk n hosszúságú járdát úgy, hogy ne keletkezzen szabályosan lezárt, $n-1$ hosszúságú járda. Ezzel további $f(n-2)$ féle járdát kapunk.

Egy szabályosan lezárt, $n-3$ hosszúságú járdából szintén egyféleképpen alakítható ki n hosszúságú járda úgy, hogy ne legyen átfedés az előző esetekkel, azaz ne keletkezzen sem szabályosan lezárt $n-1$ hosszúságú, sem pedig szabályosan lezárt $n-2$ hosszúságú járda. Így további $f(n-3)$ féle járdát készíthetünk.

---

[7] Az eredeti feladat szerint *jarda* néven.

[8] A feladat kitűzői sajnálatos módon nem adták meg, hogy Visual Basic-ben *Long* (vagy *Decimal*) típust célszerű alkalmazni.



n hosszúságú járda kialakítása n–2, illetve n–3 hosszúságú járdából

n-4 hosszúságú járdából nem alakítható ki *n* hosszúságú járda úgy, hogy ne legyen átfedés az előző esetekkel. Rövidebb járdák esetén viszont nagyon gyorsan növekszik a lehetőségek száma. A megoldás alapján majd kiderül például, hogy *n*–8 esetén négyféleképpen alakítható ki *n* hosszúságú járda, *n*–17 esetén ez a szám már 100 fölé emelkedik, *n*–41 hosszúságú járdánál pedig már egymilliónál is nagyobb. Ezek áttekintéséhez újabb rekurziót kell bevetnünk.

Nevezzük elemi eseteknek azokat az *i* ($1 \le i \le n$) hosszúságú, szabályosan lezárt járdákat, amelyek nem bonthatóak fel további, szabályosan lezárt járdákra! Az *i* hosszúságú elemi esetek számát jelöljük *e(i)*-vel! Könnyű belátni, hogy

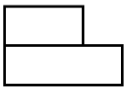
$e(1) = e(2) = e(3) = 1$, továbbá $e(4) = 0$.

Ekkor a következőt mondhatjuk. *n*–*i* hosszúságú, szabályosan lezárt járda $f(n\text{-}i)$ módon készíthető el. Ezt $e(i)$ módon bővíthetjük *n* hosszúságú járdává úgy, hogy ne legyen átfedés a hosszabb járdákból kiinduló bővítésekkel. Így $f(n–i)\cdot e(i)$ számú járdát kapunk. Az összes lehetőséget figyelembe véve:

$$f(n) = f(n–1)\cdot e(1) + f(n–2)\cdot e(2) + f(n–3)\cdot e(3) + \ldots + f(1)\cdot e(n–1) + e(n)$$

Eddigi gyakorlatunknak megfelelően a 0 hosszúságú járda egyféleképpen készíthető el, mégpedig úgy, hogy nem csinálunk semmit. Ekkor $f(0)=1$, így az összefüggést tömörebben felírhatjuk:

$$f(n) = \sum_{i=0}^{N-1} \left(f(i) \cdot e(n-i) \right)$$



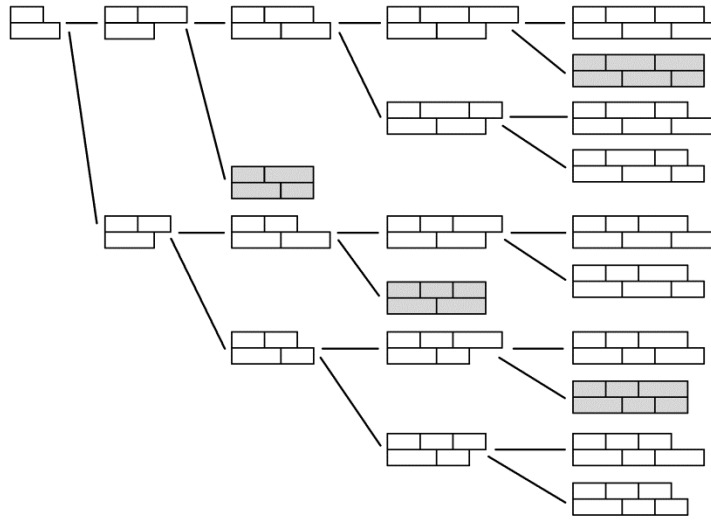
Az *e(i)* értékeket ($1 \le i \le n$) az *ElemiEsetekSzáma*(*a*, *b*) rekurzív eljárással fogjuk meghatározni. Az eljárás *a* és *b* paramétere jelöli a járda egyes sorainak a hosszát. Induljunk ki a mellékelt ábrán látható elrendezésből (a=2, b=3)!

Természetesen olyan elemi esetek is kialakíthatók, melyeknél az első sor kezdődik 1×3-as lappal, a második pedig 1×2-essel. Így azonban a fenti kiinduló elemből kapható járdák tükörképei adódnak (a vízszintes tengelyre tükrözve). Ezért elegendő csak az egyik esetet megvizsgálni, és ha találunk egy elemi esetet, akkor az $e(a)$ értékét 2-vel növeljük.

A járdát a következő módon építjük tovább. Ha a két sor hossza különböző, akkor a rövidebb sor végére lerakhatjuk az 1×2-es vagy az 1×3-as lapot is. Tehát, ha az *a*-val jelölt sor a rövidebb, akkor meghívjuk az *ElemiEsetekSzáma* eljárást az (a+3, b) és az (a+2, b) argumentumokkal, ha pedig a *b*-vel jelölt sor a rövidebb, akkor az (a, b+3) és az (a, b+2) értékekkel.

Ha a sorok hossza megegyezik, akkor elemi esethez jutottunk. A fentiek alapján az $e(a)$ értékét 2-vel megnöveljük. Vigyázat! Ilyenkor nem hívhatjuk meg az *ElemiEsetekSzáma* eljárást, mert ha egy elem végére további lapokat raknánk, akkor egy

olyan járdát kapnánk, ami felosztható rövidebb, szabályosan lezárt járdákra, így nem bizonyulna elemi esetnek.



A szürkével jelölt elemi esetek kialakítása a kiinduló elrendezésből

A programban a járda hosszát (a feladatban szereplő *N*-et) a *Hossz* változóban tároljuk. A tömböknél ügyeljünk az elegendően nagy értékkészletet biztosító típus választására!

```
VÁLTOZÓ F(0…50), E(1…50) MINT Egész
VÁLTOZÓ Hossz MINT Egész
```

Az elemi esetek számát meghatározó eljárás a fentiek alapján:

```
ELJÁRÁS ElemiEsetekSzáma(A, B MINT Egész)
    HA A ≤ Hossz ÉS B ≤ Hossz AKKOR
        HA A < B AKKOR
            ElemiEsetekSzáma(A+3, B)
            ElemiEsetekSzáma(A+2, B)
        EGYÉBKÉNT HA A > B AKKOR
            ElemiEsetekSzáma(A, B+3)
            ElemiEsetekSzáma(A, B+2)
        EGYÉBKÉNT
            E(A) = E(A) + 2
        ELÁGAZÁS VÉGE
    ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

A járda kialakítására vonatkozó lehetőségek számát (az *F* és *E* tömbök elemeit) a következő eljárással számítjuk ki:

```
ELJÁRÁS Járda(Hossz MINT Egész)
    F() = 0, E() = 0
    F(0) = 1
    E(1) = 1, E(2) = 1, E(3) = 1
    ElemiEsetekSzáma(2, 3)
```

```
    CIKLUS N = 1-től Hossz-ig
        CIKLUS I = 0-tól N–1-ig
            F(N) = F(N) + F(I) * E(N–I)
        CIKLUS VÉGE
    CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Hasonlítsuk össze az $f(n)$-re kapott képlettel!

A főprogramban beolvassuk a járda hosszát, meghívjuk a *Járda*(*Hossz*) eljárást, majd kiírjuk az *F*(*Hossz*) értékét. A teljes megoldást a *ketsoros-1* program tartalmazza.

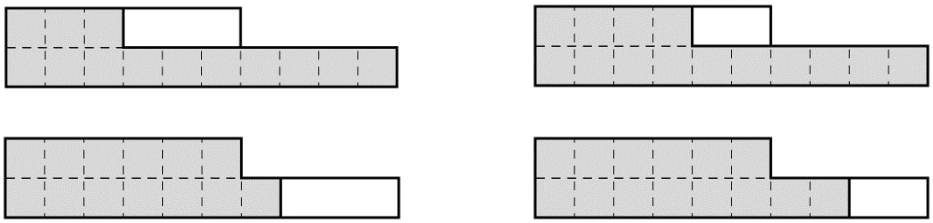
Megoldás dinamikus programozással

A feladatot megoldhatjuk dinamikus programozással, az elemi esetek megszámolása nélkül. Nevezzük (i, j) méretű járdának azt a járdát, amelyiknek az első sora i, a második sora pedig j hosszúságú! Ezek számát jelöljük $f(i, j)$-vel! Feladatunk az $f(n, n)$ érték meghatározása.

Könnyű belátni, hogy $f(0, 2) = f(2, 0) = f(1, 1) = 1$, $f(2, 2) = 2$, továbbá $(0, 1)$, $(1, 0)$, $(1, 2)$ és $(2, 1)$ méretű járda nem készíthető. 0 hosszúságú járda itt is csak egyféleképpen hozható létre: $f(0, 0) = 1$.

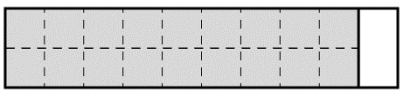
Vizsgáljuk meg, hogyan alakítható ki (i, j) méretű járda a kisebb járdákból!

a) Az $(i–3, j)$ méretű járdák első sorába 1×3-as lapot teszünk. Így $f(i–3, j)$ féle járdához jutunk.
b) Az $(i–2, j)$ méretű járdák első sorába 1×2-es lapot teszünk. Ezzel $f(i–2, j)$ féle járdát kapunk.
c) Az $(i, j–3)$ méretű járdák második sorába 1×3-as lapot teszünk. Így $f(i, j–3)$ féle járdát készíthetünk.
d) Az $(i, j–2)$ méretű járdák második sorába 1×2-es lapot teszünk. Ezen a módon további $f(i, j–2)$ féle járda készíthető.



(6, 10) méretű járda kialakítása

e) Ha $i=j$, akkor az $(i–1, j–1)$ méretű járdák végére függőlegesen letehetünk egy 1×2-es lapot. Így tehát további $f(i–1, j–1)$ féle lehetőség adódik.



Járda kialakítása (i–1, j–1) méretű járdából, ha i=j

A fentiek alapján összesen $f(i–3, j) + f(i–2, j) + f(i, j–3) + f(i, j–2)$, valamint, ha $i=j$, akkor még további $f(i–1, j–1)$ számú járdát kapunk.

De így az $(i–3, j–3)$ méretű járdákat, az $(i–3, j)$ és az $(i, j–3)$ méretű járdáknál is megszámoltuk, tehát az $f(i–3, j–3)$ értéket ki kell vonnunk az összegből.

Ehhez hasonlóan az $(i–2, j–2)$, $(i–3, j–2)$ és az $(i–2, j–3)$ méretű járdákat szintén kétszer számoltuk, tehát ezek számát is ki kell vonni.

Így az alábbi képlethez jutunk:

$$f(i, j) = f(i–3, j) + f(i–2, j) + f(i, j–3) + f(i, j–2) – f(i–3, j–2) – f(i–2, j–3) – \\ – f(i–3, j–3) – f(i–2, j–2) \quad \text{ha} \quad i \neq j$$

$$f(i, j) = f(i–3, j) + f(i–2, j) + f(i, j–3) + f(i, j–2) + f(i–1, j–1) – f(i–3, j–2) – \\ – f(i–2, j–3) – f(i–3, j–3) – f(i–2, j–2) \quad \text{ha} \quad i = j$$

A fenti összefüggés alapján meghatározhatjuk az $f(i, j)$ értékeit. A táblázat első 3 sorában, illetve oszlopában az összegből ki kell hagyni azokat a tagokat, amelyekben valamelyik index negatív lenne.

	0	1	2	3	4	5	6	7	8
0	1	0	1	1	1	2	2	3	4
1	0	1	0	1	1	1	2	2	3
2	1	0	2	1	2	3	3	5	6
3	1	1	1	4	2	5	6	7	11
4	1	1	2	2	7	4	9	11	13
5	2	1	3	5	4	15	9	19	24
6	2	2	3	6	9	9	30	18	39
7	3	2	5	7	11	19	18	60	37
8	4	3	6	11	13	24	39	37	123

Az f(i, j) első néhány értéke

Programunkban az F függvény értékeit kétdimenziós tömbben tároljuk. A tömbelemek típusának megválasztásánál ügyeljünk az elegendően nagy értékkészletre!

```
VÁLTOZÓ F(0…N, 0…N) MINT Egész
VÁLTOZÓ N MINT Egész
```

A függvényértékeket meghatározó eljárás mondatszerű leírása:

```
ELJÁRÁS Járda(N MINT Egész)
    F(,)=0
    F(0, 0) = 1, F(0, 2) = 1
    F(1, 1) = 1
    F(2, 0) = 1, F(2, 2) = 2
    CIKLUS I = 3-tól N-ig
        F(0, I) = F(0, I–2) + F(0, I–3)
        F(I, 0) = F(I–2, 0) + F(I–3, 0)
        F(1, I) = F(0, I–1)       ' = F(1, I - 2) + F(1, I - 3)
        F(I, 1) = F(I–1, 0)       ' = F(I - 2, 1) + F(I - 3, 1)
        F(2, I) = F(2, I–3) + F(2, I–2) + F(0, I) – F(0, I–2) – F(0, I–3)
        F(I, 2) = F(I–3, 2) + F(I–2, 2) + F(I, 0) – F(I–2, 0) – F(I–3, 0)
    CIKLUS VÉGE
```

```
    CIKLUS I = 3-tól N-ig
        CIKLUS J = 3-tól N-ig
            F(I, J) = F(I–3, J) + F(I–2, J) + F(I, J–3) + F(I, J–2) –
                    F(I–3, J–3) – F(I–2, J–2) – F(I–3, J–2) – F(I–2, J–3)
            HA I = J AKKOR
                F(I, J) += F(I–1, J–1)
            ELÁGAZÁS VÉGE
        CIKLUS VÉGE
    CIKLUS VÉGE
ELJÁRÁS VÉGE
```

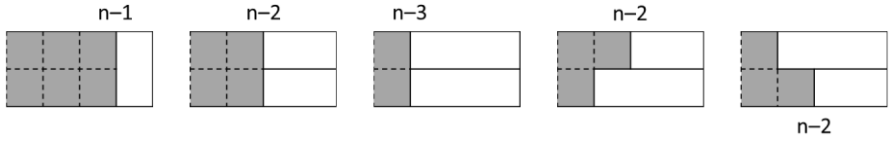
A főprogramban beolvassuk a járda hosszát, meghívjuk a *Járda*(*N*) eljárást, majd kiírjuk az *F*(*N*, *N*) értékét.

A teljes megoldást a *ketsoros-2* program mutatja be. Hasonlítsuk össze a futásidőt az elemi esetek megszámolását végző algoritmus futásidejével!

Megoldhattuk volna az előző (*Járdakövezés–3*) feladatot dinamikus programozással? Ha igen, írjuk meg így is a programot!

Megoldás közvetett rekurzióval[9]

Közvetett rekurzió alkalmazásával az eddigieknél egyszerűbb megoldást kapunk. Kezdjük el még egyszer az *n* hosszúságú, szabályosan lezárt járda visszavezetését rövidebb járdákra! Jelöljük *f*(*n*)-nel az *n* hosszúságú, szabályosan lezárt járdák számát!

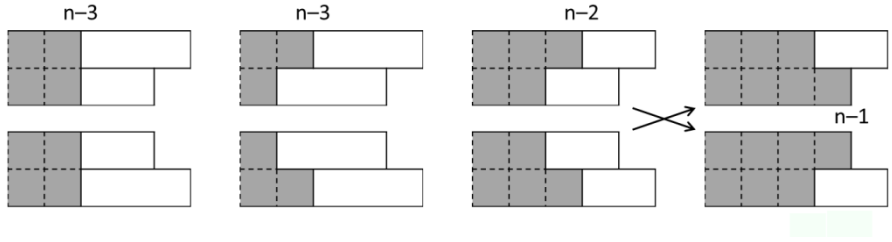


1. *2.* *3.* *4.* *5.*

n hosszúságú járda kialakítása rövidebb járdákból

A fenti ábra alapján *n*–1, *n*–2 és *n*–3 hosszúságú járdából egy-egyféleképpen kaphatunk *n* hosszúságú járdát (1–3. eset). Kialakíthatjuk azonban *n*–2 hosszúságú, foghíjas járdákból is. Szimmetriaokokból nyilván ugyanannyi alul (4. eset), mint felül foghíjas (5. eset) járda van. Jelöljük az alul foghíjas járdák számát *g*(*n*)-nel! Így:

$$f(n) = f(n–1) + f(n–2) + f(n–3) + 2 \cdot g(n–2)$$



1. *2.* *3.* *4.*

n hosszúságú, foghíjas járda kialakítása rövidebb járdákból

[9] Horváth Gyula – Szlávi Péter előadásainak felhasználásával: Rekurzió
A Neumann János Tehetséggondozó Program szakköri anyaga (ELTE IK, 2013)

A fenti ábra alapján (alul vagy felül) foghíjas járdát kaphatunk n–3 hosszúságú, szabályosan lezárt, illetve foghíjas járdából (1–2. eset), továbbá n–2, illetve n–1 hosszúságú foghíjas járdából (3–4. eset). De ha a 3. esetnél az utolsó, dupla hosszúságú követ töröljük, akkor éppen a 4. eset ábráit kapjuk meg, csak felcserélődik a foghíj helye. A 4. eset tehát magában foglalja a 3. esetet. Így

$g(n) = f(n–3) + g(n–3) + g(n–1)$

Könnyű belátni, hogy a báziskritériumokat a következő értékek szolgáltatják (most nincs szükségünk az egyszerűsítéshez a 0 indexű elemekre):

$f(1) = 1$, $f(2) = 2$, $f(3) = 4$
$g(1) = 0$, $g(2) = 0$, $g(3) = 1$

A programban rekurzió helyett iterációt használunk. Az f és g értékeket tömbökben tároljuk, melyek elemeit ciklussal határozzuk meg.

```
VÁLTOZÓ F(1…N), G(1…N) MINT Egész
VÁLTOZÓ N MINT Egész
Be: N
Járda(N)
Ki: F(N)

ELJÁRÁS Járda(N MINT Egész)
    F() = 0, G() = 0
    F(1) = 1, F(2) = 2, F(3) = 4
    G(3) = 1
    CIKLUS I=4-től N-ig
        F(I) = F(I–1) + F(I–2) + F(I–3) + 2*G(I–2)
        G(I) = F(I–3) + G(I–3) + G(I–1)
    CIKLUS VÉGE
ELJÁRÁS VÉGE
```

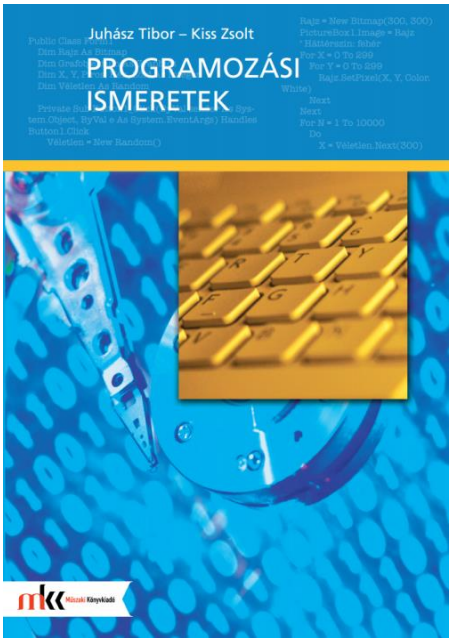
A teljes megoldást a *ketsoros-3* program tartalmazza.

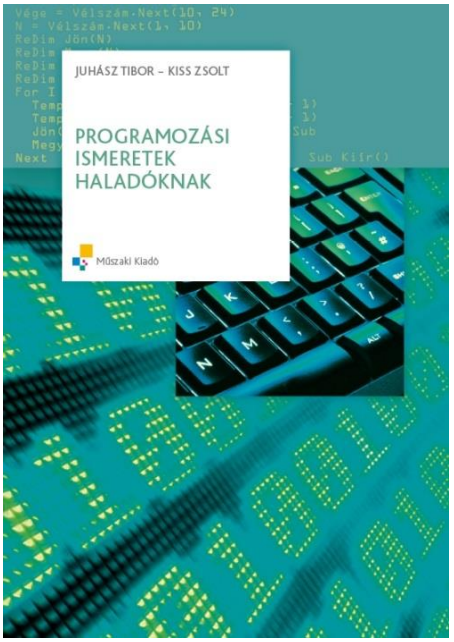
Alkalmazhatjuk a közvetett rekurzió gondolatmenetét az előző, *Járdakövezés–3* feladat megoldásánál is? Javasoljuk az Olvasónak, hogy a tanultak felhasználásával határozza meg, hányféleképpen lehet a 3 egység széles és n egység hosszú járdát kikövezni 1×2 egység méretű lapokkal!

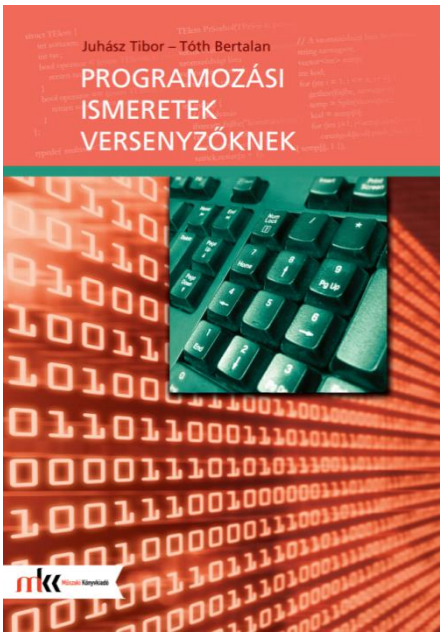
Ezzel a járdás feladat változatait alaposan kimerítettük. ☺

További feladatok a Fibonacci-sorozat általánosítására

OITV 1994-2f2	Bacilus
OITV 2012-3f2	Róka
OKTV 2006-3f3	Színezés
OKTV 2012-3f3	Nyúl







Juhász Tibor – Kiss Zsolt:
Programozási ismeretek
(Műszaki Kiadó, 2011, 2015)

Juhász Tibor – Kiss Zsolt:
Programozási ismeretek haladóknak
(Műszaki Kiadó, 2012)

Juhász Tibor – Tóth Bertalan:
Programozási ismeretek versenyzőknek
(Műszaki Kiadó, 2015)

www.zmgzeg.sulinet.hu/programozas