Mérés feladat

Egy mérőműszerrel felszerelt repülőgép az óceániai szigetvilág területe felett, előre adott útvonalon repülve a felszín terepviszonyaival kapcsolatos méréseket végez. Repülése közben százméterenként megméri annak a tereppontnak a tengerszint feletti magasságát, amely felett éppen elhalad. A repülőgép útjának 72%-a a tenger felett vezet. Ha tenger felett repül, a mért érték minden esetben nulla, egyébként a szárazföldi terepviszonyoknak megfelelő pozitív egész szám.

Készíts ezer mérési értéket tartalmazó tömböt, amely nem negatív egészek véletlen sorozatával van feltöltve. A véletlen egész értékek a [0, 640] intervallumból kerüljenek ki úgy, hogy a nulla értékek 72%-os elméleti valószínűséggel forduljanak elő, a pozitív értékek előfordulási valószínűsége pedig legyen azonos!

Válaszolj a következő kérdésekre!

Mekkora a legmagasabb tereppont magasság értéke az út során?

Melyik mérési pontnál mérték a legnagyobb magasság értéket?

Útjának hányad részében repült ténylegesen szárazföld felett a repülőgép?

Mekkora volt a leghosszabb szárazföld felett megtett út?

Hány sziget felett repült át a repülőgép?

Hányadik szigeten volt a mért legmagasabb tereppont?
(Feltételezzük, hogy minden szárazföldi tereppont szigethez tartozik.)

A válaszok az alábbi mintának megfelelően jelenjenek meg:

A legmagasabb tereppont magasság értéke: 632 méter.

A legnagyobb magasság értéket 57,6 kilométernél mérték.

A repülőgép útjának 29,3 %-ában szárazföld felett haladt.

A leghosszabb szárazföld felett megtett út 0,6 km volt.

A repülőgép 192 sziget felett repült át.

A mért legmagasabb tereppont a(z) 49. szigeten volt.

Szorgalmi feladat

A szigetek száma a megadott feltételek mellett nagyjából 180 és 220 között szór, a szigetek feletti repülés átlagos hossza pedig kevesebb 0,2 kilométernél. Generáljunk olyan véletlen sorozatokat, melyekben 4 ± 1 km hosszúságban repülünk egy-egy sziget felett és ezt 6 ± 1 km hosszúságban tenger feletti útszakasz követi!

Megoldás

Eljárás Mérés

```
mérésDarab=1000
mérés Tömb(1-mérésDarab)
ciklus i= 1 - mérésDarab
  Ha véletlen()<=0,72 akkor
    mérés(i)=0
  egyébként
    mérés(i) = véletlen(1, 640)
  elágazás vége
ciklus vége

földDarab=0
hossz=0; maxHossz=0
vízFelőlHalad=igaz; szigetDarab=0
maxMagasság=0; maxHely=0; maxhelySziget=0
ciklus i= 1 - mérésDarab
  Ha 0<mérés(i)akkor
    földDarab=földDarab+1
    hossz=hossz+1
    Ha maxHossz<hossz akkor
      maxHossz=hossz
    elágazás vége
    Ha vízFelőlHalad akkor
      szigetDarab=szigetDarab+1
      vízFelőlHalad=hamis
    elágazás vége
  egyébként
    hossz=0
    vízFelőlHalad=igaz
  elágazás vége
  Ha maxMagasság<mérés(i) akkor
    maxMagasság=mérés(i)
    maxHely=i
    maxhelySziget=szigetDarab
  elágazás vége
ciklus vége

Ki:
A legmagasabb tereppont magasság értéke: maxMagasság méter
A legnagyobb magasság értéket maxHely/10 kilométernél mérték
A repülőgép útjának 100*földDarab/mérésDarab %-ában szárazföld
felett haladt
A leghosszabb szárazföld felett megtett út maxHossz/10 km volt
A repülőgép szigetDarab sziget felett repült át
A mért legmagasabb tereppont a(z)maxhelySziget. szigeten volt
```

Eljárás vége

A következőkben megmutatjuk a *Mérés feladat* egy másik lehetséges megoldását a megoldás egy-egy érdekes részletén keresztül. Új megoldásunk mellőzi a korábbi megoldás mind a négy elágazását. Ez különösen a maximum-kiválasztás tekintetében figyelemre méltó.

A mérés sorozat egy lehetséges kimenete tizenkét elemű tömbre:

```
0   0   228   0   341 509 216   0    0   72   19   0
A legmagasabb tereppont magasság értéke: 509 méter
A legnagyobb magasság értéket 0.5 kilométernél mérték
A repülőgép útjának 50.0 %-ában szárazföld felett haladt
A leghosszabb szárazföld felett megtett út 0.3 km volt
A repülőgép 3 sziget felett repült át
A mért legmagasabb tereppont a(z) 2. szigeten volt
```

Megoldásunk új változatában az eredeti méréssorozat elemein egymást követő transzformációkat hajtunk végre:

	0	0	228	0	341	509	216	0	0	72	19	0
1.	**0**	**0**	**1**	**0**	**1**	**2**	**3**	**0**	**0**	**1**	**2**	**0**
2.	-1	-1	0	-1	0	1	2	-1	-1	0	1	-1
3.	-1	-1	0	-1	0	1	1	-1	-1	0	1	-1
4.	1	1	0	1	0	1	1	1	1	0	1	1
5.	0	0	1	0	1	0	0	0	0	1	0	0

Az *1.* transzformáció neve *föld*. A transzformáció végrehajtásához a Signum függvényt használjuk fel. A Signum, azaz Előjel függvény +1-et rendel minden pozitív számhoz, -1-et minden negatív számhoz, és 0-át a nulla számhoz.

föld(i)=[Előjel(m_i)+föld(i-1)] * Előjel(m_i)

Az *5.* transzformáció neve *földKezdet*. Ehhez a sorozathoz négy további transzformációval jutunk el. A *2.* lépésben minden *föld* elemből kivonunk 1-et, majd a *3.* lépésben újra felhasználjuk az Előjel függvényt: *Előjel(föld(i)-1).* Ezt követően a *4*. lépésben a létrejött elemek abszolút értékét képezzük, végül az így kiszámolt értékeket kivonjuk 1-ből.

földKezdet(föld(j))=1-Abs[Előjel(föld(j)-1)]

A *földKezdet* sorozatot felhasználva könnyen meg tudjuk válaszolni az ötödik kérdést, anélkül, hogy elágazást használnánk:

Szigetek száma (m_i Tömb)
föld(0)=0; szigetDarab=0;
ciklus i= 1 – 12
szigetDarab=szigetDarab+földKezdet(föld(i));
ciklus vége
Ki: szigetDarab;
Eljárás vége

Megjegyzések:

A *föld* függvény képletében a föld elemek két különböző index értékkel szerepelnek: *i* és *i - 1*. Erre szükség is van, hiszen az egyes transzformált értékek kialakításakor az őket közvetlenül megelőző transzformált értéket is figyelembe kell venni. Amennyiben el szeretnénk menteni a transzformáció során létrejövő föld értékek sorozatát, egy *mérésdarab+1* elemű tömböt kellene tehát létrehoznunk. Függetlenül az értékek elmentésének igényétől a *föld(0)* segéd értéket mindenképpen inicializálnunk kell a függvény helyes működése érdekében: *föld(0)=0.*

Azokat az egész értékeken értelmezett diszkrét függvényeket, amelyeknek adott helyen felvett értéke csak értelmezési tartományuk megelőző, egész helyen felvett értéke alapján számítható ki *rekurzív* függvényeknek nevezzük. A matematikatörténet több híres rekurzív függvényét is ismerjük. Egyike ezeknek például Newton közelítő gyökképlete. Az *N* szám *R(N)* négyzetgyökét a következő közelítő képlettel határozhatjuk meg: $\mathbf{R_n(N)=[R_{n-1}(N) + N/R_{n-1}(N)] / 2}$. A képletből látható, hogy az *n*-edik közelítő értéket csak a megelőző *n-1*-edik közelítő érték ismeretében számíthatjuk ki. Tegyük fel például, hogy 10 négyzetgyökét szeretnénk kiszámítani a képlettel, négy tizedes jegy pontossággal. Legyen $R_1(10)=3$! Ekkor a képlet alapján $R_2(10)=3{,}166666$, $R_3(10)=3{,}162280$, és így tovább, ameddig csak akarjuk. A tényleges érték: $R(10)=3{,}162277\ldots$, azaz már a sorozat harmadik eleme megfelelő pontosságú értéket szolgáltat.

Hasonlóan érdekes a londoni Királyi Csillagászati Társaság ülésén 1822-ben Charles Babbage által bemutatott másodfokú polinom függvény rekurzív változata. Babbage szerette volna gépesíteni a matematikai táblázatokban szereplő, sokszor több ezer szükséges adat kiszámítását az általa megálmodott mechanikus *differenciagép*pel. Gépe elvi működését egy példán keresztül mutatta be: tegyük fel, hogy az $f(x)=x^2+x+41$ függvény egész helyeken felvett értékeire van szükségünk. Ha ismerjük a függvény $x_0=n$ helyen felvett értékét, könnyen kiszámíthatjuk az $x_1=n+1$ helyen felvett értékét is. Ekkor ugyanis az $f(x_1)$ - $f(x_0)$ differencia értéke $(n + 1)^2 + (n + 1) + 41 - (n^2 + n + 41) = 2 (n + 1)$, azaz $\mathbf{f(n+1)=f(n)+(n+1)+(n+1)}$. A polinom egymást követő értékeinek kiszámításához kizárólag az összeadás műveletét kell felhasználnunk, amire már a Pascal által 1642-ben épített mechanikus összeadó gép is képes volt. Például, ha az $x_0=9$ helyen ismert a függvény értéke, ami 131, akkor az $x_1=10$ helyen a függvény értékét könnyen kiszámíthatjuk a 131+10+10 képlettel. Beállítva az x=0 értékhez tartozó 41 kezdő értéket, a gép elindítása után órákig monoton ütemben képes „ontani” a számértékeket. A beszámolók szerint Babbage differenciagépe 32 jegyű számokból percenként 44 darabot állított elő. A gép tengelyezett fémtárcsáinak működtetéséhez szükséges energiát egy korabeli ingaóra mechanikája szolgáltatta, ami a felemelt órasúly helyzeti energiáját képes volt a tárcsák mozgási energiájává alakítani.

A rekurzió a matematikai bizonyításokban is megjelenik. A *teljes indukció* bizonyítási módszere tipikusan ilyen. Akkor alkalmazzuk, ha egy állítást nem tudunk, vagy legalábbis csak nagy nehézségek árán tudnánk egy

felsorolás típusú végtelen sokaság minden elemére egyszerre bizonyítani. Ekkor azzal az ötlettel élünk, hogy feltesszük, van olyan eleme a sokaságnak, amelyre az állítás igaz, majd ebből kiindulva megpróbáljuk bizonyítani az állítást a sokaság következő elemére: $A_n \Rightarrow A_{n+1}$. Lényegében egy „önjáró" logikai automatát generálunk, amely képes végiglépkedni az A_i állítások alkotta logikai „lépcsőn". Ezután már csak egyetlen dolgunk van: automatánkat rá kell tenni az egyik lépcsőfokra, azaz be kell látnunk legalább egyetlen *k* indexre, hogy A_k igaz. A legtöbb esetben *k=1*, de vannak olyan esetek, amikor *k=2, k=3* vagy valamely véges konstans. Az esetleg ily módon előforduló néhány kivétel nem gyengíti a sokaság *k*-nál nagyobb vagy egyenlő indexű végtelen sok elemére bizonyított tétel erejét.

A *föld(**i**)=[Előjel(m_i)+föld(**i-1**)]*Előjel(m_i)* rekurzív képletet valójában írhatjuk így is: *föld(**i**)=[Előjel(m_i)+föld(**i**)]*Előjel(m_i)*. A matematikában az ilyen definíciókat önhivatkozásnak nevezzük és hibásnak tartjuk, ugyanis az éppen születőben lévő *föld(i)* elem meghatározásához úgy használjuk fel magát a *föld(i)* elemet, mintha az már stabilan definiálva lenne, holott éppen most készülünk definiálni azt! A *matematikai-térben* ezért egy x=x+const típusú egyenlet mindig ellentmondáshoz vezet const≠0 esetén. Például az x=x+1 egyenlet rendezés után 0=1 alakú lesz!

A számítógép hardverének szintjén definiálható *folyamat-térben* az x=x+const típusú egyenlet nem vezet ellentmondásra! A folyamat első fázisában ugyanis a processzor meghívja az *x* értéket az operatív tárból címe alapján, és elhelyezi műveleti regiszterei egyikében. A második fázisban generálja a szükséges konstans értéket egy még üres műveleti regiszterében. A harmadik fázisban létrehozza a művelet eredményét az eredmény regiszterben, végül a kapott eredménnyel felülírja az operatív tárnak az *x* változó címéhez tartozó korábbi tartalmát. Ebben semmi ellentmondás nincs. A *folyamat-tér* számára mindez azért lehetséges, mert a *folyamat-tér*ben van idő dimenzió, szemben a *matematikai-tér*rel, amelyben nincs! Legszembeötlőbben a két egyenlőségjelet a Pascal nyelv különíti el azzal, hogy a *folyamat-tér* egyenlőségjelét := (legyen egyenlő!) jellel jelöli.

A maximális hossz kiválasztása érdekében az eredeti sorozat elemein végzett első transzformációs lépés *1.* eredmény sorozatát két újabb, az eddigiektől különböző transzformációnak vetjük alá:

1.	**0**	**0**	**1**	**0**	**1**	**2**	**3**	**0**	**0**	**1**	**2**	**0**
2.′	**1**	**1**	**2**	**1**	**2**	**3**	**4**	**1**	**1**	**2**	**3**	**1**
3.′	**1**	**1**	**2**	**2**	**2**	**3**	**4**	**4**	**4**	**4**	**4**	**4**

A kapott *3.′* sorozatból megtudhatjuk a leghosszabb szárazföld felett megtett út hosszát (4. kérdés: (4-1)*100 m). A *2.′* sorozat egymást követő elemeit most nem egy elágazás feltételeként, explicit módon hasonlítjuk, hanem implicit összehasonlítást végzünk. Az implicit összehasonlítás algoritmusának elemzése után érdekes matematika-filozófiai következményekkel szembesülünk.

Az egyes összehasonlító algoritmusok szerkezetének könnyebb áttekinthetősége érdekében rövidítéseket vezetünk be:

2.′ ≡ föld′(i) = föld(i)+1 ≡ f(i)

Explicit összehasonlítás:

maximumérték=f(1)

ciklus i= 2 – 12

Ha maximumérték < f(i) akkor maximumérték = f(i)

ciklus vége

Implicit összehasonlítás:

Előjel [AlsóEgész(a / b)] ≡ $\lfloor\!\lfloor a, b \rfloor\!\rfloor$ *// b ≠ 0*

$$Max[a, b]=\frac{a * \lfloor\!\lfloor a, b \rfloor\!\rfloor + b * \lfloor\!\lfloor b, a \rfloor\!\rfloor}{\lfloor\!\lfloor a, b \rfloor\!\rfloor + \lfloor\!\lfloor b, a \rfloor\!\rfloor}$$

maximumérték=f(1) *// f(1) ≠ 0*

ciklus i= 2 – 12

maximumérték = Max[maximumérték, f(i)]

ciklus vége

Az implicit összehasonlításnál alkalmazott *Max* függvény lényegében súlyozott átlagot számol, a nagyobb értéket eggyel, a kisebb értéket nullával súlyozva. A súlyok összegével történő osztásnak egyenlő értékek esetén jut fontos szerep. Ekkor ugyanis mindkét súly értéke egy, így az osztás nélkül megduplázódna a maximumérték.

Matematika-filozófiai következmény:

Egy véges halmaz legnagyobb elemének kiválasztása akkor sikerülhet, ha a halmaz bármely két eleméről el tudjuk dönteni, hogy kettőjük közül melyik a nagyobb. Az explicit összehasonlítást alkalmazó algoritmusban éppen ezt a lehetőséget használjuk ki. Hogyan sikerülhetett mégis kikerülni a második algoritmusban a nagyság entitás vizsgálatát? A nullával való osztás elkerülése érdekében bevezetett *2.′* sorozat elemeinek implicit hasonlítása során a hasonlított mennyiségekből tört számot képeztünk, felhasználva egyben a kapott tört reciprokát is. A törtek képzése által megvalósuló osztásban viszont rejtve mégiscsak megjelenik a nagyság entitás, hiszen az osztás ismételt kivonás, és kisebb értékből nem lehet legalább egyszer kivonni egy nagyobb értéket. Egy nagyobb szám tehát nem azért viselkedik nagyobbként egy összehasonlítási műveletben, mert „eredendően ismeri”, hogyan kell viselkednie, hanem azért, mert mi a műveleti szabályokon keresztül előírjuk, hogyan viselkedjen, hogyan „szokás” viselkednie. *A matematikai-térben található számok teste belülről üres, a számok tulajdonságait a gondolat erejével tartjuk fent!*