Számjegyek és oszthatóság

Ionescu Klára

clara@cs.ubbcluj.ro

Általánosságok

- A számjegyekkel foglalkozó feladatok sok más feladat részfeladataiként jelentkeznek.
- Gyakran szükségünk van az „előfordulások”, illetve a „gyakoriság” tömbre.
- Ezek használata sok esetben vezet jobb (optimális) megoldásokhoz.
- Az oszthatóság vizsgálata szintén gyakran jelenik meg.
- Legnagyobb közös osztó, legkisebb többszörös, prímszámok, törzstényezők stb.

Kettőhatvány

*Adott az **n** nem nulla természetes szám* ($0 < n \leq 10^9$). *Döntsük el, hogy kettőhatvány-e! Ha az adott szám nem kettőhatvány, bontsuk kettőhatványok összegére!*

Elemzés

- Generáljuk a kettőhatványokat, amíg kisebbek, mint ***n***
- Miután leállt a generálás, megvizsgáljuk, hogy a legutolsó kettőhatvány egyenlő-e ***n***-nel és így eldöntjük, a választ az kérdésre
- Ha az utolsó kettőhatvány nagyobb, mint ***n***, osztjuk kettővel, így megvan a felbontás első tagja
- Ezt kivonjuk ***n***-ből

Kettőhatvány

- A következő tagokat megkaphatnánk, ha újra végrehajtanánk az előbb leírt műveletsort, amíg ***n*** nullává nem válik
- Észrevétel: így újból elölről kezdenénk a kettőhatványok generálását; ha visszafele haladunk, osztva kettővel az aktuális kettőhatványt, hatékonyabb megoldáshoz jutunk
- Ugyanakkor, ha nem lenne a második követelmény, a döntést sokkal egyszerűbben így oldhatnánk meg:

```
Algoritmus kettőHatvány(n):
        térítsd n & (n - 1) = 0
Vége(algoritmus)
```

```
Algoritmus kettőHatvány(n, m):                    // m kimeneti paraméter
    m ← 1
    Amíg m < n végezd el:
        m ← m * 2
    vége(amíg)
    térítsd m = n
Vége(algoritmus)

Algoritmus felbontás(n, m, k, tagok):// ha k értéke 0 marad ⇒ n kettőhatvány
    k ← 0
    Ha nem kettőHatvány(n, m) akkor
        Amíg n > 0 végezd el:
            Amíg m > n végezd el:
                m ← m / 2
            vége(amíg)
            k ← k + 1
            tagok[k] ← m
            n ← n – m
        vége(amíg)
    vége(ha)
Vége(algoritmus
```

Hasonló számok

*Legyen **x** és **y** két természetes szám, amelyeknek legtöbb 9 számjegyük van. Döntsük el, hogy a két szám **hasonló-e**. Két számot hasonlónak nevezünk, ha számjegyeik halmaza azonos:* 2131 *és* 32211 *hasonló mivel számjegyeik halmaza ugyanaz:* {1,2,3}.

- a) Két tömb használatával
- b) Egy tömb segítségével
- c) Tömbhasználat nélkül

Hasonló számok

Elemzés

a) a feladat megoldása két tömb használatával rendkívül egyszerű:

- meghatározzuk ***x*** és ***y*** *előfordulási* tömbjét;
- összehasonlítjuk a két tömböt, és eldöntjük, hogy a két tömb azonos-e.

b) A feladat megoldása egy tömb használatával szintén egyszerű:

- meghatározzuk ***x*** *előfordulási* tömbjét;
- meghatározzuk, rendre az ***y*** szám számjegyeit, és vizsgáljuk, hogy ez a számjegy előfordult-e az ***x*** számban; ha ***y*** minden számjegyét megtaláljuk, következik, hogy ***x*** és ***y*** azonos. De ezt fordítva is megvizsgáljuk.

c) Ha nem szabad tömböt használni, meg kell vizsgálnunk, hogy ***x*** minden számjegye előfordul-e ***y*** -ban, és fordítva

```
Algoritmus előfordul(x, előf):                                  // a)
                    // létrehozzuk az x szám számjegyeinek előfordulástömbjét
    Mnden i = 0, 9 végezd el:
        előf[i] ← hamis
    vége(minden)
    Amíg x > 0 végezd el:
        előf[x mod 10] ← igaz
        x ← x /10
    vége(amíg)
Vége(algoritmus)

Algoritmus azonos_1(a, b): // eldöntjük, hogy a és b azonosak-e
    előfordul(a, előfA); előfordul(b, előfB)
    i ← 0
    Amíg i ≤ 9 és előfA[i] = előfB[i] végezd el:
        i ← i + 1
    vége(amíg)
    térítsd i > 9
Vége(algoritmus)
```

```
Algoritmus ab(a, b):                                    // b)
// az a előfordulástömbjében vizsgáljuk a b számjegyeinek megfelelő elemeket
    előfordul(a, előfA)
    Amíg b > 0 és előfA[b mod 10] végezd el:
        b ← b /10
    vége(amíg)
    térítsd b = 0
Vége(algoritmus)

Algoritmus azonos_2(x, y):
// x számjegyeit keressük y számjegyei között és fordítva
    térítsd ab(x, y) és ab(y, x)
Vége(algoritmus)
```

```
Algoritmus AszámjegyeiBben(a, b):                    // c)
// vizsgáljuk, hogy a minden számjegye előfordul-e b-ben
    auxB ← b                                          // másolat b-ről
    Amíg a > 0 végezd el:
        aUtolsóSzj ← a mod 10
        Amíg auxB > 0 és aUtolsóSzj ≠ auxB mod 10 végezd el:
            auxB ← auxB / 10
        vége(amíg)
        Ha auxB = 0 akkor
                    // ha auxB 0 lett, az a aktuális számjegye nincs meg b-ben
            térítsd hamis
        vége(ha)
        auxB ← b                                      // visszaállítjuk b értékét
        a ← a / 10                          // levágjuk az a szám utolsó számjegyét
    vége(amíg)
    térítsd igaz
Vége(algoritmus)

Algoritmus azonos_3(x, y):
// vizsgáljuk, hogy x minden számjegye előfordul-e y-ban, és fordítva
    térítsd AszámjegyeiBben(x, y) és AszámjegyeiBben(y, x)
Vége(algoritmus)
```

Rómaiból arab

Adva van egy (garantáltan helyes) római szám, írjuk ki arab számjegyekkel!

Elemzés

- A feladat garantálja, hogy a római szám helyes, így ezt nem szükséges vizsgálni.
- Ismerjük, hogy a római számok számjegyei: 'M', 'D', 'C', 'L', 'X', 'V', 'I', amelyek rendre az 1000, 500, 100, 50, 10, 5, 1 értékeknek felelnek meg.
- Az algoritmus akkor lesz hatékony, ha nem kisiskolás módon alakítjuk át a római számot, hanem találunk egy tömörebb megoldást.

```
Algoritmus arabÉrték(c):
    Esetek c:
        'M' : érték ← 1000
        'D' : érték ← 500
        'C' : érték ← 100
        'L' : érték ← 50
        'X' : érték ← 10
        'V' : érték ← 5
         'I' : érték ← 1
       'm' : érték ← 1000
        …
         'i' : érték ← 1
    vége(esetek)
    térítsd érték
Vége(algoritmus)
```

```
Algoritmus arabSzám(római):
    szám ← 0
    arab1 ← arabÉrték(római[0])
    i ← 1
    Amíg római[i] végezd el:
        arab2 ← arabÉrték(római[i])
        Ha arab1 < arab2 akkor
            szám ← szám - arab1
        különben
            szám ← szám + arab1
        vége(ha)
        arab1 ← arab2
        i ← i + 1
    vége(amíg)
    szám ← szám + arab1
    térítsd szám
Vége(algoritmus)
```

Arabból római

Adva van egy arab szám (< 5000)*, írjuk ki római számjegyekkel!*

Elemzés

- Újból fennáll a veszély, hogy egy „gyerekes” megoldást adjunk
- Észrevesszük, hogy a római számjegyek a következőképpen csoportosíthatók:

('M', 'D', 'C'); ('C', 'L', 'X'); ('X', 'V', 'I')

- A megfelelő értékek:

(1000, 500, 100); (100, 50, 10); (10, 5, 1)

- Bevezetünk egy ***k*** változót, amelynek kezdőértéke 100 (ezzel dolgozunk az első csoporthoz tartozó értékekkel), majd ***k***-t elosztjuk kétszer 10-zel a másik két csoporthoz tartozó értékek feldolgozásának érdekében.

```
Algoritmus arabbóRómai(arabSzám):
    k ← 100; m ← 'M'; d ← 'D'; c ← 'C'
    Ki: "romai szam = "
    Amíg arabSzám ≥ 1000 végezd el:
        Ki: m; arabSzám ← arabSzám - 1000
    vége(amíg)
    Amíg k > 0 végezd el:
        Ha arabSzám ≥ 9*k akkor
            Ki: c, m; arabSzám ← arabSzám - 9*k
        vége(ha)
        Ha arabSzám ≥ 5*k akkor
            Ki: d; arabSzám ← arabSzám - 5*k
        vége(ha)
        Ha arabSzám ≥ 4*k akkor
            Ki: c, d; arabSzám ← arabSzám - 4*k
        vége(ha)
```

```
Amíg arabSzám ≥ k végezd el:
    Ki: c; arabSzám ← arabSzám - k
 vége(amíg)
k ← k / 10
Ha k = 10 akkor
    m ← 'C'
    d ← 'L'
    c ← 'X'
 különben
    m ← 'X'
    d ← 'V'
    c ← 'I'
 vége(ha)
vége(amíg)
Vége(algoritmus)
```

Legnagyobb palindrom

Egy természetes szám palindrom, ha egyenlő azzal a számmal, amelyet úgy kapunk, hogy számjegyeit fordított sorrendben írjuk fel. Adott az ***n*** (0 < ***n*** < 2^{31}) *természetes szám. Határozzuk meg azt a* ***maxPal*** *számot, amely a legnagyobb palindrom, amelyet az* ***n*** *szám minden számjegyének átrendezése által kaphatunk. Ha nem lehet kialakítani a palindromot, amelyben az* ***n*** *szám minden számjegye szerepeljen, akkor* ***maxPal*** *értéke* -1.

1. *Példa:* ha ***n*** = 21523531, akkor ***maxPal*** = 53211235.

2. *Példa:* ha ***n*** = 12272351, akkor ***maxPal*** = -1.

Legnagyobb palindrom

Elemzés

- Létrehozzuk az ***n*** számjegyeinek gyakoriságtömbjét, amelynek feldolgozása lehetővé teszi, hogy a kért számot hatékonyan generálhassuk
- Ha a számban csak páros gyakoriságok léteznek, elhelyezünk az új számba **gyak[i] / 2** darab ***i*** számjegyet, értékeik csökkenő sorrendjében, majd elhelyezünk az új számba **gyak[i] / 2** darab ***i*** számjegyet, értékeik növekvő sorrendjében
- Ha az ***i*** számjegy gyakorisága páratlan, és a szám közepének megfelelő hely még nem foglalt, ide teszünk egy ***i*** számjegyet
- Ha megjelenik még egy számjegy, amelynek gyakorisága páratlan, nem lehetséges palindromszámot generálni

```
Algoritmus palindrom(n):
    Minden i = 0, 9 végezd el:
        gyak[i] ← 0
    vége(minden)
    Amíg n > 0 végezd el:
        gyak[n mod 10] ← gyak[n mod 10] + 1
        n ← n / 10
    vége(amíg)
    szám ← 0                          // a legnagyobb palindromszám
    közép ← -1    // a közép indexű hely még nincs elfoglalva
    i ← 9     // a számjegyeket csökkenő sorrendben dolgozzuk fel
    …
```

```
Amíg i ≥ 0 végezd el:
    Ha gyak[i] mod 2 = 1 akkor
                        // az i számjegy páratlan számszor fordul elő
        Ha közép = -1 akkor     // ha a közép indexű hely szabad
            közép ← i       // elhelyezzük az i szj-t a szám közepére
            gyak[i] ← gyak[i] – 1  // elhasználtunk egy i számjegyet
            i ← i + 1
         különben
            térítsd -1          // ha a közép indexű hely foglalt, nem
                                // tudunk palindromszámot generálni
        vége(ha)
    különben
        hány ← gyak[i] / 2
                // az i számjegyből hány darabot építünk a számba
        ...
```

```
        Minden j = 0, hány – 1 végezd el:
            szám ← szám * 10 + i
        vége(minden)
        gyak[i] ← gyak[i] – hány      // hány darab i-t felhasználtunk
    vége(ha)
    i ← i - 1                          // a következő (kisebb) számjegyérték
vége(amíg)
Ha közép ≠ -1 akkor                    // ha a közép kapott értéket
    szám ← szám * 10 + közép           // beépítjük a számba
vége(ha)
Minden i = 0, 9 végezd el:
            // most növekvő sorrendben dolgozzuk fel a számjegyeket
    Minden j = 0, gyak[i] - 1 végezd el:
            // minden számjegyet beépítünk annyiszor, ahány maradt
        szám ← szám * 10 + i
térítsd szám
Vége(algoritmus)
```

„Erős” számok

- Egy nullától különböző ***sz*** természetes számnak az *erőssége* ***k***, ha bináris alakjában pontosan ***k*** darab 1-es számjegy található.
- Például, a 23 erőssége 4 (kettes számrendszerben felírva, 4 darab 1-es számjegye van).
- Adott számsorozat ***k*** *erősségű csoport*jának nevezzük azt a részsorozatot, amely a sorozat ***k*** erősségű elemeit tartalmazza, az elemek eredeti sorrendjében.
- Például, az ***s*** = (7, 12, 3, 13, 24, 19), sorozat ***k*** = 2 *erősségű csoportja* a (12, 3, 24) részsorozat.

„Erős” számok

- Határozzunk meg minden *erősségi csoportot*, amelyek az ***n*** (1 < ***n*** < 100) elemű ***x*** sorozat elemeiből létrehozhatók. Az elemek különböző természetes számok és kisebbek, mint 30000. Kimenet: a ***csSzáma*** (a csoportok száma) és a ***csoportok*** (a létrehozott csoportok, erősségük szerint növekvően rendezve; a csoporton belül az elemek sorrendje tetszőleges).
- ***Példa:*** ha ***n*** = 6 és ***x*** = (12, 3, 24, 16, 15, 32), akkor ***csSzáma*** = 3, és ***csoportok***: (16, 32), (12, 3, 24), (15).

„Erős” számok

Elemzés

- A megoldás alapművelete az erősség meghatározása vagyis a szám bináris alakjában található 1-es számjegyek számának meghatározása (tanultunk ma valamit?)
- A következő gondunk: hogyan ábrázoljuk a csoportokat?
- A megoldásban szétosztjuk a számokat erősségük szerint egy kétdimenziós tömbbe.
- Az ***i*** erősségű számok az ***i***. sorba kerülnek.
- A csoport számosságát a 0 indexű elem tárolja.

```
Algoritmus erősség(szám):

    erő ← 0
    Ismételd
        szám ← szám & (szám – 1)                    // bitenként
        erő ← erő + 1
    ameddig szám = 0                                // kilépünk, ha szám 0
    térítsd erő
Vége(algoritmus)
```

```
Algoritmus csoportok(n, x, f):
// csoportok meghatározása
// az aktuális elemet az erősségnek megfelelő sorba helyezzük
// a csoport számosságát a 0 indexű elemben tároljuk
    cs ← 0
    Minden i = 1, 16 végezd el:
        f[i][0] ← 0                    // a csoportok számosságának kezdőértékei
    vége(minden)
    Minden i = 1, n végezd el:                    // feldolgozzuk a sorozatot
        erő ← erősség(x[i])                        // az x[i] elem erőssége
        Ha f[erő][0] = 0 akkor                // az x[i]-vel új csoport kezdődik
            cs ← cs + 1                            // nő a csoportok száma
        vége(ha)
        f[erő][0] ← f[erő][0] + 1  // nő az aktuális erőnek megfelelő sor hossza
        f[erő][f[erő][0]] ← x[i]
                        // elhelyezzük az elemet az erősségének megfelelő sorba
    vége(minden)
    térítsd cs
Vége(algoritmus)
```

Előszelet

Sors-számjegynek hívjuk azt a természetes számot, amelyet adott természetes számra a következőképpen számítunk ki: összeadjuk a szám számjegyeit, majd a kapott összeg számjegyeit, és így tovább, amíg a kapott összeg nem válik egyszámjegyű számmá.

Például, a 182 *sors-számjegye* 2 (1 + 8 + 2 = 11, 1 + 1 = 2).

Egy pontosan ***k*** számjegyű ***p*** számot egy legkevesebb ***k*** számjegyű ***q*** szám *előszeleté*nek nevezünk, ha a ***q*** szám első ***k*** számjegyéből alkotott szám (balról jobbra tekintve) egyenlő ***p***-vel.

Például, 17 előszelete 174-nek, és 1713 előszelete 1 713 242-nek.

Előszelet

Legyen az ***sz*** természetes szám ($0 < sz \leq 10\,000$) és az ***m*** sorral és ***n*** oszloppal ($0 < m \leq 100$, $0 < n \leq 100$) rendelkező ***A*** mátrix (kétdimenziós tömb), amelynek elemei 30 000-nél kisebb természetes számok.

Írjunk programot, amely meghatározza és kiírja az ***sz*** szám *leghosszabb előszeletét*, amelyet az adott mátrix elemeinek megfelelő *sors-számjegyeiből* fel lehet építeni. Egy ilyen sors-számjegyet akárhányszor fel lehet használni.

Ha nem építhető fel előszelet, a program írja ki a „*nem létezik előszelet*” üzenetet.

```
Algoritmus sorsSzámjegy(x):
    Amíg x > 9 végezd el:   // x-nek több, mint egy számjegye van
        y ← x                                   // másolat x-ről
        s ← 0                                   // x számjegyeinek összege
        Amíg y > 0 végezd el:
            s ← s + y mod 10
            y ← y / 10
        vége(amíg)
        x ← s                   // x-et felülírjuk számjegyeinek összegével
    vége(amíg)
    térítsd x                               // x-nek most egy számjegye van
Vége(algoritmus)
```

```
Algoritmus prefixMaxSzámjeggyel(szám, m, n, A,
                                számjegyek, számjegyekSzáma):
    Minden i = 0, maxSzjSz -1 végezd el:// előfordulások tömbje
        előfordulások[i] ← 0
    vége(minden)
    Minden i = 0, m - 1 végezd el:
        Minden j = 0, n - 1 végezd el:
            előfordulások[sorsSzámjegy(A[i][j])] ← 1
        vége(minden)
    vége(minden)
    számjegyekSzáma ← 0  // a szam számjegyeinek darabszáma
    Amíg szám > 0 végezd el:
        számjegyekSzáma ← számjegyekSzáma + 1
        számjegyek[számjegyekSzáma] ← szám mod 10
        szám ← szám / 10
    vége(amíg)
```

```
    i ← számjegyekSzáma – 1    // bejárjuk a számjegyek sorozatát
    Amíg i ≥ 0 és előfordulások[számjegyek[i]] végezd el:
    // létezik sorsszámjegy, amely egyenlő az aktuális számjeggyel
        i ← i - 1
    vége(amíg)
    térítsd számjegyekSzáma - i - 1
Vége(algoritmus)
```