Az SMS feladat 8.) részfeladata a következő:

> Határozza meg, hogy a szógyűjteményben mely kódokhoz tartozik több szó is! Írassa ki a képernyőre ezeket a szavakat a kódjukkal együtt egymás mellé az alábbi mintának megfelelően (a szavak sorrendje ettől eltérhet): `baj : 225; bal : 225; arc : 272; apa : 272; eb : 32; fa : 32; dal : 325; fal : 325; eltesz : 358379; elvesz : 358379; fojt : 3658; folt : 3658; ...`

Megoldási lehetőségek:

a.) A hármas feladatban beolvastuk a ***szó*** tömbbe a txt állomány szavait. A ***szó*** tömb elemei tehát alkalmasak az összehasonlítások elvégzésére. Az összehasonlításokhoz ugyanakkor kellenek az egyes szavak kódjai is. Mivel a szavakat többször hasonlítjuk, nem célszerű minden hasonlítás előtt újra és újra generálni, majd ”eldobni” a hasonlított szavak kódjait.

b.) Vegyük észre, hogy a 6.) feladatban egymás után generálni kellett az összes szó kódját. Használjuk fel ezt a kódrészletet újra és a generált kódokat mindjárt írjuk ki egy segéd fájlba, vagy tömbbe, noha a feladat ezt nem kéri. Innentől kezdve konstansként közvetlenül adottak a kódok az összehasonlításokhoz, nem kell azokat ”röptében” generálni. (Amennyiben van elég memóriánk a konstans tömb foglalásához, ez mindig kényelmesebb megoldást jelenthet.) Használjuk tehát a ***kódok*** tömb elemeit!

A feladat megoldására a *megszámolási tétel* egy variánsát használjuk. Az alapötlet a következő: egyesével haladunk a *kódok tömb* elemein és összehasonlítjuk azokat az összes többi tömb elemmel. Így biztosan mindenkit hasonlítottunk, és egyben mindenkit mindenkivel is hasonlítottunk:

```
ciklus i = 1-től  kód darab-ig
    db(i)=0;
    ciklus j = 1-től   kód darab-ig
        ha kód(i)= kód(j) akkor
            db(i)= db(i)+1;
        elágazás vége;
    ciklus vége
ciklus vége
```

A vázolt algoritmus alapján egyrészt látszik, hogy létre kell hoznunk egy *db tömb*-öt, amely tárolja az i-edik kóddal egyező kódok darabszámát. A *db(i)* minden esetben legalább 1, hiszen minden kód megtalálja legalább önmagát. Ha db(i)>1, akkor az i-edik kódból több is van a *kódok tömb*ben. Másrészt az is látszik, hogy az algoritmus nem eléggé hatékony. Öt egyező kód esetén pl. a sorrendben később következő négy további azonos kódhoz való hasonlítás teljesen felesleges, hiszen ugyanazt az öt elemet fogjuk egyezőnek találni, mint az elsőhöz történt hasonlítások alkalmával. Sok tehát a felesleges lépés. (n elem esetén n^2 hasonlítást végzünk, amelyből a példa szerint máris 4n összehasonlítás felesleges) További probléma: az algoritmus nem nyújt információt arról, hogy hány különböző kód van a *kódok tömb*ben.

Javítsuk az algoritmust:

A felesleges hasonlításokat úgy küszöbölhetjük ki, hogy feljegyezzük azoknak az elemeknek a sorszámát, amelyek egyezőnek találtattak egy korábbi összehasonlítás alkalmával és ezekhez az elemekhez, ha rájuk kerül a sor a külső ciklusban, már nem hasonlítunk. Az is belátható, hogy a belső ciklus számlálóját nem kell *1*-től indítani, hiszen az első meg nem talált új elem előtt nem lehet vele azonos elem. Elég tehát a *j* ciklusszámlálót *i* aktuális értékétől, sőt az önmagával való egyezés vizsgálatának elhagyhatósága miatt *i + 1* értéktől indítani.

A felesleges hasonlítások kihagyásának programozására háromféle lehetőség is kínálkozik:

- Vannak nyelvek, amelyek megengedik a számlálós ciklus számlálójának manipulálását a ciklus belsejében abból a célból, hogy ilyenkor átléphessünk a feleslegessé vált futtatásokon:

```
ciklus i=1-től  n-ig
    …
    ha T(i) akkor
       i=i+1;
    elágazás vége
ciklus vége
```

  Ha a *T* tulajdonság igaz értéket vesz fel *i*-re, kikényszerítjük *i* értékének növelését. Mivel a ciklus maga is növeli a ciklusszámláló aktuális értékét a ciklusmag minden végrehajtása után, így *i+1* értékre kimarad a ciklus futása. A Turbo Pascal megengedi ezt, a Free Pascal hibaüzenettel tér vissza. *Meg kell jegyezni, ez a beavatkozás ellentétes a számlálós ciklus szervezési elvével!*
- Feltételes ciklust szervezünk, hiszen a kihagyások miatt előre nem ismerjük a lépésszámot.
- Végül elképzelhető az a megoldás is, hogy megtartjuk a számlálós ciklust, de feltételes elágazással letiltjuk a ciklusmag nem kívánt futásait.

Az alábbi táblázat egy egyszerű mintapéldán követi végig a javított algoritmus egyes lépéseit. Példánkban egy kilencelemű tömb különböző elemeit számoljuk meg. A tömb 'szavait' úgy válogattuk, hogy első betűik azonosíthassák a szavakat az egyszerűbb áttekinthetőség érdekében. A *db tömb*höz funkcionálisan hasonló *adminisztrációs tömb* elemei rekordok. A *azonos* mező arról tájékoztat, hogy azonosnak találtuk-e már az elemet valamely **másik** elemmel, a *db* mező pedig az adott sorszámú elemmel azonosnak talált elemek darabszámát mutatja. (kezdetben minden elem önmagával azonos) Az *i* a külső, *j* a belső ciklus számlálójának aktuális értékeit mutatja. Ha egy elemet az összehasonlítások során azonosnak találunk azzal a minta elemmel, amelyhez hasonlítottuk, akkor *true* bejegyzésre javítjuk a hasonlított elem *azonos* mezejének tartalmát: ***ha*** *szó(i)=szó(j)* ***akkor*** *admin(j).azonos= true*. Amikor a külső ciklusban majd erre az elemre kerül a sor, éppen ezért kimarad. Ezzel egyidejűleg még két bejegyzést teszünk ilyenkor. Egyrészt eggyel növeljük a minta elem *db* mezőjének értékét, hiszen vele egyezőt találtunk: *admin(i).db=admin(i).db+1*. Másrészt a *true* bejegyzés mellett *0*-ra állítjuk a hasonló elem *db* mezőjét: *admin(j).db=0*. Ezzel azt jelezzük, hogy a *j*-edik elem nem önálló érték, csak másolata az *i-edik* elemnek.

sor-szám	szó	admin. azonos	admin. db	i	j		i	j		i	j		i	j		i	j	
				1	2 - 9		2	3 - 9		3	4 - 9		5	6 - 9		8	9 - 9	
1	dm	false	1	d	f	3												
2	kt	false	1	k	f	1	k	f	2									
3	bi	false	1	b	f	1	b	f	1	b	f	1						
4	dn	false	1	d	t	0	d	-	-	d	-	-						
5	aw	false	1	a	f	1	a	f	1	a	f	1	a	f	2			
6	az	false	1	a	f	1	a	f	1	a	f	1	a	t	0			
7	kv	false	1	k	f	1	k	t	0	k	-	-	k	-	-			
8	ce	false	1	c	f	1	c	f	1	c	f	1	c	f	1	c	f	1
9	do	false	1	d	t	0	d	-	-	d	-	-	d	-	-	d	-	-
összehasonlítások száma:				8			5			3			2			0		

Látható, hogy a hasonlítások száma $n^2 = 81$ helyett mindössze 18.

Az admin tömb tartalma (keretezve) a megszámolás után:

36	**dm**	**1**	**f**	**3**
58	**kt**	**2**	**f**	**2**
24	**bi**	**3**	**f**	**1**
36	**dn**	**4**	**t**	**0**
29	**aw**	**5**	**f**	**2**
29	**az**	**6**	**t**	**0**
58	**kv**	**7**	**t**	**0**
23	**ce**	**8**	**f**	**1**
36	**do**	**9**	**t**	**0**

Ha összeszámoljuk a végeredmény tömbben a nem *0* bejegyzéseket, megkapjuk az egymástól különböző elemek darabszámát (5 féle). Kikereshetjük, melyik elemből van a legtöbb (az elsőből 3). A legtöbbször előforduló elemhez tartozó *i* sorszámérték alapján (*i = 1*) megkaphatjuk a leggyakoribb kódot (*kod(1)=36*). Ennek alapján kiválogathatjuk a kapott leggyakoribb kódhoz tartozó azonos kódok sorszámát(*1, 4, 9*), mely sorszámok felhasználásával a forrás szavakat is kiírathatjuk (*szo(1)=dm, szo(4)=dn, szo(9)=do*).

Végül észrevehetjük, hogy az admin tömbünk redundáns. A *0* bejegyzések ugyanis egyértelműen azonosítják a hasonlított ’másolat’ elemeket, tehát a *true* és *false* bejegyzések elhagyhatók. Ennek figyelembe vételével a megszámolási algoritmus így alakul:

```
ciklus i = 1-től   N-ig
      db(i)=1;
ciklus vége
ciklus i = 1-től  (N-1)-ig
      ciklus j = (i+1)-től  N-ig
            ha  ( (db(i)≠0 ) és ( db(j) ≠ 0) )  akkor
                ha  kód(i)= kód(j) akkor
                    db(i)= db(i)+1;
                    db(j)=0;
                elágazás vége;
            elágazás vége;
      ciklus vége
ciklus vége.
```