## Informatika felvételi feladatok – megoldásokkal

A kolozsvári Babeş-Bolyai Tudományegyetem Matematika és Informatika Karán először az idén lehetett informatikából felvételizni. Az új felvételi rendszer értelmében csupán két írásbeli vizsga volt (algebra és matematikai analízis, valamint mértan és trigonometria vagy informatika). A végeredménybe az érettségi átlag is beszámított 33%-ban. Azoknak a felvételizőknek, akik matematika vagy informatika tantárgyversenyen országos szakaszig jutottak (a X-XII. osztályok valamelyikében) mindkét vizsgát eleve tízesnek ismerték el. Aki megyei versenyen díjat kapott (szintén a X-XII. osztályok valamelyikében), annak egyik vizsgáját ismerték el tízesnek (a felvételiző választotta meg, hogy melyiket). Ugyancsak a felvételiző döntött, hogy mértanból vagy informatikából vizsgázik, függetlenül attól, hogy melyik szakra jelentkezett. Idén három szakra lehetett jelentkezni: informatika, matematika-informatika és matematika. Magyar nyelven mindegyik szakon 25-25 hely volt. (Román nyelven ezenkívül még hároméves informatika szak is létezik). Az alábbiakban közöljük a feladatokat és azok megoldását is.

**I.** Bontsunk tényezőkre egy adott pozitív egész számot, mint az alábbi példában:

700=2^2*5^2*7^1,

ahol a ^ jel a hatványozást jelenti. Írjuk le a megoldási módszert, az algoritmust pszeudokódban és megjegyzésekkel ellátott Pascal programozási nyelven!

**II.** Ellenőrizzük, hogy az $f:\{0, 1, ..., n\} \rightarrow \{0, 1, ..., n\}$ függvény (ahol $n$ 1000-nél nem nagyobb természetes szám) bijektív-e, és ha igen, akkor adjuk meg az inverzét is! Írjuk le a megoldási módszert, és adjunk meg egy megjegyzésekkel ellátott Pascal-programot! Ellenőrizni kell a bemeneti adatok helyességét.

**III.** Adottak az $S_1, S_2, ...., S_n$ egész számokat tartalmazó halmazok. Határozzuk meg az $S_1 \times S_2 \times ... \times S_n$ Descartes-szorzat azon $s = (s_1, s_2, ..., s_n)$ elemeit, amelyekre $s_1 < s_2 < ... < s_n$. Írjuk le a megoldási módszert, és adjunk meg egy megjegyzésekkel ellátott Pascal-programot. A bemeneti adatokat helyesnek tekintjük.

**IV.** Írjunk egy Pascal-eljárást két, egész számokat tartalmazó, növekvő sorrendbe rendezett sorozat összefésülésére úgy, hogy az eredmény egy csökkenő sorrendbe rendezett sorozat legyen! (A bemeneti számok nem feltétlenül különbözőek.)

**V.** Adott a $P(X) = a_0X^n + a_1X^{n-1} + ... + a_n$ egész együtthatójú polinom. Határozzuk meg a polinom egész gyökeit, valamint azok multiplicitását! Írjuk le a megoldási módszert, valamint egy megjegyzésekkel ellátott Pascal-programot! A bemeneti adatokat helyesnek tekintjük.

### Megoldások

**I.** A megadott számot mindaddig osztjuk 2-vel ameddig osztható. Ezzel megkapjuk az osztó multiplicitását is. Ugyanezt folytatjuk 3-mal, majd egyesével növelve minden számmal ameddig $n$ 1 nem lesz. Jelöljük $i$-vel az aktuális osztót, $k$-val pedig a multiplicitását. Így mindig csak prím osztókat kapunk. (Természete-

sen nem kellene minden *k*-t megvizsgálni, elég lenne csak a 2-t és az egynél nagyobb páratlan számokat.)

Az algoritmus pszeudokódban:

```
Adott n
i:=2
Ciklus Amíg n<>1 végezd el
      k:=0
      Ciklus Amíg i | n végezd el
            k:=k+1
            n:=n/i
      Ciklus vége
      Eredmény: i törzstényezô k-szoros
      i:=i+1
Ciklus vége
```

A Pascal-program a következő:

```
program torzs; {Torzstenyezokre bont egy egesz szamot}
var n, i: longint;
   k: integer;

BEGIN
 repeat write ('n='); readln (n) until n>0;
 write (n, '=');      {kiiras megkezdese}
 i:=2;          {i lehetseges primoszto}
 if n=1 then write('1^1'); {sajatos eset kiirasa}
 while n<>1 do
 begin
   k:=0;
   while n mod i=0 do begin  {primoszto keresese}
            k:=k+1; {multiplicitasa}
            n:=n div i
            end;
   if k<>0 then begin    {primtenyezo kiirasa}
          write(i,'^',k);
          if n<>1 then write('*');
         end;
   i:=i+1;         {ujabb oszto keresese}
 end;
 readln;
END.
```

Az újabb osztó keresésekor az *i*:= *i+1* sort helyettesíteni lehet a következővel:

```
if i=2 then i:=i+1 else i:=i+2; {ujabb oszto keresese}
```

Tehát csak a 2-t és a páratlan számokat vizsgáljuk, ez meggyorsítja a programot.

**II.** Beolvassuk a függvényértékeket, és megvizsgáljuk különbözőek-e, ha igen, akkor a függvény injektív, tehát bijektív (Eleve szürjektív, mivel minden argumentumra beolvasunk egy értéket). A függvényt egy kétdimenziós tömbben őrizzük, az első eleme az argumentum, a második a függvényérték. Az inverz függvény kiírásához rendezzük a tömb elemeit a függvényértékek szerint, hogy a kiírás növekvő sorrendbe történjék.

```
program bijektiv;     {bijektivitas ellenorzese}
const max=1000;
```

```
var f: array[0..max,1..2] of integer;
  n, i, j, x, y : integer;
  bij   : boolean;
BEGIN
  repeat
    write('n='); readln (n)
  until n<=max;
  {fuggvenyerteket beolvasasa ellenorzessel}
  for i:=0 to n do
    begin
      f[i,1] := i;
      repeat
       write('f(',i,')=');
       readln(f[i,2])
      until (f[i,2]>=0) and (f[i,2]<=n)
    end;
  {bijektivitas ellenorzese}
  bij := true;
  for i:= 0 to n do
   for j:=i+1 to n do if f[i,2]=f[j,2] then bij:=false;
  {fuggveny inverze -
     rendezni kell a fuggvenyertekek szerint}
  if not bij
  then write ('A fuggveny nem bijektiv')
  else begin
      for i:= 0 to n do
       for j:= i+1 to n do
        if f[i,2] > f[j,2] then
         begin
          x:=f[i,2];    y:=f[i,1];
          f[i,2]:=f[j,2]; f[i,1]:=f[j,1];
          f[j,2]:=x;    f[j,1]:=y;
         end;
      writeln ('Az inverz fuggveny:');
      for i := 0 to n do
        writeln ('g(', f[i,2], ')=', f[i,1])
    end;
   readln;
END.
```

**III.** Ez a feladat a visszalépéses (backtracking) módszerrel oldható meg. Megoldásunk rekurzívan oldja meg a feladatot (tulajdonképpen rejtett backtracking). A rekurzívhívást az első halmaz minden elemére indítjuk. A rekurzív eljárás sorra megvizsgálja a következő halmaz elemeit, és amelyek nagyobbak nála, azokra újra hívja önmagát. Az elemeket egy $r$ vektorban őrizzük meg. Egyéb változók:

$s$ – kétdimenziós tömb, amely az $S_i$ halmazok elemeit őrzi soronként,
$p$ – tömb, amely az $S_i$ halmazok elemszámát tartalmazza

```
program Descartes; {Descartes-szorzat novekvo elemekkel}
var s  : array[1..50, 1..50] of integer;
    p,r : array[1..50] of byte;
    n, i, j, k : integer;

procedure keres (a,k:integer); {kovetkezo elem vizsgalata}
var i:integer;
begin
 if k<=n    {ujabb elem keresese}
```

```
 then begin r[k-1] := a;
      for i:=1 to p[k] do
       if a < s[k,i] then keres (s[k,i],k+1);
   end
 else begin  {egy megoldas kiirasa}
     r[n]:=a;
     for i:= 1 to n do write (r[i]:3);
     writeln
   end;
end;

BEGIN
  write ('n='); readln (n);
  for i:=1 to n do      {elemek beolvasasa}
    begin
     write (i, '. halmaz elemszama: ');
     readln (p[i]);
     for j:=1 to p[i] do
       begin
         write (' ':5, j, '. elem: ');
         readln (s[i,j])
       end;
    end;
  {Az elso halmaz minden elemevel keresest inditunk}
  for i:=1 to p[1] do keres (s[1,i],2);
  readln;
END.
```

**IV.** Az összefésülés rendezett sorozatokra alkalmazható (pl. növekvő sorozatokra). Összehasonlítjuk a kér sorozat első elemét, a kisebbiket beírjuk az eredménysorozatba, majd a megfelelő sorozatban (ahonnan a kisebbik elemet vettük) továbblépünk. Ha valamelyik sorozat befejeződik, akkor a másikat egyszerűen csak átmásoljuk. Az eljárás lényege, hogy mindegyik sorozaton csak egyszer megy végig. Ez a feladat annyiban különbözik a szokásos összefésülésnél, hogy növekvően rendezett sorozatokat egy csökkenő sorozattá kell összefésülni. A két sorozatot visszafele olvassuk (tehát pl. *n*-től *1*-ig). Az eljárás paraméterei:

*a,b* – a két bemeneti növekvő sorozat, *na* illetve *nb* elemszámúak.
*c* – eredménysorozat, ennek elemszáma *nc* (természetesen *nc=na+nb*)

```
procedure ossze (a:sorozat; na:integer;
        b:sorozat; nb:integer;
        var c:sorozat; var nc:integer);
var i,j:integer;

begin
 i:=na; j:=nb; nc:=0;
 while (i>0) and (j>0) do
   begin
     nc:=nc+1;
     if a[i] > b[j]
       then begin c[nc]:=a[i]; i:=i-1 end
       else begin c[nc]:=b[j]; j:=j-1 end;
   end;
 while i>0 do begin nc:=nc+1; c[nc]:=a[i]; i:=i-1 end;
 while j>0 do begin nc:=nc+1; c[nc]:=b[j]; j:=j-1 end;
end;
```

**V.** A program lényege, hogy a szabad tag osztóit sorra behelyettesíti (először pozitív, majd negatív előjellel véve) a polinomba. Ha valamelyik *a* osztó gyök, akkor a polinomot *x–a*-val osztva a maradék polinomot újra megvizsgáljuk. Így a többszörös gyököket is megkapjuk. A programban használt tömbök:

*a* – a polinom együtthatóit őrzi,
*d* – a szabad tag osztóit őrzi,
*s* – az egész gyököket őrzi, a többszörös gyököket többször egymás után (a kiírásnál megszámoljuk a multiplicitást)

```
program p;   {egesz gyokok, tobbszorosek is}
type vector = array[0..100] of integer;
var m,n,i,k,t : integer;
  a,d,s,r: vector;

procedure Horner (c:integer);  {Horner-sema}
var p:vector;   {a,s,n,t globalis valtozok}
  k:integer;
begin
 p[0]:=a[0];
 for k:=1 to n do p[k]:=p[k-1]*c+a[k];
 if p[n]=0 then
 begin
 {ha c gyok, a maradek polinomra ujra megvizsgaljuk}
   t:=t+1; s[t]:=c; n:=n-1;
   for k:=0 to n do a[k]:=p[k];
   Horner (c);   {rekurziv hivas}
 end;
end;

BEGIN
 write('n='); readln(n);
 for i := 0 to n do
 begin
   write('a',i,': '); readln(a[i]);
 end;
 writeln('Egyutthatok:');  {egyutthatok kiirasa}
 for i:=0 to n do write(a[i]:5); writeln;
 m := 0; t := 0;    {s-ben orizzuk a gyokoket}
 while (a[n]=0) and (n>0) do
   begin t:=t+1; s[t]:=0; n:=n-1 end;  {0 multiplicitasa}
 for i:= 1 to abs(a[n]) do
  if a[n] mod i = 0 then begin m:=m+1; d[m]:=i end;
    {d-ben orizzuk a szabad tag osztoit}
 for i:=1 to m do
 begin
  Horner (d[i]); {gyok vizsgalata, tobbszorossege is}
  Horner (-d[i]);
 end;
 writeln('Egesz gyokok:');
 if t=0
  then write('  nincsenek')
  else begin  {gyokok kiirasa, multiplicitassal}
      i:=1;
      while i<=t do
       begin
        k:=1;
        write (s[i]:5);
        i:=i+1;
```

```
          while (i<=t) and (s[i]=s[i-1])
           do begin k:=k+1; i:=i+1 end;
           writeln (' multiplicitasa: ', k)
         end;
      end;
  readln;
END.
```

A kiírás egyszerűbb lehet, ha nem kérjük külön a gyök multiplicitását, hanem annyiszor kiírjuk, ahányszor gyökként megjelent:

```
if t=0 then write('  nincsenek')
   else begin
      for i:=1 to t do write(s[i]:5);
      writeln;
     end;
```

Természetesen, csak egy-egy lehetséges megoldást adtunk. Még nagyon sok más, jó megoldás is elképzelhető. Fontos, hogy a feladatot helyesen értelmezzük, és annak megfelelően oldjuk meg. Lényeges, hogy betartsuk mindazt, amit a feladat kimondottan kér. Például, ahol a feladat kéri a bemeneti adatok helyességét, akkor azért biztos pont jár. A feladatok általában azonos értékűek, tehát mindegyiket 1-től 10-ig osztályozzák (hivatalból jár egy pont, tehát tulajdonképpen 2-től osztályoznak). Optimizálni, szépíteni már csak a helyes megoldást érdemes.

**Kása Zoltán**

# *Híradó*

## Pécsi Kémikus Diákszimpózium

Az 1999. április végén sorra kerülő szimpóziumon a résztvevő diákok a tantervben előírt kötelezettségeken felül végzett munkáikat tudományos előadások keretében mutathatják be és vitathatják meg. A legjobb előadások díjakban részesülnek.

Plenáris előadásokon egyetemi oktatók, kutatók mutatják be legújabb tudományos eredményeiket.

Az előzetes jelentkezés beküldési határideje 1998 december 1.

A tudományos és fejlesztő munka iránt elkötelezettséget érző 8. osztályos, középiskolás, vagy 1999-ben már I. éves hallgatók jelentkezését várják a felkészítő tanáraikkal együtt.

A részletek iránt az EMT székhelyén lehet érdeklődni.