O problemă de COMBINATORICĂ

Tiberiu Socaciu

Dedic acest articol domnului Mihail Hudrea, profesor la Liceul de Informatică "Tiberiu Popoviciu" din Cluj, pentru că nu pot uita orele de ASSIRIS (limbajul de asamblare al calculatoarelor FELIX 256) pe care le ținea cu noi acum 15 ani (pe atunci eram elev al Liceului de Matematică-Fizică Nr. 2 - Informatică din Cluj). Chiar dacă domnia sa se ocupă acum mai mult de matematică, sper că va citi acest articol cu plăcere.

O problemă de numărare

Enunțul acestei probleme de combinatorică l-am primit de la *dl. Mihail Hudrea*: "*Fie n un număr natural. Notăm cu S_n mulțimea tuturor numerelor formate numai cu cifrele 1 și 3, astfel încât suma cifrelor să fie n. Să se afle numărul de elemente ale mulțimii S_n.*"

Rezolvarea problemei

Vom încerca să scriem un număr din S_n folosind un număr de a cifre de 1 și b cifre de 3; deoarece suma cifrelor este n, înseamnă că a și b verifică ***ecuația diofantică*** (adică o ecuație ai cărei coeficienți și ale cărei soluții sunt numere întregi) $a + 3b = n$, cu condițiile evidente $a \geq 0$ și $b \geq 0$, adică $a = n - 3b$, unde b ia valori succesive între 0 și $[n / 3]$, unde $[x]$ reprezintă partea întreagă a numărului real x. Pentru o soluție (a, b) a ecuației diofantice există un număr de $C_{a+b}^{a} = C_{a+b}^{b}$ elemente distincte ale mulțimii S_n care respectă condiția menționată. Prin urmare, numărul de elemente ale lui S_n este $C_n^0 + C_{n-2}^1 + C_{n-4}^2 + \cdots + C_{n-2\cdot[n/3]}^{[n/3]}$, unde suma are $[n / 3] + 1$ termeni.

Enunțul original

Enunțul original cerea să se arate că în anumite condiții impuse asupra valorii n, numărul de elemente ale mulțimii S_n este un pătrat perfect. Rezolvarea se bazează pe scrierea unor relații de recurență de forma $X_n - X_{n-1} = X_{n-3}$, unde cu X_n am notat numărul de elemente ale mulțimii S_n.

Generalizarea problemei

O primă generalizare este imediată: "*Fie n un număr natural și a o cifră diferită de 1 și 0. Notăm cu S_n mulțimea tuturor numerelor formate numai cu cifrele 1 și a, astfel încât suma cifrelor să fie n. Să se afle numărul de elemente ale mulțimii S_n.*"

Rezolvarea problemei generalizate

Procedând analog, obținem că numărul de elemente ale mulțimii S_n este $C_n^0 + C_{n-(a-1)}^1 + C_{n-2\cdot(a-1)}^2 + \cdots + C_{n-[n/a]\cdot(a-1)}^{[n/a]}$, unde suma are $[n / a] + 1$ termeni.

Evident, pentru $a = 3$ obținem problema inițială. Scăzând termenii consecutivi ai șirului X_n (definit ca în paragraful anterior) și aplicând regula lui *Pascal* pentru toți termenii, mai puțin primul (care se reduce deoarece are valoarea 1) obținem $X_n - X_{n-1} = X_{n-a}$ (trebuie remarcat faptul că pentru $n \equiv 0 \pmod a$ în X_n și X_{n-a} apare încă un termen egal cu 1). Evident, valorile de pornire sunt $X_i = 1$ pentru $i = 1, ..., a - 1$ și $X_a = 2$.

Implementarea

Pentru implementare am ales un șir X de dimensiune a în care reținem de fiecare dată numai ultimii a termeni ai șirului, adică exact termenii de care avem nevoie pentru a calcula termenul curent cu ajutorul recurenței. Pentru implementare a fost ales limbajul *Java* și a fost testat pe o mașină *Linux* cu *JDK 1.2* (vezi [*Java*]).

Listing CALCUL.JAVA:

```
public class calcul{
  private static int rezolva(int a,
                             int n){
    if ((n<=1)||(a<=1))
      return 0;
```

```
    int x[]=new int[a];
    x[0]=2;
    for(int i=1;i<a;i++)
      x[i]=1;
    for(int i=a+1;i<=n;i++)
      x[i%a]=x[(i-1)%a]+x[(i-a)%a];
    return x[n%a];
  }

  public static void main(String
                                  args[]){
    System.out.println(rezolva(3,20));
  }
}
```

O generalizare mai puternică

Încercăm acum o generalizare mai puternică, în sensul că vom lăsa pentru alegerea cifrelor o mai mare libertate.

"*Fie n un număr natural și fie $M = \{a_1, \ldots, a_p\}$ o mulțime de p cifre într-o bază oarecare de numerație. Notăm cu S_n mulțimea tuturor numerelor formate numai cu cifre din M, astfel încât suma cifrelor să fie m. Să se afle numărul de elemente ale mulțimii S_n.*"

De remarcat este faptul că dacă $M = \{1, \ldots, n\text{-}1\}$, atunci problema este echivalentă cu determinarea numărului de partiții ale lui *n*.

Combinări binomiale și combinări multinomiale

Din definiția combinatorială a combinărilor rezultă imediat ca proprietate formula de dezvoltare binomială a lui *Newton*: $(a+b)^n = C_n^0 a^n + \cdots + C_n^k a^{n-k} b^k + \cdots + C_n^n b^k$, unde $C_n^k = \frac{n!}{k!(n-k)!}$. (Datorită proprietăților binomiale, acestea combinări se numesc și ***combinări binomiale***). Justificarea coeficienților formulei lui *Newton* se face pe baza numărării termenilor monomiali asemenea, obținuți în urma dezvoltării (vezi [*Manual*]).

Prin extindere, vom denumi ***combinare multinomială*** coeficientul din dezvoltarea unui multinom la o putere *n* dată. Vom nota cu $(n \mid k_1, \ldots, k_p)$ coeficientul termenului $x_1^{k_1} \cdots x_p^{k_p}$ din dezvoltarea $(x_1 + \ldots + x_p)^n$. Acesta este egal cu $\frac{n!}{k_1! \cdots k_p!}$.

Demonstrația este relativ simplă și se bazează pe metoda inducției matematice. Este evident că $k_1 + \ldots + k_p = n$, deoarece fiecare monom din dezvoltare are gradul *n*. Pentru $p = 2$, obținem combinările binomiale: $(n \mid k, n - k) = C_n^k = C_n^{n-k}$.

Pentru implementare am preferat să folosim definiția cu factoriali a combinărilor (vezi funcția `Combinare` din programul **`CALCUL.PAS`**).

Rezolvarea noii probleme generalizate

Încercăm să scriem un număr din S_n folosind x_1 cifre a_1, …, x_p cifre a_p. Deoarece suma cifrelor este *n*, avem că x_i verifică ecuația diofantică $a_1x_1 + \ldots + a_px_p = n$, respectând condițiile evidente $x_i \geq 0$.

Pentru o soluție $(x_1, \ldots, x_p)$ a ecuației diofantice există un număr de $(x_1 + \ldots + x_p \mid x_1, \ldots, x_p)$ elemente distincte ale mulțimii S_n dacă se respectă condiția menționată.

Prin urmare, numărul de elemente ale mulțimii S_n este suma combinărilor multinomiale corespunzătoare pentru fiecare soluție a ecuației diofantice.

Backtracking și funcția de continuare pentru rezolvarea ecuației diofantice

Din cele expuse anterior, dificultatea o constituie rezolvarea ecuației diofantice $a_1x_1 + \ldots + a_px_p = n$ în numere naturale. Rezultă că, din punct de vedere tehnic, limitările pentru x_i sunt date de intervalul $[0, n / a_i]$.

Rezolvarea noastră se va baza pe metoda *backtracking*. Pentru aceasta, avem nevoie de o funcție de continuare cu proprietățile din [*Socaciu*].

O bună condiție de continuare poate fi obținută în mod curent din condiții de necesitate. Astfel, presupunând că $(x_1, \ldots, x_p)$ este o soluție, rezultă că se verifică ecuația diofantică și, din pozitivitatea soluțiilor, obținem că $a_1x_1 + \ldots + a_kx_k \leq n$ pentru orice $k = 1, p - 1$. Prin urmare, am obținut o funcție de continuare bună (vezi funcția `fi` din programul **`CALCUL.PAS`**).

Implementarea

Implementarea a fost realizată în *Pascal*, programul fiind compilat pe o platformă *Win32* cu compilatorul *FreePascal 0.99.14* distribuit sub licență *GNU* (vezi [*FreePascal*]).

Teoretic, ar trebui să reușiți compilarea acestui program fără probleme pe orice platformă cu acest compilator, dar și sub clasicele compilatoare *Turbo/Borland* pentru *DOS*.

Listing CALCUL.PAS

```
Program calcul;
     { datele problemei si rezultatul }
type sir=array[1..100] of Integer;
var rezultat,n,p:Integer;
    a: sir;

function combinare(p:Integer;x:sir):
                                 Integer;
{ calculeaza o combinare multinomiala }
{ de forma (x1+...+xp|x1,...,xp)      }
var c:Real;
    s,i:Integer;
begin
  if p=1 then
    if x[1]=0 then combinare:=0
    else combinare:=1
```

```
  else
    begin
    c:=combinare(p-1,x);
    s:=0;
    for i:=1 to p-1 do
      s:=s+x[i];
    for i:=1 to x[p] do
    begin
      Inc(s);
      c:=c*s/i
    end;
    combinare:=trunc(round(c))
  end
end;

{ solutia ecuatiei                    }
{     a1 * x1 + ... + ap * xp = n     }
{ in numere naturale                  }
var x:sir;

function fi(k:Integer):Boolean;
{ functia de continuare }
var i,s:Integer;
begin
  s:=0;
  for i:=1 to k do
    Inc(s,a[i]*x[i]);
  fi:=((k=p) and (s=n)) or
      ((k<p) and (s<=n))
end;

procedure rezolva(k:Integer);
{ determina solutiile ecuatiei }
{ folosind metoda backtracking }
var i:Integer;
begin
  if k=p+1 then
    Inc(rezultat,combinare(p,x))
  else
    for i := 0 to n div a[k] do
    begin
      x[k]:=i;
      if fi(k) then rezolva (k+1)
    end
end;

procedure citeste;
{ citeste datele }
var i:Integer;
begin
  Read(p);
  for i:=1 to p do
    Read(a[i]);
  Read(n)
end;
```

```
Begin
  rezultat:=0;
  citeste;
  rezolva(1);
  writeln(rezultat);
End.
```

Teme

Încercați-vă puterile cu următoarele probleme:

Problema 1

Modificați funcția `rezolva()` din cadrul programului **`CALCUL.JAVA`** prin eliminarea operatorului `%` (modulo) și folosirea operatorului `+=`.

Problema 2

Modificați cele două programe prezentate în cadrul acestui articol astfel încât să accepte datele de intrare dintr-un fișier după un anumit șablon.

Problema 3

Realizați o rutină de verificare a unicității elementelor a_i (elementele șirului *a* trebuie să fie distincte).

Problema 4

Găsiți o relație de recurență pentru combinările multinomiale (analoagă relației triunghiulare a lui *Pascal* de la combinările binomiale) și reimplementați optimal funcția `combinare`. (Eventual, puteți ține cont și de ordinea de apelare a acestei funcții - în versiunea prezentată se apelează în ordinea găsirii soluțiilor ecuației diofantice).

Problema 5

Modificați programul astel încât să se construiască efectiv toate elementele mulțimii S_n (eventual folosind metoda *backtracking*). Care dintre următoareale abordări vi se pare mai bună: imbricarea a două *backtracking*-uri (întâi cel "mare" care rezolvă ecuația, apoi cel "mic" care determină soluțiile de același tip) sau un singur *backtracking*? Argumentați varianta aleasă.

Problema 6

Demonstrați că numărul de elemente ale mulțimii S_n poate fi scris sub forma $k_1(m_1)^n + \ldots + k_q(m_q)^n$. Deoarece există posibilitatea ca cel puțin valoarea absolută a uneia dintre valorile m_i să fie subunitară, rezultă că sunt șanse ca numărul de elemente ale lui S_n să aibă o creștere exponențială, astfel tipul `Integer` să fie repede depășit. Modificați programul astfel încât acest risc să fie eliminat.

Bibliografie

[*FreePascal*] http://www.freepascal.org

[*Java*] http://www.java.sun.com

[*Manual*] Manuale de *Algebră* pentru liceu

[*Socaciu*] Tiberiu Socaciu, *Metoda Backtracking*, revista *if*, nr. 5(16)/1992, pag. 22-25.