Pregătirea onLine a lotului naţional de juniori

- martie 2008 -

Introducere în combinatorică

Autor: prof. Alin Burța
C.N. "B.P. Hasdeu"

Cuprins:

Scurtă prezentare:

Acest minicurs reprezinta o scurta introducere in combinatorica, definind cateva dintre notiunile de baza şi insistand mai mult asupra algoritmilor de generare in ordine lexicografica a permutarilor, aranjamentelor, combinarilor, submultimilor si a produsului cartezian.

Sunt necesare cunoştințe matematice de bază din teoria mulțimilor, baze de numerație, reguli de calcul in baza 2.

Cunoştințe şi abilități algoritmice: algoritmi cu tablouri unidimensionale şi bidimensionale, lucrul cu structuri repetitive si, eventual, functii.
Algoritmii prezentați nu necesita utilizarea functiilor recursive, deşi se pot da astfel de implementari pentru toti algoritmii prezentati.

Tema de lucru:

1. Parcurgeti cursul si implementati algoritmii prezentați.

1. Permutări şi factoriale

O permutare de n obiecte este un mod de aranjare a acestora prin modificarea pozițiilor.

Să presupunem că avem 3 obiecte, notate cu a,b, respectiv c. Avem 6 modalități de aranjare (permutări) şi anume:

abc, acb, bac, bca, cab, cba

În combinatorică apar frecvent probleme care cer fie determinarea permutărilor de n obiecte (generarea permutărilor) care satisfac, eventual, anumite condiții, fie numărarea acestora.

În continuare încercăm să găsim un algortim pentru generarea permutărilor de n obiecte şi să determinăm şi numărul acestora.

Pentru n=1 şi obiectul {a}, avem o singură modalitate de aranjare: *a*
Pentru n=2 şi obiectele notate {a,b}, se observă uşor că avem permutările: *ab* şi *ba*
Pentru n=3:
Să construim permutările de 3 obiecte pe baza permutărilor de două obiecte generate la pasul anterior.
Pentru permutarea ab vom insera al treilea obiect, c, în toate pozițiile posibile (înaintea lui a, între a şi b, respectiv după b):

cab, acb, abc

Aplicăm acelaşi procedeu pentru permutarea *ba*:

cba, bca, bac

Astfel am obținut cele 6 permutări enumerate mai sus.

Pentru n obiecte, algoritmul de generare poate fi rezumat astfel:
- Considerăm, pe rând, fiecare permutare de n-1 obiecte şi obținem alte n permutări de n obiecte, prin inserarea obiectului n în şirul celor n-1 obiecte inițiale, în toate pozițiile posibile.

În privința numărului permutărilor de n obiecte, sa-l notam P_n avem:

$P_n = n * P_{n-1} = n *(n-1) * P_{n-2} = n *(n-1) * (n-2)*P_{n-3} = ... = n *(n-1) * (n-2)*...* 3 * 2 * 1$

adică produsul primelor n numere naturale. Convenim să notăm acest produs cu n! (se citeşte n factorial). Prin convenție, 0! = 1.
Valoarea n! creşte foarte repede, chiar şi pentru valori mici ale lui n, astfel că 10! = 3 628 800 este considerat o linie de separare între lucrurile care se pot calcula în timp decent şi celelalte.

Pentru a evita calcularea valorii n! prin produse succesive, se poate folosi şi formula aproximativă următoare:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

descoperită de matematicianul James Stirling, unde e este o constanta utilizata ca baza a logaritmilor naturali (aproximativ egala cu 2,72)

Diferența dintre valoarea lui n! şi valoarea obținută cu ajutorul formulei este de aproximativ $\frac{1}{12n}$.

Algoritm pentru generarea permutărilor în ordine lexicografică

Determinarea permutărilor unei mulțimi cu n elemente se poate reduce la a genera permutările mulțimii {1,2,…,n}. Memorăm elementele mulțimii într-un tablou unidimensional şi permutăm de fapt indicii elementelor şi nu elementele propriu-zise.

Algoritmul prezentat în secțiunea precedentă nu permite generarea permutărilor în ordine lexicografică, de aceea vom arăta cum putem genera, pornind de la permutarea {1,2,…,n} toate permutările de n elemente în ordine lexicografică crescătoare.

Să analizăm puțin primele permutări de 5 elemente aranjate în ordine lexicografică:

```
1 2 3 4 5
1 2 3 5 4
1 2 4 3 5
1 2 4 5 3
1 2 5 3 4
1 2 5 4 3
1 3 2 4 5
1 3 2 5 4
1 3 4 2 5
1 3 4 5 2
1 3 5 2 4
1 3 5 4 2
1 4 2 3 5
1 4 2 5 3
1 4 3 2 5
1 4 3 5 2
1 4 5 2 3
1 4 5 3 2
1 5 2 3 4
1 5 2 4 3
1 5 3 2 4
1 5 3 4 2
1 5 4 2 3
1 5 4 3 2
2 1 3 4 5
```

Pentru a trece de la o permutare dată la următoarea permutare în ordine lexicografică, parcurgem permutarea de la dreapta la stânga şi găsim primul element p_i cu proprietatea că

$$p_i < p_{i+1} \quad \text{şi} \quad p_{i+1} > p_{i+2} > \ldots > p_n$$

Permutarea următoare se obține prin înlocuirea lui p_i cu cel mai mic dintre elementele următoare cu proprietatea de a fi mai mare decât p_i. Apoi se inversează ultimele n-i elemente pentru ca acestea să apară în ordine crescătoare.

De exemplu, trecerea de la permutarea `1 3 5 4 2 la permutarea 1 4 2 3 5` se realizează astfel: găsim $p_i = 3$; 4 este cel mai mic element mai mare decât 3, deci interschimbăm cele două valori şi ordonăm ultimele 3 elemente.

Pentru generarea tuturor celor n! permutări ale mulțimii {1,2,…,n} pornim de la permutarea identică 1,2,3,…,n şi aplicăm algoritmul de mai sus până când se obține permutarea n,n-1,…,3,2,1, care este cea mai mare permutare în ordinea lexicografică.

Programul C++ care implementează algoritmul este:

```
int main()
{
 int p[100],n,i,k,poz,min,j;
 ofstream g("lexic.out");
 cin>>n;
 for(i=1;i<=n;i++) p[i]=i;
 do{
    poz=n;
    while(p[poz]<p[poz-1] && poz>1) poz--;
    poz--;
    if(poz)
     {
      min=p[poz+1]; j=poz+1;
      for(i=poz+1;i<=n;i++)
        if(min>p[i] && p[i]>p[poz]) min=p[i], j=i;
      k=p[poz], p[poz]=p[j]; p[j]=k;
      for(i=1;i<=(n-poz)/2;i++) k=p[poz+i],p[poz+i]=p[n-i+1],p[n-i+1]=k;
      for(i=1;i<=n;i++) g<<p[i]<<" "; g<<endl;
     }
 }while(poz);
 g.close();
 return 0;
}
```

2. Aranjamente

Să considerăm n obiecte. O modalitate de ***a selecta şi aranja*** *k obiecte distincte dintre cele n date se numeşte aranjament. Evident, k<=n !*
Fie mulțimea M={a_1,a_2,…a_n} Toate modalitățile de a selecta şi aranja k elemente din cele n elemente ale mulțimii M poartă numele de aranjamente de k elemente din mulțimea M sau aranjamente de n elemente luate cate k.
Numărul de aranjamente se notează cu A^k_n.

Exemplu: Pentru M={1,2,3}, aranjamentele de 2 obiecte sunt:

1, 2
1, 3
2, 3
2, 1
3, 1
3, 2

Observați că aranjamentele (1,2) si (2,1) sunt considerate distincte.

Se poate demonstra ca $A^k_n = \frac{n!}{(n-k)!}$

Algoritm pentru generarea aranjamentelor de k obiecte in ordine lexicografica

Pas1. Consideram pentru inceput cel mai mic aranjament, in ordine lexicografica, de k elemente, anume:

{1, 2, ..., k}

Pas2. Presupunem ca aranjamentul curent este {a_1,a_2,…a_k}. Determinarea aranjamentului succesor se reduce la:

- Determinarea celui mai mare indice i cu proprietatea că a_i mai poate fi marit, adica macar una din valorile a_i+1, a_i+2,…,n nu a fost deja inclusa in aranjament;
- Inlocuim valorile a_i,a_{i+1},…a_k cu cele mai mici valori disponibile (nealese), ordonate crescator

Pas 3. Se repeta PAS2 până când nu mai poate fi determinat un indice cu proprietea de mai sus. Asta înseamnă că s-a ajuns la cel mai mare aranjament în ordine lexicograficp şi anume:

{n,n-1, … , n-k+1}

3. Submulțimi şi combinări

O modalitate de a alege k obiecte din n obiecte date, fără a conta ordinea, se numeşte combinare.
Fie mulțimea $M=\{a_1,a_2,\ldots a_n\}$ *Toate modalitățile de a selecta k elemente distincte din cele n elemente ale mulțimii M poartă numele de combinări de k elemente din mulțimea M sau combinari de n elemente luate câte k.*
Numărul de combinari se notează cu C^k_n.

Exemplu:

Pentru M={1,2,3}, combinarile de 2 obiecte sunt:

1, 2
1, 3
2, 3

Exprimarea lui C^k_n se poate face observand ca, in cazul aranjamentelor de n obiecte luate cate k, fiecare alegere apare de k! ori: alegand, de exemplu elementele 1, 2 , 3 din multimea {1,2,3,4} se vor alege si :

1, 3, 2
2, 1, 3
2, 3, 1
3, 1, 2
3, 2, 1

Asta inseamna $C^k_n=\frac{A^k_n}{k!}$ => $C^k_n=\frac{n!}{k!(n-k)!}$

Citind cu atentie definitia şi considerațiile de mai sus, putem trage concluzia că o combinare de k elemente reprezintă de fapt o submulțime cu k elemente a mulțimii M considerate iar C^k_n reprezintă.numărul acestor submulțimi.
După cum se ştie din teoria mulțimilor, numărul de submulțimi ale unei mulțimi cu n elemente, inclusiv mulțimea vidă, este de 2^n, astfel avem:

$$C^0_n+C^1_n+\ldots+C^n_n=2^n$$

C^0_n – *numarul modalitatilor de alegere a 0 elemente din n = 1*
C^1_n – *numarul modalitatilor de alegere a submultimilor de 1 element = n*
. . .
C^n_n – *numarul modalitatilor de alegere a n elemente din n = 1*

Datorita acestei identitati matematice, combinarile mai poarta numele de **coeficienți binomiali**.

Formula obtinuta mai sus poate fi dificil de aplicat pentru calculul C^k_n, pentru ca intervin factorialele, a caror valoare creste foarte repede. O modalitate mai buna de calcul este sugerata de triunghiul lui Pascal, compus din coeficientii binomiali calculati pentru n=1,2,3,...:

K	C^k_0	C^k_1	C^k_2	C^k_3	C^k_4	C^k_5	C^k_6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	**4**	**6**	4	1		
5	1	5	***10***	10	5	1	
6	1	6	15	20	15	6	1

e.t.c

Se observa ca fiecare element, cu exceptia celor de pe diagonala principala si de pe prima coloana, se poate calcula prin insumarea celor doua elemente de pe linia precedenta aflate deasupra, respectiv la stanga sa, deci

$$C^k_n = C^{k-1}_{n-1} + C^k_{n-1}$$

Formula de mai sus ne permite sa calculam C^k_n din apropape in aproape si fara a efectua inmultiri. Deoarece valoarea C^k_n creste si ea destul de repede, calculul valorii C^{50}_{100} de exemplu, nu se poate realiza decat prin utilizarea **adunarii cu numere mari**.

Algoritm pentru generarea submultimilor unei multimi

Fie mulțimea A={a_1,a_2,…a_n}, memorata intr-un tablou unidimensional notat X. Ne propunem sa generam toate cele 2^n submulțimi ale sale.

Să consideram mai intai urmatoarea problema:
Fiind dat un numar natural n, sa se determine in cate moduri putem construi un tablou unidimensional avand elemente din multimea {0, 1} ? (problema numarului de vectori binari cu n elemente)

Primul element al tabloului, sa-l notam v[1], poate fi ales in doua moduri, la fel si v[2], v[3],…,v[n]
De exemplu, daca n=3 avem solutiile:
```
0, 0, 0
1, 0, 0
0, 1, 0
1, 1, 0
0, 0, 1
1, 0, 1
0, 1, 1
1, 1, 1
```

Se observa ca, pentru fiecare din cele doua alegeri facute pentru v[1] se pot face 2 alegeri pentru v[2] si, de asemenea, pentru fiecare alegere facuta pentru elementul v[2] se pot face 2 alegeri pentru v[3], adica 2x2x2 solutii.
Generalizand, numarul tablourilor unidimensionale cu elemente 0 si 1 este 2^n.

Rezultatul precedent ne permite sa facem "o paralela" intre problema generarii submultimilor si problema vectorilor binari. Iata cum, pentru multimea A={1, -3, 8}

Tabloul unidimensional	**Submultimea corespunzatoare**
0, 0, 0	Multimea vida
1, 0, 0	{1}
0, 1, 0	{3}
1, 1, 0	{1, -3}
0, 0, 1	{8}
1, 0, 1	{1, 8}
0, 1, 1	{-3, 8}
1, 1, 1	{1, -3, 8}

Tabloul unidimensional v se mai numeste si vector caracteristic, deoarece o submultime se construieste astfel:

> *pentru fiecare i=1,…,n, daca v[i]=1 atunci consideram elementul corespunzator X[i] ca facand parte din submultime*

de aceea, pentru rezolvarea problemei noastre initiale, va fi suficient sa generam vectorii caracteristici.

Algoritmul propriu-zis foloseste proprietatile adunarii numerelor scrise in baza 2, de fapt "tabla adunarii" din aceasta baza:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 (+ 1 de "ținut minte" = transport catre pozitia urmatoare)

(**obs.** *Reamintiti-va ca, atunci cand calculati 25 + 175 in baza 10 zicem 5+5 = 0 si una in minte, 7+5=12+1 din minte, e.t.c)*

Algoritm pentru generarea submultimilor:
Pas 1. Se porneste cu vectorul caracteristic v, avand toate cele n elemente egale cu zero si-l consideram a fi numarul 0 = 000…0 (n de 0) in baza 2

Pas 2. Adunam 1 la numarul curent si obtinem o noua submultime.
Afisam/prelucram elementele submultimii.

Pas 3. Repetam pasul 2 pana cand vectorul caracteristic contine n elemente egale cu 1

Generarea submultimilor in ordine lexicografica

La fel ca mai sus consideram mulțimea A={a_1,a_2,…a_n}, memorata intr-un tablou unidimensional, notat X si ne propunem generarea submultimilor sale in ordinea crescatoare a numarului de elemente.
Fiecare submultime va fi generata, pe rand, intr-un tablou unidimensional v, care va memora indicii elementelor din tabloul X.
Astfel, spunem ca generarea submultimilor unei multimi oarecare A cu n elemente, este echivalenta cu generarea submultimilor multimii {1,2,…,n} si de aceea, in continuare, de vom referi la aceasta.

Prima submultime cu k elemente, in ordine lexicografica, este {1,2,…,k}, deci primele pozitii ale tabloului v sunt ocupate, in ordine, de aceste valori.
Ultima submultime cu k elemente va fi {n-k+1, n-k+2,…, n-1,n}

Trecerea de la o submultime cu k elemente la submultimea urmatoare, in ordine lexicografica, se bazeaza pe urmatoarea observatie:

Pe o pozitie j a tabloului solutie v, valoarea maxima admisa este n-k+j.

Intuitiv, pe pozitia 1 a tabloului v se poate afla valoarea maxima n-k+1, deoarece ultima submultime cu k elemente este {n-k+1, n-k+2,…, n-1, n}. De aici observam ca, pe pozitia 2 se poate afla cel mult valoarea n-k+2, e.t.c

Pas1. k=1
Pas2. Determinam submultimile cu k elemente, astfel:

- Prima submultime este {1,2,…,k}
- Trecerea de la submultimea curenta la urmatoarea se face prin determinarea celui “mai din dreapta element care mai poate fi marit cu o unitate”, fie el v[j]; Marim pe v[j] cu 1 si modificam valoarea elementelor de la dreapta lui v[j] dupa formula:

$$v[p] = v[p-1]+1, \text{ oricare ar fi } p=j+1,\dots,n$$

- Repetam pasii pana la determinarea tuturor submultimilor cu k elemente

Pas3. k = k +1
Pas4. Repeta *Pas 2* pana cand k>n

Observatie:
“Subalgoritmul” pentru determinarea submultimilor cu k elemente, aplicat de n ori in algoritmul de mai sus, constituie de fapt algoritmul de generare a combinarilor de k elemente ale multimii {1,2, …, n} in ordine lexicografica.

4. Generarea produsului cartezian

Fie A şi B două mulţimi cu n, respectiv m elemente. Definim produsul cartezian al muţimilor A şi B, notat A x B, ca fiind mulţimea perechilor care au prima componentă din mulţimea A şi a doua din mulţimea B. Formal, vom scrie:

$$A \times B = \{(a,b) \ / \ a \in A \text{ si } b \in B \}$$

Numarul elementelor acestei mulţimi este n x m.

Multe aplicaţii practice necesită generarea produsului cartezian a N mulţimi.

Fie mulţimile $A_1, A_2, \dots, A_N$ având c_1, c_2, respectiv c_N elemente. Definim produsul cartezian al celor n mulţimi ca fiind mulţimea:

$$A_1 \times A_2 \times \dots \times A_N = \{(a_1, a_2, \dots, a_N) \ / \ a_i \in A_i\}$$

Multimea $A_1 \times A_2 \times \dots \times A_N$ va fi multimea tablourilor unidimensionale cu N elemente, elementul de pe pozitia i apartinand multimii A_i.

Numarul elementelor produsului cartezian a celor N multimi este $c_1 \times c_2 \times \dots \times c_N$.

Algoritm pentru generarea produsului cartezian a N multimi

Multimile noastre avand numar finit de elemente, putem gandi algoritmul pentru multimile:

$A_1 = \{1, 2, \dots, c_1\}$
$A_2 = \{1, 2, \dots, c_2\}$
. . .
$A_N = \{1, 2, \dots, c_N\}$

determinand elementele produsului cartezian in ordine lexicografica.
Vom proceda similar algoritmilor din sectiunile precedente, deci vom incepe cu elementul cel mai mic, in ordine lexicografica:

$$(1, 1, \dots, 1)$$

si ne vom opri la ultimul element in ordine lexicografica:

$$(c_1, c_2, \ldots , c_N)$$

Observam ca elementul de pe pozitia i are cel mult valoarea c_i (ultimul element al multimii corespunzatoare).

Trecerea de la un element oarecare (v[1], v[2], ..., v[N]) la urmatorul element in ordine lexicografica se face astfel:

- Determinam cel “cel mai din dreapta” indice j cu proprietatea ca v[j] mai poate fi marit cu o unitate ($v[j]<c_j$)
- Marim v[j] cu o unitate
- Initializam toate elementele v[p] cu valoarea 1, unde p=j+1, ..., N