bile - descrierea soluţiei

prof. Emanuela Cerchez, Liceul de Informatică "Grigore Moisil" Iaşi

Soluţie 1.

Problema cere să generăm toate submulţimile de `k` elemente ale mulţimii {`1`, `2`, ..., `n`} astfel încât oricare două submulţimi generate consecutiv să difere printr-un singur element.

Numărul de soluţii este, evident, `Comb(n,k)=n!/(k!*(n-k)!)`.

Algoritmul de generare este inspirat din relaţia de recurenţă a lui Pascal:

`Comb(n,k)=Comb(n-1,k)+Comb(n-1,k-1).`

Mai exact submulţimile de `k` elemente ale mulţimii {`1`, `2`, ..., `n`} pot fi partiţionate în două clase: submulţimile care conţin valoarea `n` (acestea sunt `Comb(n-1,k-1)`) şi submulţimile care nu conţin valoarea `n` (acestea sunt `Comb(n-1,k)`).

Să notăm cu `Sn,k` o soluţie corectă (adică o succesiunea combinărilor de `n` luate câte `k` care respectă condiţiile din enunţ).

$S_{n,1}$=`{1} {2} {3}... {n}`, pentru orice `n`

$S_{n,n}$=`{1, 2, 3, ..., n}`

Pentru a genera $S_{n,k}$:

1. vom genera combinările de `n-1` luate câte `k` în ordinea specificată (adică $S_{n-1,k}$)
2. vom genera combinarile de `n-1` luate cate `k-1` în ordinea specificată (adica $S_{n-1,k-1}$), la care reunim elementul `n`
3. Construim $S_{n,k}$ din $S_{n-1,k}$ urmat de $\underline{S_{n-1,k-1}}$U{n} unde cu $\underline{S_{n-1,k-1}}$ am notat $S_{n-1,k-1}$ considerat în ordine inversă.

De exemplu:

$S_{2,1}$={1}{2}

$S_{3,1}$={1}{2}{3}

$S_{3,2}$=$S_{2,2}$ $S_{2,1}$U{3}={1,2}{2,3}{1,3}

$S_{4,1}$={1}{2}{3}{4}

$S_{4,2}$=$S_{3,2}$ $S_{3,1}$U{4}={1,2}{2,3}{1,3}{3,4}{2,4}{1,4}

$S_{4,3}$=$S_{3,3}$ $S_{3,2}$U{4}={1,2,3} {1,3,4} {2,3,4}{1,2,4}

$S_{5,2}$=$S_{4,2}$ $S_{4,1}$U{5}={1,2}{2,3}{1,3}{3,4}{2,4}{1,4}{4,5}{3,5}{2,5}{1,5}

$S_{5,3}$=$S_{4,3}$ $S_{4,2}$U{5}=

{1,2,3}{1,3,4}{2,3,4}{1,2,4}{1,4,5}{2,4,5}{3,4,5}{1,3,5}{2,3,5}{1,2,5}

etc

Se poate demonstra cu uşurinţă prin inducţie că acest procedeu produce o secvenţă de combinări care respectă condiţiile din enunţ.

Observaţi că procedeul de generare este asemănător celui utilizat pentru codul *Gray*.

Vom descrie un algoritm de tip succesor pentru generarea $S_{n,k}$:

Configuraţia iniţială este `s={1,2,...,k}`

Cât timp nu am generat toate combinările:

- afişăm configuraţia curentă;
- generăm configuraţia următoare, dacă există.

Generarea succesorului:

Pentru a nu trata în mod special cazul ultimului element, `s[k+1]=n+1;`

```
for (j=1; j<=k && s[j]==j; j++);
if  (j%2!=k%2)
    if (j==1) s[1]--;
       else {s[j-1]=j; s[j-2]=j-1;}
    else
    if (s[j+1]!=s[j]+1) {s[j-1]=s[j]; s[j]++;}
      else {s[j+1]=s[j]; s[j]=j
```

Soluţie 2 (Zoltan Szabo)

Se cere generarea tuturor combinărilor a n elemente luate câte k astfel încât două mulţimi succesive să difere printr-un singur element.

Să generăm aceste combinări cu metoda backtracking, folosind o stivă st în care vom pune k elemente în ordine strict crescătoare, generând toate soluţiile în ordine lexicografică.

Putem observa, că în toate cazurile în care ultimul element al stivei are succesor ($st[k]<n$), cerinţa cerută va fi respectată (fiindcă ultimul elemente ceşte cu 1, şi toate celelalte elemente rămân neschimbate).

În cazurile în care ultimul element al mulţimii nu are succesor ($st[k]=n$), trecerea către următorul element, nu va respecta cerinţa dorită.

Să observăm, că elementul $st[p]$ nu are successor în ordine lexicografică, dacă $st[p]-p=n-k$. Deci valorile posibile pentru $st[p]$ sunt incluse în intervalul $[st[p-1], n-p-k]$.

Vom modifica algoritmul anterior, astfel încât elementele stivei să poată să aibă la diferite nivele paşii 1 sau -1 astfel:

- pentru pasul 1 valoarea $st[p]=n-p-k$ nu are succesor,

- pentru pasul -1 valoarea $st[p]=st[p-1]$ nu are predecesor .

Combinări în ordine lexicografică	Pasul corespunzător fiecarui element din stivă	Generarea bilelor	Pasul corespunzător fiecarui element din stivă	Explicaţie
1 2 3 4	(1,1,1,1)	1 2 3 4	(1,1,1,1)	Pornim cu prima combinare banală şi pt. fiecare nivel pas=1
1 2 3 5	(1,1,1,1)	1 2 3 5	(1,1,1,1)	
1 2 3 6	(1,1,1,1)	1 2 3 6	(1,1,1,1)	
1 2 4 5	(1,1,1,1)	1 2 4 6	(1,1,1,-1)	Când ultimul nu mai are succesor, revenim, generăm succesorul, iar pentru elementele
1 2 4 6	(1,1,1,1)	1 2 4 5	(1,1,1,-1)	

1 2 5 6	(1,1,1,1)			schimbăm pasul
1 3 4 5	(1,1,1,1)			
1 3 4 6	(1,1,1,1)	1 2 5 6	(1,1,1,1)	Penultimul şi ultimul nivel nu mai au succesor
1 3 5 6	(1,1,1,1)			
1 4 5 6	(1,1,1,1)	1 3 5 6	(1,1,-1,-1)	La nivelul 2 am trecut la succesor, iar nivelele superioare şi-au schimbat pasul
2 3 4 5	(1,1,1,1)	1 3 4 6	(1,1,-1,-1)	
2 3 4 6	(1,1,1,1)	1 3 4 5	(1,1,-1,-1)	
2 3 5 6	(1,1,1,1)	1 4 5 6	(1,1,1,1)	
2 4 5 6	(1,1,1,1)	2 4 5 6	(1,-1,-1,-1)	Etc.etc.
3 4 5 6	(1,1,1,1)	2 3 5 6	(1,-1,-1,-1)	
		2 3 4 6	(1,-1,-1,-1)	
		2 3 4 5	(1,-1,-1,-1)	
		3 4 5 6	(1,1,1,1)	

Să observăm că prin acest procedeu, după ce am generat un element, restul elementelor stivei se pot completa instantaneu până la nivelul *k*. Astfel avem un algoritm backtracking optimizat, care generează toate soluţiile în ordine corectă fără să treacă prin valori succesive nevalide.

Soluţie 3 (80 puncte):

Andrei Grigorean, Universitatea Bucureşti

Vom construi o procedura recursiva care primind doi parametri N si K va genera toate permutarile de N elemente luate cate K care sa resepecte proprietatea din enunt. Vom incerca sa generam mai intai toate combinarile care nu contin elementul N, urmate de toate combinarile care il contin pe N. Pentru a face acest lucru la inceputul procedurii noastre vom face 2 apeluri recursive cu urmatorii parametrii : N-1 si K pentru a genera combinarile care nu il contin pe N, urmat de un apel cu parametrii N-1 si K-1. Pentru combinarile rezultate in urma celui de-al doilea apel, vom adauga la sfarsit elementul N. Astfel am obtinut ceea ce ne-am propus: toate combinarile de N elemente luate cate K. Problema care apare este ca nu avem garantia faptului ca ultima combinare din primul apel si prima din al doilea respecta conditiile din enunt. Ne vom folosi de urmatoarea observatie:

Avand un sir de combinari de N elemente luate cate K care respecta proprietatea din enunt si o functie bijectiva f:N->N, inlocuind peste tot un element x cu valoarea f(x), sirul de combinari care rezulta respecta la randul sau proprietatea din enunt.

Consideram ca ultima combinare rezultata in urma primului apel este v_1, v_2,.. , v_{K-1}, v_K, iar prima combinare rezultata in urma celui de-al doilea apel este u_1, u_2,..., u_{K-1}. Ne vom alege o functie f:(N-1)->(N-1) astfel incat $f(u_1) = v_1$,.... $f(u_{K-1}) = v_{K-1}$ (pentru elementele x din intervalul [1...N-1] care nu se regasesc printre valorile u_i, f(x) poate lua orice valoare care pastreaza bijectivitatea). Nu este deloc dificil sa gasim o astfel de functie bijectiva. Inlocuind fiecare element x din combinarile rezultate in urma celui de al doilea apel cu f(x), obtinem o solutie valida.