Algoritmi de tip succesor. Permutări, aranjamente. Aplicații

Algoritmii de tip succesor se aplică problemelor caracterizate prin următoarele:

- soluția se reprezintă sub forma unui vector $x=(x_1, x_2,..., x_n) \in S$, *cu* $S = S_1 \times S_2 \times ... \times S_n$, unde mulțimile $S_1, S_2,..., S_n$ sunt mulțimi finite şi ordonate;
- se cer toți vectorii care îndeplinesc anumite condiții, sau doar vectorul/vectorii care optimizează o anumită funcție;
- nu este cunoscut un alt algoritm care să construiască direct şi corect componentele vectorului x.

Esența modului de construire a soluției constă în evitarea generării succesiunilor de elemente posibile care să fie apoi verificate şi păstrate sau respinse, după caz. Această alternativă ar conduce la un consum enorm şi inutil de timp, datorită dimensiunii exponențiale a mulțimii S. Modalitatea de construire a soluției constă în renunțarea, cât mai devreme cu putință, la generarea succesiunilor de elemente sortite eşecului (care un au şansa să facă parte din soluție) .

Condițiile pe care componentele vectorului **x** trebuie să le satisfacă se numesc **condiții interne**. Componentele vectorului $\boldsymbol{x}$ se generează pe rând, în ordinea $x_1, x_2,..., x_k,...$ şi, de îndată ce s-a ales pentru x_k o valoare posibilă din S_k, există următoarele posibilități:

1. Nu există nici un vector $\boldsymbol{x}$ care să aibă pe primele $\boldsymbol{k}$ locuri valorile $\boldsymbol{x_1, x_2,..., x_k}$. În acest caz se merge înapoi, la componenta $\boldsymbol{x_{k-1}}$, şi se caută aici o nouă valoare posibilă.
2. Pentru moment valoarea aleasă pentru $\boldsymbol{x_k}$ pare bună, neexistând motive de îndoială în privința condițiilor pe care în final vectorul $\boldsymbol{x}$ va trebui să le satisfacă. În acest caz se verifică dacă ceea ce s-a construit în vectorul x este chiar o soluție a problemei.
   - Dacă da, ea este afişată sau doar reținută pentru unele verificări ulterioare privind optimalitatea ei.
   - Dacă nu, se trece la alegerea primei valori posibile pentru următoarea componentă a lui x.

Condițiile pe care trebuie să le satisfacă $\boldsymbol{x_1, x_2,..., x_k}$ şi pe care le deducem din cele interne se numesc **condiții de continuare**. Esențial este prin urmare ca, din codițiile interne să deducem condiții de continuare cât mai avantajoase, care să ne ajute să evităm construirea unor succesiuni de elemente care nu au nici o şansă să fie finalizate într-o soluție a problemei.

Aplicații

1. Generarea permutărilor

Se dă un număr $n \in$ N*. Să se genereze toate permutările mulțimii $A = \{1, 2,..., n\}$.
De exemplu, pentru n=3, programul va afişa:

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Soluție

O permutare a mulțimii A este o succesiune de n elemente în care fiecare element din A apare exact o dată.

Construim permutările mulțimii A folosind vectorul $x = (x_1, x_2,..., x_n)$, $x_i \in A$.

Condiții interne:

- $\forall i$, $\forall j$ astfel încât $i \neq j$, avem $x_i \neq x_j$
- $x[i] \in \{1, 2,..., n\}$, $\forall i \in \{1, 2,..., n\}$

Condiții de continuare:

- $x_k \neq x_i$, $\forall i \in \{1, 2,..., k-1\}$

```
#include <fstream>
using namespace std;
ifstream fin("permutare.in");
ofstream fout("permutare.out");
int x[20], n;
int valid(int k)
{
int i;
for(i=1;i<k;i++)
        if(x[k]==x[i]) return 0;
return 1;
}

int main()
{
        int k, v, i;
        fin>>n;
        x[1]=0;
        k=1;                                    // k= nivelul curent pe care se construieste solutia
        while(k>0)
                {v=0;                                   //v=1 cand pentru nivelul curent k, exista succesor valid
                 while (v==0 && x[k]+1<=n)
                        {
                           x[k]++;                  //se incearca plasarea pe nivelul k a unei valori urmatoare
                           if (valid(k))v=1;     //daca aceasta valoare este valida, at v=1
                         }                              //daca nu mai exista nici o posibilitate de plasare a unei
                                                        //valori bune pe nivelul k, atunci se revine la nivelul anterior
                        if(v==0)  k--;
                        else
                                if (k<n)
                                        {
                                                k++;          //dupa identificarea unei valori prezumtiv bune
                                                              //pentru x[k] se trece la constructia
                                                x[k]=0;   //nivelului urmator din vectorul solutie
                                        }
                                else                            //s-au construit toate cele n elemente ale unei solutii
                                        {
                                                for (i=1;i<=n;i++)
                                                        fout<<x[i]<<' ';
                                                fout<<'\n';
                                          }
                }
fout.close();
return 0;
}
```

2. Generarea aranjamentelor

Se dau două numere naturale n şi $m \in$ N*, cu m≤n. Se cere generarea tuturor aranjamentelor de n elemente luate câte m, unde $A = \{1, 2,..., n\}$.
De exemplu, pentru n=3 şi m=2 programul va afişa:

1 2
1 3
2 1
2 3
3 1
3 2

Soluție

Condiții interne:

- $\forall i$, $\forall j$ astfel încât $i \neq j$, avem $x_i \neq x_j$
- x[i] $\in \{1, 2,..., n\}$, $\forall i \in \{1, 2,..., m\}$

Condiții de continuare:

- $x_k \neq x_i$, $\forall i \in \{1, 2,..., k-1\}$

```
#include <fstream>
using namespace std;
ifstream fin("aranjamente.in");
ofstream fout("aranjamente.out");

int x[20], n ,m;
int valid(int k)
{
	int i;
	for(i=1;i<k;i++)
		if(x[k]==x[i])	return 0;
return 1;
}

int main()
{
	int k,v,i;
	fin>>n>>m;
	if(n<m)
		fout<<"Nu exista solutii\n";
	else{
		x[1]=0; k=1;
		while(k>0)
			{
				v=0;
				while (v==0 && x[k]+1<=n)
					{
						x[k]++;
						if (valid(k))v=1;
	    }
				if(v==0) k--;
				else if (k<m)	//numarul de elemente din solutie este m
```

```
                    {
                        k++;
                        x[k]=0;
                    }
                else
                    {
                        for (i=1;i<=m;i++)
                            fout<<x[i]<<' ';
                        fout<<'\n';
                    }
            }
        }
fout.close();
return 0;
}
```

Aplicații

1.Problema reginelor

Să se determine toate modalitățile în care pot fi plasate *n* regine pe o tablă de şah de dimensiune *n* x *n* (0<n<10) astfel încât oricare două regine să nu atace (să nu existe două regine pe o aceeaşi linie, coloană sau diagonală).

2. *Permutări fără puncte fixe*

Pentru numărul natural $n \in$ N* dat, se cere să se genereze toate permutările mulțimii {1, 2,..., *n*} care au proprietatea de a nu conține puncte fixe. O permutare are puncte fixe dacă există măcar o componentă a permutării care coincide cu poziția sa. De exemplu, permutarea *p* =(2 1 3) are un singur punct fix (*p*[3] = 3).

3. *Permutări speciale 1*

Pentru numărul natural $n \in$ N* dat, se cere să se genereze toate permutările mulțimii {1, 2,..., *n*} care au proprietatea că fiecare componentă în afară de prima are în fața ei măcar o componentă de care să difere printr-o unitate.

4. *Permutări speciale 2*

Pentru numărul natural $n \in$ N*, dat se cere să se genereze toate permutările mulțimii {1, 2,..., *n*} care au proprietatea că pe poziții pare există doar valori pare. De exemplu, pentru *n* = 4, *p* = (3 4 1 2) îndeplineşte condiția.

5. Se citesc 0<n≤9 cifre distincte. Construiți şi afişați toate numerele de m (m≤n) cifre care se pot forma din cele n.

6. Să se afişeze toate drapelele cu trei culori care se pot forma cu şase culori: alb, galben, roşu, verde, albastru şi negru, care au în mijloc culoarea alb, verde sau roşu.

7. campion.edu.ro/arhiva

- jucarii
- anag
- reteta2

Bibliografie

Cornelia, Ivaşc, Anamaria, Rusu, Mona, Prună, *Tehnici de programare. Aplicații, Ed. Alexandru Myller, Iaşi, 2006*

http://www.campion.edu.ro/arhiva