Complexitatea algoritmilor

Prof. Diaconescu Corneliu

Şcoala Gimnazială Balcani

În cele ce urmează vom prezenta o problemă tratată prin mai multe metode şi vom prezenta metodic diferențele complexității algoritmilor corespunzători. Este important să alegem un algoritm eficient atât din perspectiva complexității timp cât şi din punctul de vedere al complexității spațiu. Dacă spațiul în general nu reprezintă o problemă, nu acelaşi lucru îl putem spune despre timp, deoarece aici putem obține şi algoritmi netractabili.

Permutarea

Se citesc n numere întregi. Să se verifice dacă aceste numere pot fi o permutare a mulțimii M={1, 2, 3, …, n}.

Rezolvare: Amintim că o permutare a unei mulțimi este formată din elementele aceleiaşi mulțimi considerate eventual în altă ordine. O primă idee de lucru ar fi ordonarea elementelor în ordine crescătoare urmată apoi de o verificare dacă pe fiecare poziție în şir avem un element cu valoarea egală cu numărul de ordine în şir. Dacă şirul citit este a atunci trebuie pe fiecare poziție în şir să avem a[i] = i. Dacă pentru unul din elemente nu este satisfăcută relația înseamnă că un element lipseşte şi atunci se poate opri verificarea. Această parcurgere nu consumă decât un număr de n operații, dar în schimb sortarea va fi de un ordin mai mare decât cel liniar. Prezentăm în continuare algoritmul în pseudocod.

```
Citeşte n                          1 operație
Pentru i←1, n, 1
  execută
      Citeste ai                   n operații
Sfârşit pentru
Pentru i←1, n-1, 1   execută
    Pentru j← i+1, n, 1
      Daca a[i]>a[j] atunci //3*n*(n-1)/2 operații
            aux← a[i]; a[i]← a[j]; a[j]← aux;
          Sfârşit dacă
    Sfârşit pentru
Sfârşit pentru
Ok← True                           //1 operație
Pentru i← 1, n, 1 execută
      Dacă a[i]<> i
           Atunci  Ok ← false //n operații
      Sfârşit dacă
Sfârşit pentru
Scrie Ok                           //1 operație
```

Algoritmul prezentat mai sus este de ordinul $O(n^2)$.

Vom încerca în cele ce urmează să prezentăm un algoritm de complexitate mai mică şi anume vom căuta un algoritm liniar. Ideea de lucru este următoarea. Dacă vom avea toate elementele permutării în şirul de valori citite (şirul **a**), atunci vom marca într-un al doilea şir **b** că am găsit acel element. Ideea de lucru ar putea fi următoarea:

- Vom citi n şi elementele şirului a,
- Inițializăm toate elementele din şirul b cu 0,
- Dacă o anumită valoare k este găsită în şirul a, de exemplu pe poziția i, atunci vom marca în şirul b acest lucru prin o atribuire de forma: b[a[i]]←1, adică b[k]←1,
- Inițializăm cu **True** o variabilă OK,
- Parcurgem şirul **b** şi în cazul în care un element al său este 0 înseamnă că şirul a nu este permutare. În acest caz facem variabila OK **False** şi ne oprim din parcurgere,
- Afişăm variabila OK.

În final dacă variabila OK a ramas cu valoarea True înseamna că în şirul **a** am găsit toate elementele permutării primelor n numere naturale, în caz contrar a nu este o permutare. Vom prezenta în continuare algoritmul corespunzător:

```
Citeşte n                              1 operație
Pentru i←1, n, 1
  execută
       Citeste ai                      n operații
        bi← 0                          n operații
Sfârşit pentru
Pentru i←1, n, 1   execută
    b[a[i]]←1                          n operații
Sfârşit pentru
Ok← True                               1 operație
Pentru i← 1, n, 1 execută
      Dacă b[i]<> 1
        Atunci  Ok ← falsen operații
      Sfârşit dacă
Sfârşit pentru
Scrie Ok                               1 operație
```

Observăm că acest algoritm are 4*n + 3 operații deci ordinul său de complexitate este O(n). Aşadar acest algoritm este mult mai eficient decât primul. În acest algoritm am trecut şi inițializarea lui **b** cu zero, dar amintim că în ambele limbaje utilizate în articol variabilele globale sunt inițializate în mod implicit cu valoarea zero.

Vom scrie în continuare programul corespunzător în cele două limbaje de programare Pascal şi C/C++:

LIMBAJUL PASCAL	LIMBAJUL C/C++

Program permutare;
Var a, b:array [1..200] of integer;
N,i: integer; OK : boolean;
begin
write ('n='); readln(n);
for i:=1 to n do
begin
write ('a[', i, ']=');
readln (a[i]);
b[i]:=0; {poate lipsi}
end;
for i:=1 to n do
b[a[i]]:=1;
OK:=true;
for i:=1 to n do
if b[i]<>1 then
begin
OK:=false; break;
end;
write(OK);
end. Ordinul algoritmului este O(n).
	# include <iostream.h>
	int a[201], b[201], n, i, OK;
	
	void main()
	{
	cout<<"n="; cin >> n;
	for (i=1; i<=n; i++)
	{
	cout<<"a["<<i<<"]=";
	cin>>a[i];
	b[i]=0;
	}
	for (i=1; i<=n; i++) b[a[i]]=1;
	OK=1;
	for (i=1; i<=n; i++)
	if (b[i]!=1)
	{
	OK=0; break;
	}
	cout<<OK;
	} Ordinul algoritmului este O(n).