Curs 5: Metode elementare de sortare
- Problematica sortării
- Sortare prin inserţie
- Sortare prin selecţie
- Sortare prin interschimbarea elementelor vecine
- Exerciţii

1 Problematica sortării

Se consideră un set finit de obiecte, fiecare având asociată o caracteristică, numită *cheie*, care ia valori într-o mulţime pe care este definită o relaţie de ordine. *Sortarea* este procesul prin care elementele setului sunt rearanjate astfel încât cheile lor să se afle într-o anumită ordine.
Exemplul 1. Considerăm setul de valori întregi: (5,8,3,1,6). În acest caz cheia de sortare coincide cu valoarea elementului. Prin sortare crescătoare se obţine setul (1,3,5,6,8) iar prin sortare descrescătoare se obţine (8,6,5,3,1).
Exemplul 2. Considerăm un tabel constând din nume ale studenţilor şi note: ((Popescu,9), (Ionescu,10), (Voinescu,8),(Adam,9)). În acest caz cheia de sortare poate fi numele sau nota. Prin ordonare crescătoare după nume se obţine ((Adam,9),(Ionescu,10),(Popescu,9), (Voinescu,8)) iar prin ordonare descrescătoare după notă se obtîne ((Ionescu,10),(Popescu,9),(Adam,9),(Voinescu,8)).

Pentru a simplifica prezentarea în continuare vom considera că setul prelucrat este constituit din valori scalare care reprezintă chiar cheile de sortare şi că scopul urmărit este ordonarea crescătoare a acestora. Astfel, a ordona setul $(x_1, x_2, \ldots, x_n)$ este echivalent cu a găsi o permutare de ordin n, $(p(1), p(2), \ldots, p(n))$ astfel încât $x_{p(1)} \leq x_{p(2)} \leq \ldots \leq x_{p(n)}$.

De asemenea vom considera că elementele setului sunt stocate pe un suport de informaţie care permite accesul aleator la date. În acest caz este vorba despre *sortare internă*. În cazul în care suportul de informaţie permite doar accesul secvenţial la date trebuie folosite metode specifice încadrate în categoria metodelor de *sortare externă*.

Pe de altă parte, în funcţie de spaţiul de manevră necesar pentru efectuarea sortării există:

- Sortare ce foloseşte o zonă de manevră de dimensiunea setului de date. Dacă setul iniţial de date este reprezentat de tabloul $x[1..n]$, cel sortat se va obţine într-un alt tablou $y[1..n]$.
- Sortare în aceeaşi zonă de memorie (sortare pe loc - *in situ*). Elementele tabloului $x[1..n]$ îşi schimbă poziţiile astfel încât după încheierea procesului să fie ordonate. Este posibil ca şi în acest caz să se folosească o zonă de memorie însă aceasta este cel mult de dimensiunea unui element şi nu de dimensiunea întregului tablou.

În continuare vom analiza doar metode de sortare internă în aceeaşi zonă de memorie. Metodele de sortare pot fi caracterizate prin:

- *Simplitate.* O metodă este considerată simplă dacă este intuitivă şi uşor de înţeles.
- *Eficienţă.* O metodă este considerată eficientă dacă nu necesită un volum mare de resurse. Din punctul de vedere al spaţiului de memorie o metodă in situ este mai eficientă decât una decât una bazată pe o zonă de manevră de dimensiunea tabloului. Din punct de vedere al timpului de execuţie este important să fie efectuate cât mai puţine operaţii. În general în

analiză se iau în considerare doar operaţiile efectuate asupra elementelor tabloului (comparaţii şi mutări).

- *Naturaleţe.* O metodă de sortare este considerată naturală dacă numărul de operaţii scade odată cu *distanţa* dintre tabloul iniţial şi cel sortat. Un măsură a acestei distanţe poate fi numărul de inversiuni al permutării corespunzătoare tabloului iniţial.

- *Stabilitate.* O metodă de sortare este considerată stabilă dacă ordinea relativă a elementelor ce au aceeaşi valoare a cheii nu se modifică în procesul de sortare. De exemplu dacă asupra tabelului cu note se aplică o metodă stabilă de ordonare descrescătoare după notă se obţine ((Ionescu,10),(Popescu,9),(Adam,9),(Voinescu,8)) pe când dacă se aplică una care nu este stabilă se va putea obţine ((Ionescu,10),(Adam,9),(Popescu,9),(Voinescu,8)).

În continuare vom considera câteva metode *elementare* de sortare caracterizate prin faptul că sunt simple, nu sunt cele mai eficiente metode dar reprezintă punct de pornire pentru metode avansate. Pentru fiecare dintre aceste metode se va prezenta: principiul, verificarea corectitudinii şi analiza complexităţii.

2 Sortare prin inserţie

Principiul acestei metode este:

> *Începând cu al doilea element, fiecare este inserat pe poziţia adecvată în subşirul care îl precede.*

La fiecare etapă a sortării, elementului x_i i se caută poziţia adecvată în subşirul destinaţie $x[1..i-1]$ (care este deja ordonat) comparând pe $x[i]$ cu elementele din $x[1..i-1]$ începând cu $x[i-1]$ şi creând spaţiu prin deplasarea spre dreapta a elementelor mai mari decât $x[i]$. Astfel structura generală a algoritmului este:

```
inserţie(x[1..n])
FOR i ← 2, n DO
    ||< inserează x[i] în subşirul x[1..i − 1] astfel încât x[1..i] să fie ordonat>
```

iar cea detaliată este:

```
inserţie1(x[1..n])
FOR i ← 2, n DO
    ||j ← i − 1                          { i − 1 este indicele de la care se porneşte căutarea }
    ||aux ← x[i]                         { salvarea valorii lui x[i] }
    ||WHILE aux < x[j] ∧ j ≥ 1 DO        { căutarea poziţiei pentru inserţie }
    ||    ||x[j + 1] ← x[j]              { deplasarea lui x[j] cu o poziţie la dreapta }
    ||    ||j ← j − 1
    ||x[j + 1] ← aux                     { inserarea valorii analizate pe poziţia adecvată }
RETURN(x[1..n])
```

Există şi alte metode de implementare a metodei. Astfel, pentru a evita efectuarea comparaţiei $j \geq 1$ la fiecare iteraţie interioară se plasează pe poziţia 0 în tabloul x valoarea lui $x[i]$, această poziţie jucând rolul unui *fanion*. Astfel cel mai târziu când $j = 0$ se ajunge la $x[j] = x[i]$. Această variantă poate fi descrisă după cum urmează:

```
inserţie2(x[1..n])
FOR i ← 2, n DO
    ‖x[0] ← x[i]                { plasarea valorii lui x[i] pe poziţia fanionului }
    ‖j ← i − 1
    ‖WHILE x[0] < x[j] DO
    ‖     ‖x[j + 1] ← x[j]
    ‖     ‖j ← j − 1
    ‖x[j + 1] ← x[0]            { inserarea valorii fanion pe poziţia adecvată}
RETURN(x[1..n])
```

Verificarea corectitudinii. Precondiţia problemei este $\{n \geq 1\}$ iar postcondiţia este $\{x[1..n]$ este ordonat crescător$\}$. Pentru ciclul exterior (după i) demonstrăm că proprietatea invariantă este $\{x[1..i-1]$ este ordonat crescător $\}$. La începutul ciclului $i = 2$ deci $x[1..1]$ este implicit crescător. La sfârşitul prelucrării ($i = n + 1$) invariantul implică evident postcondiţia. Rămâne să arătăm că proprietatea rămâne adevărată şi după efectuarea prelucrărilor din cadrul ciclului. Pentru aceasta este suficient să arătăm că pentru ciclul interior (după j) aserţiunea $\{x[1..j]$ este crescător şi $aux \leq x[j+1] = x[j+2] \leq \ldots \leq x[i]\}$ este invariantă. La sfârşitul ciclului WHILE aceasta proprietate ar implica una dintre relaţiile:

- $x[1] \leq \ldots \leq x[j] \leq aux \leq x[j+1] = x[j+2] \leq \ldots \leq x[i]$ în cazul în care condiţia de ieşire din ciclu este $aux \geq x[j]$;
- $aux \leq x[1] = x[2] \leq \ldots \leq x[i]$ în cazul în care condiţia de ieşire din ciclu este $j = 0$.

Oricare dintre aceste două relaţii ar conduce prin atribuirea $x[j+1] \leftarrow aux$ la $x[1] \leq \ldots \leq x[j] \leq x[j+1] \leq x[j+2] \leq \ldots \leq x[i]$ iar prin trecerea la următoarea valoare a contorului ($i \leftarrow i + 1$) la faptul că $x[1..i-1]$ este crescător.

Rămâne doar să justificăm invariantul ciclului WHILE. La intrarea în ciclu au loc relaţiile: $j = i - 1$, $aux = x[i]$ deci $aux = x[j+1] = x[i]$ iar cum $x[1..i-1]$ este crescător rezultă că proprietatea invariantă propusă pentru WHILE este satisfăcută. Arătăm că ea nu este alterată de prelucrările din cadrul ciclului: dacă $aux < x[j]$, prin atribuirea $x[j+1] \leftarrow x[j]$ se obţine: $aux < x[j] = x[j+1] \leq \ldots \leq x[i]$ iar după $j \leftarrow j - 1$ va fi adevărată aserţiunea: $\{ x[1..j]$ crescător şi $aux < x[j+1] = x[j+2] \leq \ldots \leq x[i]$. Cum $x[j+1] = x[j+2]$ prin $x[j+1] \leftarrow x[j]$ nu se pierde informaţie din tablou.

Oprirea algoritmului este asigurată de utilizarea câte unui contor pentru fiecare dintre cele două cicluri.

Analiza complexităţii. Vom lua în considerare doar operaţiile de comparare şi mutare asupra elementelor tabloului. Fie $T_C(i)$ şi $T_M(i)$ numărul comparaţiilor respectiv al deplasărilor efectuate asupra elementelor tabloului pentru fiecare $i = \overline{2,n}$. Cazul cel mai favorabil corespunde şirului ordonat crescător iar cel mai defavorabil celui ordonat descrescător.

În cazul cel mai favorabil: $T_C(i) = 1$ iar $T_M(i) = 2$ (este vorba de operaţiile care implică variabila ajutătoare aux şi care dealtfel în acest caz nu au nici un efect) deci $T(n) \geq \sum_{i=2}^{n}(T_C(i) + T_M(i)) = 3(n-1)$. În cazul cel mai defavorabil $T_C(i) = i$ iar $T_M(i) = i - 1 + 2 = i + 1$, obţinându-se că $T(n) \leq \sum_{i=2}^{n}(2i+1) = n^2 + 2n - 3$.

Prin urmare $3(n-1) \leq T(n) \leq n^2 + 2n - 3$ adică algoritmul sortării prin inserţie se încadrează în clasele $\Omega(n)$ şi $O(n^2)$.

Alte proprietăţi. Algoritmul sortării prin inserţie este natural şi atât timp cât condiţia de continuare a ciclului interior este $aux < x[j]$ este şi stabil. Dacă însă se foloseşte $aux \leq x[j]$ metoda nu mai este stabilă.

3 Sortare prin selecţie

Principiul acestei metode este

> *Pentru fiecare poziţie i, începând cu prima, se selectează din subşirul ce începe cu acea poziţie cel mai mic element şi se amplasează pe locul respectiv (prin interschimbare cu elementul curent de pe poziţia i)*

Structura generală a algoritmului este:

```
selecţie(x[1..n])
FOR i ← 1, n − 1 DO
    ||< se determină valoarea minimă din x[i..n] şi se interschimbă cu x[i] >
```

Ciclul FOR continuă până la $n-1$ deoarece subşirul $x[n..n]$ conţine un singur element care este plasat chiar pe poziţia potrivită, ca urmare a interschimbărilor efectuate anterior.
Descrierea detaliată a algoritmului este:

```
selecţie(x[1..n])
FOR i ← 1, n − 1 DO
    ||k ← i                          { indicele curent al minimului }
    ||FOR j ← i + 1, n DO            { ciclu pentru căutarea minimului }
    ||    ||IF x[k] > x[j] THEN k ← j {noul indice al minimului}
    ||IF k ≠ i THEN x[k] ↔ x[i]      {plasarea minimului pe poziţia adecvată}
RETURN(x[1..n])
```

Verificarea corectitudinii. Arătăm că un invariant al ciclului exterior (după i) este: $\{x[1..i-1]$ este ordonat crescător şi $x[i-1] \leq x[j]$ pentru $j = \overline{i,n}\}$. La început $i = 1$ deci $x[1..0]$ este şir vid. Ciclul FOR interior (după j) determină valoarea minimă din $x[i..n]$. Aceasta este plasată prin interschimbare pe poziţia i. Se obţine astfel că $x[1..i]$ este ordonat crescător şi că $x[i] \leq x[j]$ pentru $j = \overline{i+1,n}$. După incrementarea lui i (la sfârşitul ciclului după i) se reobţine proprietatea invariantă. La final, $i = n$ iar invariantul conduce la $x[1..n-1]$ crescător şi $x[n-1] \leq x[n]$ adică $x[1..n]$ crescător.

Analiza complexităţii. Indiferent de aranjarea iniţială a elementelor, numărul de comparaţii efectuate este:

$$T_C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} (n-i) = n(n-1) - \frac{n(n-1)}{2} = \frac{n(n-1)}{2}$$

În cazul cel mai favorabil (şir ordonat crescător) numărul interschimbărilor este $T_M(n) = 0$. În cazul cel mai defavorabil (şir ordonat descrescător) pentru fiecare i se efectuează o interschimbare (trei atribuiri), deci costul interschimbărilor este $T_M(n) = 3\sum_{i=1}^{n-1} 1 = 3(n-1)$.

Astfel algoritmul sortării prin selecţie aparţine clasei $\Theta(n^2)$.

Alte proprietăţi. Algoritmul este parţial natural (numărul de comparaţii nu depinde de gradul de sortare al şirului). În varianta prezentată (când minimul este interschimbat cu poziţia curentă) algoritmul nu este stabil. Dacă însă în locul unei singure interschimbări s-ar realiza deplasarea elementelor subtabloului $x[i..k-1]$ la dreapta cu o poziţie (ca în algoritmul inserţiei) iar $x[k]$ (salvat în prealabil într-o variabilă auxiliară) s-ar transfera in $x[i]$ algoritmul ar deveni stabil.

4 Sortare prin interschimbarea elementelor vecine

Principiul acestei metode de sortare este:

> *Se parcurge şirul şi se compară elementele vecine iar dacă acestea nu se află în ordinea corectă se interschimbă. Parcurgerea se reia până când nu mai este necesară nici o interschimbare.*

Structura generala a algoritmului este:

```
interschimbări (x[1..n])
REPEAT
    ||< se parcurge şirul şi dacă două elemente vecine nu sunt în ordinea
    ||corectă sunt interschimbate >
UNTIL < nu mai este necesară nici o interschimbare >
```

Să considerăm secvenţa interschimbării elementelor vecine în cazul în care nu sunt în ordinea corectă:

```
FOR i ← 1, n − 1 DO
    ||IF x[i] > x[i + 1] THEN x[i] ↔ x[i + 1]
```

Folosind ca invariant al prelucrării repetitive proprietatea $\{x[i] \geq x[j],\ j = \overline{1,i}\}$ se poate arăta că prelucrarea de mai sus conduce la satisfacerea postcondiţiei: $\{x[n] \geq x[i],\ i = \overline{1,n}\}$. Pentru $i = 1$ proprietatea invariantă este adevărată întrucât $x[1] \geq x[1]$. Presupunem că proprietatea este adevărată pentru i. Dacă $x[i] \leq x[i+1]$ atunci nu se efectuează nici o prelucrare şi rămâne adevărată şi pentru $i+1$. Dacă în schimb $x[i] > x[i+1]$ atunci se efectuează interschimbarea astfel că $x[i] < x[i+1]$ deci proprietatea devine adevărată şi pentru $i+1$.

Pe baza acestei proprietăţi a secvenţei de interschimbări se deduce că este suficient să aplicăm această prelucrare succesiv pentru $x[1..n]$, $x[1..n-1]$, ..., $x[1..2]$. Rezultă că algoritmul poate fi descris prin:

```
interschimbări1(x[1..n])
FOR i ← n, 2, −1 DO
    ||FOR j ← 1, i − 1 DO
    ||IF x[j] > x[j + 1] THEN x[j] ↔ x[j + 1]
RETURN(x[1..n])
```

Verificarea corectitudinii. Întrucât s-a demonstrat că efectul ciclului interior este că plasează valoarea maximă pe poziţia i rezultă că pentru ciclul exterior poate fi considerată ca invariantă proprietatea $\{x[i+1..n]$ este crescător iar $x[i+1] \geq x[j]$ pentru $j = \overline{1,i}\}$.

Analiza complexităţii. Numărul de comparaţii efectuate nu depinde de gradul de sortare al şirului iniţial fiind în orice situaţie:

$$T_C(n) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} 1 = \sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}.$$

În schimb, numărul de interschimbări depinde de proprietăţile şirului astfel: în cazul cel mai favorabil (şir sortat crescător) se obţine $T_M(n) = 0$ iar în cazul cel mai defavorabil (şir sortat descrescător) se obţine $T_M(n) = 3n(n-1)/2$ (o interschimbare presupune efectuarea a 3 atribuiri). Astfel numărul

de prelucrări analizate satisface: $n(n-1)/2 \leq T(n) \leq 2n(n-1)$ adică algoritmul prezentat mai sus aparţine clasei $\Theta(n^2)$.

Algoritmul poate fi însă îmbunătăţit prin reducerea numărului de comparaţii efectuate, în sensul că nu este necesar întotdeauna să se parcurgă tabloul pentru $i = \overline{2,n}$. De exemplu, dacă tabloul este de la început ordonat ar fi suficientă o singură parcurgere care să verifice că nu este necesară efectuarea nici unei interschimbări. Pornind de la această idee se ajunge la algoritmul:

```
interschimbări2(x[1..n])
REPEAT
    ||inter ← F
    ||FOR i ← 1, n − 1 DO
    ||    ||IF x[i] > x[i + 1] THEN
    ||    ||    ||x[i] ↔ x[i + 1]
    ||    ||    ||inter ← A
UNTIL inter = F
RETURN x[1..n]
```

Pentru acest algoritm numărul de comparaţii efectuate în cazul cel mai favorabil este $T_C(n) = n-1$ iar în cazul cel mai defavorabil este $T_C(n) = \sum_{j=1}^{n}(n-1) = n(n-1)$. În ceea ce priveşte numărul de interschimbări acesta este acelaşi ca pentru prima variantă a algoritmului şi anume $0 \leq T_M(n) \leq 3n(n-1)/2$. Prin urmare numărul total de repetări ale prelucrărilor analizate satisface:

$$n - 1 \leq T(n) \leq \frac{5n(n-1)}{2}$$

adică algoritmul este din $\Omega(n)$ şi $O(n^2)$.

Întrucât la sfârşitul şirului se formează un subşir crescător rezultă că nu mai este necesar să se facă comparaţii în acea porţiune. Această porţiune este limitată inferior de cel mai mare indice pentru care s-a efectuat interschimbare. Astfel algoritmul poate fi rescris astfel:

```
interschimbări3(x[1..n])
m ← n                          { la început se va parcurge tot tabloul }
REPEAT
    ||t ← 0                    { în t se va reţine cel mai mare indice pentru }
    ||                         { care se face interschimbare }
    ||FOR i ← 1, m − 1 DO
    ||    ||IF x[i] > x[i + 1] THEN
    ||    ||    ||x[i] ↔ x[i + 1]
    ||    ||    ||t ← i        { se reţine indicele ultimei interschimbări }
    ||m ← t
UNTIL t ≤ 1
RETURN x[1..n]
```

Atâta timp cât condiţia pentru interschimbare este specificată prin inegalitate strictă ($x[i] > x[i+1]$) oricare dintre variantele algoritmului este stabilă.

Valorile comparative ale numărului de prelucrări (comparaţii şi mutări asupra elementelor tabloului) pentru algoritmii de sortare descrişi sunt prezentate în tabelul următor.

Algoritm	Caz favorabil		Caz defavorabil		Clase de complexitate
	$T_C(n)$	$T_M(n)$	$T_C(n)$	$T_M(n)$	$T(n)$
inserţie1	$n-1$	$2(n-1)$	$\frac{n^2+n-2}{2}$	$\frac{n^2+3n-4}{2}$	$3(n-1) \leq T(n) \leq n^2+2n-3$ $(\Omega(n), O(n^2))$
selecţie	$\frac{n(n-1)}{2}$	0	$\frac{n(n-1)}{2}$	$3(n-1)$	$\frac{n(n-1)}{2} \leq T(n) \leq \frac{n^2+5n-6}{2}$ $(\Theta(n^2))$
interschimbări1	$\frac{n(n-1)}{2}$	0	$\frac{n(n-1)}{2}$	$3\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2} \leq T(n) \leq 2n(n-1)$ $(\Theta(n^2))$
interschimbări2	$n-1$	0	$n(n-1)$	$3\frac{n(n-1)}{2}$	$n-1 \leq T(n) \leq \frac{5n(n-1)}{2}$ $(\Omega(n), O(n^2))$

5 Exerciţii

1. Să se propună un algoritm care construieşte tabloul sortat crescător $y[1..n]$ pornind de la tabloul $x[1..n]$.

2. Se consideră un tabloul ale cărui elemente conţin două informaţii: nume şi nota. Să se ordoneze descrescător după notă, iar pentru aceeaşi notă crescător după nume.

3. Să se analizeze complexitatea algoritmului interschimbări3.

4. Se consideră o matrice cu m linii şi n coloane de elemente reale. Să se reorganizeze matricea prin interschimbări de linii şi coloane astfel încât elementele diagonalei principale să fie ordonate crescător.