9 Sortare în timp liniar

În capitolele precedente au fost prezentaţi câţiva algoritmi de complexitate $O(n \lg n)$. Această margine superioară este atinsă de algoritmii de sortare prin interclasare şi heapsort în cazul cel mai defavorabil; respectiv pentru quicksort corespunde în medie. Mai mult decât atât, pentru fiecare din aceşti algoritmi putem efectiv obţine câte o secvenţă de n numere de intrare care determină execuţia algoritmului în timpul $\Omega(n \lg n)$.

Aceşti algoritmi au o proprietate interesantă: *ordinea dorită este determinată în exclusivitate pe baza comparaţiilor între elementele de intrare*. Astfel de algoritmi de sortare se numesc ***sortări prin comparaţii***. Toţi algoritmii de sortare prezentaţi până acum sunt de acest tip.

În secţiunea 9.1 vom demonstra că oricare sortare prin comparaţii trebuie să facă, în cel mai defavorabil caz, $\Omega(n \lg n)$ comparaţii pentru a sorta o secvenţă de n elemente. Astfel, algoritmul de sortare prin interclasare şi heapsort sunt asimptotic optimale şi nu există nici o sortare prin comparaţii care să fie mai rapidă decât cu un factor constant.

Secţiunile 9.2, 9.3 şi 9.4 studiază trei algoritmi de sortare – sortare prin numărare, ordonare pe baza cifrelor şi ordonare pe grupe – care se execută în timp liniar. Se subînţelege că aceşti algoritmi folosesc, pentru a determina ordinea de sortare, alte operaţii decât comparaţiile. În consecinţă, lor nu li se aplică marginea inferioară $\Omega(n \lg n)$.

9.1. Margini inferioare pentru sortare

Într-o sortare prin comparaţii folosim numai comparaţiile dintre elemente pentru a câştiga informaţii de ordine despre o secvenţă de intrare $\langle a_1, a_2, ..., a_n \rangle$. Respectiv, date fiind elementele a_i şi a_j, facem testele $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$, sau $a_i > a_j$ pentru a determina ordinea lor relativă. Nu putem să inspectăm valorile elementelor sau să obţinem informaţii asupra ordinii lor în nici un alt mod.

În această secţiune vom presupune, fără a pierde din generalitate, că toate elementele de intrare sunt distincte. Dată fiind această presupunere, comparaţiile de tipul $a_i = a_j$ sunt nefolositoare, deci putem să afirmăm că nu sunt făcute comparaţii de acest fel. Studiind comparaţiile $a_i \leq a_j$, $a_i \geq a_j$, $a_i > a_j$ şi $a_i < a_j$ observăm că acestea sunt echivalente, deoarece aduc informaţie identică despre ordinea relativă a lui a_i şi a_j. De aceea vom considera că toate comparaţiile au forma $a_i \leq a_j$.

Modelul arborelui de decizie

Sortările prin comparaţii pot fi viziualizate în mod abstract în termenii ***arborilor de decizie***. Un arbore de decizie reprezintă comparaţiile realizate de un algoritm de sortare când acesta operează asupra unor date de intrare având o mărime dată. Controlul, mişcarea datelor şi toate celelalte aspecte ale algoritmului sunt ignorate. Figura 9.1 vizualizează un arbore de decizie corespunzător algoritmului de sortare prin inserţie din secţiunea 1.1 ce operează cu o secvenţă de intrare având trei elemente.

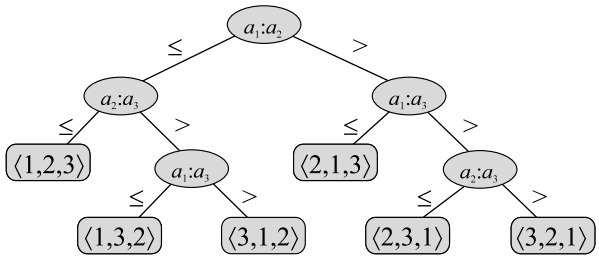

Figura 9.1 Arborele de decizie corespunzător sortării prin inserţie care operează cu trei elemente. Sunt posibile $3! = 6$ permutări între elementele de intrare, deci arborele de decizie trebuie să aibă cel puţin 6 frunze.

Într-un arbore de decizie fiecare nod intern este etichetat prin $a_i : a_j$ pentru i şi j din intervalul $1 \leq i, j \leq n$, unde n este numărul de elemente din secvenţa de intrare. Fiecare frunză este etichetată cu o permutare $\langle \pi(1), \pi(2), ..., \pi(n) \rangle$. (Vezi secţiunea 6.1 pentru noţiuni de bază asupra permutărilor.) Execuţia algoritmului de sortare corespunde trasării unui drum de la rădăcina arborelui de decizie la o frunză. La fiecare nod intern este făcută o comparaţie $a_i \leq a_j$. Subarborele stâng dictează comparaţiile următoare pentru $a_i \leq a_j$, iar subarborele drept dictează comparaţiile următoare pentru $a_i > a_j$. Când ajungem la o frunză, algoritmul de sortare a stabilit ordonarea $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$. Pentru ca algoritmul de sortare să ordoneze adecvat, fiecare dintre cele $n!$ permutări de n elemente trebuie să apară în dreptul unei frunze a arborelui de decizie.

O margine inferioară în cel mai defavorabil caz

Lungimea celui mai lung drum de la rădăcina unui arbore de decizie la oricare dintre frunzele sale reprezintă numărul de comparaţii, în cel mai defavorabil caz, pe care le realizează algoritmul de sortare. În consecinţă, numărul de comparaţii, în cel mai defavorabil caz, pentru o sortare prin comparaţii corespunde înălţimii arborelui de decizie. O margine inferioară a înălţimilor arborilor de decizie este astfel o margine inferioară a timpului de execuţie al oricărui algoritm de sortare prin comparaţii. Următoarea teoremă stabileşte o astfel de margine inferioară.

Teorema 9.1 Orice arbore de decizie care sortează n elemente are înălţimea $\Omega(n \lg n)$.

Demonstraţie. Să considerăm un arbore de decizie având înălţimea h şi care sortează n elemente. Întrucât există $n!$ permutări ale celor n elemente, fiecare permutare reprezentând o ordine distinctă de sortare, arborele trebuie să aibă cel puţin $n!$ frunze. Deoarece un arbore binar având înălţimea h nu are mai mult de 2^h frunze, avem

$$n! \leq 2^h,$$

iar, prin logaritmare, rezultă

$$h \geq \lg(n!),$$

deoarece funcţia lg este monoton crescătoare. Din aproximarea lui Stirling (2.11), avem

$$n! > \left(\frac{n}{e}\right)^n,$$

unde $e = 2.71828...$ este baza logaritmilor naturali; astfel

$$h \geq \lg\left(\frac{n}{e}\right)^n = n\lg n - n\lg e = \Omega(n\lg n).$$

■

Corolarul 9.2 Metoda heapsort şi metoda de sortare prin interclasare sunt metode de sortare prin comparaţii asimptotic optimale.

Demonstraţie. În cazul metodelor de sortare heapsort şi sortarea prin interclasare, marginile superioare $O(n\lg n)$ asupra duratei timpilor de execuţie corespund marginii inferioare $\Omega(n\lg n)$ stabilite de teorema 9.1 pentru cazul cel mai defavorabil. ■

Exerciţii

9.1-1 Care este cea mai mică adâncime posibilă a unei frunze într-un arbore de decizie pentru un algoritm de sortare?

9.1-2 Obţineţi margini asimptotic strânse la $\lg(n!)$ fără a folosi aproximaţia Stirling. În loc de această aproximaţie, evaluaţi suma $\sum_{k=1}^{n} \lg k$ folosind tehnicile prezentate în cadrul secţiunii 3.2.

9.1-3 Arătaţi că nu există nici o sortare prin comparaţii având timpul de execuţie liniar pentru cel puţin jumătate din cele $n!$ intrări de lungime n. Ce se poate spune în cazul unei fracţiuni $1/n$ din datele de intrare de lungime n? Dar despre o fracţiune $1/2^n$?

9.1-4 Profesorul Solomon afirmă că marginea inferioară $\Omega(n\lg n)$ pentru sortarea a n numere nu se aplică în cazul mediului de care dispune, unde după o singură comparaţie $a_i : a_j$, programul se ramifică în trei direcţii, după cum $a_i < a_j, a_i = a_j$ respectiv $a_i > a_j$. Demonstraţi că profesorul nu are dreptate demonstrând că numărul comparaţiilor tridirecţionale cerut de sortarea celor n elemente este tot $\Omega(n\lg n)$.

9.1-5 Demonstraţi că, pentru a interclasa două liste ordonate ce conţin fiecare câte n elemente, sunt necesare $2n - 1$ comparaţii în cazul cel mai defavorabil.

9.1-6 Se consideră o secvenţă de n elemente care trebuie sortate. Datele de intrare sunt formate din n/k subsecvenţe, fiecare conţinând k elemente. Toate elementele dintr-o subsecvenţă dată sunt mai mici decât elementele din subsecvenţa următoare şi mai mari decât elementele din subsecvenţa precedentă. Astfel, pentru sortarea întregii secvenţe de lungime n este suficientă sortarea celor k elemente din fiecare dintre cele n/k subsecvenţe. Găsiţi o margine inferioară $\Omega(n\lg k)$ a numărului de comparaţii necesar pentru rezolvarea acestei variante a problemei de sortare. (*Indicaţie:* combinarea simplă a marginilor inferioare pentru subecvenţele individuale nu este riguroasă.)

9.2. Sortarea prin numărare

Sortarea prin numărare presupune că fiecare dintre cele n elemente ale datelor de intrare este un număr întreg între 1 şi k, pentru un număr întreg k oarecare. Când $k = O(n)$, sortarea se execută în timpul $O(n)$.

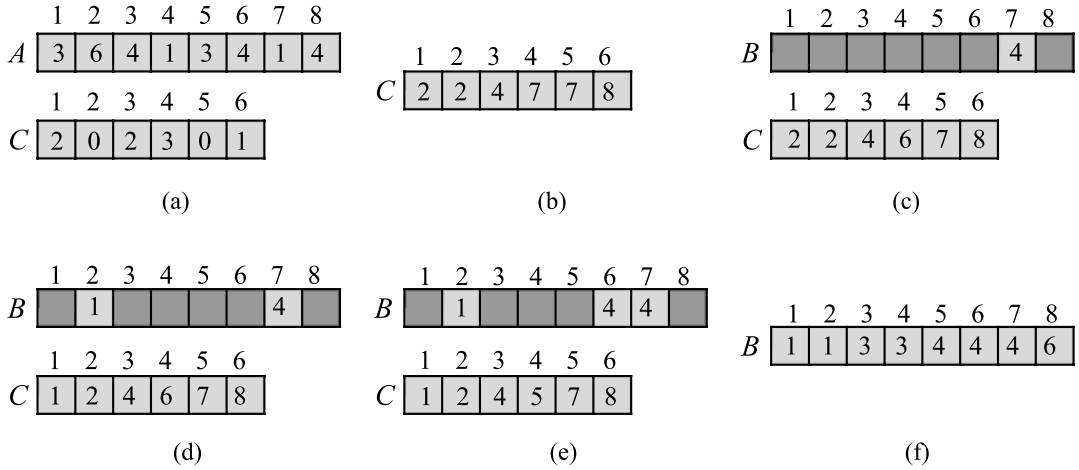

Figura 9.2 Modul de funcţionare al algoritmului SORTARE-PRIN-NUMĂRARE pe un tablou $A\,[1..8]$ de intrare, unde fiecare element din tabloul A este un număr întreg pozitiv nu mai mare decât $k = 6$. **(a)** Tabloul A şi tabloul auxiliar C după liniei 4. **(b)** Tabloul C după executarea liniei 7. **(c)-(e)** Tabloul de ieşire B şi tabloul auxiliar C după unul, două, respectiv trei iteraţii ale buclei din liniile 9–11. Doar elementele haşurate din tabloul B au fost completate. **(f)** Tabloul B sortat, furnizat la ieşire.

Ideea de bază în sortarea prin numărare este de a determina numărul de elemente mai mici decât x, pentru fiecare element x din datele de intrare. Acestă informaţie poate fi utilizată pentru a poziţiona elementul x direct pe poziţia sa din tabloul de ieşire. De exemplu, dacă există 17 elemente mai mici decât x, atunci x va fi pe poziţia 18 în tabloul de ieşire. Această schemă trebuie uşor modificată în situaţia în care există mai multe elemente având aceeaşi valoare, întrucât nu dorim să le aşezăm pe toate în aceeaşi poziţie.

În algoritmul pentru sortarea prin numărare presupunem că datele de intrare formează un tablou $A[1..n]$, şi deci $lungime\,[A] = n$. Sunt necesare alte două tablouri suplimentare, tabloul $B\,[1..n]$, care cuprinde datele de ieşire sortate, şi tabloul $C\,[1..k]$, pentru stocarea temporară în timpul lucrului.

SORTARE-PRIN-NUMĂRARE(A, B, k)

1: **pentru** $i \leftarrow 1, k$ **execută**
2: $\quad C\,[i] \leftarrow 0$
3: **pentru** $j \leftarrow 1, lungime\,[A]$ **execută**
4: $\quad C\,[A\,[j]] \leftarrow C\,[A\,[j]] + 1$
5: ▷ $C[i]$ conţine acum numărul elementelor egale cu i.
6: **pentru** $i \leftarrow 2, k$ **execută**
7: $\quad C\,[i] \leftarrow C\,[i] + C\,[i-1]$
8: ▷ $C[i]$ conţine numărul elementelor mai mici sau egale cu i.
9: **pentru** $j \leftarrow lungime\,[A]\,, 1, -1$ **execută**
10: $\quad B\,[C\,[A\,[j]]] \leftarrow A\,[j]$
11: $\quad C\,[A\,[j]] \leftarrow C\,[A\,[j]] - 1$

Algoritmul de sortare prin numărare este ilustrat în figura 9.2. După iniţializarea din liniile

1–2, în liniile 3–4 se contorizează fiecare element de intrare. Dacă valoarea unui element de intrare este i, se incrementează valoarea lui $C[i]$. Astfel, după liniile 3–4, $C[i]$ păstrează un număr de elemente de intrare egal cu i pentru fiecare număr întreg $i = 1, 2, \ldots, k$. În liniile 6–7 se determină, pentru fiecare $i = 1, 2, \ldots, k$, numărul elementelor de intrare mai mici sau egale decât i; aceasta se realizează prin păstrarea în $C[k]$ a sumei primelor k elemente din tabloul iniţial.

În final, în liniile 9–11, fiecare element $A[j]$ se plasează pe poziţia sa corect determinată din tabloul de ieşire B, astfel încât acesta este ordonat. Dacă toate cele n elemente sunt distincte, la prima execuţie a liniei 9, pentru fiecare $A[j]$, valoarea $C[A[j]]$ este poziţia finală corectă a lui $A[j]$ în tabloul de ieşire, întrucât există $C[A[j]]$ elemente mai mici sau egale cu $A[j]$. Deoarece elementele ar putea să nu fie distincte, decrementăm valoarea $C[A[j]]$ de fiecare dată când plasăm o valoare $A[j]$ în tabloul B; aceasta face ca următorul element de intrare cu o valoare egală cu $A[j]$, dacă există vreunul, să meargă în poziţia imediat dinaintea lui $A[j]$ în tabloul de ieşire.

Cât timp necesită sortarea prin numărare? Bucla **pentru** din liniile 1–2 necesită timpul $O(k)$, bucla **pentru** din liniile 3–4 necesită timpul $O(n)$, bucla **pentru** din liniile 6–7 necesită timpul $O(k)$, iar bucla **pentru** din liniile 9–11 necesită timpul $O(n)$. Deci timpul total este $O(k + n)$. În practică se utilizează sortarea prin numărare când avem $k = O(n)$, caz în care timpul necesar este $O(n)$.

Sortarea prin numărare are un timp mai bun decât marginea inferioară egală cu $\Omega(n \lg n)$ demonstrată în secţiunea 9.1, deoarece nu este o sortare prin comparaţii. De fapt, în cod nu apar comparaţii între elementele de intrare. În schimb, sortarea prin numărare foloseşte valorile elementului ca indice într-un tablou. Marginea inferioară $\Omega(n \lg n)$ nu se aplică atunci când ne îndepărtăm de modelul sortării prin comparaţii.

O proprietate importantă a sortării prin numărare este ***stabilitatea***: numerele cu aceeaşi valoare apar în tabloul de ieşire în aceeaşi ordine în care se găsesc în tabloul de intrare. Adică, legăturile dintre două numere sunt distruse de regula conform căreia oricare număr care apare primul în vectorul de intrare, va apărea primul şi în vectorul de ieşire. Desigur, stabilitatea este importantă numai când datele învecinate sunt deplasate împreună cu elementul în curs de sortare. Vom vedea în secţiunea următoare de ce stabilitatea este importantă.

Exerciţii

9.2-1 Folosind figura 9.2 ca model, ilustraţi modul de operare al algoritmului SORTARE-PRIN-NUMĂRARE pe tabloul $A = \langle 7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3 \rangle$.

9.2-2 Demonstraţi că SORTARE-PRIN-NUMĂRARE este stabil.

9.2-3 Se presupune că bucla **pentru** din linia 9 a procedurii SORTARE-PRIN-NUMĂRARE este rescrisă sub forma:

pentru $j \leftarrow 1$, $lungime[A]$ **execută**

Arătaţi că algoritmul modificat astfel funcţionează corect. Algoritmul modificat este stabil?

9.2-4 Se presupune că ieşirea algoritmului de sortare este un flux de date, de exemplu, o afişare în mod grafic. Modificaţi algoritmul SORTARE-PRIN-NUMĂRARE pentru a obţine rezultatul în ordinea sortată fără a utiliza vectori suplimentari în afara lui A şi C. (*Indicaţie:* legaţi elementele tabloului A având aceleaşi chei în liste. Unde există un spaţiu "liber" pentru păstrarea pointerilor listei înlănţuite?)

9.2-5 Date fiind n numere întregi din intervalul [1,k], preprocesaţi datele de intrare şi apoi răspundeţi într-un timp $O(1)$ la orice interogare privind numărul total de valori întregi dintre cele n, care se situează în intervalul $[a..b]$. Algoritmul propus ar trebui să folosească un timp de preprocesare $O(n+k)$.

9.3. Ordonare pe baza cifrelor

Ordonarea pe baza cifrelor este algoritmul folosit de către maşinile de sortare a cartelelor, care se mai găsesc la ora actuală numai în muzeele de calculatoare. Cartelele sunt organizate în 80 de coloane, şi fiecare coloană poate fi perforată în 12 poziţii. Sortatorul poate fi "programat" mecanic să examineze o coloană dată a fiecărei cartele dintr-un pachet şi să distribuie fiecare cartela într-una dintre cele 12 cutii, în funcţie de poziţia perforată. Un operator poate apoi să strângă cartelele cutie cu cutie, astfel încât cartelele care au prima poziţie perforată să fie deasupra cartelelor care au a doua poziţie perforată ş.a.m.d.

Pentru cifre zecimale sunt folosite numai 10 poziţii în fiecare coloană. (Celelalte două poziţii sunt folosite pentru codificarea caracterelor nenumerice). Un număr având d cifre ar ocupa un câmp format din d coloane. Întrucât sortatorul de cartele poate analiza numai o singură coloană la un moment dat, problema sortării a n cartele în funcţie de un număr având d cifre necesită un algoritm de sortare.

Intuitiv, am putea dori să sortăm numere în funcţie de *cea mai semnificativă* cifră, să sortăm recursiv fiecare dintre cutiile ce se obţin, şi apoi să combinăm pachetele în ordine. Din nefericire, întrucât cartelele din 9 dintre cele 10 cutii trebuie să fie pastrate pentru a putea sorta fiecare dintre cutii, această procedură generează teancuri intermediare de cartele care trebuie urmărite. (Vezi exerciţiul 9.3-5.)

Ordonarea pe baza cifrelor rezolvă problema sortării cartelelor într-un mod care contrazice intuiţia, sortând întâi în funcţie de *cea mai puţin semnificativă* cifră. Cartelele sunt apoi combinate într-un singur pachet, cele din cutia 0 precedând cartelele din cutia 1, iar acestea din urmă precedând pe cele din cutia 2 şi aşa mai departe. Apoi întregul pachet este sortat din nou în funcţie de a doua cifră cea mai puţin semnificativă şi recombinat apoi într-o manieră asemănătoare. Procesul continuă până când cartelele au fost sortate pe baza tuturor celor d cifre. De remarcat că în acel moment cartelele sunt complet sortate în funcţie de numărul având d cifre. Astfel, pentru sortare sunt necesare numai d treceri prin lista de numere. În figura 9.3 este ilustrat modul de operare al algoritmului de ordonare pe baza cifrelor pe un "pachet" de şapte numere de câte trei cifre.

Este esenţial ca sortarea cifrelor în acest algoritm să fie stabilă. Sortarea realizată de către un sortator de cartele este stabilă, dar operatorul trebuie să fie atent să nu schimbe ordinea cartelelelor pe măsură ce acestea ies dintr-o cutie, chiar dacă toate cartelele dintr-o cutie au aceeaşi cifră în coloana aleasă.

Într-un calculator care funcţionează pe bază de acces secvenţial aleator, ordonarea pe baza cifrelor este uneori utilizată pentru a sorta înregistrările de informaţii care sunt indexate cu chei având câmpuri multiple. De exemplu, am putea dori să sortăm date în funcţie de trei parametri: an, lună, zi. Am putea executa un algoritm de sortare cu o funcţie de comparare care, considerând două date calendaristice, compară anii, şi dacă există o legătură, compară lunile, iar dacă apare din nou o legătură, compară zilele. Alternativ, am putea sorta informaţia de trei ori cu o sortare

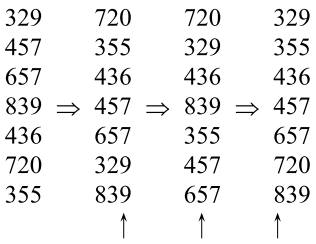

Figura 9.3 Modul de funcţionare al algoritmului de ordonare pe baza cifrelor pe o listă de şapte numere a câte 3 cifre. Prima coloană este intrarea. Celelalte coloane prezintă lista după sortări succesive în funcţie de poziţiile cifrelor în ordinea crescătoare a semnificaţiei. Săgeţile verticale indică poziţia cifrei după care s-a sortat pentru a produce fiecare listă din cea precedentă.

stabilă: prima după zi, următoarea după lună, şi ultima după an.

Pseudocodul pentru algoritmul de ordonare pe baza cifrelor este evident. Următoarea procedură presupune că într-un tablou A având n elemente, fiecare element are d cifre; cifra 1 este cifra cu ordinul cel mai mic, iar cifra d este cifra cu ordinul cel mai mare.

ORDONARE-PE-BAZA-CIFRELOR(A, d)
1: **pentru** $i \leftarrow 1, d$ **execută**
2: folosește o sortare stabilă pentru a sorta tabloul A după cifra i

Corectitudinea algoritmului de ordonare pe baza cifrelor poate fi demonstrată prin inducţie după coloanele care sunt sortate (vezi exerciţiul 9.3-3). Analiza timpului de execuţie depinde de sortarea stabilă folosită ca algoritm intermediar de sortare. Când fiecare cifră este în intervalul $[1, k]$, iar k nu este prea mare, sortarea prin numărare este opţiunea evidentă. Fiecare trecere printr-o mulţime de n numere a câte d cifre se face în timpul $\Theta(n + k)$. Se fac d treceri, astfel încât timpul total necesar pentru algoritmul de ordonare pe baza cifrelor este $\Theta(dn + kd)$. Când d este constant şi $k = O(n)$, algoritmul de ordonare pe baza cifrelor se execută în timp liniar.

Unii informaticieni consideră că numărul biţilor într-un cuvânt calculator este $\Theta(\lg n)$. Pentru exemplificare, să presupunem că $d \lg n$ este numărul de biţi, unde d este o constantă pozitivă. Atunci, dacă fiecare număr care va fi sortat încape într-un cuvânt al calculatorului, îl vom putea trata ca pe un număr având d cifre reprezentat în baza n. De exemplu, să considerăm sortarea a 1 milion de numere având 64 de biţi. Tratând aceste numere ca numere de patru cifre în baza 2^{16}, putem să le sortăm pe baza cifrelor doar prin patru treceri, comparativ cu o sortare clasică prin comparaţii de timp $\Theta(n \lg n)$ care necesită aproximativ $\lg n = 20$ de operaţii pentru fiecare număr sortat. Din păcate, versiunea algoritmului de ordonare pe baza cifrelor care foloseşte sortarea prin numărare ca sortare intermediară stabilă nu sortează pe loc, lucru care se întâmplă în cazul multora din sortările prin comparaţii de timp $\Theta(n \lg n)$. Astfel, dacă se doreşte ca necesarul de memorie să fie mic, atunci este preferabil algoritmul de sortare rapidă.

Exerciţii

9.3-1 Folosind figura 9.3 ca model, ilustraţi modul de fucţionare al algoritmului ORDONARE-PE-BAZA-CIFRELOR pe următoarea listă de cuvinte în limba engleză: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

9.3-2 Care din următorii algoritmi de sortare sunt stabili: sortarea prin inserţie, sortarea prin interclasare, heapsort şi sortarea rapidă? Realizaţi o schemă simplă care să facă stabil orice algoritm de sortare. Cât timp şi spaţiu suplimentar necesită schema realizată?

9.3-3 Demonstraţi prin inducţie că algoritmul de ordonare pe baza cifrelor funcţionează corect. Unde avem nevoie în demonstraţie de presupunerea că sortarea intermediară este stabilă?

9.3-4 Arătaţi cum se sortează n întregi din intervalul $[1, n^2]$ în timp $O(n)$.

9.3-5 $\star$ Determinaţi numărul exact de treceri necesare pentru a sorta numere zecimale având d cifre, în cel mai defavorabil caz, pentru primul algoritm din acest capitol pentru ordonarea cartelelor. De câte teancuri de cartele ar avea nevoie un operator pentru a urmări această sortare?

9.4. Ordonarea pe grupe

Ordonarea pe grupe se execută, în medie, în timp liniar. Ca şi sortarea prin numărare, Ordonarea pe grupe este rapidă pentru că face anumite presupuneri despre datele de intrare. În timp ce sortarea prin numărare presupune că intrarea constă din întregi dintr-un domeniu mic, ordonarea pe grupe presupune că intrarea este generată de un proces aleator care distribuie elementele în mod uniform în intervalul $[0, 1)$. (Vezi secţiunea 6.2 pentru definiţia distribuţiei uniforme.)

Principiul algoritmului de ordonare pe grupe este de a împărţi intervalul $[0, 1)$ în n subintervale egale, numite ***grupe*** (engl. *buckets*) şi apoi să distribuie cele n numere de intrare în aceste grupe. Întrucât datele de intrare sunt uniform distribuite în intervalul $[0, 1)$, nu ne aşteptăm să fie prea multe numere în fiecare grupă. Pentru a obţine rezultatul dorit, sortăm numerele din fiecare grupă, apoi trecem prin fiecare grupă în ordine, listând elementele din fiecare.

Pseudocodul nostru pentru ordonarea pe grupe presupune că datele de intrare formează un tablou A având n elemente şi că fiecare element $A[i]$ satisface relaţia $0 \leq A[i] < 1$. Codul necesită un tablou auxiliar $B[0..n-1]$ de liste înlănţuite (reprezentând grupele) şi presupune că există un mecanism pentru menţinerea acestor liste. (În secţiunea 11.2 se descrie cum se implementează operaţiile de bază pe listele înlănţuite.)

ORDONARE-PE-GRUPE(A)
1: $n \leftarrow lungime[A]$
2: **pentru** $i \leftarrow 1, n$ **execută**
3: inserează $A[i]$ în lista $B\left[\lfloor nA\left[i\right]\rfloor\right]$
4: **pentru** $i \leftarrow 0, n-1$ **execută**
5: sortează lista $B[i]$ folosind sortarea prin inserţie
6: concatenează în ordine listele $B\left[0\right], B\left[1\right], ..., B\left[n-1\right]$

Figura 9.4 ilustrează modul de funcţionare al algoritmului de ordonare pe grupe pe un tablou de intrare cu 10 numere.

Pentru a demonstra că acest algoritm funcţionează corect, se consideră două elemente $A\left[i\right]$ şi $A\left[j\right]$. Dacă aceste elemente sunt distribuie în aceeaşi grupă, ele apar în ordinea relativă adecvată în secvenţa de ieşire, deoarece grupa lor este sortată de sortarea prin inserţie. Să presupunem, totuşi, că ele sunt distribuite în grupe diferite. Fie aceste grupe $B[i']$ şi respectiv $B[j']$, şi să

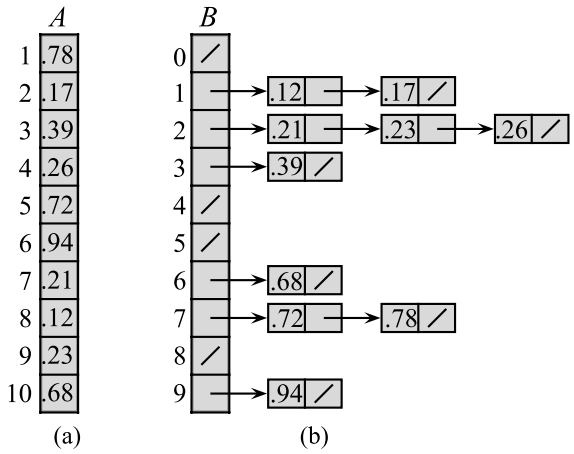

Figura 9.4 Funcţionarea algoritmului ORDONARE-PE-GRUPE. **(a)** Tabloul de intrare $A[1..10]$. **(b)** Tabloul $B[0..9]$ al listelor (reprezentând grupele) sortate după linia a cincea a algoritmului. Grupa i cuprinde valorile din intervalul $[i/10, (i+1)/10)$. Ieşirea sortată constă dintr-o concatenare în ordine a listelor $B[0]$, $B[1], ..., B[9]$.

presupunem, fără a pierde din generalitate, că $i' < j'$. Când listele lui B sunt concatenate în linia 6, elementele grupei $B[i']$ apar înaintea elementelor lui $B[j']$, şi astfel $A[i]$ precede $A[j]$ în secvenţa de ieşire. Deci trebuie să arătăm că $A[i] \leq A[j]$. Presupunând contrariul, avem

$$i' = \lfloor nA[i] \rfloor \geq \lfloor nA[j] \rfloor = j',$$

ceea ce este o contradicţie, întrucât $i' < j'$. Aşadar, ordonarea pe grupe funcţionează corect.

Pentru a analiza timpul de execuţie, să observăm mai întâi că toate liniile, cu excepţia liniei 5, necesită, în cel mai defavorabil caz, timpul $O(n)$. Timpul total pentru a examina în linia 5 toate grupele este $O(n)$ şi, astfel, singura parte interesantă a analizei este timpul consumat de sortările prin inserţie din linia 5.

Pentru a analiza costul sortărilor prin inserţie, fie n_i o variabilă aleatoare desemnând numărul de elemente din grupa $B[i]$. Întrucât sortarea prin inserţie se execută în timp pătratic (vezi secţiunea 1.2), timpul necesar sortării elementelor în grupele $B[i]$ este $E\left[O\left(n_i^2\right)\right] = O\left(E\left[n_i^2\right]\right)$. În consecinţă, timpul mediu total necesar sortării tuturor elementelor în toate grupele va fi

$$\sum_{i=0}^{n-1} O\left(E\left[n_i^2\right]\right) = O\left(\sum_{i=0}^{n-1} E\left[n_i^2\right]\right). \tag{9.1}$$

Pentru a evalua această sumă trebuie să determinăm distribuţia pentru fiecare variabilă aleatoare n_i. Avem n elemente în n grupe. Probabilitatea ca un element dat să intre într-o grupă $B[i]$ este $1/n$, întrucât fiecare grupă este responsabilă pentru $1/n$ elemente din intervalul $[0, 1)$. Astfel, situaţia este asemănătoare cu exemplul aruncatului mingii din secţiunea (6.6.2): avem n mingi (elemente) şi n cutii (reprezentând grupele), şi fiecare minge este aruncată independent, cu probabilitatea $p = 1/n$ de a intra într-o anume grupă. Astfel, probabilitatea ca $n_i = k$ urmează o distribuţie binomială $b(k; n, p)$ care are media $E[n_i] = np = 1$ şi varianta $Var[n_i] = np(1-p) =$

$1 - 1/n$. Pentru fiecare variabilă aleatoare X, ecuaţia (6.30) este echivalentă cu

$$E\left[n_i^2\right] = Var\left[n_i\right] + E^2\left[n_i\right] = 1 - \frac{1}{n} + 1^2 = 2 - \frac{1}{n} = \Theta\left(1\right).$$

Utilizând acest salt în ecuaţia (9.1) concluzionăm că timpul mediu pentru sortarea prin inserţie este $O(n)$. Astfel, întregul algoritm de ordonare pe grupe se execută în timp mediu liniar.

9.4.1. Exerciţii

9.4-1 Utilizând ca model figura 9.4, ilustraţi modul de funcţionare al algoritmului ORDONARE-PE-GRUPE pe tabloul $A = \langle .79, .13, .16, .64, .39, .20, .89, .53, .71, .42 \rangle$.

9.4-2 Care este timpul de execuţie în cazul cel mai defavorabil pentru algoritmul de ordonare pe grupe? Ce modificare simplă a algoritmului păstrează timpul mediu liniar de execuţie şi face ca timpul de execuţie în cazul cel mai defavorabil să fie $O(n \lg n)$?

9.4-3 $\star$ Se dau n puncte în cercul unitate, $p_i = (x_i, y_i)$, astfel încât $0 < x_i^2 + y_i^2 \leq 1$ pentru $i = 1, 2, ..., n$. Presupuneţi că punctele sunt uniform distribuite, adică probabilitatea de a găsi un punct în orice regiune a cercului este proporţională cu aria acelei regiuni. Scrieţi un algoritm în timpul mediu $\Theta\left(n\right)$ care să sorteze cele n puncte în funcţie de distanţele faţă de origine $d_i = \sqrt{x_i^2 + y_i^2}$. (*Indicaţie:* proiectaţi dimensiunile grupelor din ORDONAREA-PE-GRUPE pentru a reflecta distribuţia uniformă a punctelor în cercul unitate.)

9.4-4 $\star$ O ***funcţie de distribuţie a probabilităţii*** $P(x)$ pentru variabila aleatoare X este definită de $P(x) = \Pr\{X \leq x\}$. Consideraţi o listă de n numere cu o funcţie de distribuţie de probabilitate continuă P care este calculabilă în timpul $O(1)$. Arătaţi cum se sortează numerele în timp mediu liniar.

Probleme

9-1 Marginile inferioare în cazul mediu la sortarea prin comparaţii
În această problemă demonstrăm o margine inferioară $\Omega\left(n \lg n\right)$ a timpului mediu de execuţie pentru orice sortare prin comparaţii deterministă sau aleatoare efectuată pe n date de intrare. Începem prin examinarea unei sortări prin comparaţii deterministe A având arborele de decizie T_A. Presupunem că fiecare permutare a intrărilor lui A apare cu o probabilitate constantă.

a. Să presupunem că fiecare frunză a lui T_A este etichetată cu probabilitatea de a fi atinsă în cazul unor date de intrare aleatoare. Demonstraţi că exact $n!$ frunze sunt etichetate cu $1/n!$ şi că restul sunt etichetate cu 0.

b. Fie $D(T)$ lungimea drumului extern a arborelui T; adică, $D(T)$ este suma adâncimilor tuturor frunzelor lui T. Fie T un arbore având $k > 1$ frunze, şi fie LT şi RT subarborii stâng şi drept ai lui T. Arătaţi că $D(T) = D(LT) + D(RT) + k$.

c. Fie $d(k)$ valoarea minimă a lui $D(T)$ pe toţi arborii de decizie T având $k > 1$ frunze. Arătaţi că $d(k) = \min_{1 \leq i \leq k-1} \{d(i) + d(k-i) + k\}$. (*Indicaţie:* Consideraţi un arbore T având k frunze care atinge minimul. Fie i numărul de frunze din LT şi $k - i$ numărul de frunze din RT.)

d. Demonstraţi că pentru o valoare dată a lui $k > 1$ şi i din domeniul $1 \leq i \leq k-1$, funcţia $i \lg i + (k-i) \lg(k-i)$ este minimă pentru $i = k/2$. Arătaţi că $d(k) = \Omega(k \lg k)$.

e. Demonstraţi că $D(T_A) = \Omega(n! \lg(n!))$ pentru T_A, şi concluzionaţi că timpul mediu pentru sortarea a n elemente este $\Omega(n \lg n)$.

Acum să considerăm o sortare *aleatoare* prin comparaţii B. Putem extinde modelul arborelui de decizie pentru a gestiona caracterul aleator prin încorporarea a două feluri de noduri: noduri de comparaţie şi noduri de "randomizare". Un astfel de nod "modelează" o selectare aleatoare de forma ALEATOR$(1, r)$, făcută de către algoritmul B; nodul are r descendenţi, fiecare dintre ei având aceeaşi probabilitate de a fi selectat în timpul execuţiei algoritmului.

f. Arătaţi că, pentru orice sortare aleatoare prin comparaţii B, există o sortare deterministă prin comparaţii A care nu face în medie mai multe comparaţii decât B.

9-2 Sortarea pe loc în timp liniar

a. Să presupunem că avem de sortat un tablou cu n articole şi cheia fiecărui articol are valoarea 0 sau 1. Realizaţi un algoritm simplu, de timp liniar, care să sorteze pe loc cele n articole. Pe lângă memoria necesară reprezentării tabloului, puteţi folosi un spaţiu suplimentar de memorie având dimensiune constantă.

b. Poate sortarea realizată la punctul (a) să fie utilizată pentru o ordonare pe baza cifrelor de timp $O(bn)$ a n articole având chei de b-biţi? Explicaţi cum sau de ce nu.

c. Să presupunem că cele n înregistrări au chei în intervalul $[1, k]$. Arătaţi cum se poate modifica sortarea prin numărare astfel încât înregistrările să poată fi sortate pe loc în timpul $O(n+k)$. Puteţi folosi un spaţiu suplimentar de memorie de ordinul $O(k)$, pe lângă tabloul de intrare. (*Indicaţie:* Cum aţi realiza acest lucru pentru $k = 3$?)

Note bibliografice

Modelul arborelui de decizie pentru studiul sortărilor prin comparaţii a fost introdus de către Ford şi Johnson [72]. Tratatul cuprinzător al lui Knuth asupra sortării [123] acoperă multe variaţii pe problema sortării, inclusiv marginea inferioară teoretică asupra complexităţii sortării prezentate aici. Marginile inferioare pentru sortare folosind generalizările modelului arborelui de decizie au fost studiate exhaustiv de către Ben-Or [23].

Knuth îi atribuie lui H.H. Seward inventarea sortării prin numărare în 1954 şi a ideii de combinare a sortării prin numărare cu ordonare pe baza cifrelor. Ordonarea pe baza cifrelor după cea mai puţin semnificativă cifră pare să fie un algoritm popular, larg utilizat de către operatorii maşinilor mecanice de sortare a cartelelor. Conform lui Knuth, prima referinţă publicată a metodei este un document din 1929 aparţinând lui L.J. Comrie, ce descrie echipamentul pentru cartele perforate. Ordonarea pe grupe a fost utilizată începând cu 1956, când ideea de bază a fost propusă de E.J. Isaac şi R.C. Singleton.