

Cuprins

Prefață	9
1 Despre algoritmi	13
1.1 Limbaj algoritmic	13
1.2 Probleme, algoritmi și programe	27
1.3 Măsurarea performanțelor unui algoritm	30
1.4 Referințe bibliografice	40
2 Tipuri de date de nivel înalt	41
2.1 Liste liniare	41
2.2 Liste liniare ordonate	51
2.3 Liste circulare	55
2.4 Stive	57
2.5 Cozi	62
2.6 Liste generalizate	66
2.7 Arbori binari	71
2.8 Coadă cu priorități	79
2.9 Grafuri	83
2.10 <i>Union-find</i>	101
2.11 Analiza amortizată	104
2.12 Referințe bibliografice	108
3 Derecursivare	109
3.1 Parcurgerea în inordine a arborilor binari	109
3.2 Recursia liniară	113
3.3 Recursia în cascadă	116
3.4 Memorarea într-un tabel a rezultatelor subproblemelor	120
3.5 Referințe bibliografice	122
4 Sortare internă	123
4.1 Sortare bazată pe comparații	124
4.2 Sortare prin distribuire	138
4.3 Sortare prin numărare	142
4.4 Sortare topologică	144

4.5	Referințe bibliografice	146
5	Căutare	147
5.1	Căutare în liste liniare	148
5.2	Modelul de calcul al arborilor de decizie pentru căutare	149
5.3	Arbori binari de căutare	155
6	Tipuri de date avansate pentru căutare	161
6.1	Arbori echilibrați	161
6.2	Dispersia (hashing)	184
6.3	Arbori digitali (tries)	190
6.4	Referințe bibliografice	198
7	Căutare peste șiruri	201
7.1	Căutarea naivă	202
7.2	Algoritmul Knuth-Morris-Pratt	202
7.3	Algoritmul Boyer-Moore	205
7.4	Algoritmul Rabin-Karp	206
7.5	Expresii regulate	208
7.6	Mai multe patternuri și înlocuire	211
7.7	Exerciții	214
7.8	Referințe bibliografice	215
8	Despre paradigmele de proiectare	217
8.1	Aspecte generale	217
8.2	Un exemplu simplu de paradigmă: eliminarea	218
8.3	Alte considerații privind paradigmele de proiectare	222
8.4	Referințe bibliografice	223
9	Algoritmi <i>greedy</i>	225
9.1	Memorarea eficientă a programelor	225
9.2	Prezentare intuitivă a paradigmei	226
9.3	Arbori ponderați pe frontieră optimi	227
9.4	Arbori Huffman	232
9.5	Interclasare optimă pe două căi	234
9.6	Problema rucsacului I (variantea continuă)	236
9.7	Secvențializarea optimă a activităților	239
9.8	Problema instructorului de schi	242
9.9	Arborele parțial de cost minim	244
9.10	Prezentare formală a paradigmei	246
9.11	Exerciții	248
9.12	Referințe bibliografice	251

10	<i>Divide-et-impera</i>	253
10.1	Prezentare generală	253
10.2	Sortare prin interclasare (<i>Merge Sort</i>)	255
10.3	Sortarea rapidă (<i>Quick Sort</i>)	258
10.4	Selecționare	262
10.5	Linia orizontului	264
10.6	Transformata Fourier discretă	270
10.7	Exerciții	274
10.8	Referințe bibliografice	275
11	Programare dinamică	277
11.1	Exemplu: drum optim într-o rețea piramidală de numere	277
11.2	Prezentarea intuitivă a paradigmei	278
11.3	Alocarea resurselor	279
11.4	Drumurile cele mai scurte între oricare două vârfuri ale unui digraf	282
11.5	Problema rucsacului II (varianta discretă)	285
11.6	Subsecvența crescătoare maximală	293
11.7	Distanța între șiruri	295
11.8	Arbori binari de căutare optimali	298
11.9	Prezentarea formală a paradigmei	301
11.10	Exerciții	304
11.11	Referințe bibliografice	308
12	<i>Backtracking și branch-and-bound</i>	309
12.1	Organizarea spațiului soluțiilor candidat	310
12.2	<i>Backtracking</i>	319
12.3	<i>Branch-and-bound</i>	320
12.4	Colorarea grafurilor	325
12.5	Problema celor n regine	327
12.6	Submulțime de sumă dată	330
12.7	Problema rucsacului II (continuare)	332
12.8	Perspico	337
12.9	Exerciții	340
12.10	Referințe bibliografice	342
13	Probleme NP-complete	343
13.1	Algoritmi nedeterminiști	343
13.2	Clasele \mathbb{P} și \mathbb{NP}	344
13.3	Probleme NP-complete	348
13.4	Exerciții	354
13.5	Referințe bibliografice	354
	Bibliografie	355
	Index	359

2.7.3 Implementarea cu tablouri

Nodurile arborelui binar sunt memorate într-un tablou de structuri. Valorile câmpurilor de legătură nu mai sunt adrese de variabile ci indici în tablou. O reprezentare statică a arborelui din figura 2.10 este dată în figura 2.12. Valoarea -1 joacă rolul pointerului NULL. Descrierile operațiilor sunt asemănătoare cu cele de la implementarea dinamică. Prezentăm, ca exemplu, parcurgerea în ordine:

```

procedură inordine(t, i, viziteaza())
    if (i ≠ -1)
        then inordine(t, t.tab[i].stg, viziteaza())
            viziteaza(t, i)
            inordine(t, t.tab[i].drp, viziteaza())
    end

```

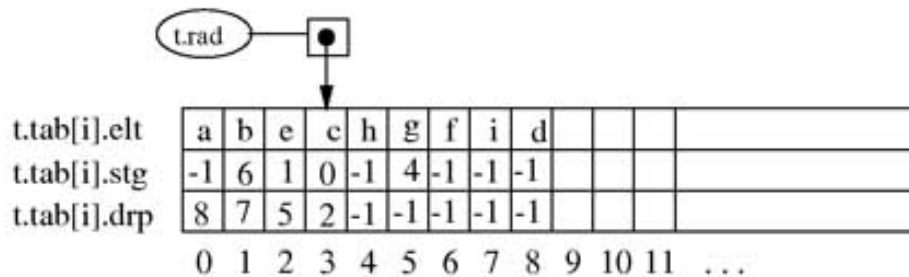


Figura 2.12: Reprezentarea cu tablouri a arborelui binar

2.7.4 Exerciții

Exercițiul 2.7.1. Să se scrie proceduri care implementează operațiile pentru arborii binari reprezentați cu tablouri.

Exercițiul 2.7.2. Asociem vârfurilor unui arbore binar adrese simbolice, ce sunt secvențe de 0 și 1 construite astfel: rădăcina are adresa simbolică 1; dacă vârful v are adresa simbolică ρ , atunci fiul aflat la stânga lui v are adresa simbolică $\rho 0$, iar fiul aflat la dreapta are adresa $\rho 1$.

- Să se scrie un program care, pentru un arbore dat, enumeră adresele simbolice ale tuturor vârfurilor în preordine.
- Să se scrie un program care, având la intrare adresele simbolice ale tuturor vârfurilor, construiește o listă înlănțuită care reprezintă arborele.

Exercițiul 2.7.3. Să se scrie un program care, având la intrare listele liniare ale nodurilor unui arbore binar în preordine, respectiv în inordine, construiește lista înlănțuită care reprezintă arborele. Mai este posibil același lucru dacă se consideră la intrare oricare altă pereche de liste (preordine și postordine sau inordine și postordine)?

Exercițiul 2.7.4. Fie t un arbore binar cu n noduri, v_1, \dots, v_n lista nodurilor în preordine și v_{p_1}, \dots, v_{p_n} lista nodurilor în inordine. Să se arate că permutarea p_1, \dots, p_n se poate obține cu o mașină de la exercițiul 2.4.1 și reciproc, orice permutare obținută cu o mașină de la exercițiul 2.4.1 corespunde unui arbore binar.

Exercițiul 2.7.5. Presupunem că Elt este total ordonat. Peste arborii binari se definește relația \prec astfel: $t_1 \prec t_2$ dacă și numai dacă:

$$\begin{aligned} t_1 &= [] \text{ sau} \\ t_1 &= [a_1](t'_1, t''_1), t_2 = [a_2](t'_2, t''_2) \text{ și} \\ & a_1 < a_2 \text{ sau} \\ & a_1 = a_2 \text{ și } t'_1 \prec t'_2 \text{ sau} \\ & a_1 = a_2 \text{ și } t'_1 = t'_2 \text{ și } t''_1 \prec t''_2. \end{aligned}$$

- Să se arate că, pentru orice doi arbori binari t_1 și t_2 , are loc $t_1 \prec t_2$, $t_1 = t_2$ sau $t_2 \prec t_1$.
- Să se scrie un program care, având la intrare doi arbori t_1 și t_2 , decide dacă $t_1 \prec t_2$ sau $t_2 \prec t_1$ sau $t_1 = t_2$.

Exercițiul 2.7.6. (*Arbori binari însăilați la dreapta.*) Presupunem că structura unui nod cuprinde un câmp suplimentar **tdrp** cu următoarea semnificație:

- $v \rightarrow \text{tdrp} = \text{false}$ semnifică faptul că v nu are fiu dreapta, iar $v \rightarrow \text{drp}$ va conține adresa succesivului lui v din lista inordine. Un astfel de pointer se numește *însăulare*;
- $v \rightarrow \text{tdrp} = \text{true}$ semnifică faptul că v are fiu dreapta.

În plus, se presupune că primul nod este succesivul-inordine al ultimului nod din lista inordine. Un astfel de arbore se numește *însăilat la dreapta*.

- Să se scrie un subprogram **sucInord**(p , t) care determină succesivul-inordine al unui nod arbitrar p în arborele t . Procedura va utiliza $O(1)$ spațiu suplimentar. Care este timpul de execuție a algoritmului descris de subprogram pentru cazul cel mai nefavorabil?
- Este posibil să se scrie o procedură **sucInord** pentru arborii binari neînsăilați la dreapta? Dacă da, să se arate cum, iar dacă nu să se spună de ce.
- Să se descrie o procedură de parcurgere a arborilor binari însăilați la dreapta, utilizând procedura **sucInord**. Să se arate că algoritmul de parcurgere descris necesită $O(n)$ timp (n este numărul de noduri din arbore).

Exercițiul 2.7.7. Se consideră următoarea modificare a structurii de arbore binar:

- zona de legături a fiecărui nod conține două câmpuri suplimentare: **tstg** și **tdrp** cu semnificațiile:
 - $v \rightarrow \text{tstg} = \text{true}$, dacă v are subarbore stânga,
 - $v \rightarrow \text{tstg} = \text{false}$, dacă v nu are subarbore stânga și în acest caz $v \rightarrow \text{stg}$ este adresa predecesivului lui v în inordine,
 - $v \rightarrow \text{tdrp} = \text{true}$, dacă v are subarbore dreapta,
 - $v \rightarrow \text{tdrp} = \text{false}$, dacă v nu are subarbore dreapta și în acest caz $v \rightarrow \text{drp}$ este adresa succesivului lui v în inordine.

Un asemenea arbore se numește *însăilat*.

- Să se scrie o procedură de parcurgere în inordine a arborilor însăilați.

- (ii) Să se scrie o procedură care transformă un arbore binar obișnuit într-un arbore înșăilat.

Exercițiul 2.7.8. Să se scrie o procedură care să numere nodurile de pe frontiera unui arbore binar.

Exercițiul 2.7.9. Se consideră arbori binari care au ca informație în noduri numere întregi. *Lungimea externă ponderată* a arborelui t este

$$\sum \{v \rightarrow \text{inf} * \text{lung}(v) \mid v \text{ este nod pe frontiera lui } t\},$$

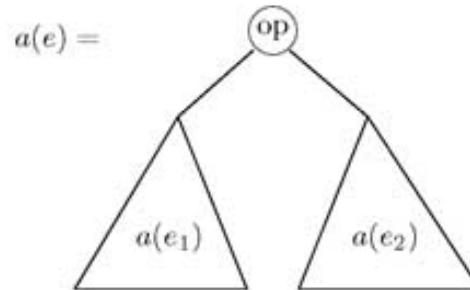
unde $\text{lung}(v)$ este lungimea drumului de la rădăcină la nodul v . Să se scrie un subprogram care determină lungimea externă a unui astfel de arbore.

Exercițiul 2.7.10. (*Reprezentarea expresiilor aritmetice*) Considerăm expresii formate numai din numere întregi și operatorii binari $+$, $-$, $/$, $*$, $\%$. Unei expresii e i se poate asocia un arbore binar $a(e)$ astfel:

- dacă e este un număr întreg atunci $a(e)$ este format dintr-un singur nod $[e]$:

$$a(e) = \textcircled{e}$$

- dacă $e = e_1 \text{ op } e_2$ atunci $a(e)$ este arborele $[\text{op}](a(e_1), a(e_2))$:



Rădăcina lui $a(e)$ are eticheta „op”, subarboarele din stânga rădăcinii este $a(e_1)$ iar subarboarele din dreapta rădăcinii este $a(e_2)$.

Să se scrie un subprogram care are la intrare o listă liniară ce reprezintă o expresie și oferă la ieșire arborele binar asociat expresiei. Arborele asociat este unic? Să se justifice cum rezolvă programul problema unicității.

Exercițiul 2.7.11. Să se arate că notația postfixată se obține prin parcurgerea postordine.

Exercițiul 2.7.12. Să se arate că notația prefixată se obține prin parcurgerea pre-ordine.

Exercițiul 2.7.13. Fie T un arbore binar reprezentat prin structuri dinamice în-lănțuite. Să se scrie un subprogram care să numere frunzele lui T care au frate.

Exercițiul 2.7.14. Fie T un arbore binar reprezentat prin structuri dinamice în-lănțuite. Să se scrie un subprogram care să numere nodurile de pe frontiera lui T care au nivelul egal cu înălțimea arborelui.

2.8 Coadă cu priorități

2.8.1 Tipul de date abstract coadă cu priorități

2.8.1.1 Obiectele

O *coadă cu priorități* este o structură de date în care elementele sunt numite *atom*, iar fiecare atom conține un câmp-cheie care ia valori dintr-o mulțime total ordonată. Valoarea acestui câmp-cheie se numește *prioritate*. Operațiile de citire/eliminare se referă întotdeauna la atomul cu prioritatea cea mai mare. Interpretarea noțiunii de *prioritate* poate diferi de la caz la caz. Există situații când atomii cei mai prioritari sunt cei cu cheile valorilor mai mici și există situații când atomii cei mai prioritari sunt cei cu cheile valorilor mai mari. Noi considerăm aici ultimul caz.

2.8.1.2 Operații

CoadăVidă.

Intrare: – nimic;
Ieșire: – coada cu priorități vidă.

Elimină.

Intrare: – o coadă cu priorități Q ;
Ieșire: – Q din care s-a eliminat atomul cu cheia cea mai mare (dacă există).

Inserează.

Intrare: – o coadă cu priorități Q și un atom a ;
Ieșire: – Q la care s-a adăugat a .

Citește.

Intrare: – o coadă cu priorități Q ;
Ieșire: – atomul cu cheia cea mai mare din coada Q .

2.8.2 Implementarea cu max-heap-uri

2.8.2.1 Descrierea max-heap-urilor

Definiția 2.1. *Un max-heap este un arbore binar complet cu proprietățile:*

1. *informațiile din noduri sunt valori dintr-o mulțime total ordonată numite chei;*
2. *pentru orice nod intern v , cheia memorată în v este mai mare sau egală cu cheia oricăruia dintre fi.*

În continuare vom arăta cum max-heap-urile sunt reprezentate prin tablouri 1-dimensionale.

Definiția 2.2. *Pentru un $n \in \mathbb{N}^*$, arborele binar complet atașat lui n este definit prin*

$$H_n = (\{0, \dots, n-1\}, A)$$

unde A este mulțimea de arce $\{(\lfloor \frac{i-1}{2} \rfloor, i) \mid i = 1, \dots, n-1\}$.

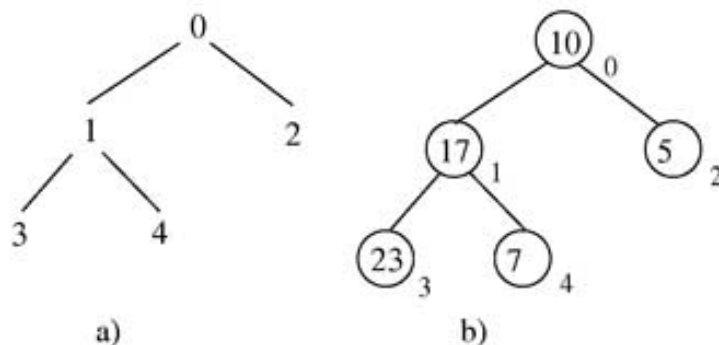


Figura 2.13: Arbore binar complet

Exemplu. Pentru $n = 5$, arborele H_5 este reprezentat în figura 2.13a. Mulțimea arcelor este $A = \{(0, 1), (0, 2), (1, 3), (1, 4)\}$. sfex

Lema 2.1. *Au loc următoarele proprietăți ale arborelui H_n :*

1. Vârful i are succesorii $2i + 1$ și $2i + 2$.
2. Vârful i are ca vârf-tată pe $\lfloor \frac{i-1}{2} \rfloor$, dacă $i \geq 1$.
3. Arborele are $\lfloor \log_2 n \rfloor + 1$ nivele.
4. Pe nivelul k se găsesc vârfurile $2^k - 1, 2^k, \dots, \min\{2^{k+1} - 2, n - 1\}$.¹

Definiția 2.3. Fie $a = (a_0, \dots, a_{n-1})$. Prin $H_n(a)$ notăm arborele obținut din H_n prin etichetarea vârfurilor i cu elementele a_i în mod corespunzător.

Exemplu. Dacă $a = (10, 17, 5, 23, 7)$, atunci arborele $H_5(a)$ este reprezentat în figura 2.13b. sfex

Definiția 2.4. a) Tabloul $(a[i] \mid i = 0, \dots, n - 1)$ are proprietatea MAX-HEAP, dacă $(\forall k)(1 \leq k < n \Rightarrow a[\lfloor \frac{k-1}{2} \rfloor] \geq a[k])$. Notăm această proprietate prin MAX-HEAP(\mathbf{a}).

b) Tabloul \mathbf{a} are proprietatea MAX-HEAP începând cu poziția ℓ dacă $(\forall k)(\ell \leq \lfloor \frac{k-1}{2} \rfloor < k < n \Rightarrow a[\lfloor \frac{k-1}{2} \rfloor] \geq a[k])$. Notăm această proprietate cu MAX-HEAP(\mathbf{a}, ℓ).

Vom da câteva proprietăți ale predicatelor MAX-HEAP(\mathbf{a}) și MAX-HEAP(\mathbf{a}, ℓ).

Lema 2.2. 1. Tabloul $(a[i] \mid i = 0, \dots, n - 1)$ are proprietatea MAX-HEAP (adică MAX-HEAP(\mathbf{a}) = true) dacă și numai dacă pentru orice vârf $a[i]$ din $H_n(a)$, $a[i]$ este mai mare decât sau egal cu orice succesori $a[j]$ ai săi în $H_n(a)$.

2. Pentru orice tablou $(a[i] \mid i = 0, \dots, n - 1)$ are loc MAX-HEAP($\mathbf{a}, \lfloor \frac{n}{2} \rfloor$).

3. Dacă MAX-HEAP(\mathbf{a}, ℓ), atunci pentru orice $j > \ell$ are loc MAX-HEAP(\mathbf{a}, j).

4. Dacă MAX-HEAP(\mathbf{a}) atunci $a[0]$ este elementul maxim din tablou.

5. Dacă MAX-HEAP(\mathbf{a}) atunci $H_n(a)$ este un max-heap.

În figura 2.14 este prezentat un exemplu de max-heap însoțit de tabloul cu reprezentarea sa.

Teorema 2.1. Fie $a = (a_0, \dots, a_{n-1})$. Atunci $H_n(a)$ este max-heap dacă și numai dacă MAX-HEAP(\mathbf{a}).

¹Aici nivelurile sunt numerotate începând cu 0; rădăcina este pe nivelul 0.

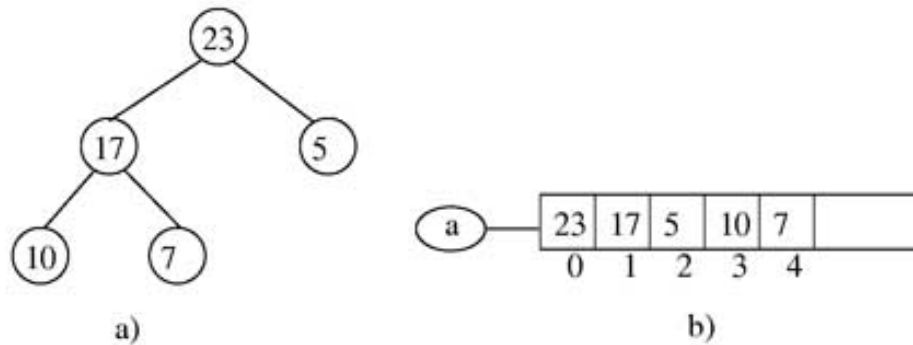


Figura 2.14: Exemplu de max-heap

2.8.2.2 Implementarea operațiilor

CoadăVidă. Este reprezentată de tabloul vid.

Elimină. Atomul cu cea mai mare cheie se află în rădăcina max-heap-ului (prima poziție în tablou). Ștergerea acestuia lasă un loc liber în rădăcină în care copiem atomul de pe ultima poziție din tablou. Dimensiunea max-heap-ului este decremențată cu 1. Refacerea proprietății MAX-HEAP se realizează prin parcurgerea unui drum de la rădăcină spre frontieră conform următorului algoritm:

1. Presupunem că vârful curent este j . Inițial avem $j = 0$.
2. Dacă $2*j + 1 \leq n - 1$, atunci determină vârful cu valoarea maximă dintre fiii lui j ; fie acesta k . Altfel, refacerea proprietății este terminată.
3. Dacă $a[j] \geq a[k]$, atunci refacerea proprietății este terminată.
4. Dacă $a[j] < a[k]$, atunci
 - (a) Interschimbă $a[j]$ cu $a[k]$.
 - (b) Repetă pasul 2 cu vârful curent $j \leftarrow k$.

Descrierea completă a algoritmului de eliminare este:

```

procedure elimina(a, n)
  a[0] ← a[n-1]
  n ← n-1
  j ← 0
  esteHeap ← false
  while ((2*j+1 ≤ n-1) and not esteHeap)
    k ← 2*j+1
    if ((k < n-1) and (a[k] < a[k+1])) then k ← k+1
    if (a[j] < a[k])
      then swap(a[j], a[k])
      else esteHeap ← true
    j ← k
  esteHeap ← true
end

```