Cuprins

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 variabil,e 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 operaţiilor sunt asemănătoare cu cele de la implementarea dinamică. Prezentăm, ca exemplu, parcurgerea inordine:

```
procedure 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
```

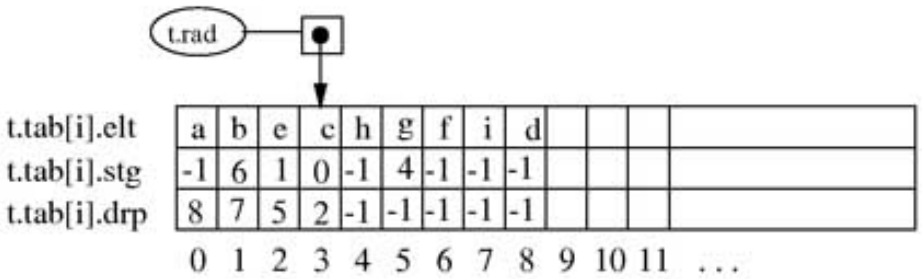


Figura 2.12: Reprezentarea cu tablouri a arborelui binar

2.7.4 Exerciţii

Exerciţiul 2.7.1. Să se scrie proceduri care implementează operaţiile pentru arborii binari reprezentaţi cu tablouri.

Exerciţiul 2.7.2. Asociem vârfurilor unui arbore binar adrese simbolice, ce sunt secvenţe de 0 şi 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 $\rho 1$.

a) Să se scrie un program care, pentru un arbore dat, enumeră adresele simbolice ale tuturor vârfurilor în preordine.

b) Să se scrie un program care, având la intrare adresele simbolice ale tuturor vârfurilor, construieşte o listă înlănţuită care reprezintă arborele.

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

Exerciţiul 2.7.4. Fie t un arbore binar cu n noduri, $v_1, \ldots, v_n$ lista nodurilor în preordine şi $v_{p_1}, \ldots, v_{p_n}$ lista nodurilor în inordine. Să se arate că permutarea $p_1, \ldots, p_n$ se poate obţine cu o maşină de la exerciţiul 2.4.1 şi reciproc, orice permutare obţinută cu o maşină de la exerciţiul 2.4.1 corespunde unui arbore binar.

Exerciţiul 2.7.5. Presupunem că Elt este total ordonat. Peste arborii binari se defineşte relaţia $\prec$ astfel: $t_1 \prec t_2$ dacă şi numai dacă:

$t_1 = [\,]$ sau
$t_1 = [a_1](t_1', t_1''),\ t_2 = [a_2](t_2', t_2'')$ şi
$a_1 < a_2$ sau
$a_1 = a_2$ şi $t_1' \prec t_2'$ sau
$a_1 = a_2$ şi $t_1' = t_2'$ şi $t_1'' \prec t_2''$.

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

Exerciţiul 2.7.6. (*Arbori binari însăilaţi la dreapta.*) Presupunem că structura unui nod cuprinde un câmp suplimentar `tdrp` cu următoarea semnificaţie:

- *v->tdrp =false* semnifică faptul că v nu are fiu dreapta, iar *v->drp* va conţine adresa succesorului lui v din lista inordine. Un astfel de pointer se numeşte *însăilare*;
- *v->tdrp =true* semnifică faptul că v are fiu dreapta.

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

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

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

- zona de legături a fiecărui nod conţine două câmpuri suplimentare: `tstg` şi `tdrp` cu semnificaţiile:
  - *v->tstg =true*, dacă v are subarbore stânga,
  - *v->tstg =false*, dacă v nu are subarbore stânga şi în acest caz *v->stg* este adresa predecesorului lui v în inordine,
  - *v->tdrp =true*, dacă v are subarbore dreapta,
  - *v->tdrp =false*, dacă v nu are subarbore dreapta şi în acest caz *v->drp* este adresa succesorului lui v în inordine.

Un asemenea arbore se numeşte *însăilat*.

(i) Să se scrie o procedură de parcurgere în inordine a arborilor însăilaţi.

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

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

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

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

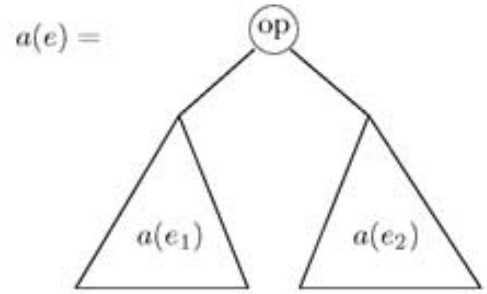
unde $lung(v)$ este lungimea drumului de la rădăcină la nodul v. Să se scrie un subprogram care determină lungimea externă a unui astfel de arbore.

Exerciţiul 2.7.10. (*Reprezentarea expresiilor aritmetice*) Considerăm expresii formate numai din numere întregi şi 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 $e = e_1 \,\mathrm{op}\, e_2$ atunci $a(e)$ este arborele $[\mathrm{op}](a(e_1), a(e_2))$:



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

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

Exerciţiul 2.7.11. Să se arate că notaţia postfixată se obţine prin parcurgerea postordine.

Exerciţiul 2.7.12. Să se arate că notaţia prefixată se obţine prin parcurgerea preordine.

Exerciţiul 2.7.13. Fie T un arbore binar reprezentat prin structuri dinamice înlănţuite. Să se scrie un subprogram care să numere frunzele lui T care au frate.

Exerciţiul 2.7.14. Fie T un arbore binar reprezentat prin structuri dinamice înlănţuite. Să se scrie un subprogram care să numere nodurile de pe frontiera lui T care au nivelul egal cu înălţimea arborelui.

2.8 Coada cu priorităţi

2.8.1 Tipul de date abstract coadă cu priorităţi

2.8.1.1 Obiectele

O *coadă cu priorităţi* este o structură de date în care elementele sunt numite *atomi*, iar fiecare atom conţine un câmp-cheie care ia valori dintr-o mulţime total ordonată. Valoarea acestui câmp-cheie se numeşte *prioritate*. Operaţiile de citire/eliminare se referă întotdeauna la atomul cu prioritatea cea mai mare. Interpretarea noţiunii de *prioritate* poate diferi de la caz la caz. Există situaţii când atomii cei mai prioritari sunt cei cu cheile valorilor mai mici şi există situaţii când atomii cei mai prioritari sunt cei cu cheile valorilor mai mari. Noi considerăm aici ultimul caz.

2.8.1.2 Operaţii

CoadaVidă.

Intrare: – nimic;
Ieşire: – coada cu priorităţi vidă.

Elimină.

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

Inserează.

Intrare: – o coadă cu priorităţi Q şi un atom a;
Ieşire: – Q la care s-a adăugat a.

Citeşte.

Intrare: – o coadă cu priorităţi Q;
Ieşire: – 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

Definiţia 2.1. *Un max-heap este un arbore binar complet cu proprietăţile:*

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

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

Definiţia 2.2. *Pentru un $n \in N^*$,* arborele binar complet *ataşat lui n este definit prin*

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

unde A este mulţimea de arce $\{(\left[\frac{i-1}{2}\right], i) \mid i = 1, \ldots, n-1\}$.

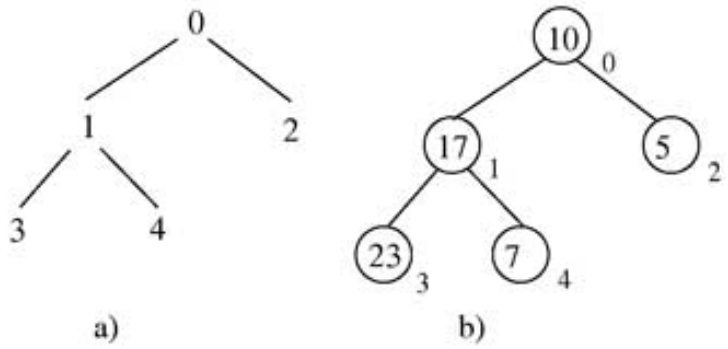


Figura 2.13: Arbore binar complet

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

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

1. *Vârful* i *are succesorii* $2i+1$ *şi* $2i+2$.
2. *Vârful* i *are ca vârf-tată pe* $\left[\frac{i-1}{2}\right]$*, dacă* $i \geq 1$.
3. *Arborele are* $[\log_2 n] + 1$ *nivele.*
4. *Pe nivelul* k *se găsesc vârfurile* $2^k - 1, 2^k, \ldots, \min\{2^{k+1} - 2, n-1\}$.[1]

Definiţia 2.3. *Fie* $a = (a_0, \ldots, a_{n-1})$*. Prin* $H_n(a)$ *notăm arborele obţinut 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

Definiţia 2.4. a) *Tabloul* $(\mathtt{a[i]} \mid i = 0, \ldots, n-1)$ *are proprietatea* MAX-HEAP*, dacă* $(\forall k)(1 \leq k < n \Rightarrow a\left[\frac{k-1}{2}\right] \geq a[k])$*. Notăm această proprietate prin* MAX-HEAP(a).

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

Vom da câteva proprietăţi ale predicatelor MAX-HEAP(a) şi MAX-HEAP(a, ℓ).

Lema 2.2. *1. Tabloul* $(\mathtt{a[i]} \mid i = 0, \ldots, n-1)$ *are proprietatea* MAX-HEAP *(adică* MAX-HEAP(a) *=true) dacă şi numai dacă pentru orice vârf* $a[i]$ *din* $H_n(a)$*,* $a[i]$ *este mai mare decât sau egal cu orice succesor* $a[j]$ *al său în* $H_n(a)$*.*

2. Pentru orice tablou $(\mathtt{a[i]} \mid i = 0, \ldots, n-1)$ *are loc* MAX-HEAP(a, $\left[\frac{n}{2}\right]$).

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

4. Dacă MAX-HEAP(a) *atunci* $a[0]$ *este elementul maxim din tablou.*

5. Dacă MAX-HEAP(a) *atunci* $H_n(a)$ *este un max-heap.*

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

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

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

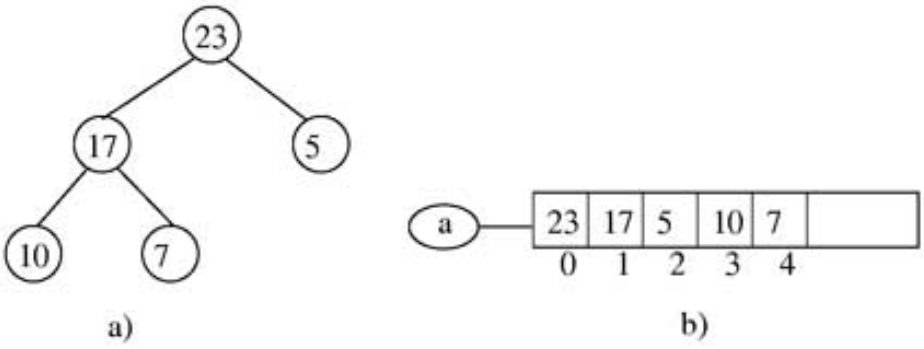


Figura 2.14: Exemplu de max-*heap*

2.8.2.2 Implementarea operaţiilor

CoadaVidă. Este reprezentată de tabloul vid.

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

1. Presupunem că vârful curent este `j`. Iniţial 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 proprietăţii este terminată.
3. Dacă `a[j]` $\geq$ `a[k]`, atunci refacerea proprietăţii 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
```