

Arbori

5 aprilie 2004

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arborii ne permit să structurăm ierarhic o mulțime de elemente

- structura de directoare și fișiere într-un calculator
- arborile genealogic (o persoană, părinții, bunicii, străbunicii, etc.)
- structura ierarhică pt. organizație (director, șefi departamente, etc.)
- circuite logice, expresii aritmetice/logice, baze de date structurate

Un arbore e format din *noduri*, din care unul e *rădăcina*.

Fiecare nod în afară de rădăcină are un nod *părinte* ⇒ ierarhizare

Definiție recursivă:

- un arbore e fie un singur nod n (care reprezintă și rădăcina arborelui)
- sau un nod n , împreună cu arborii T_1, \dots, T_k ai căror rădăcini n_1, \dots, n_k îl au pe n ca părinte

Nodurile n_i sunt *fiii* lui n , iar arborii T_i sunt *subarborii* lui n .

Un nod fără fii se mai numește nod *frunză*.

Uneori se include în definiție și *arborile vid*, fără nici un nod.

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arbori

3

Tipul de date abstract arbore

Operații pe tipul abstract arbore

- `init(arbore)` /* inițializează arborele ca fiind vid (NULL) */
- `părinte(arbore, nod)`: nod /* părintele nodului în arbore sau NULL */
- `fiu_stâng(arbore, nod)`: nod /* returnează primul fiu sau NULL */
- `frate_drept(arbore, nod)`: nod /* return. următorul frate sau NULL */
- `rădăcina(arbore)`: nod /* returnează rădăcina arborelui sau NULL */
- `crează(nod, arbore_1, ..., arbore_k)`: arbore /* creează un arbore cu rădăcina și subarborii specificați */
- `inserează(arbore, nodparinte, nodnou)` /* inserează la părinte */
- `șterge(arbore, nod)` /* șterge un nod dintr-un arbore */

Practic, cel mai des ne referim la arbore prin nodul său rădăcină
⇒ *arbore* și *nod* vor fi același tip

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arbori

4

Traversarea arborilor

În general, se ordinea în care sunt dați fii unui nod *are* importanță.

Nodurile unui arbore pot fi *traversate* (enumerare) în diverse moduri:

- Traversarea în *preordine*
 - se vizitează întâi rădăcina
 - apoi se traversează pe rând în *preordine* toți subarborii
- Traversarea în *postordine*
 - se traversează pe rând în *postordine* toți subarborii
 - apoi se vizitează rădăcina
- Traversarea în *inordine*
 - se traversează întâi în *inordine* primul subarbore (stâng)
 - se vizitează rădăcina
 - se traversează pe rând în *inordine* toți ceilalți subarborii

Obs. Definițiile de mai sus sunt *recursive*.

Cazul de bază: pentru traversarea arborelui *vid* nu se face nimic

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arbori

Traversarea arborilor (cont.)

5

```
typedef ??? node_t; /* vom discuta posibile structuri de date */
#define EMPTY ??? /* o valoare pentru arborele vid */
void preorder(tree_t n) { /* arborele e dat prin rădăcină */
    tree_t c;
    if (n == EMPTY) return;
    visit(n); /* conține ce trebuie făcut pt. fiecare nod */
    for (c = fiu_stang(n); c != EMPTY; c = frate_drept(n, c))
        preorder(c);
}
```

```
void postorder(tree_t n) {
    tree_t c;
    if (n == EMPTY) return;
    for (c = fiu_stang(n); c != EMPTY; c = frate_drept(n, c))
        postorder(c);
    visit(n);
}
```

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arbori

Traversarea arborilor (cont.)

6

```
void inorder(tree_t n) {
    tree_t c;
    if (n == EMPTY) return;
    if ((c = fiu_stang(n)) != EMPTY) inorder(c);
    visit(n);
    for (; c != EMPTY; c = frate_drept(n, c)) inorder(c);
}
```

Observații:

- procedurile de traversare sunt scrise *independent* de reprezentarea arborelui; folosesc doar operațiile (funcțiile) `fiu_stang` și `frate_drept` și valoarea `EMPTY` ⇒ s-a definit într-adevăr un *tip de date abstract*
- *preordine*: dacă trebuie transmisă informație din părinte la fii
- *postordine*: dacă trebuie transmisă informație de la fii la părinte (ex. evaluarea unei expresii; numărarea nodurilor; adâncimea arborelui)
- *inordine*: ex. pentru sortarea cu arbori binari ordonați

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arborii pot fi reprezentați, ca și listele, static, cu tablouri, folosind indici pentru referirea la nodurile fiu.

Cea mai frecventă reprezentare este însă dinamică, cu pointeri.

```
typedef struct n {
    /* aici se pune informația utilă din nod */
    struct n *fiu_stang, *frate_drept;
} node_t; /* node_t e un tip structură sinonim cu struct n */
typedef node_t *tree_t; /* un arbore e un pointer la nod */
```

O altă variantă ar fi să folosim o *listă* separată pentru fii:

```
typedef struct n {          typedef struct l {
    /* informația utilă */   struct n *nod; /* pointer la nod */
    struct l *fii;         struct l *next; /* urm. în listă */
} node_t;                  } list_t;
```

Uneori, se adaugă și un pointer (redundant) părinte, dacă pt. problema dată e necesar accesul rapid și eficient la părintele unui nod dat.

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Arbori binari ordonați (de căutare)

Fiecare nod are o cheie (valoare) a unui tip ordonat (întreg, real, șir)

Pentru fiecare nod *c* din subarborele *n.left* avem *c.key* <= *n.key*

Pentru fiecare nod *c* din subarborele *n.right* avem *c.key* >= *n.key*

Folosiți pentru a păstra o mulțime de elemente, ordonată după chei, într-o structură flexibilă (nu tablou fix), cu căutare/modificare rapidă.

```
typedef int key_t; /* sau alt tip ordonat */
typedef struct n {
    key_t key; /* sau un alt tip ordonat */
    struct n *left, *right;
} node_t;
```

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Ștergerea dintr-un arbore binar ordonat

```
void delete (node_t **n, key_t key) { /* poate modifica *n */
    while (*n) {
        if (key == (*n)->key) { /* șterge. caz simplu: 1 fiu */
            node_t *p = *n;
            if (!(*n)->left) *n = (*n)->right;
            else if (!(*n)->right) *n = (*n)->left;
            else { /* 2 fii. coboară spre dreapta în cel stâng */
                do n = &(*n)->right while (*n);
                *n = p->right; /* inserează subarborele drept la *n */
            }
            free(p); return; /* eliberează memoria pentru nodul șters */
        } else if (key < (*n)->key) n = &(*n)->left;
        else n = &(*n)->right;
    }
}
```

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Caz particular în care orice nod are doi fii: fiul stâng și cel drept

– în general: oricare (sau amândoi) pot lipsi (= arbore vid)

– uneori: arbore binar propriu-zis: fiecare nod are 0 sau 2 fii.

```
typedef struct n {
    /* informația utilă din nod */
    struct n *left, *right;
} node_t;
```

Exemple:

– reprezentarea unei expresii: nodurile intermediare conțin operatori, nodurile frunză conțin valori; calcul prin parcurgere în postordine (operatorii unari vor avea subarborele drept vid)

– arbori de decizie binari, pentru reprezentarea funcțiilor boolene

noduri intermediare: etichetate cu variabile; nodurile terminale: 0 și 1

arborele stâng: valoarea funcției când variabila respectivă e 0

arborele drept: valoarea funcției când variabila respectivă e 1

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Căutarea și inserarea într-un arbore binar ordonat

```
node_t *search(node_t *n, key_t key) {
    if (!n) return NULL; /* arbore nul, cheia nu s-a găsit */
    else if (key == n->key) return n; /* găsit, returnează nodul */
    else if (key < n->key) return search(n->left, key);
    else return search(n->right, key); /* caută într-un subarbore */
}

void insert(node_t **n, key_t key) { /* poate modifica *n */
    while (*n) /* caută un loc gol potrivit */
        /* variantă care acceptă duplicate, inserate la stânga */
        /* fără duplicate: se iese la test de egalitate */
        if (key <= (*n)->key) n = &(*n)->left; /* caută la stânga */
        else n = &(*n)->right; /* caută la dreapta */
    if (!(*n = malloc(sizeof(node_t)))) return; /* alocă nodul */
    (*n)->left = (*n)->right = NULL; /* noul nod e terminal */
    (*n)->key = key;
}
```

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea

Sortarea cu arbori binari ordonați

– se creează un arbore binar ordonat (vid)

– se inserează pe rând elementele de sortat

– se parcurge arborele în *inordine* ⇒ se obțin elementele în ordine

Complexitate:

– toate operațiile (căutare, inserție, sortare) au complexitate liniară în adâncimea *h* a arborelui

– în cazul ideal (și mediu), $h \simeq \log n$ (nr. de noduri)

– în cazul defavorabil: $h = n$ (deja sortat ⇒ arborele devine listă)

– sortarea e $O(n \log n)$ în medie, dar poate fi $O(n^2)$

Soluție: diverse tipuri de arbori binari echilibrați

Utilizarea și programarea calculatoarelor. Curs 16

Marius Minea