4 Arbori binari

4.1 Obiective

Scopul acestei lucrari de laborator este de a dezvolta si de a imbunatati cunostintele legate de stuctura de date de tip arbore binar prin dezvoltarea unei aplicatii care implica operatiile specifice acestei structuri.

In lucrare sunt prezentati arbori binari, arbori binari echilibrati si arbori oarecare. Vom insista asupra arborilor binari.

4.2 Noţiuni teoretice

Un arborele binar, este un arbore in care orice nod are cel mult doi fii (descendenti): fiul stâng şi fiul drept. Cei doi fii se numesc de obicei $left$ (stang) şi $right$ (drept).

Arborele binar este prin urmare o structura recursiva de date: un arbore binar nevid se reduce fie la radacina, fie cuprinde radacina si cel mult doi subarbori binari. Figura 4.1 prezinta un arbore binar.

Un arbore binar se poate implementa foarte usor cu ajutorul pointerilor, fiecare element cuprinzand in afara de informatia propriu-zisa asociata nodului, adresa fiului stang si adresa fiului drept, acestea exprimand legaturile existente intre noduri.

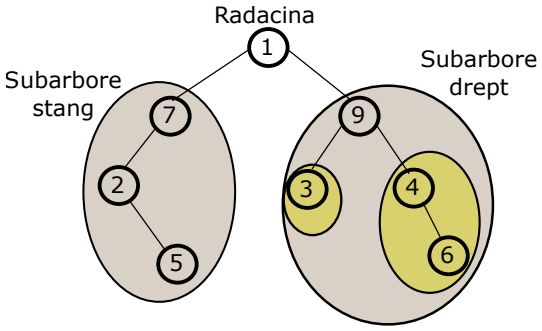


Figure 4.1: Exemplu de arbore binar

Crearea unui arbore binar

Construirea unui arbore binar se face citind în $preordine$ (RadacinaStangaDreapta) de la tastatură sau dintr-un fişier informaţiile din nodurile arborelui. Subarborii vizi trebuie să fie notaţi cu un semn distinctiv, cum ar fi '0'. Pentru arborele din figura 4.1 fisierul de intrare, Arbori.txt are urmatorul continut (obtinut din parcurgerea in preordine cu includerea nodurilor vide):

```
1 7 2 0 5 0 0 0 9 3 0 0 4 0 6 0 0
```

Structura unui nod este:

```
typedef struct node_type
{
  int id; /* node name */
  struct node_type *left, *right;
} NodeT;
```

Construirea unui arbore binar cu informatia citita din fisier se face conform functiei de mai jos. Atentie o valoare de 0 inseamna un arbore vid (NULL). Folosim aceasta valoare in scop didactic. Presupunem ca toate nodurile au valori mai mari decat 0 in arborele creat.

```
NodeT *createBinTreeFromFile(FILE* f)
{
  NodeT *p;
  int c;

  /* se citeste id-ul nodului */
  fscanf(f, "%d", &c);
```

```
  if ( c == 0 )
    return NULL; /* arbore vid, nu facem nimic */
  else /* else inclus pentru claritate */
  { /* construim nodul la care pointeaza p */
    p = ( NodeT *) malloc( sizeof( NodeT ));
    if ( p == NULL )
      fatalError( "Nu mai avem memorie in createBinTree" );
    /* se populeaza nodul */
    p->id = c;
    p->left = createBinTreeFromFile(f);
    p->right = createBinTreeFromFile(f);
  }
  return p;
}
```

Apelul funcţiei de construire a arborelui, $createBinTreeFromFile$, se va face astfel:

```
int main()
{
    NodeT *root;
    FILE *f = fopen("ArboreBinar.txt", "r");
    if (!f){
        printf("Nu se poate deschide fisierul ArboreBinar.txt\n");
        return -1;
    }
    root = createBinTreeFromFile(f);
    fclose(f);
                return 0;
}
```

Ex. 1 — Implementati functia **NodeT *createBinTreeFromFile(FILE* f)** de creare a unui arbore binar folosind datele din fisierul **ArboreBinar.txt**. Scrieti functia **main()** care apeleaza crearea arborelui.

4.3 Operaţii pe arbori binari

Operatiile posibile asupra arborilor binari sunt urmatoarele:

1. Parcugerea sau traversarea arborelui.
2. Adaugarea (inserarea) unui nod in arbore.
3. Stergerea unui nod din arbore.

Parcurgerea arborelui binar consta in parcurgerea pe rand a nodurilor pentru a prelucra informatia pe care ele o contin. Parcurgerea poate fi integrala sau partiala (in cazul in care doar se cauta anumite informatii). Traversarea arborelui presupune vizitarea fiecarui nod o singura data.

Exista trei moduri de traversare a unui arbore binar:

1. Traversare in preordine: se viziteaza intai radacina apoi tot in preordine se viziteaza nodurile sub arborelui stang si apoi acelea ale subarborelui drept.
2. Traversare in inordine: se viziteaza in inordine nodurile subarborelui stnag, apoi radacina si apoi tot in inordine nodurile subarborelui drept.
3. Traversare in postordine: se viziteaza in postordine nodurile subarborelui stang, apoi tot in postordine nodurile subarborelui drept si la sfarsit radacina.

Cele trei modalitati de traversare difera prin momentul in care se viziteaza radacina si anume in cazul:

- preordinii se viziteaza intai radacina apoi subarborele stang si dupa aceea subarborele drept.
- inordinii se viziteaza radacaina dupa vizitarea subarborelui stang
- postordinii: se viziteaza radacaina la sfarsit dupa vizitarea subarborelui stang si a celui drept.

Parcurgerea in preordine este utilizata frecvent. Aceasta parcurgere se poate descompune in trei probleme identice cu problema initiala dar de dimensiuni mai mici:

- parcurgerea radacinii – un singur nod, dimensiunea este 1.
- parcugerea subarborelui stang.
- parcugerea subarborelui drept.

Descompunerile celor doi subarbori continua pana se ajunge la subarbori vizi.

4.3.1 Exemplu de traversare in preordine a arborilor binari

Traversarea unui arbore binar se poate face în cele 3 moduri: preordine, inordine, postordine. In figura de mai jos este dat codul pentru traversarea arborelui in preordine.

Listing 4.1: Traversarea in preordine arborilor binari

```
void preorder( NodeT *p )
// p = nodul curent;
{
    if ( p != NULL )
    {
        printf( "%d ", p->id );
        preorder( p->left);
        preorder( p->right);
    }
}
```

Parcurgerea in preordine a arborelui din figura 4.1 rezulta in: 1, 7, 2, 5, 9,3,4,6.

Ex. 2 — scrieti functiile de parcurgere in inordine void inorder(NodeT *p) si parcurgere in postordine void postorder(NodeT *p) pentru un arbore binar.

4.4 Atribute ale arborilor binari

Un arbore este definit ca o multime de noduri conectate prin muchii, $T = (V, E)$ avand un nod radacina $r \in V$ si o relatie de paternitate intre noduri fapt ce impune o structura ierarhica a nodurilor.

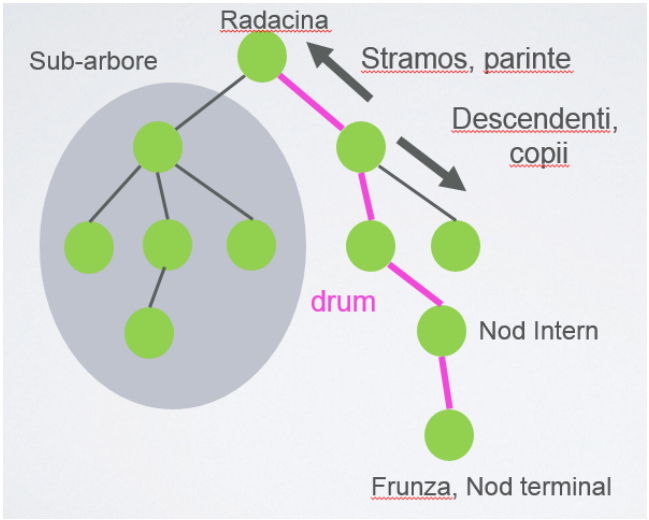


Figure 4.2: Atributele arborilor

Vom defini pe scurt atributele importante care sunt utilizate in operatiile cu arbori.

1. Intr-un arbore nodul care precede un alt nod se numeste parinte.
2. Intr-un arbore nodul care descende din alt nod se numeste copil. Un nod parinte poate avea oricate noduri copii. Intr-un arbore toate nodurile sunt copii, cu exceptia radacinii.
3. Nodurile care au acelasi parinte se numesc frati.

4. Intr-un arbore nodurile care nu au copii se numesc frunze sau noduri terminale sau noduri externe.
5. Nodurile care au cel putin un copil se numesc noduri interne sau noduri non-terminale. Radacina e considerate nod intern.
6. Drumul dintre doua noduri este secventa de noduri si muchii cuprinsa intre cele doua noduri. $(n_1, n_2, \ldots, n_k)$ astfel incat n_i = parintele lui n_{i+1} pentru $1 \leq i \leq k$. Lungimea drumului este egala nu numarul de noduri din drum -1;
7. Gradul unui nod este egal cu numarul de copii ai nodului. Gradul arborelui este maximul dintre gradurile nodurilor din arbore.
8. Inaltimea (height) unui nod v notata cu $inaltime(v)$ este lungimea celui mai lung drum de la v la o frunza. Inaltimea arborelui T este inaltimea radacinii r. $Inaltime(T) = inaltime(r)$. Inaltimea frunzelor este 0.
9. Adancimea unui nod (varf) $v \in V$ notata cu $adancime(v)$ = lungimea drumului de la radacina la v. Adancimea radacinii este 0. Adancimea arborelui este maximul dintre adancimile frunzelor.
10. Nivelul unui varf $v \in V$ este $nivel(v) = 1 + adancime(v)$.
11. Diametrul unui arbore este dat de cel mai lung drum dintre doua frunze.

Ex. 3 — Să se scrie o funcţie **int leaf_node(NodeT *node)** care determină numărul de frunze ale unui arbore binar. Sa se afiseze toate nodurile de tip frunza din arborele binar. Ideea de calcul pentru frunzele unui arbore binar este urmatoarea:

```
leaf_node(nod):
daca nod == NULL returneaza 0;
altfel daca nod->left == NULL si nod->right == NULL returneaza 1;
altfel returneaza leaf_node(nod->left)+leaf_nod(nod->right);
```

Pentru arborele din figura 4.1 se va afisa: 5, 3, 6.

Ex. 4 — Să se scrie o funcţie care determină numărul de noduri interne dintr-un arbore binar. Pentru arborele din figura 4.1 se va afisa: 5 (nodurile sunt 1, 7, 2, 9, 4).

Ex. 5 — Să se scrie o funcţie care determină înălţimea unui arbore binar. Ideea de calcul pentru inaltimea arbore binar este urmatoarea:

```
inaltime(nod):
daca (nod == NULL) returneaza -1;
altfel returneaza 1 + maxim (inaltime (nod->left), inaltime (nod->right))
```

Pentru arborele din figura 4.1 inaltimea este 3.

Ex. 6 — Sa se scrie o functie care cauta un nod in arbore **NodeT * search(NodeT *root, int key).** Functia returneaza nodul cu cheia key, sau NULL daca nu gaseste aceasta cheie in arbore. Sa se determine inaltimea nodului returnat.

4.5 Arbori binari total echilibraţi

Un **arbore binar total echilibrat** este un arbore binar care îndeplineşte următoarea condiţie: numărul nodurilor unui oricare subarbore stâng diferă cu cel mult 1 în plus faţă de numărul nodurilor subarborelui corespunzător drept. Rezultă că frunzele sale se află pe ultimele două niveluri.

Algoritmul de construire a unui arbore binar total echilibrat cu n noduri, este următorul:

1. Se desemnează un nod care este rădăcină.
2. Se consideră numărul de noduri din subarborele stâng $n_{left} = \frac{n}{2}$.
3. Se consideră numărul de noduri din subarborele drept: $n_{right} = n - n_{left} - 1$.
4. Se repetă aceşti paşi în mod recursiv, considerând că numărul de noduri este n_{left}, până când nu mai sunt noduri.
5. Se repetă aceşti paşi în mod recursiv, considerând că numărul de noduri este n_{right}, până când nu mai sunt noduri.

Codul de mai jos contine un exemplu de construire a arborilor binari total echilibrati.

Listing 4.2: Construirea arborilor binari total echilibraţi

```
NodeT *creBalBinTree( int nbOfNodes )
```

```
{
    NodeT *p;
    int nLeft, nRight;

    if ( nbOfNodes <= 0 ) return NULL;
    else
    {
        nLeft = nbOfNodes / 2;
        nRight = nbOfNodes - nLeft - 1;
        p = ( NodeT * ) malloc( sizeof( NodeT ));
        printf( "\nNode identifier = ", nLeft, nRight );
        scanf( "%d", &( p->id ));
        p->left = creBalBinTree( nLeft );
        p->right = creBalBinTree( nRight );
    }
    return p;
}

int main()
{
    NodeT *root = NULL;
    int nbOfNodes = 0;

    printf("\n Numarul de noduri din arbore este:");
    scanf("%d", &nbOfNodes );
    root = creBalBinTree( nbOfNodes );
    return 0;
}
```

Ex. 7 — Implementati codul prezentat anterior pentru construirea arborilor binari total echilibrati.

4.6 Arbori oarecare

Arborele oarecare este un arbore a cărui noduri au mai mult de doi copii (descendenţi).

Un nod are următoarea structură:

```
/* numarul maxim de copii */
#define MAX_CHILD <appropriate value>
typedef struct node_type
{
  char id; /* numele nodului */
  ... /* alte informatii utile */
  struct node_type *children[MAX_CHILD];
} NodeT;
```

Construirea arborelui se realizează astfel:

1. Pentru fiecare nod se citesc in *postordine*, câmpurile: id, alta informatie utilă [1], şi numărul de copii, iar această informaţie se pune pe stivă.
2. Când se citeşte un nod părinte, se scot adresele fiilor din stivă, şi adresele lor sunt trecute în nodul tată, după care adresa nodului tată este pusă în stivă. În final singura adresă în stivă va fi cea a rădăcinii, arborele fiind construit.

Traversarea arborelui pe orizontală (nivel după nivel) se va face astfel:

- Se utilizează o coadă pentru păstrarea adreselor nodurilor ce urmează să fie prelucrate; Iniţial coada este vidă.
- Se introduce în coada adresa rădăcinii.
- Se scoate pe rând din coadă adresa câte unui nod, se prelucrează informaţia din nod, iar apoi se introduc adresele fiilor nodului respectiv.
- Se repetă acest pas până când coada devine vidă.

4.7 Mersul lucrării

Studiati codul prezentat în laborator si utilizati acest cod pentru rezolvarea exercitiilor obligatorii, prezentate pe parcursul lucrarii. Dupa ce terminati implementarea problemelor obligatorii rezolvati problemele propuse in cele ce urmeaza.

[1] dacă e definită

4.8 Probleme

1. Rezolvati problemele obligatorii - marcate cu chenar gri !

2. Să se scrie un program care să interschimbe subarborele drept cu cel stâng pentru un nod dat.

3. Să se scrie o funcţie care determină adancimea maxima a unui arbore binar.

4. Sa se scrie o functie care determina diametrul unui arbore binar.

5. Să se scrie un program care transformă un arbore binar într-o listă dublu înlănţuită.

6. Arborele genealogic al unei persoane se reprezintă astfel: numele persoanei este cheia nodului rădăcină şi pentru fiecare nod cheia descendentului stâng este numele tatălui, iar a descendentului drept este numele mamei. Se citesc două nume de la tastatură. Ce relaţie de rudenie există între cele două persoane? Se presupune că o familie are doar un singur fiu.

7. Să se scrie un program de construire şi traversare a unui arbore oarecare conform indicaţiilor din lucrare.

8. Evaluarea unei expresii: Sa se citeasca dintr-un fisier un sir de caractere care reprezinta o expresie matematica in forma postfix. Operatorii folositi pot fi:

   - operatori binari aditivi: (+,-)
   - operatori binari multiplicativi: (*,/)
   - operatori unari de schimare de semn: (+,-).

   Sa se construiasca arborele expresiei citite din fisier. Fiecare nod contine un operator sau un operand. In cazul operatiilor de schimare de semn operandul lipsa este semnalat cu caracterul #. Operanzii pot fi orice caractere. Construiti arborele asociat expresiei.

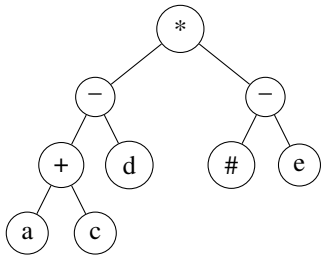
   .
   I/O description. Input: Pentru expresia generica in forma normala infix: a + c - d * (-e), forma postfix este:

   ```
   i:a␣c␣+␣d␣-␣#␣e␣-␣*
   p:
   ```

   unde `i:`= semnalizeaza ca urmeaza date de intrare, `p:`=afiseaza expresia. Iesire: un arbore afisat frumos, Ex:

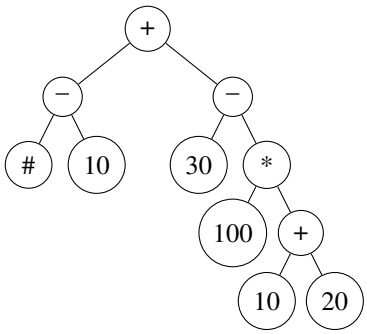
   ```
   ␣␣␣␣␣*
   ␣␣␣␣/␣\
   ␣␣␣-␣␣␣-
   ␣␣/␣\␣␣/␣\
   ␣+␣␣␣d␣#␣␣e
   /␣␣\
   a␣␣c
   ```

   Arborele este:

   Pentru expresia aritmetica in forma normala infix: 10 + 20 * 100 - 30 + (-10), forma postfix este:

   ```
   10␣20␣+␣100␣*␣30␣-␣#␣10␣-␣+
   ```



Arborele este:

9. Data fiind o matrice construiti un arbore binar parcurgand in spirala matricea. Vedeti figura din exemplu in care cu sageata albastra punctata este marcata parcurgerea in spirala. Se porneste de la elementul de pe pozitia 0,0 (coltul stanga sus). Valoarea 0 in matrice indica faptul ca un nod nu mai are descendenti. Arborele care rezulta este afisat mai jos.