Laborator 4: Arbori binari

1 Obiective

Scopul acestei lucrări de laborator este de a dezvolta și de a îmbunătăți cunoștințele legate de structura de date de tip arbore binar prin dezvoltarea unei aplicații care implică operațiile specifice acestei structuri.

În lucrare sunt prezentați arbori binari, arbori binari echilibrați și arbori oarecare. Vom insista asupra arborilor binari.

2 Noțiuni teoretice

Un *arbore binar* este un arbore în care orice nod are cel mult doi fii (descendenți): fiul stâng și fiul drept. Cei doi fii se numesc de obicei $left$ (stâng) și $right$ (drept).

Arborele binar este prin urmare o structură recursivă de date: un arbore binar nevid se reduce fie la rădăcină, fie cuprinde rădăcina și cel mult doi subarbori binari. Figura 1 prezintă un arbore binar.

Un arbore binar se poate implementa foarte ușor cu ajutorul pointerilor, fiecare element cuprinzând, în afară de informația propriu-zisă asociată nodului, adresa fiului stâng și adresa fiului drept, acestea exprimând legăturile existente între noduri.

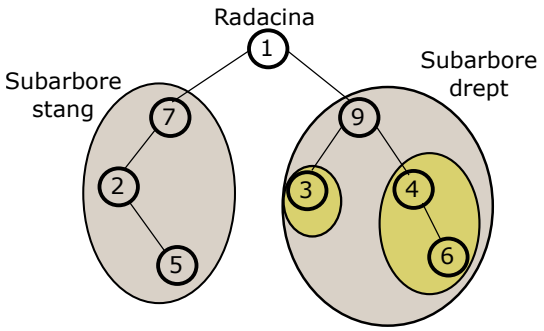


Figura 1: Exemplu de arbore binar

Operațiile posibile asupra arborilor binari sunt următoarele:

1. Crearea (construirea) arborelui
2. Parcurgerea sau traversarea arborelui
3. Căutarea unei informații în arbore
4. Aflarea anumitor proprietăți (atribute) ale lui

2.1 Crearea unui arbore binar

Construirea unui arbore binar se face citind în *preordine* (RădăcinăStângaDreapta) de la tastatură sau dintr-un fișier, informațiile din nodurile arborelui. Subarborii vizi trebuie să fie notați cu un semn distinctiv, cum ar fi '0'. Pentru arborele din figura 1 fișierul de intrare, Arbori.txt are următorul conținut (obținut din parcurgerea în 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 informația citită din fișier se face conform funcției de mai jos. Atenție, o valoare de 0 înseamnă un arbore vid (NULL). Folosim această valoare în scop didactic. Presupunem că toate nodurile au valori mai mari decât 0 în 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 ) {
      puts( "Nu mai avem memorie in createBinTree" );
      exit(1);
    }
    /* se populeaza nodul */
    p->id = c;
    p->left = createBinTreeFromFile(f);
    p->right = createBinTreeFromFile(f);
  }
  return p;
}
```

Apelul funcției 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 — Implementați funcția **NodeT *createBinTreeFromFile(FILE* f)** de creare a unui arbore binar folosind datele din fișierul **ArboreBinar.txt**. Scrieți funcția **main()** care apelează crearea arborelui.

2.2 Parcurgerea arborilor binari

Parcurgerea arborelui binar constă în parcurgerea pe rând a nodurilor pentru a prelucra informația pe care o conțin. Parcurgerea poate fi integrală sau parțială (în cazul în care doar se caută anumite informații). Traversarea arborelui presupune vizitarea fiecărui nod o singură dată.

Există trei moduri de traversare a unui arbore binar:

1. Traversare în preordine: se vizitează întâi rădăcina, apoi tot în preordine se vizitează nodurile sub-arborelui stâng și apoi cele ale subarborelui drept.
2. Traversare în inordine: se vizitează în inordine nodurile subarborelui stâng, apoi rădăcina și apoi tot în inordine nodurile subarborelui drept.
3. Traversare în postordine: se vizitează în postordine nodurile subarborelui stâng, apoi tot în postordine nodurile subarborelui drept și la sfârșit rădăcina.

Cele trei modalități de traversare diferă prin momentul în care se vizitează rădăcina, și anume:

- **preordine** - se vizitează rădăcina *înainte* de vizitarea celor doi subarbori.
- **inordine** - se vizitează rădăcina *între* vizitarea subarborelui stâng și vizitarea subarborelui drept
- **postordine** - se vizitează rădăcina *după* vizitarea celor doi subarbori.

Parcurgerea în preordine este utilizată frecvent. Această parcurgere se poate descompune în trei probleme identice cu problema inițială, dar de dimensiuni mai mici:

- parcurgerea rădăcinii – un singur nod, dimensiunea este 1.
- parcurgerea subarborelui stâng.
- parcurgerea subarborelui drept.

Descompunerile celor doi subarbori continuă până se ajunge la subarbori vizi.

Exemplu de traversare în preordine a arborilor binari

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

Exemplu de cod 1: Traversarea în preordine a arborilor binari

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

Parcurgerea în preordine a arborelui din figura 1 rezultă în: 1, 7, 2, 5, 9, 3, 4, 6.

Ex. 2 — Scrieți cele 3 funcții de parcurgere pentru un arbore binar: în preordine **void preorder(NodeT *p)**, în inordine **void inorder(NodeT *p)** și în postordine **void postorder(NodeT *p)**.

2.3 Atribute ale arborilor binari

Un arbore este definit ca o mulțime de noduri conectate prin muchii, $T = (V, E)$ având un nod rădăcină $r \in V$ și o relație de paternitate între noduri, fapt ce impune o structură ierarhică a nodurilor.

Vom defini pe scurt atributele importante care sunt utilizate în operațiile cu arbori.

1. Într-un arbore, nodul care precede un alt nod se numește părinte.

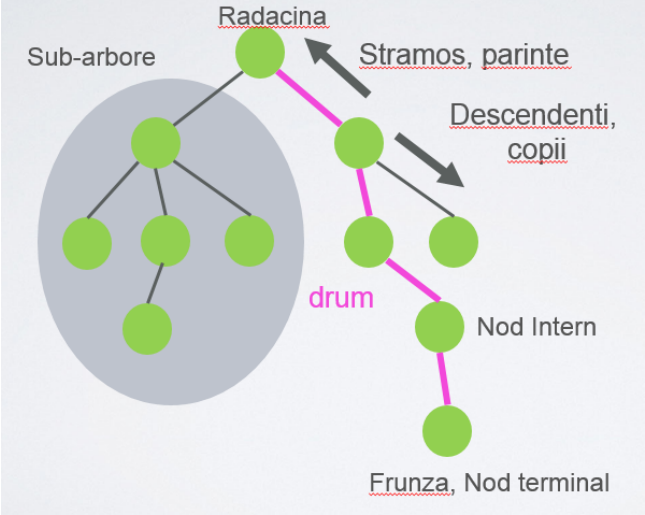


Figura 2: Atributele arborilor

2. Într-un arbore, nodul care descende din alt nod se numește copil. Un nod părinte poate avea oricâte noduri copii. Într-un arbore toate nodurile sunt copii, cu excepția rădăcinii.
3. Nodurile care au același părinte se numesc frați.
4. Într-un arbore nodurile care nu au copii se numesc frunze sau noduri terminale sau noduri externe.
5. Nodurile care au cel puțin un copil se numesc noduri interne sau noduri non-terminale. Rădăcina e considerată nod intern.
6. Drumul dintre două noduri este secvența de noduri și muchii cuprinsă între cele două noduri: $(n_1, n_2, \ldots, n_k)$ astfel încât n_i = părintele lui n_{i+1} pentru $1 \leq i \leq k$. Lungimea drumului este egală cu numărul de muchii ale drumului;
7. Gradul unui nod este egal cu numărul de copii ai nodului. Gradul arborelui este maximul dintre gradele nodurilor din arbore.
8. Înălțimea (height) unui nod v, notată cu $inaltime(v)$ este lungimea celui mai lung drum de la v la o frunză.
   Înălțimea arborelui T este înălțimea rădăcinii r, $Inaltime(T) = inaltime(r)$. Înălțimea frunzelor este 0.
9. Adâncimea unui nod (vârf) $v \in V$, notată cu $adancime(v)$ = lungimea drumului de la rădăcină la v. Adâncimea rădăcinii este 0. Adâncimea arborelui este maximul dintre adâncimile frunzelor.
10. Nivelul unui vârf $v \in V$ este $nivel(v) = 1 + adancime(v)$.
11. Diametrul unui arbore este dat de cel mai lung drum dintre două frunze.

Ex. 3 — Să se scrie o funcție **int leaf_node(NodeT *node)** care determină numărul de frunze ale unui arbore binar. Să se afișeze toate nodurile de tip frunză din arborele binar. Ideea de calcul pentru numărul de frunze ale unui arbore binar este următoarea:

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

Pentru arborele din figura 1 se va afișa: 5, 3, 6.

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

Ex. 5 — Să se scrie o funcție care determină înălțimea unui arbore binar. Ideea de calcul pentru înălțimea arborelui binar este următoarea:

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

Pentru arborele din figura 1 înălțimea este 3.

Ex. 6 — Să se scrie o funcție care caută un nod în arbore **NodeT * search(NodeT *root, int key).** Funcția returnează nodul cu cheia key, sau NULL dacă nu găsește această cheie în arbore. Să se determine înălțimea nodului returnat.

3 Arbori binari total echilibrați

Un **arbore binar total echilibrat** este un arbore binar care îndeplinește următoarea condiție: numărul nodurilor oricărui subarbore stâng diferă cu cel mult 1 faţă de numărul nodurilor subarborelui drept corespunzător. 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 acești pași în mod recursiv, considerând că numărul de noduri este n_{left}, până când nu mai sunt noduri.
5. Se repetă acești pași în mod recursiv, considerând că numărul de noduri este n_{right}, până când nu mai sunt noduri.

Codul de mai jos conține un exemplu de construire a arborilor binari total echilibrați.

Exemplu de cod 2: Construirea arborilor binari total echilibraţi

```
NodeT *createBalBinTree( 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 = " );
        scanf( "%d", &( p->id ));
        p->left = createBalBinTree( nLeft );
        p->right = createBalBinTree( 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 — Implementați codul prezentat anterior pentru construirea arborilor binari total echilibrați. Afișați arborele rezultat utilizând cele 3 parcurgeri.

4 Arbori oarecare

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

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 în *postordine*, câmpurile: id, altă informație utilă [1], și numărul de copii, iar această informație se pune pe stivă.
2. Când se citește un nod părinte, se scot adresele fiilor din stivă și 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; inițial coada este vidă.
- Se introduce în coadă adresa rădăcinii.
- Se scoate pe rând din coadă adresa câte unui nod, se prelucrează informația din nod, iar apoi se introduc adresele fiilor nodului respectiv.
- Se repetă acest pas până când coada devine vidă.

[1] dacă e definită

5 Mersul lucrării

Studiați codul prezentat în laborator și utilizați acest cod pentru rezolvarea exercițiilor obligatorii, prezentate pe parcursul lucrării. După ce terminați implementarea problemelor obligatorii rezolvați problemele propuse în cele ce urmează.

6 Probleme

1. Să se scrie un program care să interschimbe subarborele drept cu cel stâng pentru un nod dat.
2. Să se scrie o funcție care determină adâncimea maxima a unui arbore binar.
3. Sa se scrie o funcție care determina diametrul unui arbore binar.
4. Să se scrie un program care transformă un arbore binar într-o listă dublu înlănțuită.
5. Arborele genealogic al unei persoane se reprezintă astfel: numele persoanei este cheia nodului rădăcină și pentru fiecare nod cheia descendentului stâng este numele tatălui, iar a descendentului drept este numele mamei. Se citesc două nume de la tastatură. Ce relație de rudenie există între cele două persoane? Se presupune că o familie are doar un singur fiu.
6. Să se scrie un program de construire și traversare a unui arbore oarecare conform indicațiilor din lucrare.
7. Evaluarea unei expresii: Sa se citească dintr-un fișier un sir de caractere care reprezinta o expresie matematica în forma postfix. Operatorii folosiți pot fi:
   - operatori binari aditivi: (+,-)
   - operatori binari multiplicativi: (*,/)
   - operatori unari de schimbare de semn: (+,-).

   Sa se construiască arborele expresiei citite din fișier. Fiecare nod conține un operator sau un operand. În cazul operațiilor de schimbare de semn operandul lipsa este semnalat cu caracterul #. Operanzii pot fi orice caractere. Construiți arborele asociat expresiei.

   .

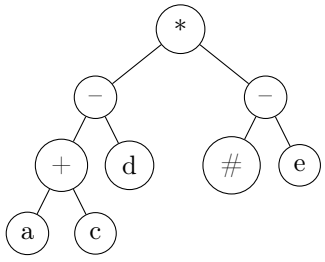
   I/O description. Input: Pentru expresia generica în forma normala infix: a + c - d * (-e), forma postfix este:

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

   unde `i:`= semnalizează că urmează date de intrare, `p:`=afișează expresia. Ieșire: un arbore afișat frumos, Ex:

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

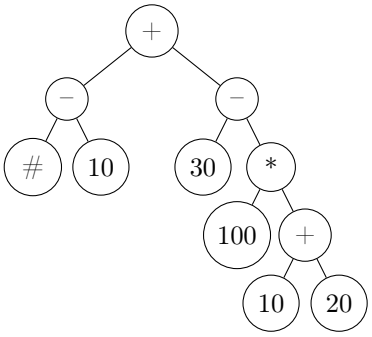
   Arborele este:



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

```
10␣20␣+␣100␣*␣30␣-␣#␣10␣-␣+
```

Arborele este:



8. Data fiind o matrice, construiți un arbore binar parcurgând în spirală matricea. Vedeți figura din exemplu în care cu săgeata albastră punctată este marcată parcurgerea în spirala. Se pornește de la elementul de pe poziția 0,0 (colțul stânga sus). Valoarea 0 în matrice indică faptul că un nod nu mai are descendenți. Arborele care rezultă este afișat mai jos.