1 Arbori binari

1.1 Obiective

În lucrare sunt prezentate operaţiile de bază asupra arborilor binari, arbori binari total echilibraţi şi arborilor oarecare.

1.2 Noţiuni teoretice

Arborele binar, foarte des întâlnit în aplicaţii, este arborele în care orice nod are cel mult doi fii (descendenţi): fiul stâng şi fiul drept. Cei doi fii se numesc de obicei $left$ şi $right$.

1.3 Operaţii pe arbori binari

Crearea unui arbore binar

Construirea unui arbore binar se face citind în $preordine$ de la intrarea standard (tastatură sau fişier) informaţiile din nodurile arborelui. Subarborii vizi trebuie să fie notaţi cu un semn distinctiv, cum ar fi '*'. De exemplu pentru arborele din figura 1.1, notând identificatorul pentru arborele vid cu * , introducerea identificatorilor nodurilor se va face astfel:

ABD*G***CE**F*H**

Structura unui nod este:

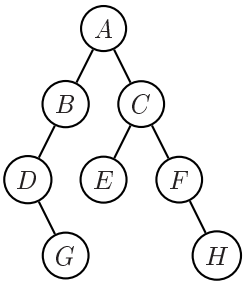

Figure 1.1: Arbore binar.

```
typedef struct node_type
{
  char id; /* node name */
  label ; /* appropriate type for label */
  struct node_type *left, *right;
} NodeT;
```

Construirea unui arbore binar se face conform funcţiei de construire, având următoarea structură:

```
NodeT *createBinTree()
{
  NodeT *p;
  char c;

  /* se citeste id-ul nodului */
  scanf("%c", &c);
  if ( c == '*' )
    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 = createBinTree();
    p->right = createBinTree();
  }
  return p;
}
```

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

```
root = createBinTree();
```

Metode de traversare a arborilor binari

Traversarea unui arbore binar se poate face în cele 3 moduri: preordine, inordine, postordine.

În programul următor 1.1 sunt implementate operaţiile de construcţie şi traversare a unui arbore binar. Nodul conţine numai identificatorul său. Afişarea nodurilor vizitate se face cu indentare.

Listing 1.1: Constructia si traversarea arborilor binari

```
#include <stdio.h>
```

```c
#include <stdlib.h>

typedef struct node_type
{
  char id; /* numele nodului */
  ... /* alte informatii  utile */
  struct node_type *left, *right;
} NodeT;

void fatalError( const char *msg )
{
 printf( msg );
 printf( "\n" );
 exit ( 1 );
}

void preorder( NodeT *p, int level )
/*
 * p = current node;
 * level = used for nice printing
*/
{
  int i;

  if ( p != NULL )
  {
    for ( i = 0; i <= level; i++ )
      printf( " " ); /* pentru afisare formatata */
    printf( "%2.2d\n", p->id );
    preorder( p->left, level + 1 );
    preorder( p->right, level + 1 );
  }
}
void inorder( NodeT *p, int level )
{
  int i;

  if ( p != NULL )
  {
    inorder( p->left, level + 1 );
    for ( i = 0; i <= level; i++ )
      printf( " " ); /* for nice listing */
    printf( "%2.2d\n", p->id );
    inorder( p->right, level + 1 );
  }
}
void postorder( NodeT *p, int level )
{
  int i;

  if ( p != NULL )
  {
    postorder( p->left, level + 1 );
    postorder( p->right, level + 1 );
    for ( i = 0; i <= level; i++ )
      printf( " " ); /* for nice listing */
    printf( "%2.2d\n", p->id );
  }
}
NodeT *createBinTree( int branch, NodeT *parent )
{
  NodeT *p;
  int id;

  /* read node id */
  if ( branch == 0 )
     printf( "Root identifier [0 to end] =" );
  else
  if ( branch == 1 )
     printf( "Left child of %d [0 to end] =",
              parent->id );
  else
     printf( "Right child of %d [0 to end] =",
              parent->id );
```

```
  scanf("%d", &id);
  if ( id == 0 )
    return NULL;
  else
  { /* construim nodul la care pointeaza p */
    p = ( NodeT *)malloc(sizeof( NodeT ));
    if ( p == NULL )
      fatalError( "Nu mai avem memorie in createBinTree" );
    /* fill in node */
    p->id = id;
    p->left = createBinTree( 1, p );
    p->right = createBinTree( 2, p );
  }
  return p;
}
int main()
{
  NodeT *root;

  root = createBinTree( 0, NULL );
  while ('\n' != getc(stdin));
  printf( "\nParcurgere in preordine\n" );
  preorder( root, 0 );
  printf( "Apasati enter ca sa continuati." );
  while ('\n' != getc(stdin));
  printf( "\nParcurgere in inorder\n" );
  inorder( root, 0 );
  printf( "Apasati enter ca sa continuati" );
  while ('\n' != getc(stdin));
  printf( "\nParcurgere in postordine\n" );
  postorder( root, 0 );
  printf( "Apasati enter ca sa continuati" );
  while ('\n' != getc(stdin));
  return 0;
}
```

1.4 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.

Listing 1.2: Construirea şi traversarea arborilor binari total echilibraţi

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node_type
{
  char id; /* numele nodului */
  ... /* alte informatii utile */
  struct node_type *left, *right;
} NodeT;

void fatalError( const char *msg )
{
 printf( msg );
 printf( "\n" );
 exit ( 1 );
}

void preorder( NodeT *p, int level )
/*
```

```
 * p = current node;
 * level = folosit pentru afisare frumoasa
*/
{
    int i;

    if ( p != NULL )
    {
        for ( i = 0; i <= level; i++ )
            printf( " " ); /* pentru afisare frumoasa */
        printf( "%2.2d\n", p->id );
        preorder( p->left, level + 1 );
        preorder( p->right, level + 1 );
    }
}
void inorder( NodeT *p, int level )
{
    int i;

    if ( p != NULL )
    {
        inorder( p->left, level + 1 );
        for ( i = 0; i <= level; i++ )
            printf( " " ); /*  pentru afisare frumoasa */
        printf( "%2.2d\n", p->id );
        inorder( p->right, level + 1 );
    }
}
void postorder( NodeT *p, int level )
{
    int i;

    if ( p != NULL )
    {
        postorder( p->left, level + 1 );
        postorder( p->right, level + 1 );
        for ( i = 0; i <= level; i++ )
            printf( " " ); /*  pentru afisare frumoasa */
        printf( "%2.2d\n", p->id );
    }
}
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 );
    while ('\n' != getc(stdin));
    printf( "\nParcurgere in preordine\n" );
    preorder( root, 0 );
    printf( "Apasati Enter pentru a continua." );
    while ('\n' != getc(stdin));
    printf( "\nParcurgere in inordine.\n" );
    inorder( root, 0 );
```

```
    printf( "Apasati Enter pentru a continua." );
    while ('\n' != getc(stdin));
    printf( "\nParcurgere in postordine.\n" );
    postorder( root, 0 );
    printf( "Apasati Enter pentru a continua." );
    while ('\n' != getc(stdin));
    return 0;
}
```

1.5 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.

1.6 Mersul lucrării

1.6.1 Probleme Obligatorii

1.6.1. Să se implementeze şi să se testeze codul dat în laborator.
1.6.2. Să se scrie un program care să interschimbe subarborele drept cu cel stâng pentru un nod dat.
1.6.3. Să se scrie o funcţie care determină înălţimea unui arbore binar.
1.6.4. Să se scrie o funcţie care determină numărul de frunze ale unui arbore binar.

1.6.2 Probleme Opţionale

1.6.5. Să se scrie un program care transformă un arbore binar într-o listă dublu înlănţuită.
1.6.6. 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.
1.6.7. Să se scrie un program de construire şi traversare a unui arbore oarecare conform indicaţiilor din lucrare.

[1] dacă e definită