Arbori binari alocaţi dinamic

Mirel Coşulschi
mirelc@central.ucv.ro

Mihai Gabroveanu
mihaiug@central.ucv.ro

July, 2023

1 Probleme

1. Considerăm un arbore binar cu n noduri în care fiecare nod este numerotat de la 1 la n şi conţine o valoare număr natural. Să se afişeze valorile din arbore în urma parcurgerii în postordine *(stâng, drept, rădăcină)*.

   Date de intrare

   Fişierul de intrare `postordine.in` conţine pe prima linie numărul n, $1 \leq n \leq 1.000$. Fiecare dintre următoarele n linii conţine câte 3 numere `X st dr`; linia $k+1$ din fişier conţine informaţii referitoare la nodul numerotat cu k: X reprezintă valoarea păstrată în nod (cheia), $1 \leq X \leq 1.000.000$, st reprezintă numărul de ordine al descendentului stâng sau 0 dacă nodul k nu are descendent stâng, iar dr reprezintă numărul de ordine al descendentului drept sau 0 dacă nodul k nu are descendent drept.

   Date de ieşire

   Fişierul de ieşire `postordine.out` va conţine pe prima linie n numere, separate prin exact un spaţiu, reprezentând valorile din nodurile arborelui, obţinute în urma parcurgerii în postordine.

   Exemplu

postordine.in
6 2 3 5 6 0 6 1 0 0 7 1 2 4 0 0 10 0 0
postordine.out
1 4 2 10 6 7

   (Postordine, `https://www.pbinfo.ro/probleme/672/postordine`)

   Rezolvare:
   Vom prezenta o modalitate de alocare dinamică a nodurilor unui arbore binar. Reprezentarea arborelui se va face cu ajutorul unei structuri de date definită de programator denumită `NODE`. Pentru a determina nodul rădăcină se defineşte un vector de frecvenţe f: în $f[i]$ vom memora dacă nodului i a apărut ca descendent stâng sau drept al unui nod în datele de intrare.

Pentru parcurgerea în postordine a nodurilor unui arbore binar vom utiliza un algoritm recursiv de parcurgere.

Listing 1: postordine_v2.c

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

#define NMAX 1000

typedef struct node {
          int lbl;            // eticheta nodului
          struct node* left;  // adresa descendentului stang
          struct node* right; // adresa descendentului drept
        } NODE;

NODE* nodes[NMAX + 1];  // nodes[k] - adresa zonei unde a fost alocat spatiul
                        //            pentru nodul k
short f[NMAX + 1];      // f[k] - nodul k a aparut ca descendent al unui nod

/*
*   Functia afiseaza intr-un fisier informatii despre un nod al arborelui
*  binar.
*  @param p - adresa unei zone de memorie ce contine informatii despre un nod
*             al arborelui binar
*  @param f - adresa unei structuri de tip FILE reprezentand fisierul unde
*             se vor afisa informatiile.
*/
void visit(NODE* p, FILE* f) {
  fprintf(f, "%d ", p->lbl);
}

/*
*   Functie recursiva ce implementeaza metoda de vizitare in postordine
*  a nodurilor unui arbore binar.
*  @param p    - adresa unei zone de memorie ce contine informatii despre
*                nodul curent
*  @param fout - adresa unei structuri de tip FILE reprezentand fisierul
*                unde se vor afisa informatiile.
*/
void postorder(NODE* p, FILE* fout) {
  if (p != NULL) {
    // vizitam descendentul stang al nodului p
    postorder(p->left, fout);
    // vizitam descendentul drept al nodului p
    postorder(p->right, fout);

    // vizitam nodul curent p
    visit(p, fout);
  }
}

int main() {
```

```
  FILE *fin, *fout;
  int n, i, rad, stang, drept;

  fin = fopen("postordine.in", "r");
  fout = fopen("postordine.out", "w");

  fscanf(fin, "%d", &n);

  // se aloca spatiu pentru cele n noduri
  for (i = 1; i <= n; i++) {
    nodes[i] = (NODE*)malloc(sizeof(NODE));
  }

  // se construieste arborele: citim etichetele nodurilor si
  // realizam legaturile catre descendentii sai stang si drept
  for (i = 1; i <= n; i++) {
    fscanf(fin, "%d %d %d", &nodes[i]->lbl, &stang, &drept);

    f[stang]++;
    f[drept]++;

    // se initializeaza descendentul stang al nodului reprezentat de nodes[i]
    if (stang == 0) {
      nodes[i]->left = NULL;
    } else {
        nodes[i]->left = nodes[stang];
    }

    // se initializeaza descendentul drept al nodului reprezentat de nodes[i]
    if (drept == 0) {
      nodes[i]->right = NULL;
    } else {
        nodes[i]->right = nodes[drept];
    }
  }

  // se determina radacina arborelui
  for (i = 1, rad = 0; (i <= n) && (rad == 0); i++) {
    if (f[i] == 0) {
      rad = i;
    }
  }

  // se apeleaza functia de vizitare in postordine
  postorder(nodes[rad], fout);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

2. Se consideră un arbore binar în care nodurile memorează numere naturale nenule. Să se afişeze valorile memorate în subarborele stâng al rădăcinii în urma parcurgerii în preordine.

Date de intrare

Fişierul de intrare `preordine1.in` conţine pe prima linie lista valorilor memorate în nodurile arborelui, obţinute în urma parcurgerii în preordine *(rădăcină, stâng, drept)*. Dacă un nod nu are descendent stâng, în listă va apare valoarea 0. Dacă un nod nu are descendent drept, în listă va apare valoarea 0.

Date de ieşire

Fişierul de ieşire `preordine1.out` va conţine pe prima linie valorile memorate în subarborele stâng al rădăcinii în urma parcurgerii în preordine, separate prin exact un spaţiu.

Precizări

Se recomandă folosirea arborilor alocaţi dinamic.

Exemplu

`preordine1.in`
67 51 18 0 0 24 0 0 48 0 11 0 0
`preordine1.out`
51 18 24

(Preordine1, `https://www.pbinfo.ro/probleme/753/preordine1`)

Rezolvare:
Arborele binar este reprezentat cu ajutorul unei structuri de date definită de programator denumită `NODE`. Vom prezenta o implementare recursivă ce construieşte în memorie un arbore binar pe baza datelor de intrare.

Listing 2: `preordine1.c`

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

typedef struct node {
          int lbl;            // eticheta nodului
          struct node* left;  // adresa descendentului stang
          struct node* right; // adresa descendentului drept
        } NODE;

/*
*    Functia aloca spatiu pentru o structura de tip NODE si initializeaza
*  campurile acesteia.
*  @param lbl - eticheta/valoarea nodului curent
*  @return    - adresa zonei de memorie a unui nod al arborelui binar,
*               zona ce a fost alocata in cadrul functiei
*/
NODE* createNode(int lbl) {
  NODE* p;

  p = (NODE*) malloc(sizeof(NODE));
  p->lbl = lbl;
```

```
  p->left = NULL;
  p->right = NULL;

  return p;
}

/*
*    Functie recursiva pentru construirea arborelui binar.
*  @param fin - adresa unei structuri de tip FILE reprezentand fisierul
*               de unde se vor prelua datele de intrare.
*  @return    - adresa radacinii arborelui binar construit
*/
NODE* buildTree(FILE* fin) {
  NODE* p;
  int v;

  fscanf(fin, "%d", &v);        // se citeste eticheta nodului curent
  if (v > 0) {                  // daca valoarea lui v este strict pozitiva
    p = createNode(v);          // se construieste un nod nou
    p->left = buildTree(fin);   // se construieste recursiv subarborele stang
    p->right = buildTree(fin);  // se construieste recursiv subarborele drept
  } else {
      p = NULL;                 // altfel nodul curent ia valoarea NULL
  }

  return p;
}

/*
*    Functie recursiva ce implementeaza metoda de vizitare in preordine
*  a nodurilor unui arbore binar.
*  @param p    - adresa unei zone de memorie ce contine informatii
*                despre nodul curent
*  @param fout - adresa unei structuri de tip FILE reprezentand fisierul
*                unde vor fi afisate informatiile.
*/
void printPreorder(NODE* p, FILE* fout) {
  if (p != NULL) {
    fprintf(fout, "%d ", p->lbl);

    printPreorder(p->left, fout);
    printPreorder(p->right, fout);
  }
}

int main() {
  FILE *fin, *fout;
  NODE* rad;

  fin = fopen("preordine1.in", "r");
  fout = fopen("preordine1.out", "w");
```

```
  rad = buildTree(fin);

  printPreorder(rad->left, fout);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

3. Se consideră un arbore binar în care nodurile memorează numere naturale nenule. Să se determine înălţimea arborelui. Înălţimea unui arbore este egală cu numărul de noduri de pe cel mai lung lanţ elementar ce uneşte rădăcina cu un nod terminal.

   Se recomandă folosirea arborilor alocaţi dinamic.

   Date de intrare

   Fişierul de intrare `inaltime1.in` conţine pe prima linie lista valorilor memorate în nodurile arborelui, obţinute în urma parcurgerii în preordine *(rădăcină, stâng, drept)*. Dacă un nod nu are descendent stâng, în listă va apare valoarea 0. Dacă un nod nu are descendent drept, în listă va apare valoarea 0.

   Date de ieşire

   Fişierul de ieşire `inaltime1.out` va conţine pe prima linie o valoare H, reprezentând înălţimea arborelui.

   Exemplu

inaltime1.in
67 51 18 0 0 24 0 0 48 0 11 0 0
inaltime1.out
3

   (Inaltime1, `https://www.pbinfo.ro/probleme/761/inaltime1`)

   Rezolvare:

Listing 3: `inaltime1.c`

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

typedef struct node {
          int lbl;            // eticheta nodului
          struct node* left;  // adresa descendentului stang
          struct node* right; // adresa descendentului drept
        } NODE;

/*
*  Functia intoarce valoarea cea mai mare dintre valorile celor doi parametri
*/
int max(int a, int b) {
  return (a > b) ? a : b;
}

/*
```

```
*    Functia aloca spatiu pentru o structura de tip NODE si initializeaza
*  campurile acesteia.
*  @param lbl - eticheta/valoarea nodului curent
*  @return    - adresa zonei de memorie a unui nod al arborelui binar,
*               zona ce a fost alocata in cadrul functiei
*/
NODE* createNode(int lbl) {
  NODE* p;

  p = (NODE*) malloc(sizeof(NODE));
  p->lbl = lbl;
  p->left = NULL;
  p->right = NULL;

  return p;
}

/*
*    Functie recursiva pentru construirea arborelui binar.
*  @param fin - adresa unei structuri de tip FILE reprezentand fisierul
*               de unde se vor prelua datele de intrare.
*  @return    - adresa radacinii arborelui binar construit
*/
NODE* buildTree(FILE* fin) {
  NODE* p;
  int v;

  fscanf(fin, "%d", &v);        // se citeste eticheta nodului curent
  if (v > 0) {                  // daca este valoarea lui v strict pozitiva
    p = createNode(v);          // se construieste un nod nou
    p->left = buildTree(fin);   // se construieste recursiv subarborele stang
    p->right = buildTree(fin);  // se construieste recursiv subarborele drept
  } else {
     p = NULL;                  // altfel nodul curent este NULL
  }

  return p;
}

/*
*    Functia determina inaltimea subarborelui avand ca radacina nodul p
*  si returneaza aceasta valoare.
*  @param p - adresa unei zone de memorie ce contine informatii despre nodul
*             curent
*  @return -  o valoare naturala ce reprezinta inaltimea arborelui avand
*             radacina p
*/
int getHeight(NODE* p) {
  if (p == NULL) {
    return 0;
  } else {
     return max(getHeight(p->left), getHeight(p->right)) + 1;
```

```
  }
}

int main() {
  FILE *fin, *fout;
  NODE* rad;

  fin = fopen("inaltime1.in", "r");
  fout = fopen("inaltime1.out", "w");

  rad = buildTree(fin);

  fprintf(fout, "%d\n", getHeight(rad));

  fclose(fin);
  fclose(fout);

  return 0;
}
```

4. Se consideră un arbore binar în care nodurile memorează numere naturale nenule şi un număr k. În arbore rădăcina se află pe nivelul 0, fii rădăcinii pe nivelul 1, fii fiilor rădăcinii pe nivelul 2 etc.

   Să se determine suma valorilor din nodurile aflate pe nivelul k.

   Se recomandă folosirea arborilor alocaţi dinamic.

   Date de intrare

   Fişierul de intrare `knivel1.in` conţine pe prima linie lista valorilor memorate în nodurile arborelui, obţinute în urma parcurgerii în preordine *(rădăcină, stâng, drept)*. Dacă un nod nu are descendent stâng, în listă va apare valoarea 0. Dacă un nod nu are descendent drept, în listă va apare valoarea 0.

   Linia a doua a fişierului conţine numărul k.

   Date de ieşire

   Fişierul de ieşire `knivel1.out` va conţine pe prima linie un singur număr S, reprezentând suma cerută.

   Exemplu

knivel1.in	knivel1.out
67 51 18 0 0 24 0 0 48 0 11 0 0 1	99

   (kNivel1, `https://www.pbinfo.ro/probleme/760/knivel1`)

   Rezolvare:

Listing 4: `knivel1.c`

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

#define NMAX 1000
```

```
typedef struct node {
          int lbl;            // eticheta nodului
          struct node* left;  // adresa descendentului stang
          struct node* right; // adresa descendentului drept
        } NODE;

/*
*  Functia intoarce cea mai mare valoare dintre valorile celor doi parametri
*/
int max(int a, int b) {
  return (a > b) ? a : b;
}

/*
*    Functia aloca spatiu pentru o structura de tip NODE si initializeaza
*  campurile acesteia.
*  @param lbl - eticheta/valoarea nodului curent
*  @return    - adresa zonei de memorie a unui nod al arborelui binar,
*               zona ce a fost alocata in cadrul functiei
*/
NODE* createNode(int lbl) {
  NODE* p;

  p = (NODE*) malloc(sizeof(NODE));
  p->lbl = lbl;
  p->left = NULL;
  p->right = NULL;

  return p;
}

/*
*   Functie recursiva pentru construirea arborelui binar.
*  @param fin - adresa unei structuri de tip FILE reprezentand fisierul
*               de unde se vor prelua datele de intrare.
*  @return    - adresa radacinii arborelui binar construit
*/
NODE* buildTree(FILE* fin) {
  NODE* p;
  int v;

  fscanf(fin, "%d", &v);
  if (v > 0) {
    p = createNode(v);

    p->left = buildTree(fin);
    p->right = buildTree(fin);
  } else {
      p = NULL;
  }
```

```
  return p;
}

/*
*    Functia determina suma valorilor nodurilor unui arbore binar avand drept
*  radacina nodul p, noduri ce se afla pe un nivel specificat al acestui
*  arbore si returneaza aceasta valoare.
*  @param level - nivelul curent din arbore
*  @param p     - adresa unei zone de memorie ce contine informatii despre
*                 nodul curent
*  @param target - nivelul din arbore pentru care se va calcula suma
*                  etichetelor nodurilor aflate pe acel nivel
*  @return      -  suma valorilor nodurilor din subarborele de radacina p
*                  aflate pe nivelul target in arborele initial
*/
int getSumNodesForLevel(int level, NODE* p, int target) {
  if (p == NULL) {
    return 0;
  } else {
      if (level < target) {
        return getSumNodesForLevel(level + 1, p->left, target)
               + getSumNodesForLevel(level + 1, p->right, target);
      } else {
          return p->lbl;
      }
  }
}

int main() {
  FILE *fin, *fout;
  NODE* rad;
  int k;

  fin = fopen("knivel1.in", "r");
  fout = fopen("knivel1.out", "w");

  rad = buildTree(fin);

  fscanf(fin, "%d", &k);

  fprintf(fout, "%d\n", getSumNodesForLevel(0, rad, k));

  fclose(fin);
  fclose(fout);

  return 0;
}
```

5. Se consideră un arbore binar în care nodurile memorează numere naturale nenule. Să se afişeze valorile din arbore în urma parcurgerii în lăţime, pornind din rădăcină.

   Se recomandă folosirea arborilor alocaţi dinamic.

Date de intrare

Fişierul de intrare `bilatime.in` conţine pe prima linie lista valorilor memorate în nodurile arborelui, obţinute în urma parcurgerii în preordine *(rădăcină, stâng, drept)*. Dacă un nod nu are descendent stâng, în listă va apare valoarea 0. Dacă un nod nu are descendent drept, în listă va apare valoarea 0.

Date de ieşire

Fişierul de ieşire `bilatime.out` va conţine pe prima linie valorile din arbore în urma parcurgerii în lăţime, pornind din rădăcină, separate prin câte un spaţiu.

La parcurgerea în lăţime, după vizitarea unui nod, se va vizita fiul stâng, apoi fiul drept.

Exemplu

bilatime.in	bilatime.out
67 51 18 0 0 24 0 0 48 0 11 0 0	67 51 48 18 24 11

(BiLatime, `https://www.pbinfo.ro/probleme/759/bilatime`)

Rezolvare:

Listing 5: `bilatime.c`

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

#define NMAX 1000

typedef struct node {
          int lbl;             // eticheta nodului
          struct node* left;  // adresa descendentului stang
          struct node* right; // adresa descendentului drept
        } NODE;

typedef struct queue {
           short f;
           short l;
           NODE* t[NMAX + 1];
        } QUEUE;

QUEUE q;  // o coada

/*
*  Functia intoarce cea mai mare valoare dintre valorile celor doi parametri
*/
int max(int a, int b) {
  return (a > b) ? a : b;
}

/*
*  Functia initializeaza coada q ca fiind vida.
*/
void init(QUEUE* q) {
  q->f = 0;
```

```
  q->l = -1;
}

/*
*   Functia verifica daca coada q este vida. Intoarce valoarea 1
*  daca coada este vida sau 0 daca coada contine cel putin un element.
*/
int isempty(QUEUE* q) {
  return ((q->l + 1) == q->f);
}

/*
*   Functia adauga in coada q elementul a carui valoare este
*  pastrata in variabila p.
*/
void enqueue(QUEUE* q,  NODE* p) {
  q->l++;
  q->t[q->l] = p;
}

/*
*  Functia sterge din coada q elementul situat la inceputul cozii.
*/
void dequeue(QUEUE* q) {
  q->f++;
}

/*
*  Functia intoarce valoarea elementului situat la inceputul cozii q.
*/
NODE* front(QUEUE* q) {
  return q->t[q->f];
}

/*
*    Functia aloca spatiu pentru o structura de tip NODE si initializeaza
*  campurile acesteia.
*  @param lbl - eticheta/valoarea nodului curent
*  @return    - adresa zonei de memorie a unui nod al arborelui binar,
*               zona ce a fost alocata in cadrul functiei
*/
NODE* createNode(int lbl) {
  NODE* p;

  p = (NODE*) malloc(sizeof(NODE));
  p->lbl = lbl;
  p->left = NULL;
  p->right = NULL;

  return p;
}
```

```
/*
*   Functie recursiva pentru construirea arborelui binar.
*  @param fin - adresa unei structuri de tip FILE reprezentand fisierul
*               de unde se vor prelua datele de intrare.
*  @return    - adresa radacinii arborelui binar construit
*/
NODE* buildTree(FILE* fin) {
  NODE* p;
  int v;

  fscanf(fin, "%d", &v);
  if (v > 0) {
    p = createNode(v);

    p->left = buildTree(fin);
    p->right = buildTree(fin);
  } else {
      p = NULL;
  }

  return p;
}

/*
*    Functia asigura parcurgerea in latime a nodurilor unui arbore binar.
*  @param root - adresa unei zone de memorie ce contine informatii despre
*                radacina arborelui binar
*  @param fout - adresa unei structuri de tip FILE reprezentand fisierul
*                unde se vor afisa informatiile.
*/
void bfs(NODE* root, FILE* fout) {
  NODE* p;

  // se marcheaza nodul curent ca fiind vizitat
  fprintf(fout, "%d ", root->lbl);

  // se initializeaza coada
  init(&q);

  // se adauga nodul curent in coada
  enqueue(&q, root);
  // cat timp coada nu este vida
  while (!isempty(&q)) {
    // valoarea elementului aflat la inceputul cozii
    p = front(&q);
    // se elimina elementul situat la inceputul cozii
    dequeue(&q);

    // daca exista descendentul stang
    if (p->left) {
      // se adauga descendentul stang in coada
      enqueue(&q, p->left);
```

```
      // se marcheaza descendentul stang ca fiind vizitat
      fprintf(fout, "%d ", p->left->lbl);
    }

    // daca exista descendentul drept
    if (p->right) {
      // se adauga descendentul drept in coada
      enqueue(&q, p->right);

      // se marcheaza descendentul drept ca fiind vizitat
      fprintf(fout, "%d ", p->right->lbl);
    }
  }
}

int main() {
  FILE *fin, *fout;
  NODE* rad;

  fin = fopen("bilatime.in", "r");
  fout = fopen("bilatime.out", "w");

  rad = buildTree(fin);

  bfs(rad, fout);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

6. Se consideră un arbore binar în care nodurile memorează numere naturale nenule. Să se afişeze valorile memorate în nodurile terminale ale arborelui, în ordine crescătoare.

   Se recomandă folosirea arborilor alocaţi dinamic.

   Date de intrare

   Fişierul de intrare `bifrunze1.in` conţine pe prima linie lista valorilor memorate în nodurile arborelui, obţinute în urma parcurgerii în preordine *(rădăcină, stâng, drept)*. Dacă un nod nu are descendent stâng, în listă va apare valoarea 0. Dacă un nod nu are descendent drept, în listă va apare valoarea 0. Arborele va avea cel mult 5.000 de noduri terminale.

   Date de ieşire

   Fişierul de ieşire `bifrunze1.out` va conţine pe prima linie valorile memorate în nodurile terminale ale arborelui, în ordine crescătoare, separate prin câte un spaţiu.

   Exemplu

bifrunze1.in
67 51 18 0 0 24 0 0 48 0 11 0 0

bifrunze1.out
11 18 24

(BiFrunze1, https://www.pbinfo.ro/probleme/791/bifrunze1)

Rezolvare:
Pe baza datelor de intrare se construieşte un arbore binar ale cărui noduri sunt alocate dinamic. Arborele construit se parcurge în postordine şi se memorează etichetele nodurilor frunză. Se ordonează crescător aceste valori cu ajutorul funcţiei `qsort()`[1] şi se afişează şirul ordonat.

Listing 6: `bifrunze1.c`

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

#define NMAX 5000

typedef struct node {
          int lbl;            // eticheta nodului
          struct node* left;  // adresa descendentului stang
          struct node* right; // adresa descendentului drept
        } NODE;

int leaves[NMAX];
int last;

/*
*    Functia aloca spatiu pentru o structura de tip NODE si initializeaza
*  campurile acesteia.
*  @param lbl - eticheta/valoarea nodului curent
*  @return    - adresa zonei de memorie a unui nod al arborelui binar,
*               zona ce a fost alocata in cadrul functiei
*/
NODE* createNode(int lbl) {
  NODE* p;

  p = (NODE*) malloc(sizeof(NODE));
  p->lbl = lbl;
  p->left = NULL;
  p->right = NULL;

  return p;
}

/*
*   Functie recursiva pentru construirea arborelui binar.
*  @param fin - adresa unei structuri de tip FILE reprezentand fisierul
*               de unde se vor prelua datele de intrare.
*  @return    - adresa radacinii arborelui binar construit
```

[1] `void qsort(void *ptr, size_t count, size_t size, int (*comp)(const void *, const void *));`

```
*/
NODE* buildTree(FILE* fin) {
  NODE* p;
  int v;

  fscanf(fin, "%d", &v);        // se citeste eticheta nodului curent
  if (v > 0) {                  // daca valoarea lui v este strict pozitiva
    p = createNode(v);          // se construieste un nod nou
    p->left = buildTree(fin);   // se construieste recursiv subarborele stang
    p->right = buildTree(fin);  // se construieste recursiv subarborele drept
  } else {
      p = NULL;                 // altfel nodul curent ia valoarea NULL
  }

  return p;
}

/*
*   Functie recursiva ce implementeaza metoda de vizitare in postordine
*  a nodurilor unui arbore binar.
*  @param p    - adresa unei zone de memorie ce contine informatii despre
*                nodul curent
*/
void visitPostorder(NODE* p) {
  if (p != NULL) {
    visitPostorder(p->left);
    visitPostorder(p->right);

    // daca nodul p este frunza
    if ((p->left == NULL) && (p->right == NULL)) {
      // atunci se memoreaza eticheta acestuia
      leaves[last++] = p->lbl;
    }
  }
}

/*
*  Functie de comparare utilizata de functia qsort() pentru ordonarea
* crescatoare a etichetelor (valori naturale) nodurilor frunza.
*/
int cmp(const void* a, const void* b) {
  return *(int*)a - *(int*)b;
}

int main() {
  FILE *fin, *fout;
  NODE* rad;
  int i;

  fin = fopen("bifrunze1.in", "r");
  fout = fopen("bifrunze1.out", "w");
```

```
  rad = buildTree(fin);

  visitPostorder(rad);

  // se ordoneaza crescator etichetele nodurilor frunza
  qsort(leaves, last, sizeof(int), cmp);

  for (i = 0; i < last; i++) {
    fprintf(fout, "%d ", leaves[i]);
  }

  fclose(fin);
  fclose(fout);

  return 0;
}
```

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introducere în Algoritmi*, Computer Libris Agora, Cluj-Napoca, 1999.

[2] M. Coşulschi, M. Gabroveanu, *Practica programării în C*, Editura Universitaria, Craiova, 2014.