Structuri de Date
Laboratorul 6: Arbori Binari de Căutare

Dan Novischi

24 martie 2018

1. Introducere

Scopul acestui laborator îl reprezintă implementarea unui arbore binar de căutare. Obiectivul este acela de a vă familiariza cu lucrul cu arbori binari și de a face o comparație cu structurile ordonate definite in laboratoarele anterioare (mulțimi și liste ordonate).

Laboratorul are 5 cerințe care urmăresc:

- definirea interfețe de lucru cu arborii binari de cautare (BST);
- implementarea funcțiilor aferente interfeței de lucru cu BST;
- implementarea unei funcții care deterimină dacă un arbore binar este un arbore de căutare;
- crearea unei liste cu nodurile arborelui de pe ficare nivel;

2. Arbori binari de căutare

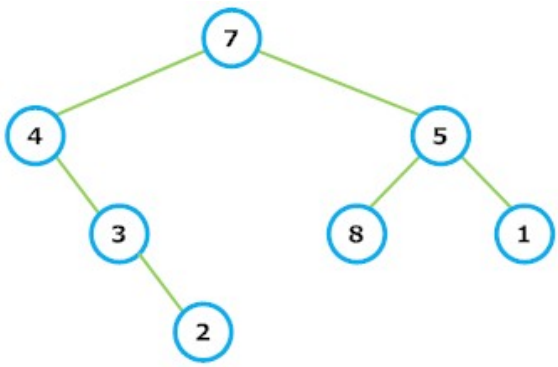

Figura 1: Arbore Binar

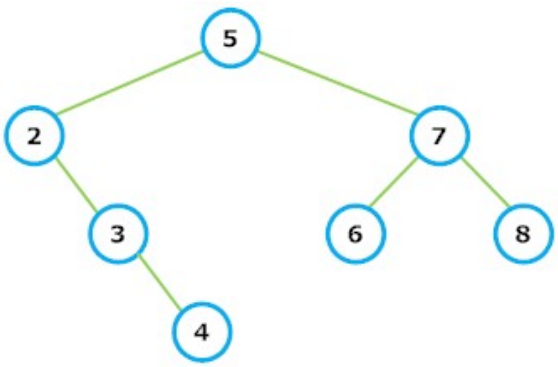

Figura 2: Arbore Binar de Căutare

Arborii sunt structuri de date ierarhice ale căror elemente se numesc **noduri** (simliar cu nodurile listelor definite în laboratorele anterioare). Nodurile, stochează valorile datelor de

interes (într-un camp denumit **elem**) și conține legături aferente pentru organizarea acestora sub formă ierarhică. Astfel, un arbore binar (vezi Figura 1) este o structură de date dinamică ale cărui noduri pot avea cel mult două legaturi ierarhice.

Vocabular:

- Cel mai de sus nod din ierarhie se numește *rădăcina* arborelui. În Figura 1 rădacina este dată de nodul cu $elem = 7$, respectiv în Figura 2 de nodul cu $elem = 5$.
- Dacă un nod x este ierarhic imediat sub alt nod y, atunci x se numește copil al nodului y, iar y se numește părinte al nodului x. În Figura 2 nodul cu $elem = 6$ este copil al nodului cu $elem = 7$. La randul lui nodul cu $elem = 7$ este parinte pentru nodurile cu $elem = 6$ și $elem = 8$.
- Nodurile care nu au copii se numesc furnze. În Figura 2 aceste sunt date de nodurile cu elementele 4, 6 și 8.

Un **arbore binar de căutare** este un arbore binar ale cărui noduri respectă următoarele relații de ordine:

```
node->l->elem < node->elem;
node->r->elem > node->elem;
```

Spre exemplu, arborele din Figura 2 este un arbore binar de căutare, deoarce toate elementele acestuia respectă relațiile de ordine de mai sus. Astfel, avem următoarele definiții pentru un abore și nodurile acestuia:

```
1 typedef struct BSTNode{
2     Item elem;
3     struct BSTNode *p;
4     struct BSTNode *l;
5     struct BSTNode *r;
6 }BSTNode;
```

```
1 typedef struct BSTree{
2     long size;
3     BSTNode *root;
4     BSTNode *nil;
5 }BSTree;
```

ATENȚIE:

- Un nod poate avea doar legăturile pentru copiii *l (left)* și *r (right)*, nefiind necesară legătura la nodul părinte *p (parent)*. Dar, untilizarea acestuia ne va simplifica implementarea anumitor operații de lucru cu arborii.
- Definiția pentru arbore de mai sus folosește două santinele: **root** pentru rădăcina arborelui și **nil** pentru elementul null.
- Convenția pentru legătura la santinela **root** este să atașăm varful arborelui întodeauna în partea stangă.
- O implementare corectă va genera un arbore echivalent celui din Figura 2 ca mai jos.

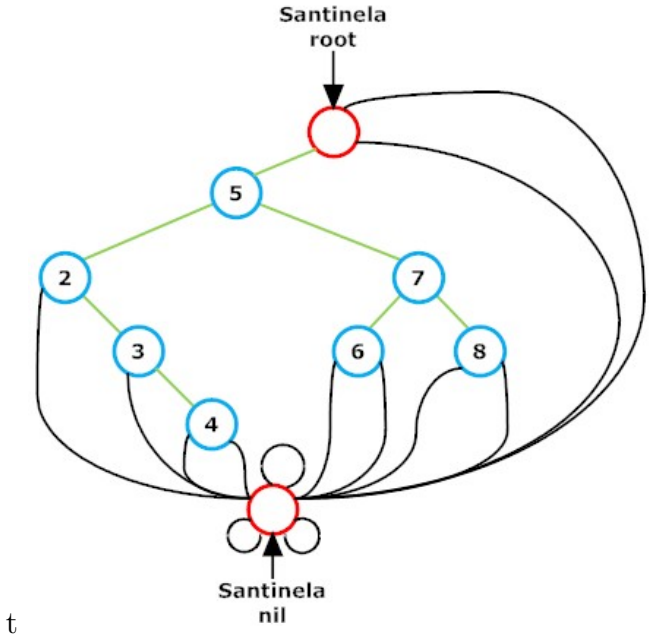

Figura 3: Arbore Binar de Căutare Echivalent

Observație: Lamuriți cu asistentul orice aspect neclar.

Cerința 1 (0p) Definiți în `BST.h` antetul (header-ele) pentru fiecare din funcțiile descrise mai jos.

- `createTree`: Nu primește nimic și întoarce un arbore (`BSTree*`).
- `isEmpty`: Verifică dacă un arbore este gol sau nu (1 sau 0).
- `insert`: Primește un arbore și un `elem`. Funcția nu întoarce nimic.
- `search`: Primește un arbore și un `elem`. Funcția întoarce o un nod al arborelui.
- `minimum`: Primește un arbore și un nod. Funcția întoarce un nod.
- `maximum`: Primește un arbore și un nod. Funcția întoarce un nod.
- `successor`: Primește un arbore și un nod. Funcția întoarce un nod.
- `predecessor`: Primește un arbore și un nod. Funcția întoarce un nod.
- `destroyTree`: Primește un arbore și nu întoarce nimic.

Observație: Pentru a verifica faptul că ați scris corect antetul, compilarea trebuie să reușească.

```
$ make
gcc -std=c9x -g -O0 BSTList.c -o BSTList -lm
gcc -std=c9x -g -O0 longTest.c -o longTest -lm
gcc -std=c9x -g -O0 parseTree.c -o parseTree -lm
gcc -std=c9x -g -O0  testBST.c -o testBST -lm
```

Cerința 2 (6p) Implementați funcțiile de mai sus în urmatoarea ordine: `createTree`, `isEmpty`, `insert`, `search`, `contains`, `minimum`, `maximum`, `successor`, `predecessor`, `delete`, `destroyTree`.

Indicații:

- `createTree`: Creează un arbore nou.
- `isEmpty`: Primul nod al arborelui se află la `root->l`.
- `insert`: Caută punctul de inserție și insearează `elem` dacă acesta nu există deja. În arbore nu se vor stoca elemente `elem` duplicat.
- `search`: Întoarce un nod cu elementul `elem` primit ca parametru dacă acesta există.
- `minimum`: Întoarce nodul cu elementul `elem` cel mai mic din arbore.
- `maximum`: Întoarce nodul cu elementul `elem` cel mai mare din arbore.
- `successor`: Întoarce succesorul nodului primit ca parametru. Acesta poate fi cel mai din stanga nod al sub-arborelui drept. Sau în cazul in care sub-arborele drept nu există, succesorul este parintele al cărui copil din stanga este un strămunic al nodului primit ca parametru.
- `predecessor`: Simetric funcției `successor`. (simetric NU echivalent! :))
- `destroyTree`: Dealocă tot arborele.

Observație: Verificați-vă pe parcurs cu testerul și respectați ordinea de implementare.

```
$ ./testBST
. Testul Init&IsEmpty a fost trecut cu succes!  Puncte: 0.05
. Testul Insert a fost trecut cu succes!  Puncte: 0.10
. Testul Search a fost trecut cu succes!  Puncte: 0.10
. Testul Minimum&Maximium a fost trecut cu succes!  Puncte: 0.10
. Testul Successor&Predecessor a fost trecut cu succes!  Puncte: 0.15
. Testul Destroy: *Se va verifica cu valgrind*  Puncte: 0.10.

Scor total: 0.60 / 0.60
```

Cerința 3 (2p) În fișierul `longTest.c` este deja implementată o funcție `buildTree` care construiește un arbore binar de căutare dintr-un array de elemente întregi (`long`). Implementați o funcție `isBST` care verifică dacă un arbore este sau nu un arbore binar de căutare. Funcția returnează 0 sau 1.

Observație: Puteți verifca corectitudinea implementării aplicand funcția pe arborele construit de funcția `buidTree`.

Cerința 4 (2p) În fișierul `BSTList.c` este deja implementată o funcție `buildTree` care construiește un arbore binar de căutare dintr-un array de elemente întregi (`long`). Implementați o funcție `bstToList` care construiește o listă cu nodurile arborelui. Ordinea nodurilor din listă este data de ordinea de pe ficare nivel (de la stanga la drepta) începand cu nivelul rădăcinii (de sus în jos). Totodată, creați o fucție `printList` care să afișeze valorile elementelor din lista creeata.

Observație: Outpul pentru arborele din Figura 4 arată astfel:

```
$ ./BSTList
5 3 7 2 4 6 8
```

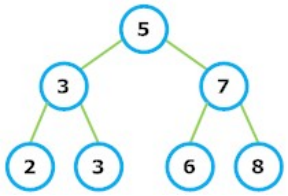

Figura 4: Arbore Binar de Căutare Echilibrat