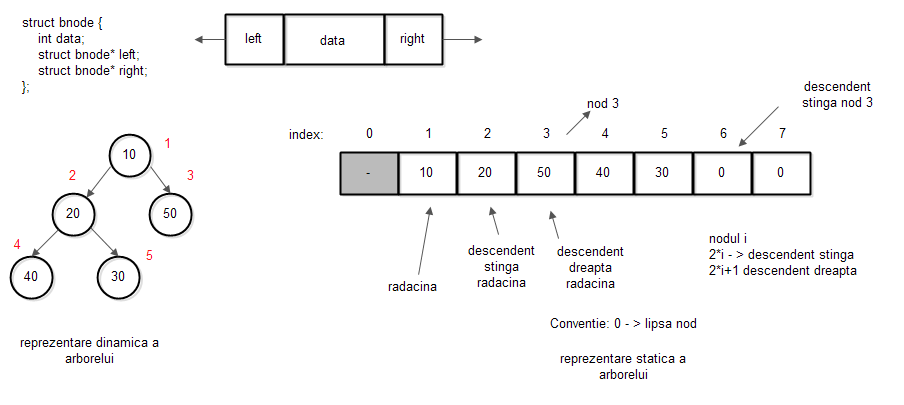
**C5 SDA**

Arbori binari

Reprezentarea arborilor binari

**Reprezentare statica** (cu tablouri) - nesecventiala -> accesul la un nod se va face direct (mai complicat insa decit la liste)



structura nodului (ca listele dublu inlantuite)

tablou cu un numar de elemente N

**distribuirea nodurilor in tablou:**

0 - nu este folosit

1 - info din nodul radacina - (nivel 1) - 1 nod

2, 3 - info din nodurile descendente ale radacinii (nivelul 2) - 2 noduri

4, 5, 6, 7 -> nivelul 3 (4 noduri)

Pentru un arbore cu *k niveluri*:

*pe nivelul k -> pot exista maxim 2^(k-1) noduri*

Regula -> nodul i -> 2\*i -> descendentul din stinga al nodului i

-> 2\*i+1 -> descendentul din dreapta al nodului i

Daca nu exista descendent - > pozitia asociata va ramine neutilizata (se va trece o valoare implicita, care nu reprezinta informatie)

**Creare nod nou in arbore**

struct bnode\* new\_tree\_node(int a)

{

struct bnode\* p;

p= (struct bnode\*) malloc(sizeof(struct bnode)); // se aloca memorie

p->data=a; //se inițalizeaza campul de informatie

p->left=NULL; //se intializeaza cele 2 legaturi catre Null

p->right=NULL;

}

*Construirea arborilor binari - implementare dinamica*

Exemplificare:

Arbori binari perfect echilibrati (abe)

**abe**- > pentru orice nod x al arborelui numarul de noduri ai subarborelui sting al nodului x si numarul de noduri ai subarborelui drept al nodului x difera cu cel mult 1.

N(x,L) - numar de noduri din subarborele sting al nodului x

N(x,R) - numar de noduri din subarborele drept al nodului x

N(x,L) - N(x, R) = {-1, 0, 1}, pentru orice x

Algoritm de construire a abe

algoritm recursiv

A = {a1, a2, ...., aN} -> multimea de elemente (info din nod) cu care se va construit abe

N trebuie sa fie cunoscut

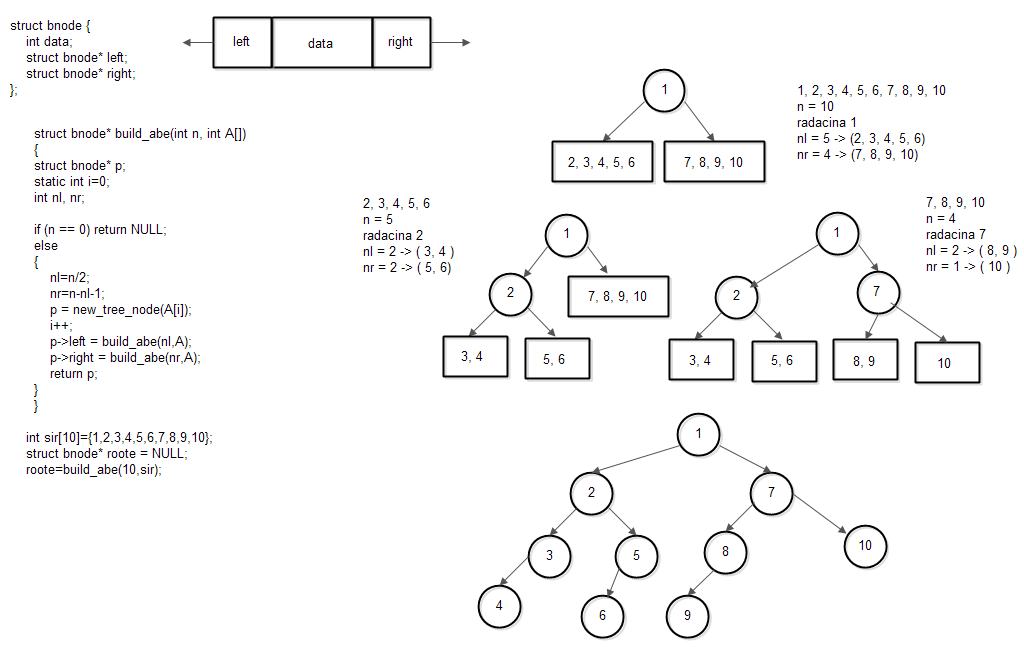
1) a1 -> radacina

2) urmatoarele nl = N/2 elemente (a2, a3, ... ak) -> vor constitui subarborele sting

3) urmatoarele nr = N - N/2 -1 elemente (ak, ak+1, ... aN) -> vor constitui subarborele drept

pasii 2) si 3) -> se efectueaza recursiv

se iese din recursivitate -> cind nl si/sau nr devin 0



struct bnode\* build\_abe(int n, int A[])

{

struct bnode\* p;

static int i=0;

int nl, nr;

if (n == 0) return NULL;

else

{

nl=n/2;

nr=n-nl-1;

p = new\_tree\_node(A[i]);

i++;

p->left = build\_abe(nl,A);

p->right = build\_abe(nr,A);

return p;

}

}

**Arbori echilibrati-> cautare eficienta**

Arbori binari de cautare (abc)

info(x) -> informatia din nodul x

x-> left : descendentul din stinga al lui x

x-> right : descendentul din dreapta al lui x

*Definitie:*

Pentru orice nod x avem:

info(x->left) < info(x) < info(x->right)

Algoritm recursiv

nu necesita cunosterea numarului de noduri -> fiecare nod se va "aseza" la locul potrivit conform relatiei din definitie

1) daca arborele nu exista - > se creaza un nod -> nod radacina

2) daca radacina exista -> se cauta o pozitie libera, pornind de radacina, pozitia noului nod, in stinga sau in dreapta (respectind conditia din definitie) si se insereaza (se "leaga") nodul nou

**Inserarea in arbori binari de cautare construire abc**

struct bnode\* build\_abc(struct bnode\*r, int a)

{

if (r==NULL) r= new\_tree\_node(a);

else

{

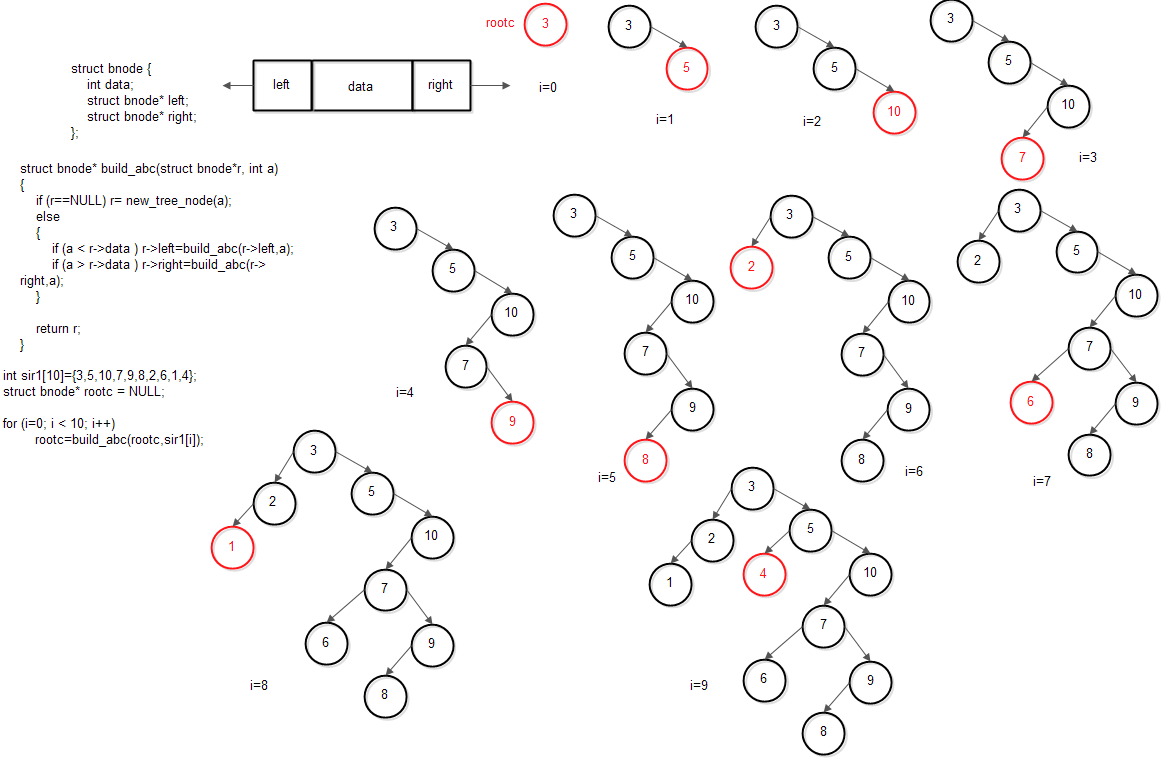
if (a < r->data ) r->left=build\_abc(r->left,a);

if (a > r->data ) r->right=build\_abc(r->right,a);

}

return r;

}



**Cautarea in arbori binari**

struct bnode\* search\_abc(struct bnode\*r, int a)

{

if (r == NULL) return NULL;

if (r->data == a) return r;

if (a < r->data) return (search\_abc(r->left,a));

if (a > r->data) return (search\_abc(r->right,a));

}

*Obs*: functia de construire a abc -> a presupus ca elementele de la intrare sint distincte (nu vor fi inserate in abc de mai multe ori)

*Tema*: modificati functia build\_abc in asa fel incit sa se inregistreze elemente multiple.

Ideea: modificati structura noduluu astfel:

struct bnode {

int data;

int cnt; // contor de numar de aparitii

struct bnode\* left;

struct bnode\* right;

};

**Parcurgerea arborilor binari**

3 moduri de parcurgere:

LDR - in ordine -> se parcurge subarborele din stinga, radacina, subarborele din dreapta

DLR - in post ordine -> se parcurge radacina, subarborele din stinga, subarborele din dreapta

LRD - in preordine -> se parcurge subarborele din stinga, subarborele din dreapta, radacina

Algoritmii de parcurgere -> recursivi

Prin simetrie:

parcurgeri: RDL, DRL, RLD

**Functii de parcurgere:**

void ldr(struct bnode\* r)

{

if(r!=NULL)

{

ldr(r->left);

printf("%d, ", r->data);

ldr(r->right);

}

}

void dlr(struct bnode\* r)

{

if(r!=NULL)

{

printf("%d, ", r->data);

dlr(r->left);

dlr(r->right);

}

}

void lrd(struct bnode\* r)

{

if(r!=NULL)

{

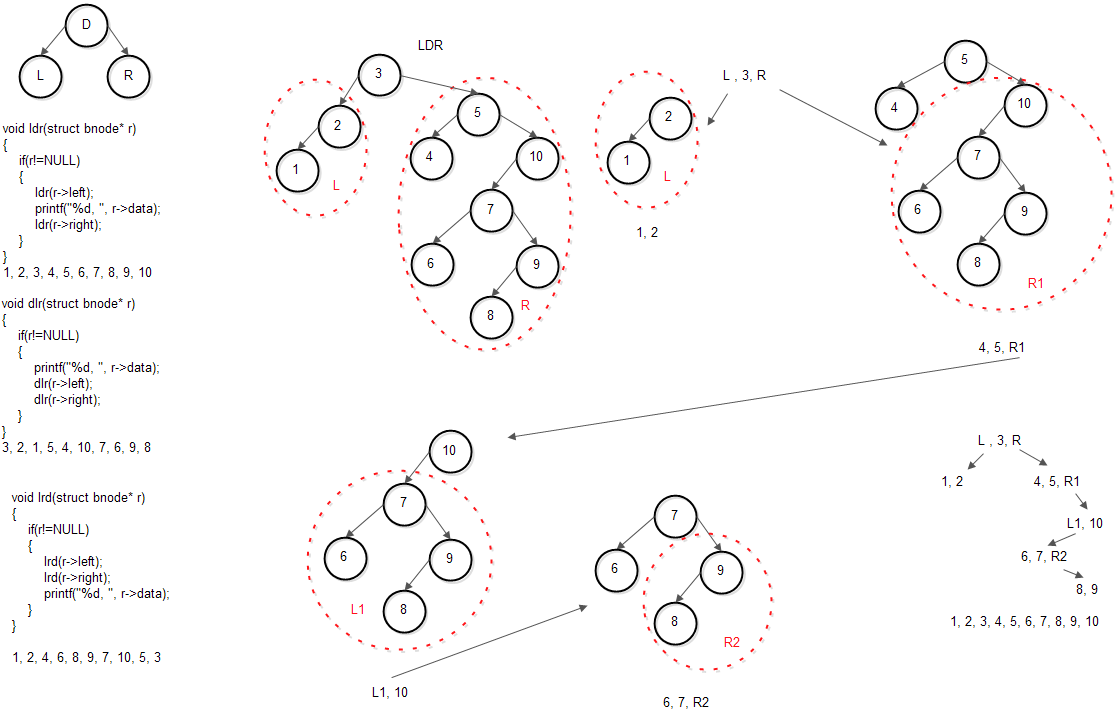
lrd(r->left);

lrd(r->right);

printf("%d, ", r->data);

}

}



parcurgerea LDR (in ordine) a unui abc -> secventa sortata crescator

parcurgerea RDL a unui abc -> secventa sortata descrescator (verificati !)

