21.06.2016

Drimba Alexandru, gr. 213

Documentație proiect SDA

17. TAD Colectie – implementare folosind un arbore binar de cautare

**1.Enunț**

Un anticariat dorește o aplicație prin care sa își gestioneze produsele (cărți, manuscrise, ziare, picturi, muzica si altle). Aplicația va avea posibilitatea de a gestiona produsele (adaugare, stergere) si de a cauta dupa nume.

Specificații:

Tipul de date folosit va fi o colecție implementată folosind un arbore binar de căutare reprezentat inlantuit cu alocare dinamica a memoriei.

Relatia de comparatie folosita pentru adaugarea în arbore va fi bazată pe DENUMIREA produsului.

Un produs este definit prin urmatoarele componente:

* Titlu/denumire,
* Categorie (cărți, manuscrise, ziare, picturi, muzica sau altele),
* Pret,
* Descriere.

**2. TAD – Specificare și interfață**

produs

creeaza(p,nume,categorie,pret,descriere)

pre:

nume: string

categorie: string

pret: real

descriere: string

post:

p: produs (cu campurile date ca preconditii)

se aloca memorie pentru un obiect de tip produs

distruge(p)

pre:

p: produs

post:

se dealoca memoria ocupata de obiect

get\_nume(p,nume)

pre:

p: produs

post:

nume: string – denumirea produsului p

get\_categorie(p,cat)

pre:

p: produs

post:

cat: string – categoria produsului p

get\_pret(p,pret)

pre:

p: produs

post:

pret: real – pretul produsului p

set\_pret(p,pret)

pre:

p: produs

pret: real

post:

p va avea un alt pret

get\_descriere(p,desc)

pre:

p: produs

post:

desc: string – descrierea produsului p

set\_descriere(p,desc)

pre:

p: produs

desc: string – descrierea produsului p

post:

p va avea o noua descriere

nod\_abc

creeaza(nod,val)

pre:

val: TElem – informatia utila din nod

post:

nod: nod\_abc

se creeaza un nod cu informatia utila val

distruge(nod)

pre:

nod: nod\_abc

post:

se dealoca memoria ocupata de nod SI DE TOTI FII SAI

get\_valoare(nod,val)

pre:

nod: nod\_abc

post:

val: TElem – info utila din nod

get\_stanga(nod,fs)

pre:

nod: nod\_abc

post:

fs: nod\_abc – fiul stang

get\_dreapta(nod,fd)

pre:

nod: nod\_abc

post:

fd: nod\_abc – fiul drept

set\_stanga(nod,st)

pre:

nod, st: nod\_abc

post:

nod va avea ca fiu stang pe st

set\_dreapta(nod,dr)

pre:

nod, dr: nod\_abc

post:

nod va avea ca fiu drept pe dr

iterator\_ABC\_RSD / iterator\_ABC\_SRD / iterator\_ABC\_SDR

creeaza(it,arb)

pre:

arb: ABC

post:

it: iterator RSD/SRD/SDR in arb

urmator(it)

pre:

it: iterator RSD/SRD/SDR

post:

muta pozitia pe elementul urmator

valid(it)

pre:

it: iterator RSD/SRD/SDR

post:

returneaza true daca it arata catre un element valid, false altfel

elem(it,e)

pre:

it: iterator RSD/SRD/SDR

post:

e: TElem – elementul curent referit de iterator

distruge(it)

pre:

it: iterator RSD/SRD/SDR

post:

se dealoca memoria ocupata de iterator

colecție\_ABC:

creează(rel,arb)

pre:

rel : Relatie

post:

arb: ABC gol cu relatia de ordine rel

adauga(arb,e)

pre:

arb: ABC

e: TElem

post:

se adauga e in arb

sterge(arb,e)

pre:

arb: ABC

e: TElem

post:

se sterge e din arb

cauta(arb,e,poz)

pre:

arb: ABC

e: TElem

post:

poz: ↑nod\_abc – nodul in care se afla elementul cautat

distruge(arb)

pre:

arb: ABC

post:

se dealoca memoria ocupata de arbore

iterator(arb,it)

pre:

arb: ABC

post:

it: iterator RSD/SRD/SDR

**3. Reprezentare**

Colectia va fi reprezentata folosind un arbore binar de cautare. Elementele din ABC se nevoiesc a fi comparabile.

produs

denumire: string

categorie: string

pret: real pozitiv

descriere: string

nod

st,dr: ↑nod\_abc

info: TElement

aparitii: intreg

iterator RSD/SRD/SDR

↑nod\_abc curent

↑ABC arb

coada c

stiva s

ABC:

↑nod varf

R – relație

**Operatii**

Functia creeaza (val)

aloca(nod)

[nod]. info ← val

[nod].st ← NIL

[nod].dr ← NIL

creeazaNod ← p

SfFunctia

subalgoritm adauga(arb,el)

varf←arb.varf

daca(varf=NIL)

varf←creeaza(el)

STOP

sf\_daca

c,prec,p: nod\_abc

c←varf

prec←NIL

p←cauta(arb,el);

daca(p!=NIL)

[p].aparitii++;

STOP

sf\_daca

cat\_timp(c!=NIL)

prec←c

daca(el<=[c].get\_data())

c←[c].get\_st()

altfel

c←[c].get\_dr()

sf\_daca

daca(el<=[prec].get\_data())

[prec].set\_st(creeaza(el));

altfel

[prec].set\_dr(creeaza(el));

sf\_daca

sf\_subalgoritm

functia cauta(arb,el)

p: ↑nod\_abc ← arb.varf;

cat\_timp(p!=NIL)

daca([p].get\_data()==el)

cauta←p;

daca(el<=[p].get\_data())

p← [p].get\_st();

altfel

p← [p].get\_dr();

sf\_daca

sf\_cat\_timp

cauta←NIL

sf\_functia

functia maxim(arb,start)

pre: start : ↑nod\_abc

p:nod\_abc←start

daca(p==NIL)

maxim←NIL

sf\_daca

cat\_timp([p].get\_dr()!=NIL)

p← [p].get\_dr();

sf\_cat\_timp

maxim←p;

sf\_functia

functia get\_parent(arb, n)

pre: n: ↑nod\_abc

daca(n==varf)

get\_parent← NIL;

sf\_daca

daca(n==NIL)

get\_parent← NIL;

sf\_daca

el:TElem← [n].get\_data()

p:↑nod\_abc=arb.varf;

cat\_timp((p!=NIL))

daca([p].get\_st()==n)

get\_parent← p;

sf\_daca

daca([p].get\_dr()==n)

get\_parent← p;

sf\_daca

daca(el<=[p].get\_data())

p← [p].get\_st();

altfel

p← [p].get\_dr();

sf\_daca

sf\_cat\_timp

get\_parent← NIL

sf\_functia

subalgoritm sterge(arb,el)

p: ↑nod\_abc←cauta(arb,el);

daca(p==NIL)

STOP

daca([p].aparitii>1)

[p].aparitii--;

STOP

sf\_daca

daca([p].get\_dr()==NIL)

daca(get\_parent(p)==NIL)

varf←[p].get\_st();

altfel

daca(el<=[get\_parent(p)].get\_data())

[get\_parent(p)].set\_st([p].get\_st());

altfel

[get\_parent(p)].set\_dr([p].get\_st());

sf\_daca

sf\_daca

[p].set\_st(NIL);

[p].set\_dr(NIL);

dealoca (p);

STOP

sf\_daca

daca([p].get\_st()==NIL)

daca(get\_parent(p)==NIL)

varf← [p].get\_dr();

altfel

daca(el<=[get\_parent(p)].get\_data())

[get\_parent(p)].set\_st([p].get\_st());

altfel

[get\_parent(p)].set\_dr([p].get\_st());

sf\_daca

sf\_daca

[p].set\_st(NIL);

[p].set\_dr(NIL);

dealoca (p);

STOP

sf\_daca

pma,ma:↑nod\_abc;

ma←maxim([p].get\_st());

pma←get\_parent(ma);

daca(p==pma)

[p].set\_st([[p].get\_st()]get\_st())

[p].set\_data([ma].get\_data())

[p].aparitii← [ma].aparitii

[ma].set\_st(NIL)

[ma].set\_dr(NIL)

dealoca (ma)

altfel

[pma].set\_dr(NIL)

[p].set\_data([ma].get\_data())

[p].aparitii← [ma].aparitii

[ma].set\_st(NIL)

[ma].set\_dr(NIL)

dealoca (ma)

sf\_daca

sf\_subalgoritm

**4. Complexități**

Presupunem n=nr. de elemente din arbore

1. adăgare:

* caz favorabil(elementul adaugat este unicul la stanga/dreapta varfului arborelui, cealalta ramura continand tot restul elementelor): Ө(1)
* caz mediu : O(log2n)
* caz defavorabil (arborele este degenerat, elementul se adauga dupa cel mai de jos nod): O(n)

1. stergere

* caz favorabil (elementul eliminat este unicul la stanga/dreapta varfului arborelui): Ө(1)
* caz mediu : Ө (log2n)
* caz defavorabil (arborele este degenerat si se elimina un nod frunza a arborelui) O(n)

1. cautare

* caz favorabil(elementul cautat este unicul la stanga/dreapta varfului arborelui): Ө(1)
* caz mediu : Ө (log2n)
* caz defavorabil (arborele este degenerat, elementul se afla la cea mai mare adancime): O(n)

1. cautare\_parinte
   * caz favorabil(se cauta parintele radacinii sau a celor doi fii ai sai): Ө(1)
   * caz mediu : Ө (log2n)
   * caz defavorabil (arborele este degenerat, elementul al carui parinte se cauta se afla la cea mai mare adancime): O(n)
2. maxim/minim
   * caz mediu : Ө (log2n)

**Deductia complexitatii pentru operatia de ADAUGARE:**

Consideram varianta recursiva a adaugarii:

adaugare (elem,[rad].get\_st()) daca elem<=[rad].data() si [rad].get\_st()!=NIL

adaugare (elem,rad) = [rad].set\_st(creeaza\_nod(elem)) daca elem<=[rad].data() si [rad].get\_st()=NIL

adaugare (elem,[rad].get\_dr()) daca elem>[rad].data() si [rad].get\_dr()!=NIL

[rad].set\_dr(creeaza\_nod(elem)) daca elem>[rad].data() si [rad].get\_dr()=NIL

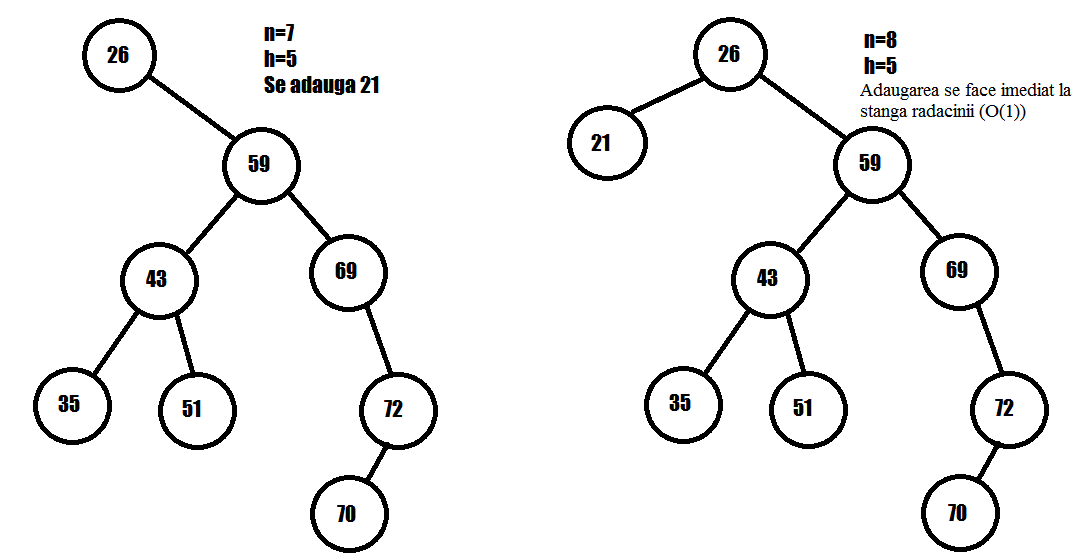
n=numarul de noduri din arbore

h=inaltimea arborelui

1. caz favorabil

Consideram un arbore in care toate elementele din el (cu exceptia radacinii) sunt fie mai mici sau mai mari ca radacina. Astfel, subarborele stang/drept al radacinii este vid, aduagarea in acel subarbore vid facandu-se in O(1).

T(n)=1 deoarce s-a gasit pozitia pe care se va adauga chiar ca fiu al radacinii.



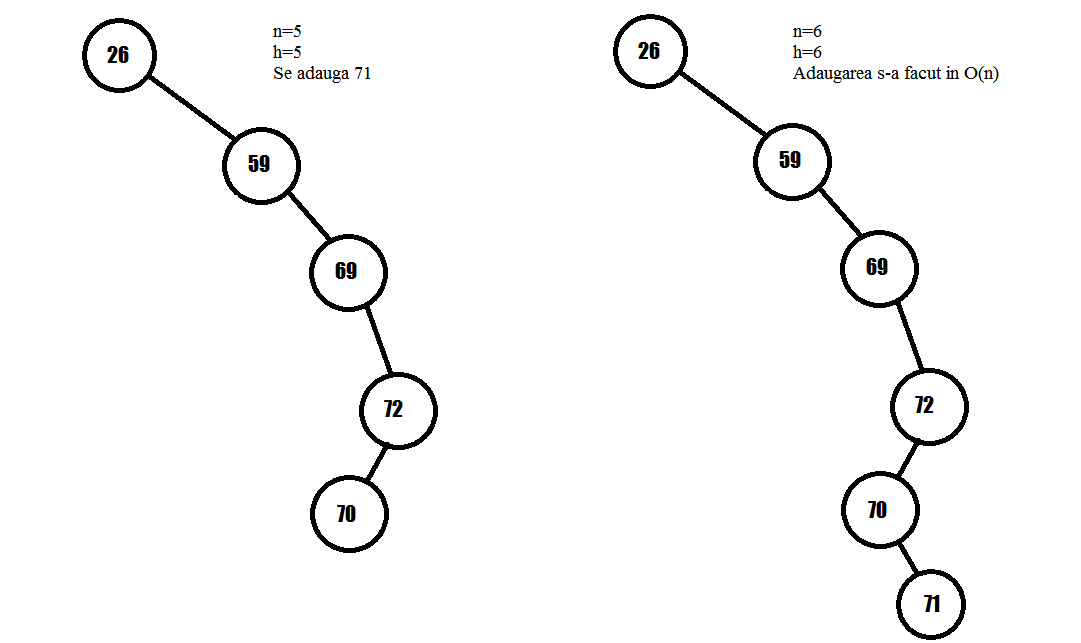
1. caz defavorabil

Pentru cazul defavorabil se considera adaugarea intr-un arbore degenerat, sub forma de lant. Daca adaugarea trebuie sa se faca la sfarsitul lantului, complexitatea devine O(n)

T(n)=T(n-1)+1

…

T(0)=1 =>T(n)=n ∈O(n)



3.caz mediu

Pentru cazul mediu consideram un arbore echilibrat.

T(n)=T(n/2)+1

…

T(1)=1

notam k=log2n

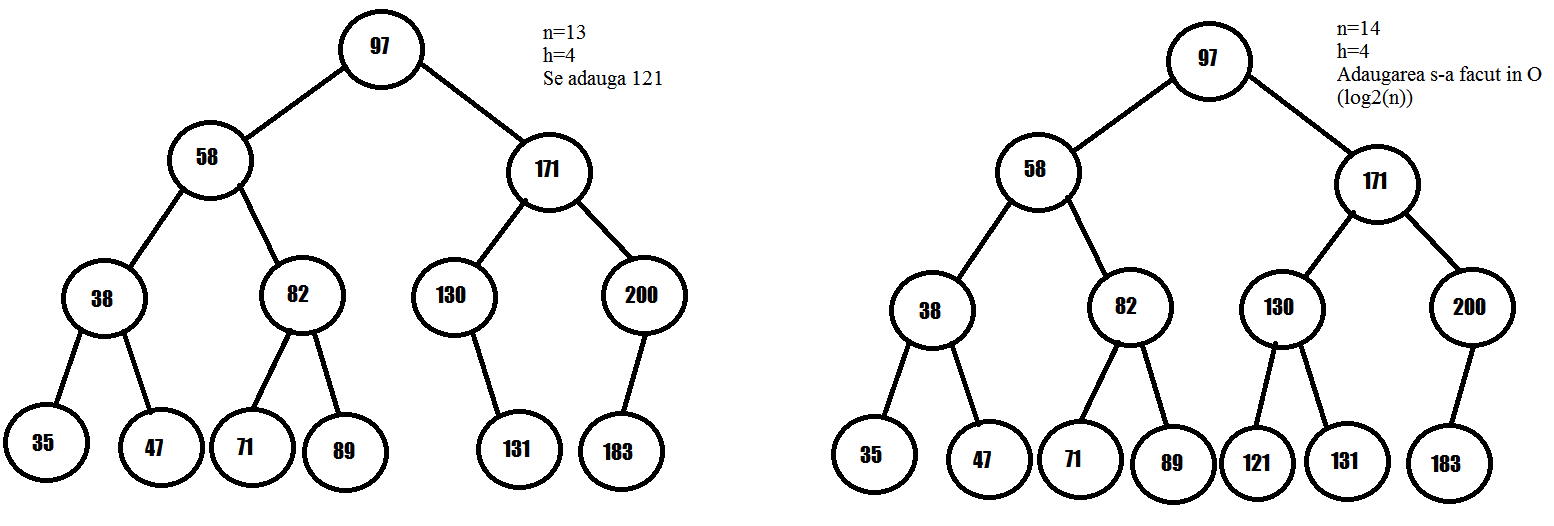
T(2k)=T(2k-1)+1

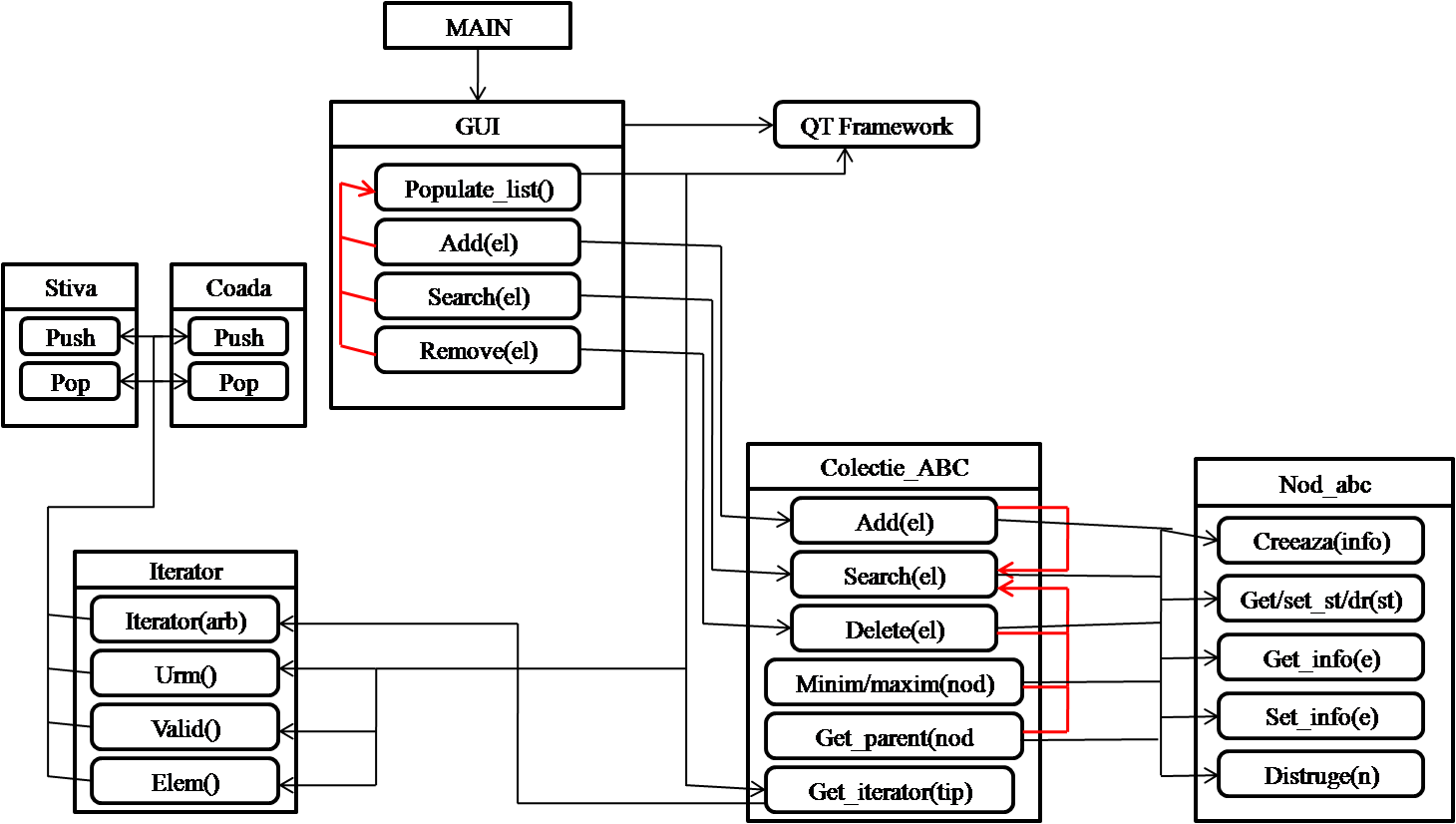
…

T(1)=1

=> T(k)=k

=>T(n)= log2n



**5. Diagrama de apeluri**

**6.Proiectarea aplicatiei**

GUI

subalgoritm populate\_list(mod)

pre:

mod: intreg ∈{1 (RSD), 2(SRD),3(SDR)}

post:

Se va afisa arborele parcurs in modul dat

it: iterator RSD/SRD/SDR ← iterator\_ RSD/SRD/SDR (arb)

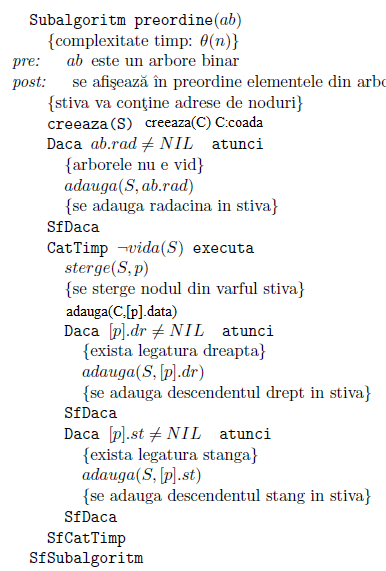
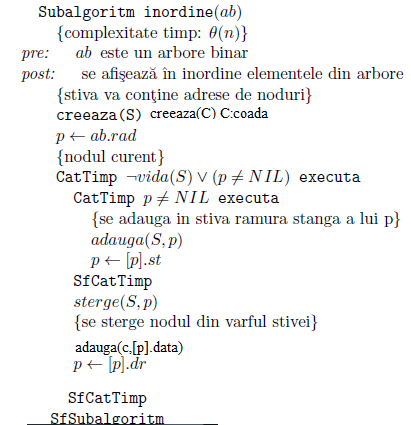
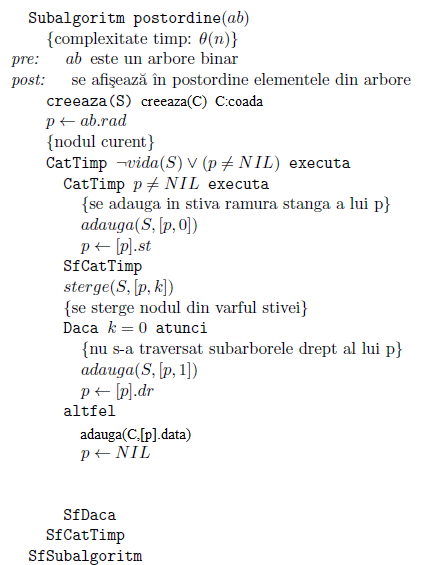
cat\_timp(valid(it))

afiseaza elem(it)

urmator(it)

sf\_cat\_timp

sf\_subalgoritm

Iteratori

subalgoritm urm(it)

pre:

it:iterator

post:

se trece la urmatorul element in iterator

daca(it.C.valid)

it.C.scoate()

sf\_daca

sf\_subalgoritm

functia valid(it)

valid←C.valid()

sf\_functia

functia elem(it)

elem←it.C.get\_primul()

sf\_functia

**7. Complexitatea operatiilor din aplicatie**

1. iterarea peste arbore: Ө(n)
2. copiere arbore: Ө(n)
3. adaugare in arbore: Ө(1)/ O(n)/O(log2n)
4. cautare in arbore: Ө(1)/ O(n)/O(log2n)
5. stergere din arbore: Ө(1)/ O(n)/O(log2n)
6. cautare parinte nod : Ө(1)/ O(n)/O(log2n)
7. minim/maxim: Ө(1)/ O(n)/O(log2n)
8. distrugere arbore: Ө(n) n-numarul total al descendentilor nodului de sters (se distrug toti descendentii)
9. construire arbore: Ө(1)
10. constructor iterator: Ө(n)
11. urmator iterator: Ө(1)
12. element iterator: Ө(1)
13. valid iterator: Ө(1)
14. constructor nod\_abc: Ө(1)
15. constructor copiere nod\_abc: Ө(n) n-numarul total al descendentilor nodului de copiat
16. get/set nod\_abc: Ө(1)
17. destructor nod\_abc: Ө(n) n-numarul total al descendentilor nodului de sters (se distrug toti descendentii)
18. constructor stiva/coada: Ө(1)
19. adaugare stiva,stergere stiva/coada, get\_first stiva/coada: Ө(1)
20. adaugare coada: Ө(n)
21. destructor coada/stiva: Ө(n)