

Lucrarile de laborator nr. 10 si 11

ARBORI BINARI

Obiective

- Aprofundarea TDA arbore binar
- Utilizarea mai multor TDA pentru realizarea programelor complexe
- Utilizarea interfetei grafice (JavaFX)
- programarea multi-thread

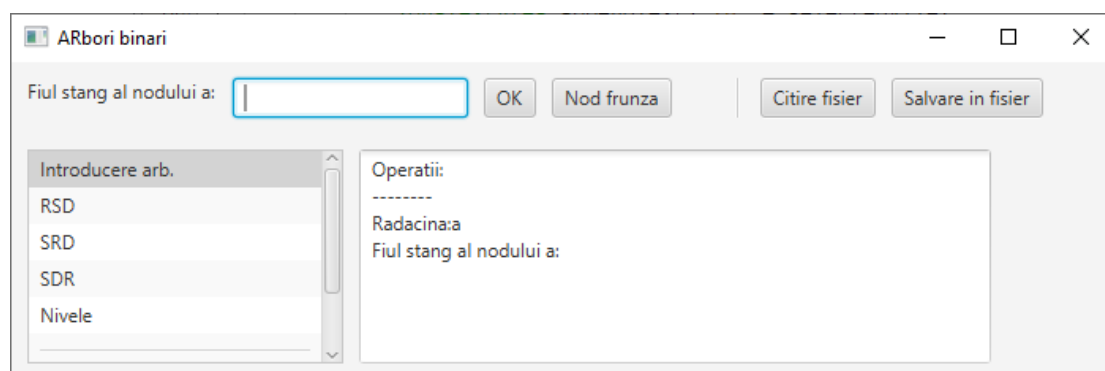
In aceasta lucrare veti utiliza

- interfata grafica construita in lucrarea precedenta pe care o veti dezvolta adaugand noi controale grafice si functionalitati butoanelor si optiunilor din lista de selectie
- clasa ArboreBinar pe care o veti generaliza astfel incat sa accepte citirea datelor din interfata grafica si din fisier
- controlul FileChooser pentru a putea deschide si salva fisiere prin interediul ferestrelor de dialog Open si Save

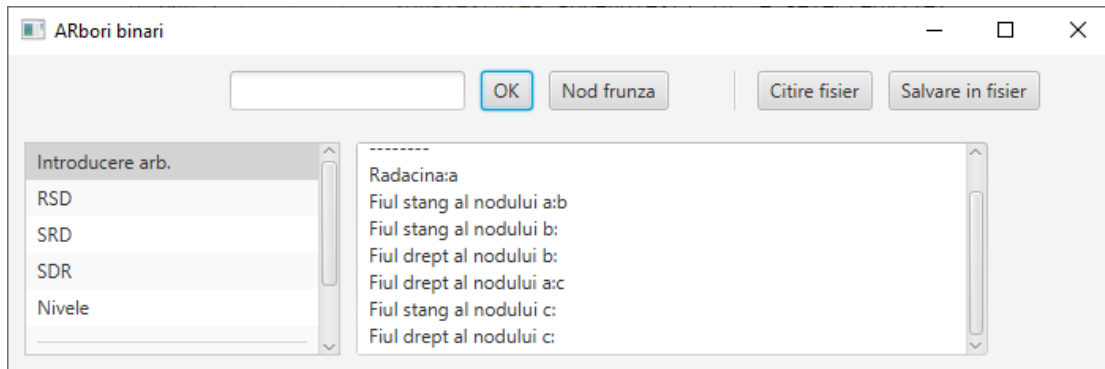


Interfata grafica

Constructia arborelui prin introducerea informatiei din noduri in campul de editare
In imaginea de mai jos este redat momentul dupa introducerea radacinii (eticheta avea valoarea „Radacina:”) si urmeaza introducerea fiului stang al radacinii.



S-au introdus informatiile pentru un arbore cu 3 noduri (radacina „a”, iar „b” si „c” fiul stang, respectiv drept). Daca nu exista fiu se introduce un sir vid (se face click pe butonul OK). Butonul Nod frunza trebuie sa simplifice dialogul. Astfel, la introducerea nodului b daca se apasa pe Nod frunza si nu pe OK utilizatorul nu mai este intrebat care sunt fiii acestuia.



Dupa ce s-au introdus toate nodurile eticheta se stege (dupa fiecare introducere a unui nod in eticheta si in campul de editare se afiseaza sirul vid).

Citirea salvarea din/in fisier

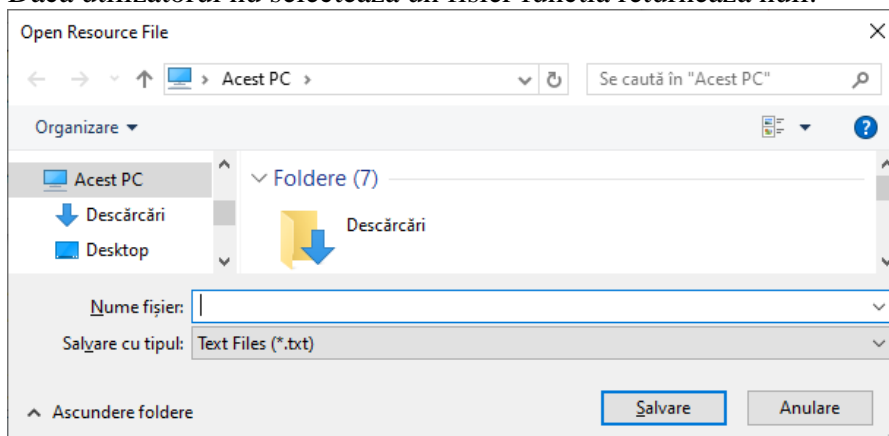
Pentru deschiderea unui fisier folositi FileChooser. Utilizati secventa de la adresa <https://docs.oracle.com/javase/8/javafx/api/javafx/stage/FileChooser.html>

Pentru Save exista functia ***showSaveDialog(mainStage)***

Functia urmatoare deschide un fisier in citire (citire=true) sau in scriere (false).

```
public File deschideFisier(boolean citire) {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Open Resource File");
    fileChooser.getExtensionFilters().addAll(
        new ExtensionFilter("Text Files", "*.txt"),
        new ExtensionFilter("Arbori", "*.arb"),
        new ExtensionFilter("All Files", "*.*"));
    File selectedFile = citire? fileChooser.showOpenDialog(mainStage)
        : fileChooser.showSaveDialog(mainStage);
    return selectedFile;
}
```

Daca utilizatorul nu selecteaza un fisier functia returneaza null.



Trebuie sa continuati programarea citirii/salvării arborelui.

TEMA DE EFECTUAT IN LABORATORUL 10

1. Adaugati la interfata grafica realizata in laboratorul anterior cele 2 butoane „Citire fisier”, „Salvare in fisier”

Punctaj:

- **2p** pentru aspect grafic (butoane si separator vertical plus separator orizontal in ListView),
- **4p** pentru citirea in **TextArea** a unui fisier existent cu mai multe linii (ex. un fisier *.java – atentie trebuie pusa si extensia .java)
- **4p** pentru salvarea unui fisier multilinie din **TextArea**

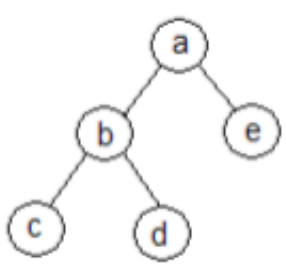
Pentru Laboratorul 11 toti studentii trebuie sa aiba functionale interfata grafica si citirea/salvarea unui fisier!

Tema de efectuat in Laboratorul 11

1. **Construirea unui arbore** pe baza datelor citite dintr-un fisier si efectuarea a cel putin 3 traversari ale sale cu optiunile din **ListView**. **Salvarea unui arbore** intr-un fisier. Aceasta operatie nu inseamna copierea fisierului citit pentru a construi arborele, ci parcurgerea arborelui si salvarea informatiilor in fisier conform formatului din exemplul de mai jos. Punctaj:**5p** daca se efectueaza in lab.10, **2.5p** pentru lab. 11.

Pentru a se putea utiliza metoda **constArbore()** din clasa **ArboreBinari** arborele trebuie memorat in fisier cu informatiile din noduri in inordine (RSD), iar acolo unde nu exista legatura la dreapta sau la stanga se va indica un sir vid.

Exemplu de format de fisier:

Arbore	Continut fisier
	a b c d e

Important. Pentru test se vor utiliza 2 fisiere: cel prezentat mai sus si un fisier care va contine descrierea arborelui reprezentand expresia aritmetica urmatoare: **(a-b)*(a+b)**

Indicatie. Scrieti o clasa **CitireNoduriDinFisier** care sa implementeze **CitireNod** si apelati constructorul **ArboreBinar(CitireNod citnod)** utilizand o instanta a acestei noi clase. Pentru salvarea arborelui adaugati clasei **ArboreBinar** o metoda care sa parcurga arborele binar creat adaugand un terminator de linie pentru a obtine un fisier ca cel din exemplu.

2. Programarea traversarii pe nivele (in latime): **2p**

Indicatie. Algoritmul il gasiti in sectiunea 2. *Algoritmi iterativi de traversare a arborilor binari* din cadrul acestei lucrari de laborator.

3. Inlocuirea cozii utilizata in algoritmul de traversare a arborelui pe nivele cu stiva si identificarea traversarii ce se efectueaza astfel (**0.5p**).

4. Programati evaluarea expresiilor aritmetice reprezentate prin arbori cititi din fisier (**3p** daca expresia contine numai constante si operatori binari, + **2p** daca expresia contine si variabile a caror valoare este citita de la utilizator o singura data).

Indicatie. La sfarsitul acestei lucrari de laborator gasiti o Anexa cu programul Python prezentat la cursul de PCLP II (anul I) de evaluare a expresiilor aritmetice reprezentate sub forma de arbore. Expresiile contineau insa numai constante. Pentru variabile e necesara o structura suplimentara care sa stocheze perechile (*nume_variabila*, *valoare_variabila*).

5. Citirea unui arbore, traversarea si salvarea sa in fisier numai cu operatii din interfata grafica (**10p** daca este realizata in lab. 10, **5p** in lab.11). Dialogul a fost indicat la inceputul acestei lucrari.

Indicatie. Cititi exemplul Producator-Consumator din [pagina cursului](#).

Puteti utiliza urmatoarele 2 clase si interfata CitireNod.

Clasa ArboreBinar

```
package arbori;

public class ArboreBinar {
    private Nod rad;
    CitireNod citnod;
    static String radacina="Radacina";
    private String traversare;

    public ArboreBinar(CitireNod citnod) {
        this.citnod = citnod;
        rad = constArbore( creeazaNod(radacina, null) );
    }

    public ArboreBinar()
    {
        citnod = new CitireNod(); // un obj. cu fctia. citesteInfo default
        rad = constArbore( creeazaNod(radacina, null) );
    }

    public Nod constArbore (Nod x)
    {
        if(((String)x.info()).equals("")) return null;
        x.initStg( constArbore( creeazaNod("stang",x) ));
        x.initDr( constArbore( creeazaNod("drept",x) ));
        return x;
    }

    private Nod creeazaNod(String s,Nod tata)
    {
        if(!s.equals(radacina))
            s="Fiul "+ s + " al nodului "+ tata;

        return new Nod( citnod.citesteInfo(s) );
    }

    public String toString()
    {

```

```

    traversare="Traversare SRD: ";
    SRD(rad);
    return traversare;
}

public void SRD(Nod x)
{
    if(x==null) return;
    SRD(x.stg());
    traversare += x.info() + " ";
    SRD(x.dr());
}
public static void main(String[] args) {
    ArboreBinar a = new ArboreBinar();
    System.out.println(a);
}
}

```

Interfata CitireNod

```

package arbori;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public interface CitireNod {
    BufferedReader r = new BufferedReader( new InputStreamReader(System.in));

    public default String citesteInfo(String txt) {
        String info;
        System.out.print(txt + ":");
        try {
            info= r.readLine().trim();
        } catch(Exception e) {
            info="";
            System.out.println(e);
        }
        return info;
    }
}

```

Clasa Nod

```

package arbori;

import java.util.Objects;
public class Nod {

    private Object info;
    private Nod stg=null, dr=null;
    //

```

```

public Nod(Object info) {
    this.info = info;
}
public void initStg(Nod stg) {
    this.stg = stg;
}
public void initDr(Nod dr) {
    this.dr = dr;
}
public Object info() { return info; }
public Nod stg() { return stg; }
public Nod dr() { return dr; }
public boolean esteFrunza() { return dr==null && stg==null; }

@Override
public int hashCode() {
    int hash = 7;
    hash = 89 * hash + Objects.hashCode(this.info);
    hash = 89 * hash + Objects.hashCode(this.stg);
    hash = 89 * hash + Objects.hashCode(this.dr);
    return hash;
}
public boolean equals (Object x) { return info.equals(x); }
public String toString() { return info.toString(); }
}

```

2. Algoritmi iterativi de traversare a arborilor binari

2.1. Traversarea în lățime a arborilor binari

Se introduce în coadă nodul rădăcină și apoi *cît timp* coada nu este vidă se execută următoarele operații:

- 1) se extrage un nod din coadă;
- 2) se vizitează acel nod;
- 3) se inserează în coadă descendenții săi (dacă îi are).

Inserarea în coadă a descendenților trebuie făcută în ordinea vizitării lor. Deci mai întâi se va insera fiul stîng apoi cel drept. Rezultă procedura *travers_latime* în care *F_ST[nod]* și *F_DR[nod]* desemnează fiul stîng și drept al nodului *nod*. În procedura următoare se creează un obiect *Coadă*, iar operațiile cu acest obiect sunt reprezentate în stil Java,

```

procedura travers_latime(a) este
| Q = new Coadă ()
| Q.add(Rădăcina_arborelui)
|   cît timp NOT Q.empty() execută
| |     nod = Q.front()
| |     Vizitează(nod)
| |     Q.rem()
| |     dacă F_ST[nod] != null atunci Q.add( F_ST [nod] ) □
| |     dacă F_DR[nod] != null atunci Q.add( F_DR [nod] ) □
|   □
sfîrșit

```

2.2. Traversarea în preordine a arborilor binari

Utilizând acesastă procedură ca model și înlocuind structura de coadă cu structura stivă, dar inversînd ordinea introducerii în structura de date a celor doi fii, obținem algoritmul de traversare în pre-ordine (RSD) a unui arbore binar.

```
procedura RSD_iterativ (a) este
| S = new Stivă()
| S.push (Rădăcina_arborelui)
|   cît timp NOT S.empty() execută
| |     nod = S.top()
| |     Vizitează(nod)
| |     S.pop()
| |     dacă F_DR[nod] !=null atunci S.push( F_DR [nod] )□
| |     dacă F_ST[nod] !=null atunci S.push( F_ST [nod] )□
|   □
sfîrșit
```

2.3. Traversarea iterativă în inordine

Ideea procedurii constă în a "coborî" către stînga cît se poate de mult, pînă ce se ajunge la un nod, pe care-l notăm cu x care nu mai are descendenți; în acest timp se trec în stivă toate nodurile parcurse la stînga sa. Se vizitează nodul x, după care, dacă nodul x este frunză, se revine la "tatăl sau" care este preluat din stivă. Dacă nodul x nu este frunză se încearcă parcurgerea subarborelui sau drept. Acest lucru este corect deoarece nodul x neavînd niciun descendent la stînga sa, dar avînd unul la dreapta se poate spune că nodul x constituie Rădăcina unui nou subarbore avînd la rîndul sau un subarbore stîng vid și unul drept nevid.

```
procedura SRD_itterativ(a) este
| nod = *) Rădăcina arborelui
| *)inițializează stiva
|   repetă
| |   cît timp F_ST[nod]=\ 0 execută
| | |   push (nod)
| | |   nod = F_ST [nod]
| | |   □
| |   Vizitează(nod)
| |   cît timp F_DR[nod] = 0 execută
| | |   dacă *)stiva vidă atunci sfîrșit
| | |   nod = top_pop(stiva)
| | |   Vizitează(nod)
| | |   □
| |   nod = F_DR [nod]
| |   pînă_cînd *) la infinit
|   sfîrșit
```

Se observă deci că primul ciclu **cît timp** efectuează parcurgerea către stînga, după care urmează vizitarea nodului fără descendent stîng. Al doilea ciclu are ca efect vizitarea ascendenților acestui nod, cît timp acești ascendenți (inițial chiar nodul vizitat înainte de începerea ciclului) nu au descendenți la dreapta. De îndată ce este găsit un descendent la dreapta, acest ciclu se încheie și se reia ciclul "infinit" repetă. Procedura se încheie atunci cînd stiva este găsită vidă în cadrul vizitării ascendenților.

Algoritmi de traverasare a arborilor binari

Algoritm	RSD	SRD	SDR	lățime
recursiv	da	da	da	nu
cu SD auxiliară	stivă	stivă	stivă	coadă

Tema acasa

1. Proiectați un algoritm iterativ de traversare în postordine a unui arbore binar care sa functioneze pe interfata grafica (veti introduce o optiune noua in ListView) **(5p)**
2. Concepeți o nouă metodă de evaluare a expresiilor reprezentate prin arbori binari utilizând o metodă de traversare și evaluarea expresiilor postfixate **(5p)**
3. Realizați o clasă ArboreBinarComplet care să reprezinte arborele printr-un vector de noduri. Programați cele 4 metode remarcabile de traversare **(5p)**.
4. Programați o metodă de derivare simbolică a expresiilor reprezentate prin arbori. Creați un program demonstrativ care să calculeze valoarea unei expresii funcție de valoarea unei variabile și valoarea derivatei expresiei în funcție de aceeași variabilă **(10p)**.

ANEXA

Evaluarea expresiilor reprezentate sub forma de arbore (Python)

```
# infinit
inf = float('inf')

def este_frunza(nod):
    return nod[1]==None and nod[2]==None

def impartire(x,y):
    if y==0:
        return inf
    return x/y

def calcul(x, op, y):
    operatii = {'+': lambda x,y:x+y,
                '-': lambda x,y:x-y,
                '*': lambda x,y:x*y,
                '/': impartire }
    return operatii[op](float(x),float(y))

def evalArb(nod):
    if este_frunza(nod):
        return float(nod[0])
    return calcul(evalArb(nod[1]), nod[0], evalArb(nod[2]))

if __name__ == '__main__':
    # expr. de test = (5+9)/2
    etest = ['/', [ '+', ['5', None, None], ['9', None, None] ], ['2', None, None] ]
    # expr. cu div. la zero = (10+20) / (30 - 30)
    edivzero = ['/', [ '+', ['10', None, None], ['20', None, None] ],
                [ '-', ['30', None, None], ['30', None, None] ]
    ]
    print (evalArb (etest) )
    print (evalArb (edivzero) )
```