**Universitatea Petrol Gaze Ploiești**

**Facultatea de Litere și Științe**

**Specializarea Informatică, An III**

# Proiect inginerie software

# - Structuri de date –

**Manager de proiect: Coșereanu Emanuel**
**Programator: Crăciun Elena Bianca**

**Programator: Gheorghe Valentin Marian**

**Programator: Vasile Andrei Marian**

**Programator: Coșereanu Emanuel**

# Cuprins

**Lista de figuri**

# 1 Introducere

Ingineria software reprezintă o ramură esențială a informaticii, având ca scop gestionarea întregului proces de proiectare, dezvoltare, implementare și testare a unei aplicații software. Această disciplină îmbină tehnologii moderne și practici consacrate din domenii precum informatica, managementul proiectelor, ingineria sistemelor și designul interfețelor, pentru a obține produse software stabile, eficiente și ușor de utilizat.

Pentru a respecta criteriile și standardele specifice acestui domeniu, se va forma o echipă dedicată, alcătuită din patru persoane competente, motivate și interesate de ingineria software. Scopul acestei echipe este realizarea unui proiect bine definit, structurat logic și construit în jurul unei interfețe intuitive, prietenoase și accesibile utilizatorilor finali.

În cadrul proiectului, se vor utiliza concepte teoretice legate de structuri de date, precum și cunoștințe de programare în limbajul Java.
Pe baza acestor elemente, se va dezvolta un program care va fi planificat și organizat conform modelului cascadă, un model clasic de dezvoltare software ce presupune parcurgerea etapelor în ordine strictă.
Totodată, implementarea va urma paradigma procedurală, concentrându-se pe funcții, pași logici și structuri clare de control.

Echipa responsabilă de proiect va fi formată din patru membri, fiecare având un rol bine stabilit.
Structura echipei include un manager de proiect, responsabil de coordonare, planificare și supravegherea întregului proces, precum și trei programatori care se vor ocupa de implementarea efectivă a funcționalităților, testarea componentelor și optimizarea codului.

# 2 Management-ul proiectului

Proiectul este structurat în 9 capitole, acestea fiind împărțite pe sarcini de lucru și realizate atât individual cât și la întâlnirile de grup.

## 2.1 Planificare proiect

Primul pas în crearea proiectului este planificarea. În acest scop, s-a realizat o întâlnire cu membrii grupului pentru a dezbate tema. Întâlnirea inițială a constat în rezolvarea problemelor manageriale cum ar fi împărțirea sarcinilor pe o anumită perioadă, stabilirea întâlnirilor viitoare, prezentarea ideilor fiecărui membru din grupă, dar și rezolvarea problemelor ce au ținut de teorie și înțelegerea temei. Această primă întâlnire a avut un rol important pentru dezvoltarea proiectului atât pe partea de organizare cât și pe partea de teorie și înțelegere a cerințelor temei.

## 2.2 Timpul de lucru

Proiectul s-a finalizat în 56 zile, timpul de lucru fiind considerat din momentul primirii temelor. Prima întâlnire a avut un rol important în determinarea datei finalizării proiectului deoarece cerințele au fost sparte în subprobleme și împărțite pe intervale de timp fiecărui membru al echipei, reușindu-se astfel rezolvarea temei în timpul stabilit.

## 2.3 Documentarea teoretică

În ceea ce privește documentarea teoriei aflate în spatele cerințelor, echipa a discutat despre fiecare structură de date ce trebuie folosită. Pentru a înțelege mai bine structurile și modul de funcționare al acestora, s-au utilizat informațiile din cartea *„Data Structures & Algorithms in Java”* de Robert Lafore, editura SAMS și suportul teoretic de la cursuri și laboratoare.

## 2.4 Rezolvare cerințe

Această etapă permite fiecărui membru să reia exercițiul primit, să îl înțeleagă mai bine și să înceapă procesul de implementare. După prima întâlnire în care s-au dezbătut exercițiile, scrierea codului a devenit mai simplu de realizat. Problemele primitive au fost rezolvate atât prin muncă individuală cât și prin muncă de echipă. La finalul acestei etape, s-a testat fiecare exercițiu rezolvat pentru a vedea cât de bine funcționează.

## 2.5 Creare interfață

În ceea ce privește crearea interfeței, s-a ales o persoană din grup care să îmbine problemele pentru a crea aplicația finală. De aceea, problemele rezolvate au fost combinate într-o singură aplicație care are o interfață grafică simplă și intuitivă. După acești pași, s-a realizat testarea aplicației finale pentru a se asigura funcționalitatea corectă a acesteia.

## 2.6 Documentație

Ultima etapă și cea mai importantă pentru un programator este documentația. Pentru înțelegerea mai bună a aplicației create, s-a descris pas cu pas procesul de dezvoltare a temei finale. După realizarea unei schițe a documentației, aceasta a fost împărțită în patru părți, câte una pentru fiecare membru al echipei .

La finalul termenului, fiecare parte a documentației a fost preluată și sudată rezultând astfel o documentație finală, realizată în detaliu și ușor de înțeles.

## 2.7 Orar

Pentru o mai bună organizare și înțelegere a sarcinilor fiecărui membru s-a realizat un orar pentru structurarea activităților. Datele scrise îngroșat sunt zilele în care pe lângă finalizarea anumitor task-uri ne-am întâlnit fizic, online sau mix pentru a discuta noi îndatoriri.

| Orar | | |
|---|---|---|
| Activități | Persoane implicate | Timpul de desfășurare |
| 1.Planificare proiect | 1. Crăciun Bianca<br>2. Coșereanu Emanuel<br>3. Gheorghe Valentin<br>4. Vasile Andrei | **19 noiembrie 2025(13:30-15:30)** |
| 2.Rezolvare cerințe individual și în grup | | **19 noiembrie-17 decembrie** |
| 2.1 Problema 54 | Crăciun Bianca | 26 noiembrie-11 decembrie |
| 2.1.1 Structurarea cerințelor temei | Crăciun Bianca | 26 noiembrie |
| 2.1.2 Crearea interfeței | Crăciun Bianca | 26 noiembrie |
| 2.1.3 Implementarea arborelui și a cozii | Crăciun Bianca | 9 decembrie |
| 2.1.4 Implementarea reuniunii | Crăciun Bianca,Cosereanu Emanuel | 11 decembrie |
| 2.1.5 Implementarea intersecției | Crăciun Bianca | 12 decembrie |
| 2.2 Problema 70 | Coșereanu Emanuel | 5 decembrie-15 decembrie |
| 2.2.1 Structurarea cerințelor și implementare cod | Coșereanu Emanuel | 5 decembrie |
| 2.2.2 Crearea interfeței | Coșereanu Emanuel | 6 decembrie |
| 2.2.3 Îmbinare a 2 probleme | Coșereanu Emanuel,Vasile Andrei | 8 decembrie |
| 2.3 Problema 51 | Gheorghe Valentin | 9 decembrie-12 decembrie |
| 2.3.1 Realizarea schiței problemei | Gheorghe Valentin | 9 decembrie |
| 2.3.2 Implementare coadă principală și coadă secundară | Gheorghe Valentin | 11 decembrie |
| 2.3.3 Creare interfață | Gheorghe Valentin | 11 decembrie |
| 2.3.4 Testare proiect | Gheorghe Valentin | 12 decembrie |
| 2.4 Problema 59 | Vasile Andrei | 30 noiembrie-8 decembrie |
| 2.4.1 Proiectarea și implementarea claselor | Vasile Andrei | 30 noiembrie |
| 2.4.2 Rafinare cod,implementare interfață grafică și testare | Vasile Andrei | 5 decembrie |
| 2.4.3 Implementare intrerfața și testare | Vasile Andrei | 8 decembrie |
| 3. Creare documentație | 1. Crăciun Bianca<br>2. Coșereanu Emanuel<br>3. Gheorghe Valentin<br>4. Vasile Andrei | **5 ianuarie-14 ianuarie** |
| 3.1.1 Introducere , management și motivația alegerii tipului echipei | Crăciun Elena Bianca | 9 ianuarie |
| 3.1.2 Realizarea orarului | Crăciun Elena Bianca | 13 ianuarie |
| 3.2.1 Alocare resurse | Coșereanu Emanuel | 12 ianuarie |
| 3.2.2 Specializarea formală a cerințelor | Coșereanu Emanuel | 12 ianuarie |
| 3.2.3 Îmbinarea primelor trei seturi de cerințe | Coșereanu Emanuel | 13 ianuarie |
| 3.3.1 Modelul sistemului | Gheorghe Valentin | 8 ianuarie |
| 3.3.2 Descriere soluții(proiectare,implementare,testare) | Gheorghe Valentin | 9 ianuarie |
| 3.4.1 Înțelegerea cerințelor | Vasile Andrei | 12 ianuarie |
| 3.4.2 Redactarea documentației utilizator și a celei de sistem | Vasile Andrei | 12 ianuarie |
| 3.4.3 Finalizarea cerințelor de documentat | Vasile Andrei | 13 ianuarie |
| 3.Finalizarea documentației | Crăciun Elena Bianca<br>Coșereanu Emanuel<br>Gheorghe Valentin<br>Vasile Andrei | 14 ianuarie |

***Fig. 1*** *- Orarul*

## 2.8 Motivația alegerii tipului de echipă

Echipa este de tipul democratic. Această alege a fost făcută de comun acord la prima întâlnire a echipei. De ce echipă democratică?

Încă de la stabilirea membrilor echipei, s-a folosit de ideea că fiecare membru este liber să își spună opinia, să se simtă auzit, ascultat și valorificat. De aceea, tipul democratic a fost ales pentru echipă, permițând astfel fiecărei persoane să își prezinte liber ideile, să se consulte cu ceilalți membrii și să se creeze o atmosferă plăcută în mediul de lucru.

# 3 Alocare resurse

În cadrul proiectului s-a urmărit ca fiecare membru să se ocupe de părțile la care se simte cel mai profesionist și cât mai în largul său. Aceste aspecte vor fii prezentate mai în detaliu în rândurile ce urmează.

## 3.1 Sarcini

Fiecare membru al echipei și-a rezolvat problema separat, dar nu înainte de a ne aduna toți și a clarifica anumite neînțelegeri asupra conceptelor. Din acest motiv, pe data de 19 Noiembrie 2025 ne-am întâlnit pentru a discuta cum se implementează fiecare structură de date pe care o avem în proiect. Pentru că unele probleme necesitau aceleași structuri, bineînțeles că de-a lungul implementării aplicațiilor ne-am ajutat unii pe alții cu mici explicații atunci când a fost nevoie.

În ceea ce privește partea scrisă, am împărțit cerințele de pe site-ul unde.ro în 4 seturi: primele 3 un set, cerințele 4 și 5 al doilea set, 6 și 7 al treilea set, iar 8 și 9 al patrulea set. Deoarece suntem o echipă democratică seturile s-au ales voluntar, de fiecare membru al echipei, dând prioritate celor în cauză.

După alegerea seturilor, sarcinile au fost împărțite în felul următor:

- Setul 1 – Bianca
- Setul 2 – Emanuel
- Setul 3 – Valentin
- Setul 4 – Andrei

## 3.2 Specialiști

Fiecare a fost specialist pe partea lui dar au existat și anumite îndatoriri de care s-au ocupat anumiți membri ai echipei după cum urmează:

- Bianca – a ținut evidența întâlnirilor și a problemelor discutate.
- Andrei – implementarea citirii arborelui folosind interfața grafică.

## 3.3 Roluri

Așa cum s-a menționat mai sus echipa a fost democratică motiv pentru care s-a ales prin vot managerul de proiect și anume Coșereanu Emanuel, fiecare alt membru al echipei ocupând rolul de programator.

## 3.4 Date limită

Datele limită au fost fixate astfel încât să avantajeze pe toată lumea și să permită dezvoltarea proiectului fără a crea un mediu stresant. Cele două mari deadline-uri sunt după cum urmează:

- 17 decembrie 2026 (terminarea tuturor problemelor și combinarea lor într-o aplicație finală pentru a o prezenta la laborator).
- 5 ianuarie 2026 (împărțirea responsabilităților pentru realizarea documentației)
- 14 ianuarie 2026 (finalizarea subcapitolelor documentației pentru a le pune cap la cap, respectiv a documentației).

De menționat: În perioada 18 decembrie – 4 ianuarie nu s-a realizat nicio modificare l-a proiect (vacanță).

# 4 Specificarea formală a cerințelor și a sistemul software

Scopul proiectului este de uni într-o singură aplicație mai multe probleme de structuri de date și de permite utilizatorilor navigarea fiabilă între aceste exerciții. Din motivele prezentate mai sus  s-au propus următoarele cerințe.

## 4.1 Crearea unei interfețe grafice simple și intuitive

Interfața trebuie să:

- Conțină pe pagina principală 4 butoane, unul pentru fiecare membru al echipei și problema lui.
- Pe paginile secundare, ale fiecărui membru, trebuie să conțină un buton „Înapoi” pentru a permite utilizatorilor să revină la meniul principal fără a închide aplicația.
- În cadrul fiecărei aplicații mesajul de pe butoane și textul afișat în oricare casetă de text trebuie să fie specifice, astfel încât utilizatorului să-i fie ușor de înțeles ce face fiecare buton și ce se afișează în fiecare casetă. Exemplu pentru buton (Grupează nodurile după numărul de fii), respectiv pentru caseta text (Coada principală: 1 -> 2 -> null)

Deoarece s-a lucrat individual la probleme, cu toate că s-au folosit aceleași structuri pentru rezolvarea unora dintre ele s-a propus ca în denumirea fiecărei clase să se pună la final inițiala prenumelui studentului a cărui clasă e, pentru a evita diverse erori ce ar putea apărea din cauza faptului că fiecare își poate rescrie codul după bunul plac. Exemplu: LinkListE.java. Deși această abordare nu este deloc una dintre cele mai bune, mai ales în cadrul aplicațiilor mari, noi am folosit-o în primul rând pentru faptul că mărimea aplicației este una acceptabilă, respectiv pentru a permite libertate de mișcare fiecărui membru.

# 5 Modelul sistemului

Această secțiune descrie fluxul informațional și modul de prelucrare a datelor pentru fiecare dintre cele patru module ale aplicației. Sistemul este proiectat modular, fiecare problemă fiind tratată independent prin intermediul unei interfețe grafice dedicate care comunică cu structurile de date din backend.

## 5.1 Modulul Valentin: Coada de cozi dintr-o matrice (Problema 51)

**Enunț:** Se dă o matrice m x n oarecare cu elemente numere întregi. Se cere să se construiască o coadă de cozi, astfel încât fiecare celulă din coada principală să conțină un pointer către o coadă ce reprezintă o linie din matrice.

- **Date de intrare:**
  - ✓ Numărul de rânduri și coloane (n, m), preluate din câmpurile inputRanduri și inputColoane.
  - ✓ Valorile elementelor matricei, generate intern (secvențial de la 1 la n*m) pentru a popula structura.
- **Mod de citire a datelor:**
  - ✓ Utilizatorul interacționează cu fereastra GUI_V. La apăsarea butonului de generare, dimensiunile sunt validate și transmise constructorului clasei MatriceV.
- **Date de ieșire:**
  - ✓ Vizualizarea cozii de cozi în componenta jTextArea2. Formatul de afișare evidențiază legătura dintre nodul principal și sub-coada asociată (ex: Valoare Nod -> [Sub-coada...]).
- **Procesare și Control:**
  - ✓ Clasa MatriceV gestionează logica. Se instanțiază o CoadaV principală.
  - ✓ Matricea este parcursă linie cu linie. Pentru fiecare linie, se creează o sub-coadă (tot de tip CoadaV), populată cu elementele liniei respective.
  - ✓ Nodurile cozii principale (NodeV) sunt complexe, conținând o referință către aceste sub-cozi (NodeV.subCoada).

## 5.2 Modulul Bianca: Reuniunea și Intersecția (Problema 54)

**Enunț:** Fie două mulțimi oarecare de numere întregi, una reprezentată cu un arbore binar și una cu o coadă. Se cere să se calculeze reuniunea și intersecția lor și să se reprezinte rezultatele cu un arbore (pentru reuniune), respectiv cu o coadă (pentru intersecție).

- **Date de intrare:**
  - ✓ Nodurile arborelui binar (introduse recursiv prin dialoguri JOptionPane via metoda createRec).
  - ✓ Elementele cozii (introduse numeric).
- **Mod de citire a datelor:**
  - ✓ Arborele este construit interactiv: utilizatorul confirmă existența fiilor și introduce cheile.
  - ✓ Coada este populată element cu element prin interfața GUI_B.
- **Date de ieșire:**
  - ✓ Reuniunea: Un arbore extins care conține elementele originale plus elementele unice din coadă, afișat în zona de text reuniune.
  - ✓ Intersecția: O listă/coadă cu elementele comune, afișată în jTextArea2.
- **Procesare și Control:**
  - ✓ Intersecția: Se utilizează metoda intersectieCuCoada din TreeB. Aceasta parcurge coada și verifică existența fiecărui element în arbore folosind o traversare recursivă (existsInTree).
  - ✓ Reuniunea: Se parcurge coada, iar elementele care nu există în arbore sunt inserate păstrând proprietățile structurale.

## 5.3 Modulul Andrei: Clasificarea nodurilor (Problema 59)

**Enunț:** Să se determine și să se stocheze într-o structură de date adecvată nodurile cu 0, 1, respectiv 2 fii dintr-un arbore binar oarecare.

- **Date de intrare:**
  - ✓ Structura arborelui definită de utilizator prin GUI_A.
- **Mod de citire a datelor:**
  - ✓ Similar cu modulul Bianca, construcția arborelui se face prin metoda createRec din TreeA, utilizând ferestre de dialog pentru input.
- **Date de ieșire:**
  - ✓ Trei liste distincte afișate în interfață: Lista nodurilor frunză (0 fii), Lista nodurilor cu un singur fiu, Lista nodurilor complete (2 fii).
- **Procesare și Control:**
  - ✓ Algoritmul central este buildHashTableByChildren. Acesta inițializează un vector de liste (LinkListA[] table) de dimensiune 3.

- ✓ Se parcurge arborele (RSD - Rădăcină-Stânga-Dreapta). Pentru fiecare nod, se calculează numărul de fii și se inserează valoarea nodului în lista corespunzătoare din vector (table[nr_fii].insertLast).

## 5.4 Modulul Emanuel: Căile de la Rădăcină la Frunze (Problema 70)

**Enunț:** Se dă un arbore binar oarecare. Se cere să se construiască un vector de liste asociat care să conțină toate căile din acest arbore, de la rădăcină către frunze.

- ➢ **Date de intrare:**
  - ✓ Topologia arborelui construit în GUI_E.
- ➢ **Mod de citire a datelor:**
  - ✓ Construcție interactivă a arborelui prin dialoguri succesive.
- ➢ **Date de ieșire:**
  - ✓ Un set de liste, fiecare reprezentând o cale (ex: Calea 1: 10 -> 5 -> 2).
- ➢ **Procesare și Control:**
  - ✓ Clasa TreeE utilizează un vector temporar path[] pentru a memora nodurile vizitate curent în recursivitate.
  - ✓ Metoda buildRec verifică dacă nodul curent este frunză. În caz afirmativ, conținutul vectorului path este copiat într-o nouă listă (LinkListE) care este adăugată în tabela de rezultate.

# 6 Soluții de proiectare, implementare și testare

Această secțiune detaliază arhitectura software a aplicației, descriind structura claselor, metodele implementate și algoritmii utilizați, precum și scenariile de testare folosite pentru validarea funcționalității.

## 6.1 Descrierea structurii aplicației (Clase și Metode)

Aplicația este organizată modular, respectând principiile programării orientate pe obiecte (POO). Fiecare problemă este izolată în propriul set de clase (Model-View-Controller simplificat).

### 6.1.1 Pachetul Principal și Navigare

**Clasa Main_GUI** Aceasta este clasa de intrare în aplicație, moștenind javax.swing.JFrame.

- **Metode:**
  - ✓ main(String args[]): Metoda principală care lansează firul de execuție al interfeței grafice.
  - ✓ jButton1ActionPerformed, jButton2ActionPerformed, etc.: Metode de tip "Listener" care instanțiază și deschid ferestrele specifice fiecărui modul (Andrei, Emanuel, Valentin, Bianca). (**Fig. 2**, **Fig. 3**)

```
104     private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
107                 new GUI_B().setVisible(true);
108             }
109         });
110
111         this.setVisible(false);
112     }
```

***Fig. 2** - jButton4ActionPerformed*

```
40          jButton1.setText("Andrei");
41          jButton1.addActionListener(this::jButton1ActionPerformed);
42
43          jButton2.setText("Emanuel");
44          jButton2.addActionListener(this::jButton2ActionPerformed);
45
46          jButton3.setText("Valentin");
47          jButton3.addActionListener(this::jButton3ActionPerformed);
48
49          jButton4.setText("Bianca");
50          jButton4.addActionListener(this::jButton4ActionPerformed);
```

***Fig. 3** - Butoanele pentru aplicația fiecărui membru*

### 6.1.2 Modulul 1: Coada de cozi (Valentin)

**Clasa MatriceV** Gestionează logica de conversie a matricei în structură de tip coadă.

- **Metode:** matriceCozi() este algoritmul principal. (**Fig. 4**)

```
39      public void matriceCozi(){
40          coadaPrincipala= new CoadaV();
41
42          for(int i=1;i<=n;i++){
43             coadaPrincipala.adaugare(i);
44          }
45           NodeV curr = coadaPrincipala.getHead();
46          for(int i=0; i<n;i++){
47              for(int j=0;j<m;j++){
48                  curr.subCoada.adaugare(matrix[i][j]);
49              }
50               curr = curr.next;
51          }
52          }
```

***Fig. 4** - Funcția matriceCozi()*

**Clasa NodeV**

- **Descriere:** Nod special care conține referința către o altă coadă. (**Fig. 5**)

```
3   public class NodeV {
4       int data;
5       NodeV next;
6       CoadaV subCoada;
7
8       NodeV(int data) {
9           this.data = data;
10          this.next = null;
11          this.subCoada = new CoadaV();
12      }
13  }
14
```

***Fig. 5** - Clasa NodeV*

### 6.1.3 Modulul 2: Reuniune și Intersecție (Bianca)

**Clasa TreeB** Implementează arborele binar și operațiile pe mulțimi.

- **Metode:** intersectieCuCoada și existsInTree. (**Fig. 6**)

```java
public int[] intersectieCuCoada(LinkQueueB q) {
    initArray(); // resetam vectorul a si index
    LinkB current = q.theList.first;

    while (current != null) {
        int val = (int) current.dData;

        // daca val exista in arbore si nu e deja in vector
        if (existsInTree(root, val) && !isInArray(a, index, val)) {
            a[index++] = val;
        }

        current = current.next;
    }

    return a;
}
```

***Fig. 6** - Funcția intersectieCuCoada()*

**Clasa GUI_B**

- **Metode:** Validarea datelor la introducerea în coadă. (**Fig. 7**)

```java
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String input = JOptionPane.showInputDialog(
            this,
            "Introdu un număr pentru a-l insera în coadă:",
            "Inserare în coadă",
            JOptionPane.PLAIN_MESSAGE);

    if (input != null) { // dacă nu a apăsat Cancel
        try {
            long value = Long.parseLong(input);
            coada.insert(value);

            JOptionPane.showMessageDialog(
                    this,
                    "Element inserat: " + value,
                    "Succes",
                    JOptionPane.INFORMATION_MESSAGE
            );
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(
                    this,
                    "Te rog introduce un număr valid!",
                    "Eroare",
                    JOptionPane.ERROR_MESSAGE
            );
        }
    }
}
```

***Fig. 7** - Validarea datelor la introducerea în coadă*

### 6.1.4 Modulul 3: Clasificarea Nodurilor (Andrei)

**Clasa TreeA** Gestionează arborele binar supus analizei.

- **Metode:** buildRecRSD(NodeA current, LinkListA[] table). (**Fig. 8**)

```
129     private void buildRecRSD(NodeA current, LinkListA[] table){
130         if(current == null) return;
131
132         int children = 0;
133         if(current.leftChild != null) children++;
134         if(current.rightChild != null) children++;
135
136         table[children].insertLast(current.iData);
137
138         buildRecRSD(current.leftChild, table);
139         buildRecRSD(current.rightChild, table);
140
141     }
142 }
```

***Fig. 8** - Funcția buildRecRSD*

### 6.1.5 Modulul 4: Căile Arborelui (Emanuel)

**Clasa TreeE** Identifică toate căile de la rădăcină la frunze.

- **Metode:** buildRec (Algoritm DFS). (Fig 5.1)

```
127 private void buildRec(NodeE current, LinkListE[] table, int[] path, int pathLen){
128     if(current != null)
129     {
130         path[pathLen++] = current.iData;
131         if(current.leftChild == null && current.rightChild == null)
132         {
133             for(int i = 0; i < pathLen; i++)
134                 table[leaf].insertLast(path[i]); //adaugam in tabel nodurile
135             leaf++; //marim numarul de frunze gasite
136         }
137         else
138         {
139             buildRec(current.leftChild, table, path, pathLen);
140             buildRec(current.rightChild, table, path, pathLen);
141         }
142     }
143 }
```

***Fig. 9** - Funcția buildRec()*

## 6.2 Testarea aplicației

Procesul de testare a urmărit validarea funcționalității fiecărui modul component prin metoda „Black Box Testing”. S-au verificat corectitudinea datelor de ieșire în raport cu datele de intrare, gestionarea cazurilor limită (arbori goi, date invalide) și stabilitatea interfeței grafice.

Mai jos sunt prezentate scenariile de testare efectuate pentru cele patru probleme.

### 6.2.1 Testare Modul Valentin (Problema 51 - Cozi din Matrice)

Acest modul a fost testat pentru a verifica dacă structura ierarhică (coadă de cozi) este construită corect pe baza dimensiunilor matricei introduse.

| ID Test | Descriere Scenariu | Date de Intrare | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| **TV-01** | **Construire normală** | Rânduri: 2<br>Coloane: 2<br>Valori generate: 1, 2, 3, 4 | Coada principală are 2 noduri.<br>Nod 1 -> Subcoada: 1 -> 2<br>Nod 2 -> Subcoada: 3 -> 4 | Structura afișată în GUI corespunde descrierii. | **REUȘIT** |
| **TV-02** | **Matrice liniară** | Rânduri: 1<br>Coloane: 3 | Coada principală are 1 singur nod. | Afișare corectă, o singură sub-coadă identificată. | **REUȘIT** |

| ID Test | Descriere Scenariu | Date de Intrare | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| | | | Subcoada conține: 1 -> 2 -> 3 | | |
| **TV-03** | **Validare input numeric** | Rânduri: a<br><br>Coloane: 2 | Aplicația nu trebuie să se blocheze.<br><br>Mesaj de eroare sau ignorare input. | Excepția NumberFormatException este prinsă, aplicația rămâne stabilă. | **REUȘIT** |

### 6.2.2 Testare Modul Bianca (Problema 54 - Reuniune și Intersecție)

Testele au vizat corectitudinea operațiilor pe mulțimi (Arbore vs. Coadă) și afișarea rezultatelor.

| ID Test | Descriere Scenariu | Date de Intrare | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| **TB-01** | **Intersecție cu elemente comune** | Arbore: {10, 5, 15}<br><br>Coadă: {5, 20, 15} | Intersecția: {5, 15} (ordinea poate varia în funcție de parcurgere). | Lista afișată conține exact elementele 5 și 15. | **REUȘIT** |

| ID Test | Descriere Scenariu | Date de Intrare | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| **TB-02** | **Intersecție mulțimi disjuncte** | Arbore: {10}<br><br>Coadă: {20, 30} | Intersecția este vidă (niciun element). | Zona de text pentru intersecție rămâne goală sau afișează mesaj specific. | **REUȘIT** |
| **TB-03** | **Reuniune (Adăugare elemente noi)** | Arbore: {10}<br><br>Coadă: {5} | Arborele rezultat trebuie să conțină {10, 5}. Structura BST păstrată (5 în stânga lui 10). | Afișarea arborelui (inordine) arată: 5 10. | **REUȘIT** |
| **TB-04** | **Reuniune cu duplicate** | Arbore: {10}<br><br>Coadă: {10} | Arborele rămâne neschimbat (mulțimile nu au duplicate). | Arborele conține un singur nod cu valoarea 10. | **REUȘIT** |

### 6.2.3 Testare Modul Andrei (Problema 59 - Clasificare Noduri)

S-a verificat dacă algoritmul de parcurgere (RSD) identifică corect numărul de descendenți pentru fiecare nod.

| ID Test | Descriere Scenariu | Date de Intrare (Structură Arbore) | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| **TA-01** | **Arbore Complet (nivel 1)** | Rădăcina: 10 | **0 fii:** 5, 15 | Listele din interfață sunt populate corect conform așteptărilor. | **REUȘIT** |

| ID Test | Descriere Scenariu | Date de Intrare (Structură Arbore) | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| | | Stânga: 5<br><br>Dreapta: 15 | **1 fiu:** (gol)<br><br>**2 fii:** 10 | | |
| **TA-02** | **Arbore Degenerat (Doar stânga)** | 10 -> 5 -> 2 | **0 fii:** 2<br><br>**1 fiu:** 10, 5<br><br>**2 fii:** (gol) | Clasificare corectă, nodurile intermediare apar la "1 fiu". | **REUȘIT** |
| **TA-03** | **Nod unic** | Rădăcina: 10 (fără fii) | **0 fii:** 10<br><br>**1 fiu:** (gol)<br><br>**2 fii:** (gol) | Rădăcina este identificată corect ca frunză. | **REUȘIT** |

### 6.2.4 Testare Modul Emanuel (Problema 70 - Căi în Arbore)

Testele au urmărit corectitudinea algoritmului DFS în identificarea tuturor traseelor de la rădăcină la frunze.

| ID Test | Descriere Scenariu | Date de Intrare | Rezultat Așteptat | Rezultat Obținut | Status |
|---|---|---|---|---|---|
| **TE-01** | **Două căi distincte** | Rădăcina: 1<br><br>Stânga: 2<br><br>Dreapta: 3 | Calea 1: 1 -> 2<br><br>Calea 2: 1 -> 3 | Sunt afișate ambele căi în JTextArea. | **REUȘIT** |
| **TE-02** | **Căi de lungimi diferite** | Rădăcina 1 are fiu stânga 2.<br><br>Nodul 2 are fiu stânga 3.<br><br>Rădăcina 1 are fiu dreapta 4. | Calea 1: 1 -> 2 -> 3<br><br>Calea 2: 1 -> 4 | Algoritmul parcurge corect adâncimile diferite. | **REUȘIT** |
| **TE-03** | **Arbore vid** | Nu se introduce niciun nod. | Nu se afișează nicio cale. | Aplicația nu generează erori, lista de rezultate este goală. | **REUȘIT** |

### 6.2.5 Concluzii Testare

În urma rulării setului de teste, aplicația a demonstrat stabilitate și corectitudine logică.

1. **Validarea datelor:** Mecanismele de try-catch implementate în interfețele grafice previn erorile de execuție la introducerea datelor non-numerice.
2. **Integritatea datelor:** Structurile de date (liste, cozi, arbori) își păstrează consistența după operații repetate (ex: reuniunea nu dublează elementele, cozile nu pierd legăturile).

3. **Performanță:** Pentru seturile de date testate (dimensiuni uzuale pentru scop didactic), răspunsul interfeței este instantaneu.

# 7 Documentație Utilizator

Prezenta secțiune este destinată utilizatorului final, având scopul de a facilita interacțiunea acestuia cu aplicația, prin descrierea detaliată a fluxului de lucru și a funcționalităților disponibile. Documentația este structurată logic, începând cu pașii de bază pentru pornirea sistemului și avansând către detalii specifice fiecărui modul de calcul. Aplicația se lansează prin executarea clasei principale, *Main_GUI.java*. Din mediul NetBeans, acest lucru se realizează prin deschiderea proiectului și apăsarea butonului „Run” (sau tasta F6). Această acțiune va deschide fereastra principală care permite navigarea către cele 4 module.

## 7.1 Meniul principal

După ce se deschide fereastra principală, utilizatorul poate vedea o mică interfață cu butoane ce conțin numele fiecărui participant la dezvoltarea aplicației (vezi **Fig. 10**).

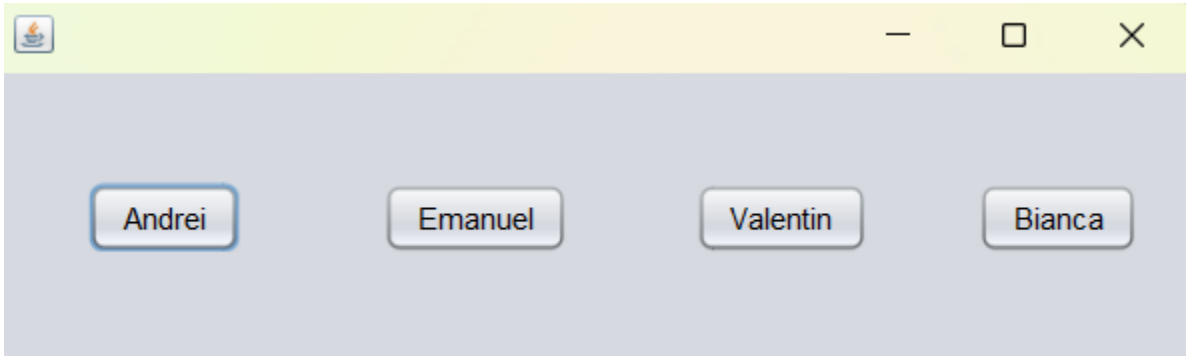


***Fig. 10*** *- Meniul Principal*

## 7.2 Butonul „Andrei”

La apăsarea unuia dintre cele 4 butoane, utilizatorul va fi redirecționat către o fereastră nouă ce conține rezolvarea problemei atribuite persoanei al cărui nume este scris pe buton. Prin urmare, la apăsarea butonului primului buton din stânga („Andrei”), se va deschide interfața din **Fig. 11**.

Enunțul problemei rezolvate aici este următorul: „Să se determine și să se stocheze într-o structură de date adecvată nodurile cu 0, 1, respectiv 2 fii dintr-un arbore binar oarecare.”.

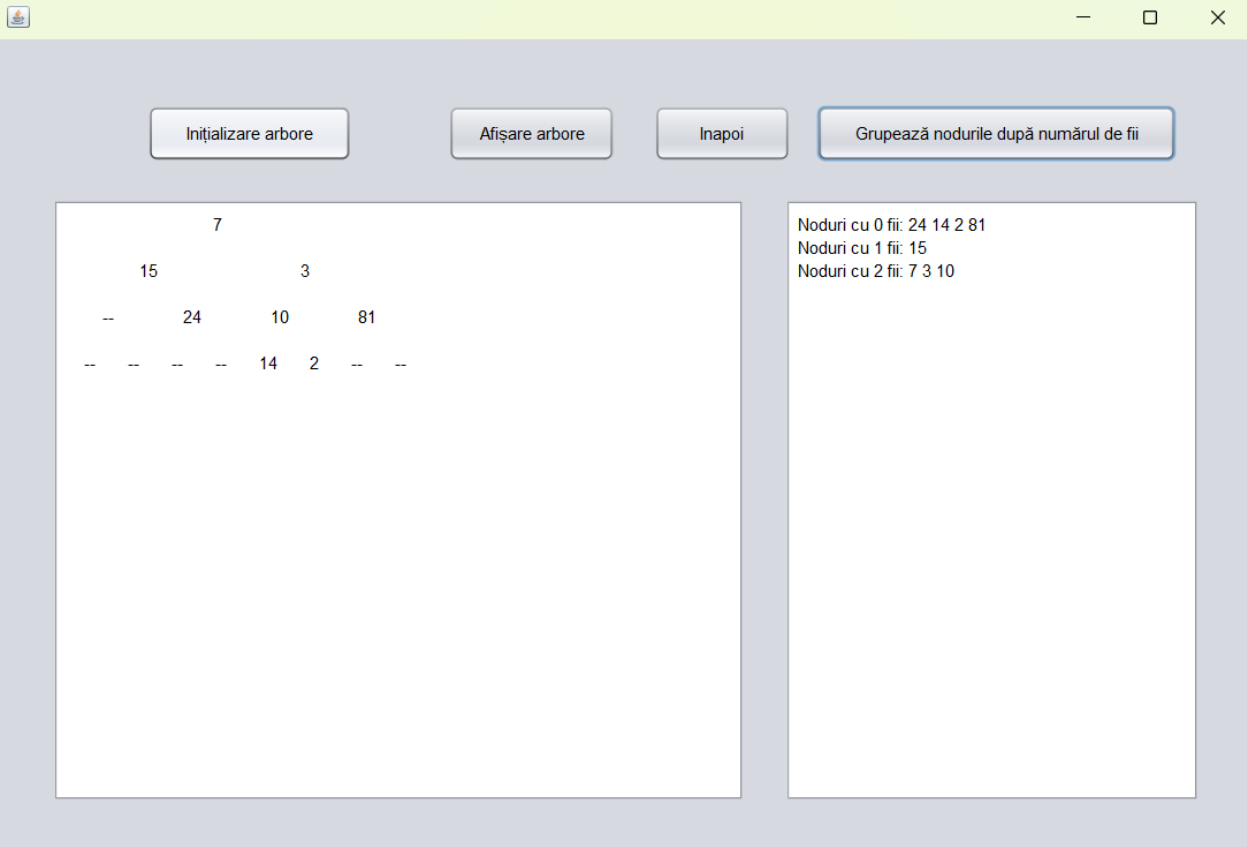


***Fig. 11** - Fereastră ce conține rezolvarea unei probleme*

După cum se poate observa, utilizatorul vede din nou 4 butoane cu nume sugestive. Dacă apasă butonul „Inițializare arbore” vor apărea ferestre mai mici (în perechi de câte trei, vezi **Fig. 12**, **Fig. 13** și **Fig. 14**) unde va avea posibilitatea să confirme dacă un anumit nod este „NULL” sau nu. În cazul selectării răspunsului „No”, va apărea o altă fereastră unde acesta va putea să introducă valoarea numerică pentru acel nod. În cazul selectării răspunsului „Yes”, întrebarea se repetă pentru celelalte noduri, până la terminarea lor.

În urma inițializării, utilizatorul poate apăsa butonul „Afișare arbore” pentru a vedea arborele introdus, poate apăsa și butonul „Grupează nodurile după numărul de fii” pentru a vedea nodurile cu 0, 1, respectiv 2 fii (vezi **Fig. 11**) sau poate apăsa butonul „Înapoi” pentru a reveni la meniul principal.

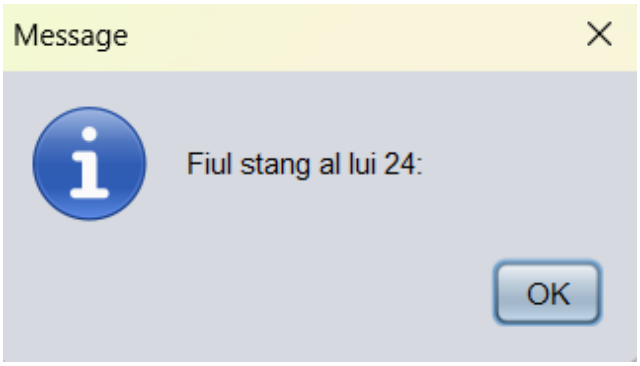


***Fig. 12*** *- Fereastră de informare a valorii nodului următor*

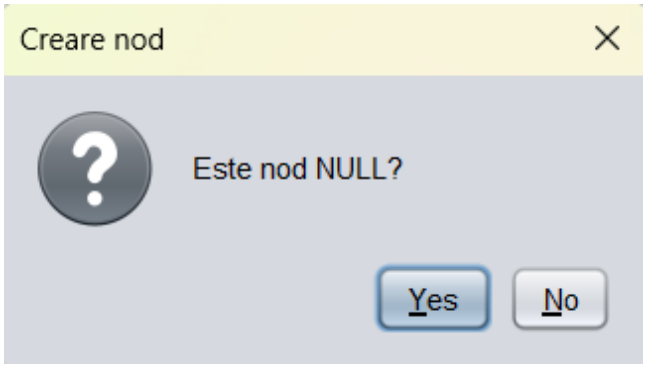


***Fig. 13*** *- Fereastră de confirmare*

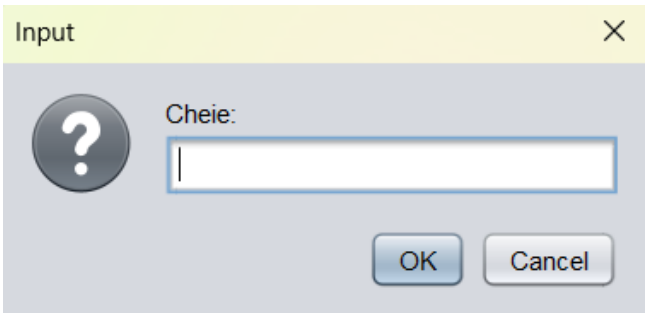


***Fig. 14*** *- Fereastră de introducere a cheii (valorii) nodului*

## 7.3 Butonul „Emanuel”

Dacă din meniul principal (vezi **Fig. 10**) se alege butonul „Emanuel”, utilizatorul va fi redirecționat către fereastra din **Fig. 15**, unde este rezolvată problema cu următorul enunț: „Se dă un arbore binar oarecare. Se cere să se construiască un vector de liste asociat care să conțină toate căile din acest arbore.”

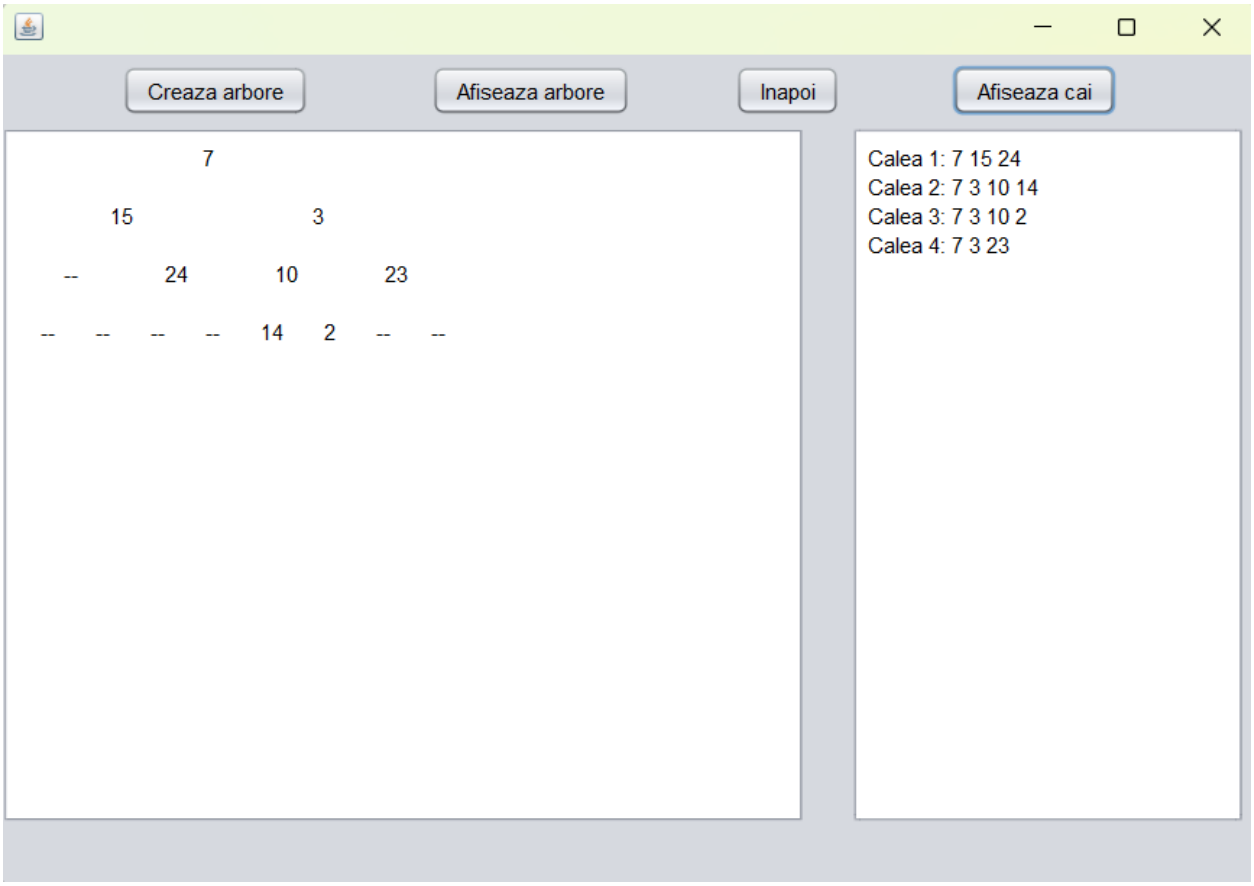


***Fig. 15** - Fereastră ce conține rezolvarea unei probleme*

Butoanele „Creaza arbore” „Afisează arbore” și „Inapoi” au aceeași funcționalitate ca și cele prezentate anterior. Butonul „Afiseaza cai” va afișa toate căile din arbore, conform cerinței.

## 7.4 Butonul „Valentin”

Dacă din meniul principal (vezi **Fig. 10**) se apasă butonul „Valentin”, se va deschide o fereastră (vezi **Fig. 16)** în care este rezolvată problema cu următorul enunț: „Se dă o matrice mxn oarecare. Se cere să se construiască o coadă de cozi astfel încât în fiecare celulă din coada principală să fie un pointer către o coadă cu o linie din matrice.”

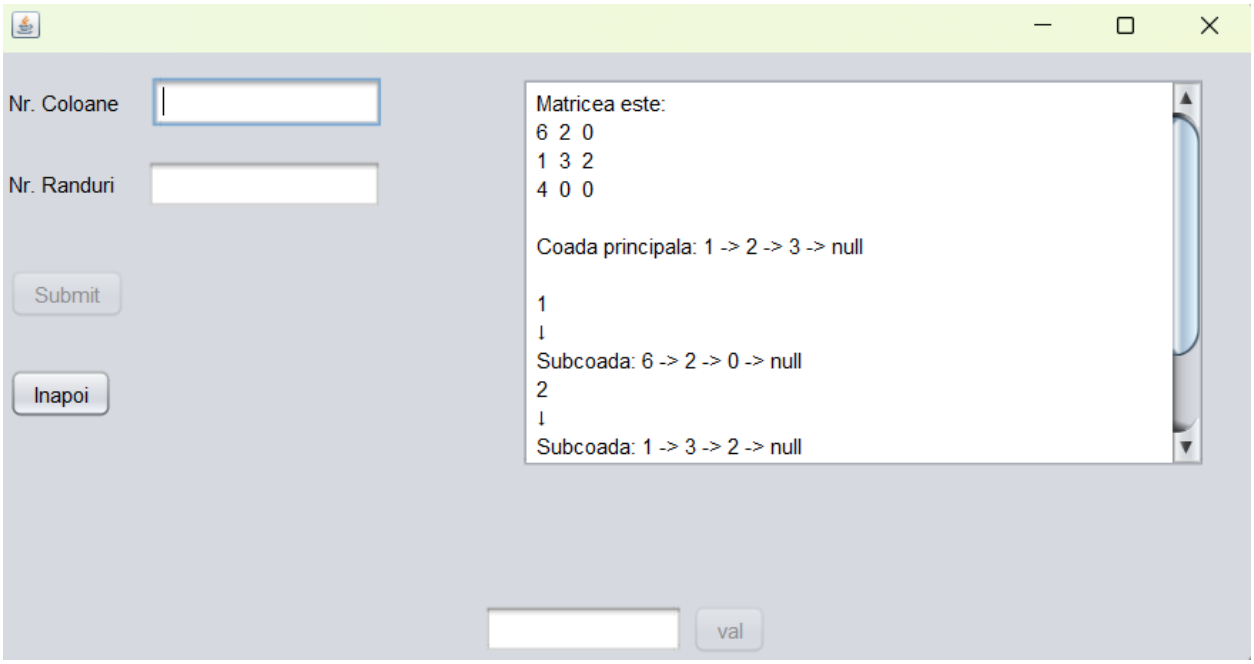


***Fig. 16** - Fereastră ce conține rezolvarea unei probleme*

Utilizatorul are posibilitatea de a introduce numărul de linii și numărul de coloane al matricei, pentru trimiterea acestor două valori fiind necesară apăsarea butonului „Submit”. În continuare, acesta va introduce valoarea fiecărui element al matricei în caseta text din partea de jos a ferestrei, iar pentru trimiterea acestora va apăsa butonul „val”. După terminarea procesului de inițializare a matricei, utilizatorul va primi o casetă de dialog cu un mesaj de informare care spune că procesul s-a terminat, întrebând în același timp utilizatorul dacă dorește să reînceapă (vezi **Fig. 17**). În urma apăsării butonului „No”, utilizatorul va vedea cozile rezultate cu elementele matricei (vezi **Fig. 16**).

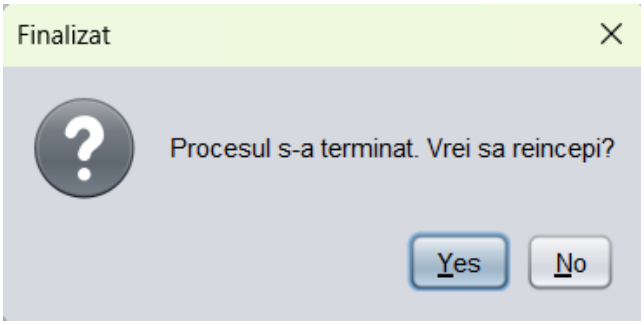


***Fig. 17** - Confirmarea terminării procesului de inițializare a matricei*

## 7.5 Butonul „Bianca”

Dacă din meniul principal (vezi **Fig. 10**) se alege butonul „Bianca”, utilizatorul va fi redirecționat către fereastra din **Fig. 18**, unde este rezolvată problema cu următorul enunț: „Fie

două mulțimi oarecare, una reprezentată cu un arbore și una cu o coadă. Se cere să se calculeze reuniunea și intersecția lor și să se reprezinte cu un arbore, respectiv coadă.”

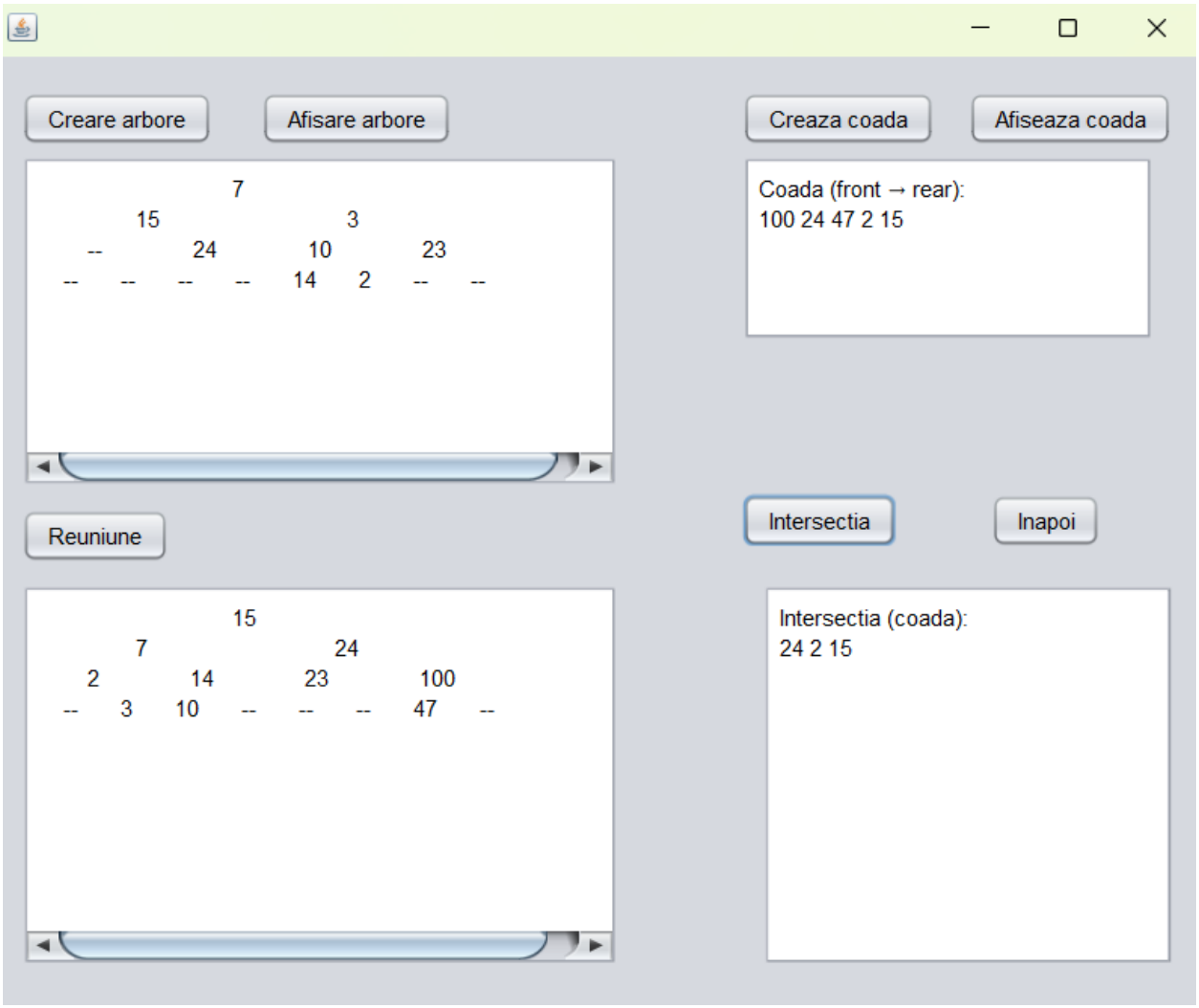


***Fig. 18** - Fereastră ce conține rezolvarea unei probleme*

Butoanele „Creare arbore” și „Afisare arbore” au aceeași funcționalitate explicată anterior. Când utilizatorul apasă butonul „Creaza coada”, se deschide o fereastră (vezi **Fig. 19**) în care acesta va introduce, pe rând, toate valorile nodurilor din coadă. Pentru fiecare valoare adăugată va primi un mesaj de confirmare (vezi **Fig. 20**)

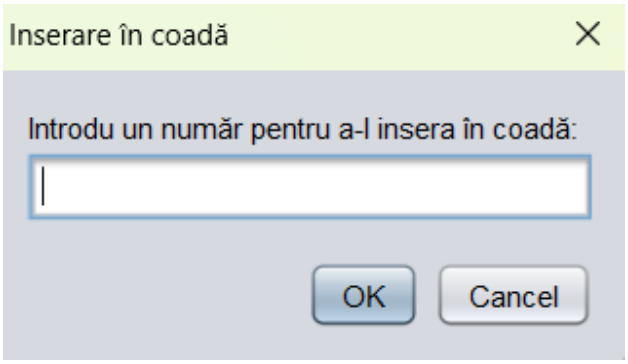


***Fig. 19*** *- Fereastră de introducere a unui element în coadă*

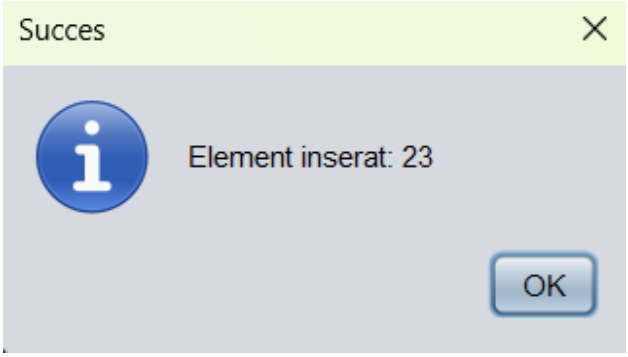


***Fig. 20*** *- Fereastră de confirmare a unui element introdus în coadă*

După inițializarea arborelui și a cozii, utilizatorul poate apăsa butonul „Reuniune” pentru a vedea arborele rezultat din reuniunea arborelui inițial cu coada introdusă. Iar pentru a vedea coada ce conține elementele mulțimii rezultate în urma intersecției arborelui cu coada inițială se poate apăsa butonul „Intersectia” (pentru rezultatele ambelor operații vezi **Fig. 18**). Când se dorește revenirea la fereastra principală se va apăsa butonul „Inapoi”.

# 8 Documentație sistem

Dacă secțiunea anterioară a fost dedicată experienței utilizatorului, acest capitol se concentrează pe arhitectura tehnică și pe mecanismele ce fac posibilă funcționarea întregului sistem. Documentația de sistem oferă o privire asupra „motorului" aplicației, explorând de la cerințele minime de rulare până la modul în care am organizat codul și structurile de date.

## 8.1 Cerințe de sistem

Performanța și stabilitatea aplicației depind și de configurarea mediului de execuție. Este esențial ca resursele hardware și software să fie aliniate cu nevoile mașinii virtuale Java. Astfel, pentru a asigura o funcționare optimă sistemul trebuie să respecte următoarele specificații tehnice:

- Cerințe Software:
  - ✓ Sistem de operare: Windows, Linux sau macOS (orice sistem care suportă Java).
  - ✓ Mediul de execuție: Java Runtime Environment (JRE) sau Java Development Kit (JDK) 21.
  - ✓ Mediul de dezvoltare (pentru inspectarea codului): NetBeans IDE (versiune compatibilă cu JDK 21).
- Cerințe Hardware minime:
  - ✓ Procesor: 2 GHz sau mai rapid, dual-core.
  - ✓ Memorie RAM: 4 GB.
  - ✓ Spațiu pe disc: minim 50 MB (pentru fișierele proiectului și resursele compilate).
  - ✓ Monitor: rezoluție minimă de 1024x768 pixeli (pentru a permite afișarea corectă a elementelor de interfață grafică).
  - ✓ Periferice: tastatură și mouse (necesare pentru introducerea datelor și navigarea între ferestre).

## 8.2 Interfața Grafică (GUI)

În continuare, se va analiza modul ales pentru construirea proiectul și ce tehnologii stau la baza funcționării acestuia. Pentru dezvoltarea interfeței grafice s-a utilizat Java Swing, componentă a Java Foundation Classes (JFC), care a permis construirea unei aplicații de tip desktop robuste și portabile. Alegerea acestui framework a facilitat gestionarea evenimentelor (event-handling) pentru cele 4 module și a asigurat o separare clară între logica algoritmilor și

prezentarea vizuală, utilizând containere de tip JFrame pentru ferestrele principale și componente precum JButton sau JTextField pentru interacțiunea cu datele.

## 8.3 Logica de Gestiune a Datelor

Un aspect central al sistemului este implementarea proprie a structurilor de date dinamice, fără utilizarea pachetului java.util.Collections. Sistemul se bazează pe o arhitectură de tip „low-level" în contextul Java, unde:

- Gestiunea Memoriei: Este realizată prin manipularea directă a referințelor către obiecte, definind clase de tip Nod pentru arbori și liste.
- Modularitatea: Fiecare dintre cele patru probleme (gruparea nodurilor unui arbore binar după numărul de fii, afișarea căilor dintr-un arbore binar, construirea unei cozi de cozi cu elementele unei matrice, reuniunea și intersecția dintre un arbore și o coadă) este încapsulată în propriul pachet logic, comunicarea dintre acestea fiind realizată prin intermediul clasei orchestrator „Main_GUI.java".
- Corectitudinea Algoritmică: Sistemul implementează manual mecanisme de tip Push/Pop pentru cozi și parcurgerile Inorder (SRD) și Preorder (RSD) pentru arbori, asigurând astfel un control mai bun asupra complexității spațiale și temporale a aplicației.

## 8.4 Structurarea Proiectului și Organizarea Modulelor

Deoarece proiectul impune implementarea manuală a structurilor de date, arhitectura este organizată în patru straturi logice, grupând clasele după rolul lor funcțional. De asemenea, unele dintre clase se pot încadra în mai multe straturi logice.

1. Nivelul de Prezentare (Modulele GUI) - Clasa „Main_GUI.java" este punctul central de lansare și meniul principal de navigare. Clasele „GUI_A.java", „GUI_E.java", „GUI_V.java", „GUI_B.java" sunt interfețe specifice pentru fiecare dintre cele 4 probleme. Acestea preiau datele de la utilizator, apelează algoritmii și afișează rezultatele.

2. Nivelul Elementelor Atomice (Data Nodes și Links) - Sunt „blocurile de bază" ale aplicației, reprezentând nodurile individuale care formează structurile complexe. Clasele de tip Node (NodeA, NodeE, NodeB) definesc structura unui nod dintr-un arbore (valoare, referință stânga, referință dreapta), iar clasa NodeV definește structura unei celule dintr-o coadă de cozi (conținând valoarea – index-ul, referință către următorul element și referință către coada spre care indică celula respectivă). Asemănător, clasele LinkA și LinkE reprezintă elementele de bază pentru listele simplu înlănțuite și conțin datele și referința către următorul element, iar clasa LinkB reprezintă elementul de bază (o celulă) pentru o coadă.

3. Nivelul Structurilor de Date - Acest strat implementează logica de gestiune a colecțiilor de date (Liste, Stive, Cozi și Arbori), fără a folosi „java.util". Clasele LinkListA și LinkListE

gestionează înlănțuirea nodurilor, operațiile de inserare, afișare și verificare listă vidă. Clasele CoadaV, LinkQueueB, MyStackA, MyStack E și StackTreeB implementează politicile de acces la date de tip LIFO (Last-In-First-Out) și FIFO (First-In-First-Out). Deși niciunul dintre enunțurile celor patru probleme nu spune ceva de vreo stivă, cele 3 clase de tip Stack sunt utilizate pentru funcțiile iterative de parcurgere a arborilor pentru afișare. Clasele de tip Tree (TreeA, TreeE, TreeB) gestionează înlănțuirea nodurilor unui arbore precum și diferite operații de parcurgere, determinarea numărului de noduri frunză și alte operații algoritmice specifice problemelor în cauză.

4. Nivelul de Logică Algoritmică - Cuprinde clasele care implementează efectiv algoritmii pentru rezolvarea problemelor (TreeA, TreeE, TreeB, MatriceV, etc). Așa cum deja am precizat, unele clase (cum sunt cele de tip Tree) se încadrează la două straturi logice, precum „Nivelul Structurilor de Date” și „Nivelul de Logică Algoritmică”. Așadar, clasele de la acest nivel încapsulează logica de procesare a structurilor ierarhice, implementând diferiți algoritmi atât recursivi cât și iterativi. Totodată, acest nivel asigură transformarea coerentă a datelor între diversele modele de reprezentare, realizând mapări între matrice, arbori binari și cozi.

# 9 Considerente referitoare la calitate

Dincolo de cerințele tehnice propriu-zise, acest capitol trece în revistă aspectele care definesc proiectul nostru în acest stadiu de învățare. Chiar dacă reprezintă o primă etapă de dezvoltare și un prototip de studiu, se vor prezenta acele puncte care arată efortul nostru de a organiza aplicația într-un mod logic, oferind un punct de plecare ce poate fi rafinat și continuat în viitor.

- Simplitate Operațională (Usability): Centralizarea celor 4 probleme într-o interfață de tip „Launcher" (Meniu Principal) elimină necesitatea ca utilizatorul să execute manual 4 clase diferite, oferind o experiență profesională.
- Independența Modulelor: Deși sunt lansate din aceeași interfață, modulele sunt decuplate logic. O eventuală eroare apărută în logica de procesare a datelor din clasa „TreeA.java", nu va afecta stabilitatea celorlalte module care utilizează tot o structură de date de tip arbore.
- Transparență Algoritmică: Deoarece nu s-au folosit biblioteci predefinite (java.util.*), calitatea sistemului este verificabilă la nivel de cod sursă. Fiecare structură de date (Coadă, Listă, Stivă, Arbore) este implementată explicit, permițând optimizarea diferiților algoritmi (de parcurgere, inserare, etc) în funcție de specificul fiecărei probleme. De asemenea, prin evitarea bibliotecilor standard complexe și utilizarea unor structuri de date minimaliste, scrise manual, overhead-ul asupra procesorului este minim, rezultând un timp de răspuns foarte bun.
- Portabilitate: Aplicația este scrisă în cod Java pur (cu Swing), deci poate fi portată pe orice sistem de operare (Windows, Linux, macOS) fără modificări de cod, respectând bineînțeles standardul minim specificat în secțiunea **Cerințe de sistem.**
- Extensibilitate: Arhitectura sistemului respectă principiul de a fi „deschis pentru extindere". Datorită decuplării dintre logica de interfață și clasele structurilor de date, adăugarea unei a cincea probleme sau a unei noi funcționalități algoritmice se poate realiza prin simpla instanțiere a claselor de bază existente (sau cel mult completarea acestora), alături bineînțeles de noul cod care tratează rezolvarea problemei respective, fără însă a fi necesară refactorizarea întregului nucleu al aplicației.
- Mentenabilitate prin Standardizare: Adoptarea unei convenții consecvente de numire în toate cele 4 module (prin utilizarea sufixelor A, B, E, V corespunzătoare membrilor echipei și a denumirilor intuitive precum Node, Link, LinkList, Tree) facilitează înțelegerea codului. Această standardizare asigură o mentenanță facilă și permite oricărui dezvoltator să intervină asupra modulelor.