

Proiect la disciplina

“Evaluarea performanțelor”

Îndrumător disciplină - Ș.I.dr. Tiberius Dumitriu

Coechipieri:

Iorga Elena - 1411B

Mihai Roxana Maria - 1411B

Pelihaci Beatrice Elena - 1411B

Cuprins

1. Enunțarea temei	2
2. Arhitectura generală	3
3. Funcționalitatea	4
3.1. Funcționalitatea algoritmului	4
3.2. Funcționalitatea aplicației	7
4. Rolul fiecărui membru al echipei	9
5. Complexitățile programului	10
5.1. Pentru funcția GenerateUserRatings	10
5.2. Pentru funcția GenerateInitialPopulation	11
5.3. Pentru funcția ReadUserRatings	13
5.4. Pentru funcția ReadPopulation	15
5.5. Pentru funcția GetCosineSimilarityV	16
5.6. Pentru funcția de fitness	17
5.7. Pentru funcția Mutation	19
5.8. Pentru funcția de încrucișare	19
5.9. Pentru funcția Selection	20
5.10. Pentru funcția Recommend	21
6. Demonstrația corectitudinii	27
6.1. Demonstrația corectitudinii funcției de citire a populației	27
6.2. Demonstrația funcției de încrucișare	31
7. Testarea programului	34
7.1. Teste pentru prima fereastră:	34
7.2. Teste pentru a doua fereastră:	36
7.3. Teste pentru a treia fereastră:	37

1. Enunțarea temei

Acest proiect își propune să implementeze o recomandare de cărți folosindu-se de algoritmul de evoluție diferențială. Programul pleacă de la premiza că alți utilizatori ai aplicației au oferit câte un punctaj fiecărui produs din lista prezentată, acestea fiind salvate anterior. Un nou utilizator intră în aplicație și își cere să selecteze cinci cărți pe care le-a citit și să le dea câte o notă de la unu la cinci. Pe baza acestora și a punctajelor oferite în trecut de ceilalți cititori, va primi o recomandare cât mai potrivită.

S-a ales acest algoritm pentru problema în cauză deoarece scopul lui fiind de a optimiza, este una dintre cele mai bune soluții pentru a face un sistem de recomandare. El a câștigat locul trei la un concurs în anul 1996 în condițiile în care primele două locuri nu erau algoritmi generali, dovedindu-și capacitatele.

2. Arhitectura generală

Programul a fost făcut în C# folosind suportul pentru crearea interfeței și funcții simple pentru a crea algoritm, acestea fiind folosite într-o funcție mai mare care le combină pe toate: Recomend function. În algoritm normal, arhitectura nu are funcția de Mutation în Crossover deoarece se face mutația și apoi se verifică dacă se aplică. Noi ca să sărim peste a face mutația de fiecare dată, am făcut-o doar dacă se aplică, din această cauză am folosit în funcția de încrucișare, dar algoritm respectă exact algoritm original și schema lui.

Interfața este alcătuită din trei form-uri care se desfășoară unul după celălalt (Form1 unde utilizatorul alege ce cărți a citit, Form2 unde le da niște note și apoi Form3 unde primește recomandarea dorită).

În Form3 se face conexiunea cu algoritmul care respectă schema algoritmului original, doar că în loc să facem mutația pentru fiecare element și apoi să vedem dacă o și luăm în considerare, o facem doar dacă trebuie să o folosim. Din această cauză deși poate părea ciudat, aplicăm mutația în funcția de crossover.

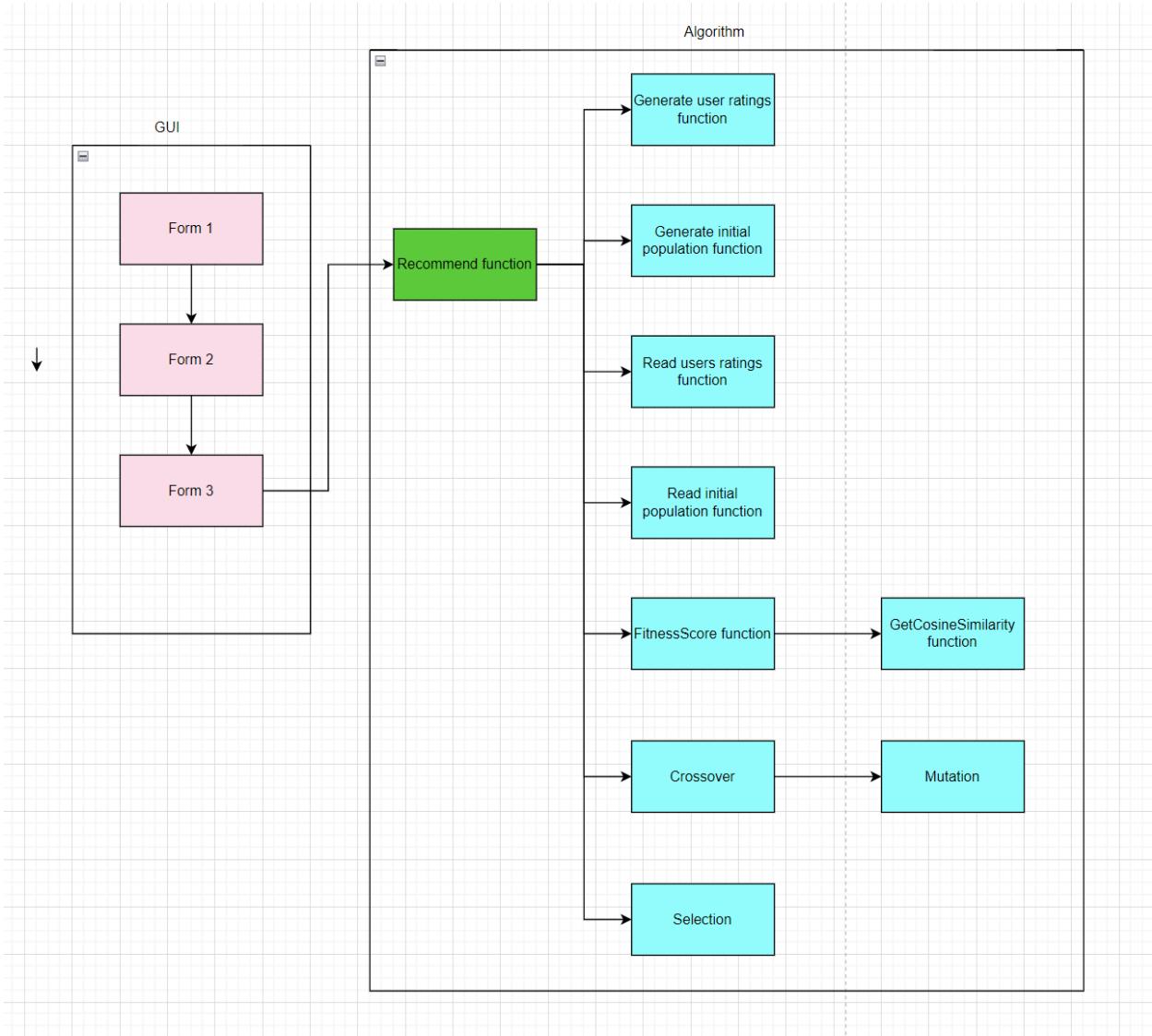


Figura 1. Arhitectura programului

3. Funcționalitatea

3.1. Funcționalitatea algoritmului

În cadrul rezolvării problemei, s-a respectat algoritmul evoluției diferențiale. Evoluția diferențială este un algoritm asemănător cu un algoritm evolutiv clasic, doar că semnificațiile

operatorilor sunt diferite, la fel și ordinea de aplicare a lor. Schema acestora este:

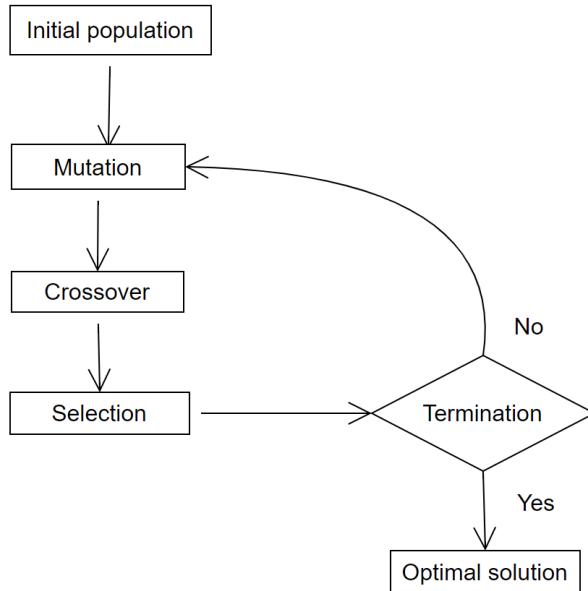


Figura 2. Schema algoritmului evolutiv diferențial

Inițializarea presupune pentru început definirea unor limite: inferioare și superioare. Apoi, populația inițială este generată în mod aleator cu valori cuprinse între cele două limite.

Mutația este caracteristică acestui algoritm presupunând operația de generare a noilor cromozomi care implică adăugarea diferenței dintre doi cromozomi la al treilea și compararea cu al patrulea:

$$v_{i,G+1} = x_{r1,G} + F * (x_{r2,G} - x_{r3,G}) \quad (1)$$

Unde:

- r1, r2, r3, i sunt indecsi diferenți care fac parte din intervalul [1, NP] (NP = dimensiunea populației)
- F este factorul de amplificare, acesta aparținând intervalului (0,2]

Încrușarea presupune generarea unui noi vector din vectorii x și v. Aceasta se folosește de un CR (crossover rate) care este rata de încrușare și alegerea unui element aleator din populație pentru ca să nu existe șansa să nu fie niciun element mutat. În această variantă a algoritmului este folosită încrușarea binomială. Ea este asemănătoare cu încrușarea uniformă de la algoritmii generici, dar aici se lucrează cu gene cu valori reale, nu cu biți. Se alege inițial un element care va fi încrușat indiferent de crossover rate. Apoi, parcurgându-se populația se

generează aleator un anumit număr între 0 și 1, iar dacă respectiva genă are acest număr mai mic decât CR sau dacă este gena aleasă aleator la început, individul devine v (vectorul calculat la mutație). Dacă nu, rămâne același.

Selecția decide dacă un cromozom va face parte din noua generație. Formula folosită este:

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{dacă } f(u_{i,G+1}) \leq f(x_{i,G}) \\ & \text{SAU} \\ & x_{i,G} \text{ în caz contrar} \end{cases} \quad (2)$$

Unde:

- f = funcția obiectiv (funcția de fitness) folosită la compararea indivizilor
- $x_{i,G+1}$, $u_{i,G+1}$, $x_{i,G}$ indivizi (noul individ ce va fi adăugat la populație, individul rezultat din mutație, vechiul individ)
- $i \in [1, N]$

Pentru inițializarea populației inițiale s-au luat ca limite valorile de 1 și de 5 întrucât un cititor poate da o notă unei cărți care face parte doar din acest interval. Apoi, pentru populația inițială, valoarea numărului de indivizi a fost aleasă ca fiind 50. Mutăția a fost făcută luând trei indivizi complet aleator dar diferenți unul de celălalt, aplicând exact formula (1).

În schimb, pentru încrucișare, pentru funcția obiectiv a fost aleasă similaritatea cosinusoidală. Aceasta a fost folosită pentru a vedea cât de apropiat este un individ din populația inițială cu cei care fac parte din colecția de punctaje date de ceilalți utilizatori tuturor produselor. Astfel, calculându-se această formulă între individul curent din populație și fiecare dintre indivizii din colecția de punctaje oferite, se alege care e individul de care este cel mai apropiat și se returnează scorul de similaritate:

$$\text{similarity} = \cos(\Theta) = \frac{A \cdot B}{\|A\| * \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} * \sqrt{\sum_{i=1}^n B_i^2}} \quad (3)$$

Facem acest lucru deoarece scopul nostru e de a găsi un individ de care este foarte apropiat. Dacă s-a găsit și unul cărui de similaritate foarte mare într-o populație, acela va fi soluția.

În cazul nostru, A este individul din populația curentă pe care vrem să o modificăm astfel încât să obținem rezultate căt mai bune, iar B este un vector din vectorii cu note oferite

produselor de către ceilalți utilizatori. Calculând similaritatea cu toți acești vectori și alegând cel mai apropiat individ, ar trebui să obținem cea mai bună recomandare. Deoarece avem scopul de a minimiza funcția, funcția de fitness va fi opusul acestei similarități pentru ca atunci când facem selecția să respectăm exact formula în care comparăm dacă funcția individului rezultat din mutație este mai mică decât cea a individului deja existent. După ce facem selecția, actualizăm populația. Funcția pentru selecție folosită este (2).

După ce facem selecția, actualizăm populația.

În funcție de parametrii aleși pentru rulare din cadrul algoritmului, recomandarea va fi una cât mai bună. Pentru rulare noi am folosit:

```
int populationCount = 50;
int maxNrOfGenerations = 6000;
int CR = 30;
double F = 0.9;
```

Figura 3. Parametrii aleși

O populație de 50, 6000 de generații, un crossover rate de 30% iar $F = 0.9$ care este factorul de amplificare.

3.2. Funcționalitatea aplicației

User-ul trebuie doar să ruleze aplicația, moment în care îi apare o listă de cărți din care trebuie să selecteze cinci pe care le-a citit deja :

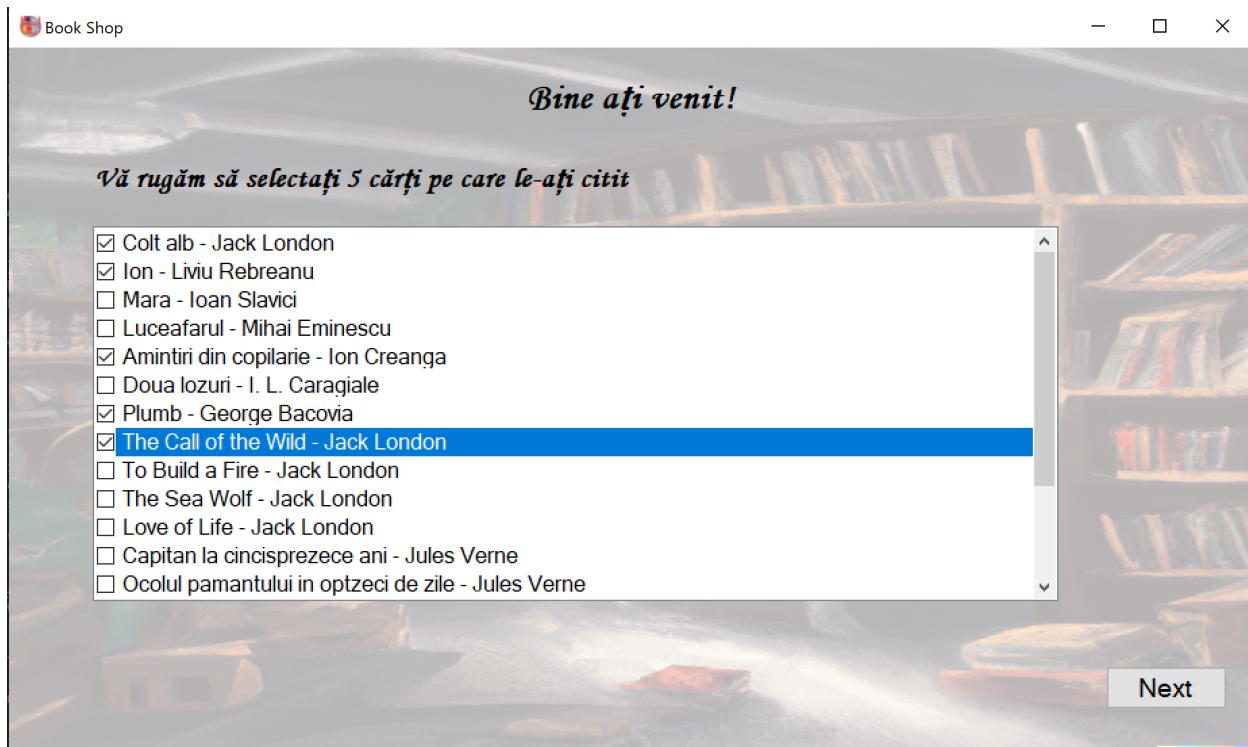


Figura 4. Captură de ecran cu prima pagină a aplicației(Form 1)

Până la selectarea celor 5 cărți, butonul Next este inactiv, iar dacă se selectează mai mult de 5 cărți, se va dezactiva butonul de Next.

După apăsarea butonului Next, îi apare a doua pagină a interfeței unde trebuie să dea note cuprinse între 1 și 5 acelor cărți alese de el:



Figura 5. Captură de ecran cu a doua pagină a aplicației (Form 2)

Apăsând Next, îi apare următoarea interfață:

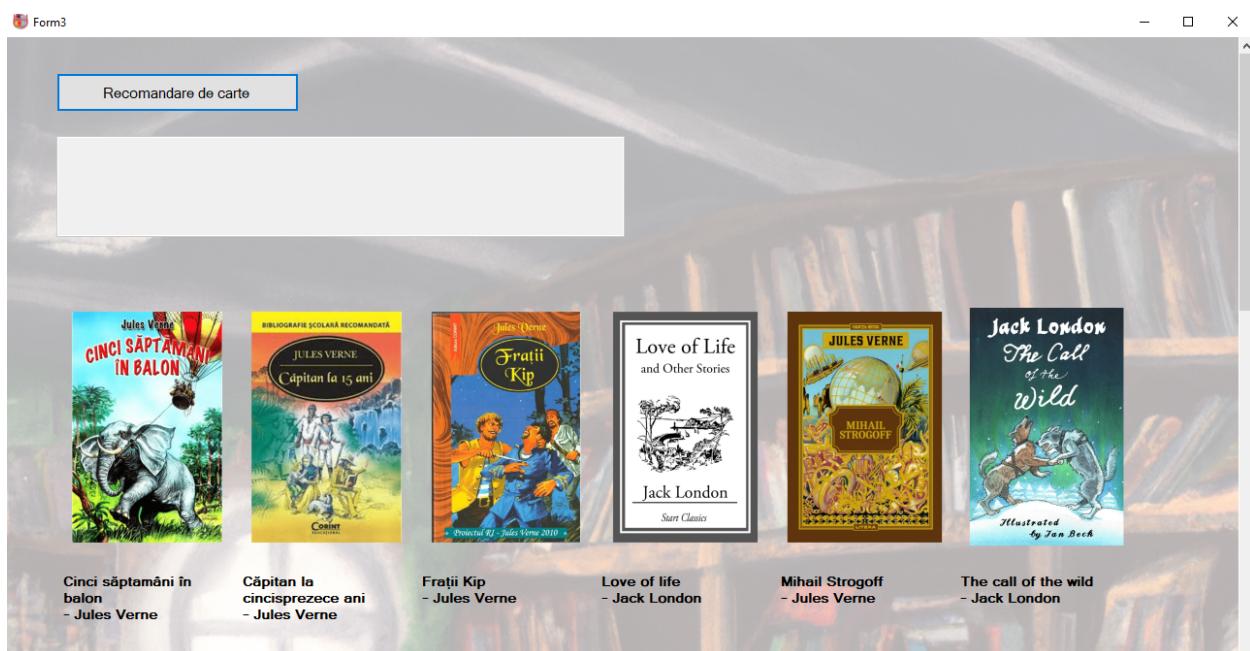


Figura 6. Captură de ecran cu a treia pagină a aplicației (Form 3)

Pe care clientul trebuie să apese pe butonul de Recomandare de carte pentru a primi recomandarea noastră:

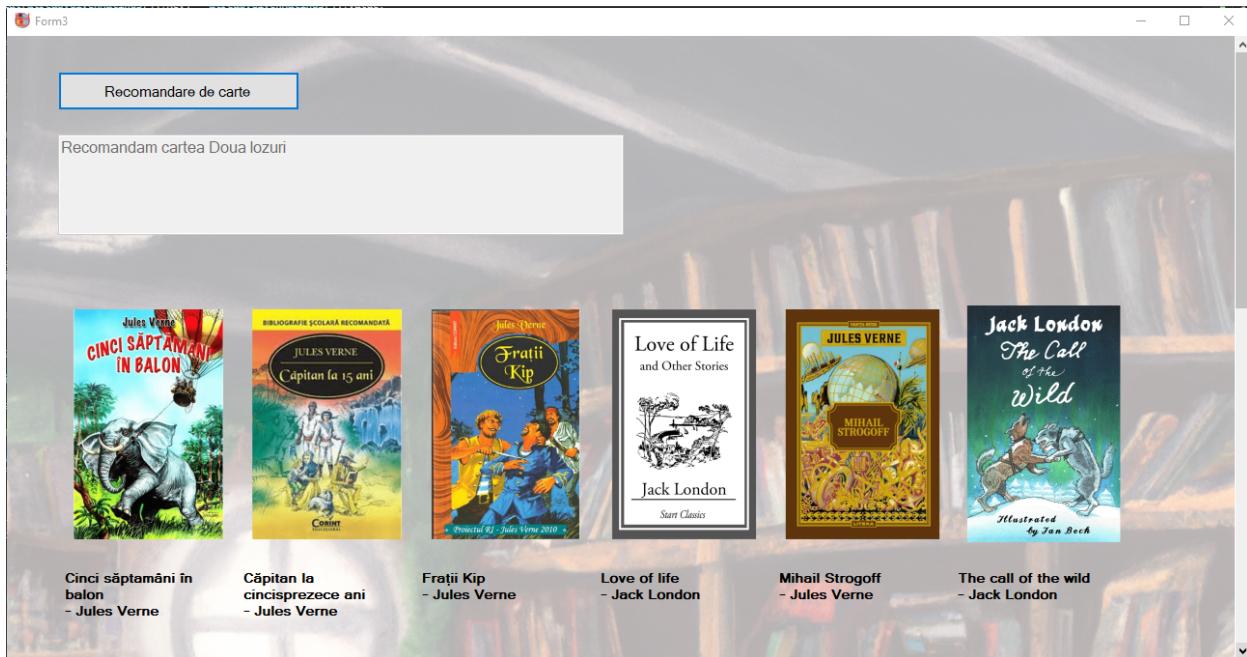


Figura 7. Captură de ecran cu a treia pagină a aplicației după apăsarea butonului de „Recomandare de carte”

4. Rolul fiecărui membru al echipei

Iorga Elena	<ul style="list-style-type: none"> - Realizarea codului - Realizarea calculului complexităților pentru funcțiile de Selection și Recommend și a arhitecturii aplicației - Demonstrarea corectitudinii
Mihai Roxana Maria	<ul style="list-style-type: none"> - Realizarea interfeței grafice - Realizarea calculului complexităților pentru funcțiile de GenerateUserRatings, GenerateInitialPopulation, ReadUserRatings - Realizarea testelor interfeței și documentarea lor
Pelihaci Beatrice	<ul style="list-style-type: none"> - Realizarea testelor - Realizarea calculului complexităților pentru ReadPopulation, GetCosineSimilarityV, FitnessScore - Realizarea documentării testelor făcute

Tabelul 1. Rolul fiecărui membru al echipei

5. Complexitățile programului

5.1. Pentru funcția GenerateUserRatings

Dimensiunea initială: noOfUsers + noOfItems = max(noOfUsers, noOfItems)

Precondiții: noOfUsers și noOfItems numere întregi pozitive diferite de zero.

Postcondiții: Este creat fișierul “UserRatings.txt” și în el sunt scrise noOfUsers * noOfItems valori reale, fiecare pe câte o linie.

```
public static void GenerateUserRatings(int noOfUsers, int noOfItems)
{
    1: double[][] userRatings = new double[noOfUsers][];
    2: Random rand = new Random();
    3: StreamWriter file = new StreamWriter("UserRatings.txt");
    4: int k = 0;
    5: for (int i = 0; i < noOfUsers; i++) {
    6:     userRatings[i] = new double[noOfItems];
    7:     for (int j = 0; j < noOfItems; j++) {
    8:         userRatings[i][j] = rand.NextDouble() * 4 + 1;
    9:         file.WriteLine(userRatings[i][j]);
    10:    }
    file.Close();
}
```

Operatie	Cost	Repetari	Total
1	noOfUsers	1	noOfUsers
2	1	1	1
3	1	1	1
4	1	1	1
5	2	noOfUsers + 1	2*(noOfUsers+1)
6	noOfItems	noOfUsers	noOfUsers*noOfItems
7	2	noOfUsers*(noOfItems+1)	2* noOfUsers*(noOfItems+1)
8	4	noOfUsers*noOfItems	4*noOfUsers*noOfItems
9	1	noOfUsers*noOfItems	noOfUsers*noOfItems
10	1	1	1

Tabelul 2. Complexitățile funcției GenerateUserRatings

$$T(\text{noOfUsers}, \text{noOfItems}) = 6 + 5\text{noOfUsers} + 8 \text{ noOfUsers} * \text{noOfItems}$$

Complexitatea va fi aceeași pentru cazul favorabil, nefavorabil și mediu întrucât numărul de iterații depinde doar de noOfUsers și noOfItems.

Deci complexitatea timp va fi $T_{\text{mediu}} = \text{noOfUsers} * \text{noOfItems} \Rightarrow$

$$O(\max\{\text{noOfUsers}, \text{noOfItems}\}^2)$$

5.2. Pentru funcția GenerateInitialPopulation

Dimensiunea inițială: alreadyGivenRatings.Count + noOfItems + populationCount

Precondiții: alreadyGivenRatings este o listă cu 5 perechi cheie-valoare unde cheia reprezintă id-ul unei cărți selectate de User, iar valoarea nota acordată; noOfItems > 0, populationCount > 0.

Postcondiții: Este creat fișierul "Population.txt" și în el sunt scrise populationCount * noOfItems valori reale, fiecare pe câte o linie.

```
public static void GenerateInitialPopulation(List<KeyValuePair<int, double>>
alreadyGivenRatings, int noOfItems, int populationCount)
{
    1. Random rand = new Random();
    2. StreamWriter file = new StreamWriter("Population.txt");
    3. double[][] population = new double[populationCount][];
    4. List<int> listOfAlreadyRatedItems = new List<int>();
    5. for (int i = 0; i < populationCount; i++) {
    6.     double[] myUserRating = new double[noOfItems];
    7.     foreach (KeyValuePair<int, double> item in alreadyGivenRatings) {
    8.         listOfAlreadyRatedItems.Add(item.Key);
    9.         myUserRating[item.Key] = item.Value;
    10.    for (int k = 0; k < noOfItems; k++)
    11.    {
    12.        if (!listOfAlreadyRatedItems.Contains(k))
    13.            myUserRating[k] = rand.NextDouble() * 4 + 1;
    14.    population[i] = myUserRating;
    15.    for (int j = 0; j < noOfItems; j++)
    16.        file.WriteLine(population[i][j]);
    17. file.Close();
}
```

Operatie	Cost	Repetari	Total
1	1	1	1
2	1	1	1
3	populationCount	1	populationCount
4	1	1	1
5	2	populationCount + 1	2 * (populationCount + 1)
6	noOfItems	populationCount	noOfItems * populationCount
7	2 (nu se încălzește paralelismul)	alreadyGivenRatings.Count * populationCount	2 * alreadyGivenRatings.Count * populationCount

8	<p>Costul operației de adăugare în listă depinde de capacitatea listei. Capacitatea listei este inițial 0. La adăugarea primului element capacitatea devine 16, urmând să fie dublată de fiecare dată când capacitatea listei nu mai poate acomoda un nou element.</p> <p>Deci, costul este:</p> <ul style="list-style-type: none"> - 1 dacă <code>listOfAlreadyRatedItems.Count <= capacitatea curentă a listei</code> -<code>listOfAlreadyRatedItems.Count</code>, dacă <code>listOfAlreadyRatedItems.Count > capacitatea curentă a listei</code> 	<code>alreadyGivenRatings.Count * populationCount</code>	16^* $[listOfAlreadyRatedItems.Count/16 + 1] + listOfAlreadyRatedItems.Count - [listOfAlreadyRatedItems.Count/16 + 1]$ <p>[] fac referire la partea întreagă</p>
9	1	<code>alreadyGivenRatings.Count * populationCount</code>	<code>alreadyGivenRatings.Count * populationCount</code>
10	2	$(noOfItems + 1) * populationCount$	$2 * (noOfItems + 1)$
11	in medie, datorita <code>Contains()</code> => <code>listOfAlreadyRatedItems.Count</code>	<code>noOfItems</code>	<code>listOfAlreadyRatedItems.Count * noOfItems</code>
12	4	<code>noOfItems - alreadyGivenRatings.Count</code>	$4 * (noOfItems - alreadyGivenRatings.Count)$
13	1	<code>populationCount</code>	<code>populationCount</code>
14	2	<code>populationCount + 1</code>	$2 * (populationCount + 1)$
15	2	<code>populationCount * (noOfItems + 1)</code>	$2 * populationCount * (noOfItems + 1)$
16	1	<code>populationCount * noOfItems</code>	<code>populationCount * noOfItems</code>
17	1	1	1

Tabelul 3. Complexitatea funcției `GenerateInitialPopulation()`

$$T(n) = 1 + 1 + \text{populationCount} + 1 + 2 * (\text{populationCount} + 1) + \text{noOfItems} * \text{populationCount} + 2 * \text{alreadyGivenRatings.Count} * \text{populationCount} + (16 * [\text{listOfAlreadyRatedItems}/16 + 1] + \text{listOfAlreadyRatedItems.Count} - [\text{listOfAlreadyRatedItems.Count}/16 + 1]) * \text{alreadyGivenRatings.Count} * \text{populationCount} + \text{alreadyGivenRatings.Count} * \text{populationCount} + 2 * (\text{noOfItems} + 1) + \text{listOfAlreadyRatedItems.Count} * \text{noOfItems} + 4 * (\text{noOfItems} - \text{alreadyGivenRatings.Count}) + \text{populationCount} + 2 * (\text{populationCount} + 1) + 2 * \text{populationCount} * (\text{noOfItems} + 1) + \text{populationCount} * \text{noOfItems} + 1$$

**alreadyGivenRatings.Count este tot timpul 5 în acest program deoarece interfața nu permite alegerea a mai mult de 5 cărți la care userul să dea rating, dar se poate modifica pe viitor.*

***listOfAlreadyRatedItems.Count == alreadyGivenRatings.Count.*

alreadyGivenRatings este o listă de perechi cheie-valoare, iar listOfAlreadyRatedItems este o listă ce conține toate cheile din alreadyGivenRatings

Se observă că operația dominantă este $2 * \text{populationCount} * (\text{noOfItems} + 1) \Rightarrow$

$T(n) = 2 * \text{populationCount} * (\text{noOfItems} + 1)$

Complexitatea va fi aceeași pentru cazul favorabil, nefavorabil și mediu întrucât numărul de operații depinde doar de n și m .

Deci complexitatea timp va fi $T(\text{mediu}) = \text{noOfUsers} * \text{noOfItems} \Rightarrow$

$$O(\max\{\text{noOfUsers}, \text{noOfItems}\}^2)$$

5.3. Pentru funcția ReadUserRatings

Dimensiunea initială: noOfUsers + noOfItems = max(noOfUsers, noOfItems)

*Precondiții: noOfUsers, noOfItems > 0, există fișierul "UserRatings.txt" și în el sunt scrise noOfUsers * noOfItems valori reale, fiecare pe căte o linie*

Postcondiții: este returnată matricea de valori reale userRatings de dimensiune noOfUsers x noOfItems

```

private static double[][] ReadUserRatings(int noOfUsers, int noOfItems)
{
    1.   StreamReader sr = new StreamReader("UserRatings.txt");
    2.   String line = "";
    3.   List<double> lstOfValues = new List<double>();
    4.   double[][] userRatings = new double[noOfUsers][];
    
    5.   while ((line = sr.ReadLine()) != null)
    {
        6.       lstOfValues.Add(Double.Parse(line));
    }
    7.   int k = 0;
    8.   for (int i = 0; i < noOfUsers; i++)
    {
        9.       userRatings[i] = new double[noOfItems];
        10.      for (int j = 0; j < noOfItems; j++)
        {
            11.          userRatings[i][j] = lstOfValues.ElementAt(k);
            12.          k++;
        }
    }
    13.   return userRatings;
}

```

Operatie	Cost	Repetari	Total
1	1	1	1
2	1	1	1
3	1	1	1
4	noOfUsers	1	noOfUsers
5	2	nr of lines in UserRatings.txt + 1 == noOfItems * noOfUsers + 1	noOfItems * noOfUsers + 1
6	- 1 dacă lstOfValues.Count <= capacitatea curentă a listei -lstOfValues, dacă lstOfValues.Count > capacitatea curentă a listei	nr of lines in UserRatings.txt == noOfItems * noOfUsers	(1 + 16* [lstOfValues.Count/16 + 1] + lstOfValues.Count - [lstOfValues.Count/16 + 1])
7	1	1	1
8	2	noOfUsers +1	2*(noOfUsers +1)
9	noOfItems	noOfUsers	noOfItems * noOfUsers
10	2	noOfUsers * (noOfItems + 1)	2 * noOfUsers * (noOfItems + 1)
11	1 + 1	noOfUsers * noOfItems	2* noOfUsers * noOfItems
12	1	noOfUsers * noOfItems	noOfUsers * noOfItems
13	1	1	1

Tabelul 4. Complexitățile funcției ReadUserRatings

$$\begin{aligned}
 T(n) = & 1 + 1 + 1 + \text{noOfUsers} + ? + (1 + 16* [\text{lstOfValues.Count}/16 + 1] + \text{lstOfValues.Count} - [\text{lstOfValues.Count}/16 + 1]) + 1 + 2*(\text{noOfUsers} + 1) + \text{noOfItems} * \text{noOfUsers} + 2 * \\
 & \text{noOfUsers} * (\text{noOfItems} + 1) + (1 + \text{lstOfValues.Count}) * \text{noOfUsers} * \text{noOfItems} + \text{noOfUsers} * \text{noOfItems} + 1
 \end{aligned}$$

* În fișierul citit, UserRatings.txt, vor exista mereu noOfItems * noOfUsers linii de valori reale

**lstOfValues.Count va fi mereu egal cu numărul de linii din fișierul citit, adică noOfItems * noOfUsers

Se observă că operația dominantă este $2 * \text{noOfUsers} * (\text{noOfItems} + 1) \Rightarrow 2 * n * (m + 1)$.

Complexitatea va fi aceeași pentru cazul favorabil, nefavorabil și mediu întrucât numărul de iterări depinde doar de noOfUsers și noOfItems.

Deci complexitatea timp va fi $T_{\text{mediu}} = \text{noOfUsers} * \text{noOfItems} \Rightarrow$

$$O(\max\{\text{noOfUsers}, \text{noOfItems}\}^2)$$

5.4. Pentru funcția ReadPopulation

Dimensiunea initială: populationCount+noOfItems

Precondiții: populationCount>0, numarul de produse noOfItems>0

Postcondiții: este returnată o matrice de tip double ce contine valori citite din fisierul generat "Population.txt"

```
public static double[][] ReadPopulation(int populationCount, int noOfItems)
{
    1. StreamReader sr = new StreamReader("Population.txt");
    2. List<double> lstOfValues2 = new List<double>();
    3. String line = "";
    4. while ((line = sr.ReadLine()) != null) {
    5.     Console.WriteLine(line);
    6.     lstOfValues2.Add(Double.Parse(line));
    7. int p = 0;
    8. double[][] population = new double[populationCount][];
    9. for (int i = 0; i < populationCount; i++) {
    10. population[i] = new double[noOfItems];
    11. for (int j = 0; j < noOfItems; j++) {
    12.     population[i][j] = lstOfValues2.ElementAt(p);
    13.     p++;
    14. return population;
}
```

Operatie	Cost	Repetari	Total
1.	1	1	1
2.	1	1	1
3.	1	1	1
4.	1	populationCount*noOfItems	populationCount*noOfItems
5.	1	populationCount*noOfItems	populationCount*noOfItems
6.	2	populationCount*noOfItems	2*populationCount*noOfItems
7.	1	1	1
8.	populationCount	1	populationCount
9.	2	populationCount+1	2*(populationCount+1)
10.	noOfItems	populationCount	noOfItems*populationCount
11.	2	populationCount*(noOfItems+1)	2*populationCount*(noOfItems+1)
12.	2	populationCount*noOfItems	2*populationCount*noOfItems
13.	1	populationCount*noOfItems	populationCount*noOfItems
14.	1	1	1

Tabelul 5. Complexitatea funcției ReadPopulation

$$T(n)=2*populationCount*(noOfItems+1)$$

Complexitatea va fi aceeași pentru cazul favorabil, nefavorabil și mediu întrucât numărul de operații depinde doar de populationCount și noOfItems.

Deci complexitatea timp va fi $T_{mediu} = populationCount * noOfItems \Rightarrow$

$$O(\max\{populationCount, noOfItems\}^2)$$

5.5. Pentru funcția GetCosineSimilarityV

Dimensiunea initială: $n + m = 2 * \max\{n, m\}$ (deși în cazul nostru, și n și m vor avea valoarea noOfItems)

Precondiții: $n, m > 0$ (ambii vectori trebuie să aibă

Postcondiții: să obținem scorul de similaritate

```
public static double GetCosineSimilarityV(double[] v1, double[] v2)
{
    double dotProduct = 0;
    double normV1 = 0;
    double normV2 = 0;
```

```

1.int N = 0;

2.N = ((v2.Length < v1.Length) ? v2.Length : v1.Length);

3.double dot = 0d;

4.double mag1 = 0d;

5.double mag2 = 0d;

6.for (int n = 0; n < N; n++) {

    7. dot += v1[n] * v2[n];

    8.mag1 += Math.Pow(v1[n], 2);

    9. mag2 += Math.Pow(v2[n], 2); }

10. return dot / (Math.Sqrt(mag1) * Math.Sqrt(mag2));
}

```

Operatie	Cost	Repetari	Total
1.	1	1	1
2.	2	1	2
3.	1	1	1
4.	1	1	1
5.	1	1	1
6.	1	max {n,m}+1	(max {n,m}+1)
7.	1	max {n,m}	max {n,m}
8.	1	max {n,m}	max {n,m}
9.	1	max {n,m}	max {n,m}
10.	1	1	1

Tabelul 6. Complexitatea funcției GetCosineSimilarityV

$$T(n) = (\max \{n, m\} + 1)$$

Complexitatea va fi aceeași pentru cazul favorabil, nefavorabil și mediu întrucât numărul de operații depinde doar de $\max \{n, m\}$.

Deci complexitatea timp va fi $T_{\text{mediu}} = \max \{n, m\} \Rightarrow$

$$O(\max\{n, m\}) \text{ (complexitate de } O(n))$$

5.6. Pentru funcția de fitness

Dimensiunea initială: noOfUsers*noOfItems + noOfItems deci noOfUsers*noOfItems (weights.length = noOfUsers)

Precondiții: weights este o matrice cu dimensiuni mai mari ca 0 și populată cu valori reale și userWeights este un vector cu dimensiuni mai mari ca 0, populat și el cu valori reale.

Postcondiții: maxScore

```
public static double FitnessScore(double[][] weights, double[] userWeights) {
    1. double sumSimilarities = 0;
    2. double score = 0d;
    3. double maxScore = 0d;
    4. for (int i = 0; i < weights.Length; i++) {
        5. double similary = GetCosineSimilarityV(weights[i],
userWeights);
        6. score += similary;
        7. if (similary > maxScore)
            8. maxScore = similary;
    }
    9. return -maxScore; //we return this so that we can compare
minimizing the function }
```

Operatie	Cost	Repetari	Total
1.	1	1	1
2.	1	1	1
3.	1	1	1
4.	2	noOfUsers+1	2*(noOfUsers+1)
5.	noOfItems	noOfUsers	noOfUsers * noOfItems
6.	1	noOfUsers	noOfUsers
7	1	noOfUsers+1	noOfUsers+1
8	1	t1(n)	t1(n)
9	1	1	1

Tabelul 7. Complexitățile funcției de fitness

Unde $t1(n) = 0$ dacă nicio similaritate nu este mai mare decât 0

= noOfUsers dacă toate similaritățile sunt mai mari

Deci complexitatea pentru cazul cel mai favorabil va fi $3 + 2*(noOfUsers+1) + noOfUsers * noOfItems + 2*noOfUsers + 2 = 4*noOfUsers + noOfUsers * noOfItems + 5$

Iar pentru cazul cel mai defavorabil $5*noOfUsers + noOfUsers * noOfItems + 5$

Observăm că atât în cazul defavorabil cât și în cel favorabil, complexitățile sunt de același ordin ($n*m$), deci complexitatea timp medie va fi $T_{mediu} = noOfUsers * noOfItems \Rightarrow$

$$O(\max\{noOfUsers, noOfItems\}^2)$$

5.7. Pentru funcția Mutation

Dimensiunea problemei: noOfUsers * noOfItems + 1 + 1

Precondiții: indivizi este o matrice cu dimensiuni mai mari ca 0 și cu valori valide (reale), $0 < \text{gena} < \text{noOfItems}$, $0 < F <= 2$

Postcondiții: la final să avem un nou individ

```
public static double Mutation(double[][] indivizi, int gena, double F)
{
    1. return indivizi[3][gena] + F * (indivizi[1][gena] -
    indivizi[2][gena]);
}
```

Operatie	Cost	Repetari	Total
1.	1	1	1

Tabelul 8. Complexitățile funcției de mutație

Deci cazul mediu va fi egal cu cazul cel mai defavorabil și cu cel mai favorabil, acestea fiind de $O(1)$.

5.8. Pentru funcția de încrucișare

Precondiții: matricea de indivizi să aibă dimensiuni mai mari ca zero și să fie populată cu elemente, F să fie o valoare între (1;2]

Postcondiții: la final să avem un individ potențial de dimensiunea noOfItems

Dimensiunea inițială: noOfItems + noOfItems * noOfUsers + listOfAlreadyRatedItems

```
public static double[] Crossover(int punctDivizare, int noOfItems, int CR,
double[][] indivizi, double F, Random rand, List<int> listOfAlreadyRatedItems)
{
    1. double[] inivididPotential = new double[noOfItems];
    2. for (int gena = 0; gena < noOfItems; gena++) {
        3. if (gena == punctDivizare || rand.Next(0, 100) < CR) {
            4. inivididPotential[gena] = Mutation(indivizi, gena, F);
        } else {
            5. inivididPotential[gena] = indivizi[0][gena];
        }
    }
    6. return inivididPotential;
}
```

Operatie	Cost	Repetari	Total
1.	noOfItems	1	noOfItems

2.	2	noOfItems+1	$2*(noOfItems+1)$
3.	1	noOfItems	noOfItems
4.	1	$t1(noOfItems,noOfUsers,listOfAlreadyRatedItems.Count)$	$t1(noOfItems,noOfUsers,listOfAlreadyRatedItems.Count)$
5.	1	$t1(noOfItems,noOfUsers,listOfAlreadyRatedItems.Count)$	$t1(noOfItems,noOfUsers,listOfAlreadyRatedItems.Count)$
6.	1	1	1

Tabelul 9. Complexitățile funcției de încrucișare

Unde $t1(noOfItems,noOfUsers,listOfAlreadyRatedItems.Count)$ în cazul cel mai favorabil este 0 , când gena nu este niciodată egală cu punctul de divizare iar random-ul generat nu este niciodată mai mic decât CR. În cazul cel mai nefavorabil este noOfItems când de fiecare dată se va genera un număr random mai mare decât CR.

Deci complexitatea în cazul cel mai favorabil al funcției este: $4 * noOfItems + 3$

Iar complexitatea în cazul ce mai nefavorabil este: $5 * noOfItems + 3$

Observăm că ambele sunt liniare deci $4*noOfItems+3 < T(n)$ (cazul mediu) < $5*noOfItems + 3$, deci cazul mediu va depinde și el liniar de noOfItems.

5.9. Pentru funcția Selection

Precondiții: userRatings este o matrice cu dimensiuni mai mari ca 0 și cu valori valide (naturale), că vectorul de individPotențial are o dimensiune mai mare ca 0, la fel și cel de individExistent

Postcondiții: la final va fi returnat fie individul potențial fie cel existent în funcție de valorile funcțiilor de fitness ale lor

Dimensiunea inițială: noOfUsers * noOfItems + noOfItems + noOfItems + 1 = noOfUsers * noOfUsers + 2 * noOfItems + 1

```
public static double[] Selection(double[][] userRatings, double[] individPotential, double[] individExistent)
{1: double fitnessIndividPotential = FitnessScore(userRatings, individPotential);
2: double fitnessIndividExistent = FitnessScore(userRatings, individExistent);
3:if (fitnessIndividPotential <= fitnessIndividExistent)
4:    return individPotential;
else
5:    return individExistent;}
```

Operatie	Cost	Repetari	Total
1	noOfUsers * noOfItems	1	noOfUsers * noOfItems

2	noOfUsers * noOfItems	1	noOfUsers * noOfItems
3	1	1	1
4	1	0 sau 1	0 sau 1
5	1	1 sau 0 (în opoziție cu op 4)	1 sau 0

Tabelul 10. Complexitățile funcției de selecție

$$T(n) = 2 * \text{noOfUsers} * \text{noOfItems} + 2$$

Deci observăm că cazul defavorabil este egal cu cazul defavorabil care este egal cu cazul mediu întrucât pentru oricare ar fi datele de intrare, avem aceeași complexitate: $\text{noOfUsers} * \text{noOfItems}$. (dacă nu luăm în considerare faptul că fitness function are și el un caz defavorabil și unul favorabil)

$$O(\max\{\text{noOfUsers}, \text{noOfItems}\}^2)$$

5.10. Pentru funcția Recommend

Precondiții: $\text{noOfItems} > 0$, $\text{noOfUsers} > 0$, $\text{maxNumberOfGenerations} > 0$, $\text{CR} \geq 0$ și trebuie să fie o valoare cuprinsă între 0 și 100, F trebuie să fie o valoare cuprinsă între 0 și 2, userRatings trebuie să fie o matrice cu dimensiuni mai mari ca 0, $\text{populationCount} > 0$, $\text{listOfItems.length} > 0$

Postcondiții: la final va fi returnat un string reprezentând recomandarea pe care o face algoritmul

Dimensiunea initială: $1 + 1 + 1 + 1 + 1 + \text{noOfItems} * \text{noOfUsers} + \text{alreadyGivenRatings.length} + 1 + \text{noOfItems} = 6 + \text{noOfItems} * \text{noOfUsers} + \text{alreadyGivenRatings} + \text{noOfItems}$

```

public static String Recommend(int noOfItems, int noOfUsers, int
maxNumberOfGenerations, int CR, double F, double[][] userRatings,
List<KeyValuePair<int, double>> alreadyGivenRatings, int populationCount, List<Carte>
listOfItems)

{
    1: Random rand = new Random();

    2: double[][] population = new double[populationCount][];
    3: GenerateInitialPopulation(alreadyGivenRatings, noOfItems,
populationCount);

    4: userRatings = ReadUsersRatings(noOfUsers, noOfItems);

    5: population = ReadPopulation(populationCount, noOfItems);

    6: for (int i = 0; i < maxNumberOfGenerations; i++) {
    7: double[][] newPopulation = new double[populationCount][];

```

```

8: int nrIndivid = 0;

9: for(int p = 0; p < population.Length; p++) {

10:    double[][] indivizi = new double[populationCount][];

11:    indivizi[0] = population[p];

12:    int firstRandomlySelectedMember = rand.Next(1, 10);

13:    int secondRandomlySelectedMember = rand.Next(1, 10);

14:    while (secondRandomlySelectedMember ==
firstRandomlySelectedMember)

15:        secondRandomlySelectedMember = rand.Next(1, 10);

16:    int thirdRandomlySelectedMember = rand.Next(1, 10);

17:    while (thirdRandomlySelectedMember == firstRandomlySelectedMember
|| thirdRandomlySelectedMember == secondRandomlySelectedMember)

18:        thirdRandomlySelectedMember = rand.Next(1, 10);

19:    indivizi[1] = population[firstRandomlySelectedMember];

20:    indivizi[2] = population[secondRandomlySelectedMember];

21:    indivizi[3] = population[thirdRandomlySelectedMember];

22:    double[] individPotential = new double[noOfItems];

23:    int punctDivizare = rand.Next(0, noOfItems);

24: int punctDivizare = rand.Next(0, noOfItems); //it's the division point where

25:    while(listOfAlreadyRatedItems.Contains(punctDivizare))

26:        punctDivizare = rand.Next(0, noOfItems);

27:    individPotential = Crossover(punctDivizare, noOfItems, CR,
indivizi, F, rand);

28: for(int j = 0; j < listOfAlreadyRatedItems.Count; j++)

29:    {individPotential[alreadyGivenRatings[j].Key] =
alreadyGivenRatings[j].Value; }

30:    newPopulation[nrIndivid] = individExistent;

31:    newPopulation[nrIndivid] = Selection(userRatings,
individPotential, individExistent, nrIndivid);

32:    nrIndivid++;}

33:    population = newPopulation; }

34:    double max = -999999;

```

```

35:     double[] solutie = new double[noOfItems];

36:     for (int i = 0; i < population.Length; i++){

37:         if (FitnessScore(userRatings, population[i]) > max) {

38:             max = FitnessScore(userRatings, population[i]);

39:             solutie = population[i];}

40:     double[] myUserRating = new double[noOfItems];

41:     List<int> listOfAlreadyRatedItems = new List<int>();

42:     foreach (KeyValuePair<int, double> item in alreadyGivenRatings){

43:         listOfAlreadyRatedItems.Add(item.Key);}

44:     double maxItem = 0;

45:     int indexSolutie = 0;

46:     for (int i = 0; i < solutie.Length; i++){

47:         if (!listOfAlreadyRatedItems.Contains(i))

48:             if (solutie[i] > maxItem){

49:                 maxItem = solutie[i];

50:                 indexSolutie = i;}}
```

51: return listOfItems.ElementAt(indexSolutie).name;}

Operatie	Cost	Repetari	Total
1	1	1	1
2	populationCount	1	populationCount
3	populationCount * noOfItems	1	populationCount * noOfItems
4	noOfItems * noOfUsers	1	noOfItems * noOfUsers
5	populationCount * noOfItems	1	populationCount * noOfItems
6	2(compară și incrementează)	maxNumberOfGenerations	2*maxNumberOfGenerations
7	populationCount	maxNumberOfGenerations	populationCount * maxNumberOfGenerations
8	1	maxNumberOfGenerations	maxNumberOfGenerations
9	2	populationCount * maxNumberOfGenerations	2*populationCount * maxNumberOfGeneration

10	populationCount	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations * populationCount
11	noOfItems (asignăm un vector de mărime noOfItems lui indivizi[0])	populationCount * maxNumberOfGenerations * populationCount	populationCount * maxNumberOfGenerations * populationCount * noOfItems
12	1	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
13	1	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
14	1	populationCount * maxNumberOfGenerations * k1 ori(depinde de câte ori generează randomul o valoare a indexului egală cu cea aleasă pentru primul individ)	populationCount * maxNumberOfGenerations * k1
15	1	populationCount * maxNumberOfGenerations *(k1 - 1) ori (nu se mai execută și pentru cazul în care al doilea index selectat e diferit de primul	populationCount * maxNumberOfGenerations *(k1 - 1)
16	1	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
17	2	populationCount * maxNumberOfGenerations * k2 ori(depinde de câte ori generează randomul o valoare a indexului egală cu cea aleasă pentru primul individ sau pentru al doilea individ)	2*populationCount * maxNumberOfGenerations * k2
18	1	populationCount * maxNumberOfGenerations *(k2 - 1)	populationCount * maxNumberOfGenerations *(k2 - 1)
19	noOfItems	populationCount * maxNumberOfGenerations * populationCount	populationCount * maxNumberOfGenerations * noOfItems
20	noOfItems	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations * noOfItems
21	noOfItems	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations * noOfItems

22	noOfItems	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations * noOfItems
23	1	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
24	1	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
25	1	populationCount * maxNumberOfGenerations * k3	populationCount * maxNumberOfGenerations * k3
26	1	populationCount * maxNumberOfGenerations * (k3-1)	populationCount * maxNumberOfGenerations * (k3-1)
27	noOfItems	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
28	2	populationCount * maxNumberOfGenerations * (alreadyGivenRatings.length+ 1)	2 * populationCount * maxNumberOfGenerations * (alreadyGivenRatings.length+1)
29	1	populationCount * maxNumberOfGenerations * alreadyGivenRatings.length	populationCount * maxNumberOfGenerations * alreadyGivenRatings.length
30	noOfItems	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
31	noOfItems (de la asignarea lui newPopulation de nrIndivid care are dimensiunea noOfItems) + noOfUsers * noOfItems (de la fitness care are complexitatea de noOfUsers ori complexitatea lui GetCosineSimilarity care are tot noOfItems)	populationCount * maxNumberOfGenerations	noOfItems * populationCount * maxNumberOfGenerations + noOfItems *noOfUsers* populationCount * maxNumberOfGenerations
32	1	populationCount * maxNumberOfGenerations	populationCount * maxNumberOfGenerations
33	1	maxNumberOfGenerations	maxNumberOfGenerations
34	1	1	1
35	noOfItems	1	noOfItems
36	2	populationCount + 1	2 * (populationCount + 1)

37	noOfItems * noOfItems (de la FitnessScore)	populationCount	populationCount * noOfItems * noOfItems
38	noOfUsers * noOfItems (de la FitnessScore)	populationCount * t1(n) ori (depinde daca cel mai mare fitnessScore e detinut de primul individ din population sau de ultimul)	noOfUsers * noOfItems * populationCount * t1(n)
39	noOfItems (asignarea de vector)	populationCount * t1(n)	noOfItems * populationCount * t1(n)
40	noOfItems	1	noOfItems
41	1	1	1
42	1	alreadyGivenRatings.length	alreadyGivenRatings.length
43	1	alreadyGivenRatings.length	alreadyGivenRatings.length
44	1	1	1
45	1	1	1
46	1	noOfItems + 1	noOfItems + 1
47	alreadyGivenRatings.length	noOfItems	alreadyGivenRatings.length * noOfItems
48	1	(noOfItems - alreadyGivenRatings.length) se va executa sigur de atâtea ori pentru că în acel vector cu dimensiune noOfItems știm sigur că avem alreadyGivenRatings.length elemente la care deja am dat o notă și nu mai e nevoie să verificăm dacă e soluția cu scorul maxim	noOfItems - alreadyGivenRatings.length
49	1	noOfItems - alreadyGivenRatings.length	noOfItems - alreadyGivenRatings.length
50	1	noOfItems - alreadyGivenRatings.length	noOfItems - alreadyGivenRatings.length
51	1	1	1

Tabelul 11. Complexitatele funcției Recommend

Unde:

$$\begin{aligned} t1(n) &= 0 \text{ dacă primul individ din populație are fitness score-ul maxim} \\ &= \text{populationCount} \text{ dacă ultimul individ din populație are fitness score-ul maxim} \end{aligned}$$

Observăm că operația care se execută de cele mai multe ori este 26 cu : noOfItems * populationCount * maxNumberOfGenerations + noOfItems * noOfUsers * populationCount * maxNumberOfGenerations, deci :

$$T(n) = \text{noOfItems} * \text{noOfItems} * \text{populationCount} * \text{maxNumberOfGenerations}$$

Dar, în codul nostru depindem și de niște valori de random, cum observăm la operațiile : 14 și 17 care depind de k1, k2 sau k3. Aceste variabile, k1, k2 și k3 semnifică că acele operații se vor repeta de câte ori este nevoie depinzând de valorile generate random. Totuși, în codul nostru se generează valori random între 1 și 10, deci șansele sunt foarte mici ca acestea să fie mai mari decât operația dominantă.

Complexitățile cazurilor favorabile și defavorabile depind de $t1(n)$ (la cel defavorabil pentru operația 33 va fi complexitatea $\text{noOfItems} * \text{populationCount} * \text{populationCount}$, iar la cel favorabil doar $\text{noOfItems} * \text{populationCount}$), dar și de random, în care îi putem pune lui cazurilor favorabile și valorile care depind de random că sunt nimerite din prima numere diferite, deci pentru operațiile 14 și 17 complexitatea de $\text{populationCount} * \text{maxNumberOfGenerations} * 1$ iar pentru 15 și 18, complexitatea de 0. Totuși, pentru cele nefavorabile nu putem determina o complexitate deoarece nu știm de câte ori poate genera valori eronate acel random și trebuie reînăscut. Deci, limita superioară nu se poate calcula.

Complexitatea întregului program este dată de această funcție Recommend care se folosește de toate funcțiile făcute.

6. Demonstrația corectitudinii

6.1. Demonstrația corectitudinii funcției de citire a populației

- *Precondiții:*
 - presupunem că în fișierul pe care îl citim sunt suficiente linii pentru a ne popula toată matricea de populație și că pe fiecare linie din el este câte o valoare de tip double
 - Presupunem că $\text{populationCount} > 0$ (număr natural valid)
 - Presupunem că $\text{noOfItems} > 0$ (număr natural valid)
- *Postcondiții:*
 - La final vom avea o matrice de double inițializată cu valori din fișier
- *Adnotarea funcției:*

```

private static double[][] ReadPopulation(int populationCount, int noOfItems)
{
    StreamReader sr = new StreamReader("Population.txt");//se atribuie fisierul streamreader-ului care are
    //rolul de a citi din fisier
    List<double> lstOfValues2 = new List<double>();//se initializeaza lista in care vor adauga valorile parase din fisier
    String line = "";//se porneste de la un string care reprezinta linia din fiecare fisier

    while ((line = sr.ReadLine()) != null)//citim toate liniile din fisier
    {
        lstOfValues2.Add(Double.Parse(line));//adaugam la lista de valori, valoarea de pe linia curenta transformata in double
    }

    int p = 0;//se initializeaza p cu 0 care este contorul pozitiei la care ne aflam pentru elementele din lista ce cuprinde
    //toate valorile din fisier
    double[][] population = new double[populationCount][];//se initializeaza matricea de populatie care trebuie sa aiba
    //in total populationCount numar de linii = numarul de indivizi din populatie
    for (int i = 0; i < populationCount; i++)//iau pe rand fiecare individ din populatie
    {
        population[i] = new double[noOfItems];//initializez individul din populatie de pe pozitia i
        for (int j = 0; j < noOfItems; j++)//pentru fiecare element din individ
        {
            population[i][j] = lstOfValues2.ElementAt(p) //initializez elementul de pe pozitia curenta din populatie cu elementul p
            //din vectorul cu toate valorile intrucat ele ar trebui sa fie in ordinea corecta in fisier

            //we do this because we extracted all the values from the file
            //in a vector and then we have to generate the matrix that represents the initial population
            p++;//incrementez p
        }
    }
    return population;//returnez populatia
}

```

Figura 8. Adnotarea funcției de ReadPopulation

La final se returnează populația care reprezintă o matrice cu valori luate din fișier.

- *Pașii de prelucrare:* se asigură că de la precondiții ajungem la postcondiții

```


    /// <summary>
    /// This is the method that reads from file the population with randomly generated value
    /// </summary>
    /// <param name="populationCount"></param> is how many individuals we have in the population
    /// <param name="noOfItems"></param> is how many items we have in store , basically the length of one individual
    /// <returns></returns>
    /// preconditii: presupunem ca in fisier sunt cate linii avem nevoie si ca pe o linie se afla o valoare de tip double
    1 reference
private static double[][] ReadPopulation(int populationCount, int noOfItems)
{
    { P }
    { P0 }
    A0: StreamReader sr = new StreamReader("Population.txt");
    { P1 }
    A1: List<double> lstOfValues2 = new List<double>(); //=I1 (INVARIANT IN BUCLA DE WHILE DE MAI JOS)
    { P2 }
    A2: String line = "";
    { P3 }
    A3: while ((line = sr.ReadLine()) != null) //=C1 (=conditia 1)
    {
        { P4 }
        A4: lstOfValues2.Add(Double.Parse(line));
    }
    { P5 }
    A5: int p = 0;
    { P6 }
    A6: double[][] population = new double[populationCount][]; //=I2 (INVARIANT IN BUCLA DE WHILE DE MAI JOS)
    { P7 }
    //for (int i = 0; i < populationCount; i++)
    A7: int i = 0;
    { P8 }
}


```

Figura 9. Etichetarea operațiilor și a pașilor de prelucrare partea 1

```


    A8: while(i<populationCount)//C2 (=conditia 2)
    {
        { P9 }
        A9: population[i] = new double[noOfItems]; //=I3 (INVARIANT IN BUCLA DE WHILE DE MAI JOS)
        { P10 }
        A10: int j = 0;
        { P11 }
        A11: while( j < noOfItems)//C3 (=conditia 3)
        {
            { P12 }
            A12: population[i][j] = lstOfValues2.ElementAt(p) ;
            { P13 }
            A13: p++;
            { P14 }
            A14: j++;
        }
        { P15 }
        A15: i++;
    }
    { P16 }
    return population;
}
{ Q }


```

Figura 10. Etichetarea operațiilor și a pașilor de prelucrare partea 2

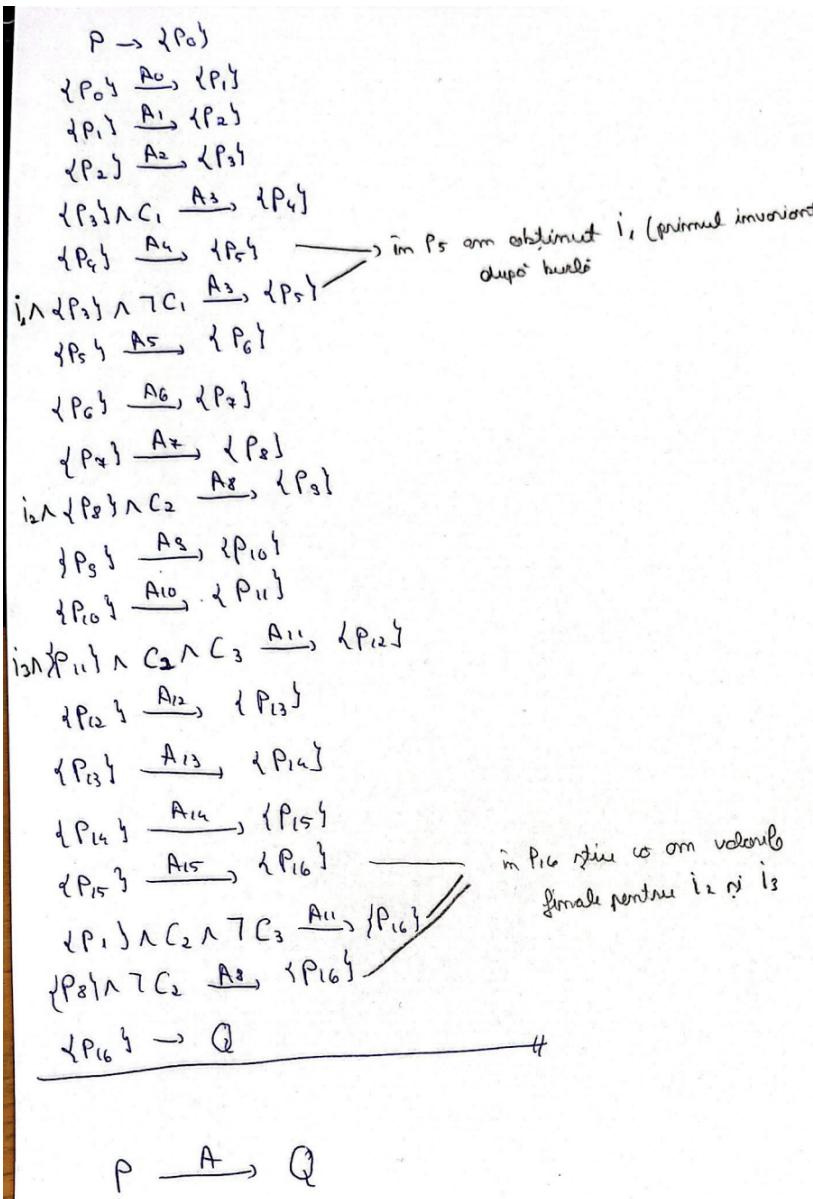


Figura 11. Descrierea formală

- *Invariante*: întrucât avem 3 bucle avem și 3 invariante

- I_1 = lista de valori extrase din fișier

La orice moment, aceasta va avea în ea valorile citite până atunci din fișier în format double. (demonstrația prin inducție și funcția de terminare se află mai jos, în imagini)

I. demonstratie prin inducție

I. Po: la început lînta este goală \Rightarrow ① dacă nu n-ai citit valori din fișier

II. Pp. $P(k) \oplus \Rightarrow$ dñm $P(k+1)$, $0 \leq k \leq (\text{nr. Linii din fișier} - 1)$

$P(k)$: Pp. că la k linie din fișier avem în lînta mecanic toate lîniile din acesta, puncte nul formă de duble

$P(k+1)$: while ((line = nr. ReadLine ()) != null) - trase de while decoune $k+1 < \text{nr. Linii din fișier}$

{ list k.Values.Add (Double.Parse (line)) }

Linie adaugă elementul de pe linia $k+1$ din fișier nul formă de duble

$\Rightarrow P(k+1)$

Dacă $\boxed{P(k) \Rightarrow P(k+1)}$

Funcție de terminare

$F(k) = \text{nr. Linii Fișier} - k + 1$; $k \in [0; \text{nr. Linii din Fișier}]$

este menționat deoarece

ajunge la 0 pentru $k = \text{nr. Linii Fișier} + 1$

Se execută pînă când nu mai sunt lîniile populate în fișier, atunci funcția de terminare va fi 0.

Figura 12. Demonstrația invariantului 1 și funcția de terminare a buclei

- I_2 = matricea corespunzătoare populației

În orice moment, aceasta va avea toate elementele populate cu ajutorul invariantului 3 care se află în bucla corespunzătoare invariantului 2 (demonstrația acestuia este sub aceasta)

I_2 demonstratie prin inducție

I_2 = matricea populației

I. Po: la început orde goală decoune nu om adaugă încă niciun individ ①

II. Pp. $P(k) \oplus \text{dñm. } P(k+1)$

$\bullet P(k) \oplus =$ la $i = k$ în populație matricea va fi întărită cu toate elementele din lîntă

$P(k+1)$: la $i = k+1$ în populație matricea va fi întărită cu toate elementele din lîntă

pînă la poziția k

$P(k+1)$: dacă în nr. k populării se adaugă un nou individ, pînă la k , la nr. $k+1$ adăugăm noul

de elem. pînă la $k+1 \Rightarrow$ dacă vom avea toate elem. pînă la $k+1$

(aici presupunem că dñm. pt i_2 ne căre o facem mai greu să se aducă înstă)

Figura 12. Demonstrația invariantului 2

Funcția de terminare
 $F(i) = \text{populationCount} - i + 1, i \in [0; \text{populationCount}]$

Este monotonă descrescătoare
 ajunge la 0 pentru $i = \text{populationCount}$

Figura 14. Funcția de terminare a buclei corespondente invariantului 2

- I3 = individul din populație de la poziția i

Acesta în buclă va fi populat mereu incremental cu câte o valoare din fișier.

Figura 15. Demonstrația invariantului 3 și funcția de terminare a buclei

6.2. Demonstrația funcției de încrucișare

- *Precondiții:*
 - Punctul de divizare să fie între 0 și noOfItems
 - $\text{noOfItems} > 0$
 - Matricea de indivizi să nu aibă dimensiuni nule

- F să fie o valoare în intervalul $(0,2]$
- $\text{Rand}(\text{random}-\text{ul})$ este inițializat
- *Postcondiții:*
 - Să îmi returneze un individ potențial pentru populația mea
- *Adnotarea funcției:*

```


    /// <summary>
    /// This is the function that will perform the crossover which will be either
    /// initialized with the mutation value or the value of the primary individual.
    /// </summary>
    /// <param name="punctDivizare"></param> is the division point used so we don't risk ending up with no mutation at all
    /// <param name="noOfItems"></param> is the number of items in the store
    /// <param name="CR"></param> is the crossover rate, a probability with which we will perform the mutation
    /// <param name="indivizi"></param> are the individuals from that population
    /// <param name="F"></param> is the amplification factor used in mutation
    /// <param name="rand"></param> is the random generator
    /// <returns></returns>
    1 reference
    public static double[] Crossover(int punctDivizare, int noOfItems, int CR, double[][] indivizi, double F, Random rand, List<int> listOfAlreadyRatedItems)
    {
        double[] individuPotential = new double[noOfItems]; //initializam individul potential care este un vector, la inceput este gol
        int gena = 0; //initializam gena cu zero

        while(gena < noOfItems) //cat timp valoarea genei este mai mica decat noOfItems
        {
            if (gena == punctDivizare || rand.Next(0, 100) < CR) //daca valoarea genei este egala cu punctul de divizare
                //sau daca o valoare random cuprinsa intre 0 si 100 este mai mica decat CR
            {
                individuPotential[gena] = Mutation(indivizi, gena, F); //individul potential va avea valoare mutatiei dintre indivizii
                //primiti ca parametru luandu-se in considerare valoarea genei si cu factorul de amplificare F
            }
            else //in caz contrar, daca valoarea genei nu este egala cu punctul de divizare si valoarea random generata este mai mare sau egala cu CR
            {
                individuPotential[gena] = indivizi[0][gena]; //individul potential este gena primului individ
            }
            gena++; //se incrementeaza valoarea genei pentru a le parurge pe toate
        }
        return individuPotential; //se returneaza vectorul reprezentat de individul potential
    }


```

Figura 16. Adnotarea funcției de crossover

- *Pașii de prelucrare:* se asigură că de la precondiții ajungem la postcondiții

```


    /// This is the function that will perform the crossover which will be either
    /// initialized with the mutation value or the value of the primary individual.
    /// </summary>
    /// <param name="punctDivizare"></param> is the division point used so we don't risk ending up with no mutation at all
    /// <param name="noOfItems"></param> is the number of items in the store
    /// <param name="CR"></param> is the crossover rate, a probability with which we will perform the mutation
    /// <param name="indivizi"></param> are the individuals from that population
    /// <param name="F"></param> is the amplification factor used in mutation
    /// <param name="rand"></param> is the random generator
    /// <returns></returns>
    1 reference
    public static double[] Crossover(int punctDivizare, int noOfItems, int CR, double[][] indivizi, double F, Random rand, List<int> listOfAlreadyRatedItems)
    {
        { P0 }
        A1: double[] individuPotential = new double[noOfItems]; //I = invariantul din bucla

        { P1 }

        //for (int gena = 0; gena < noOfItems; gena++)

        A2: int gena = 0;

        { P2 }

        A3: while(gena < noOfItems) //c1 = conditia ca gena < noOfItems
        {
            { P3 }

            A4: if (gena == punctDivizare || rand.Next(0, 100) < CR) //c2 : gena = punctDivizare sau rand.Next(0,100)<CR
            {
                { P4 }

                A5: individuPotential[gena] = Mutation(indivizi, gena, F);
            }
            else //C2 NEGAT
            {
                { P5 }
                A6: individuPotential[gena] = indivizi[0][gena];
            }
            { P6 }
            A7: gena++;
            { P7 }

        }
        A8: return individuPotential;
        { P8 }
    }


```

Figura 17. Etichetarea operațiilor și a pașilor de prelucrare

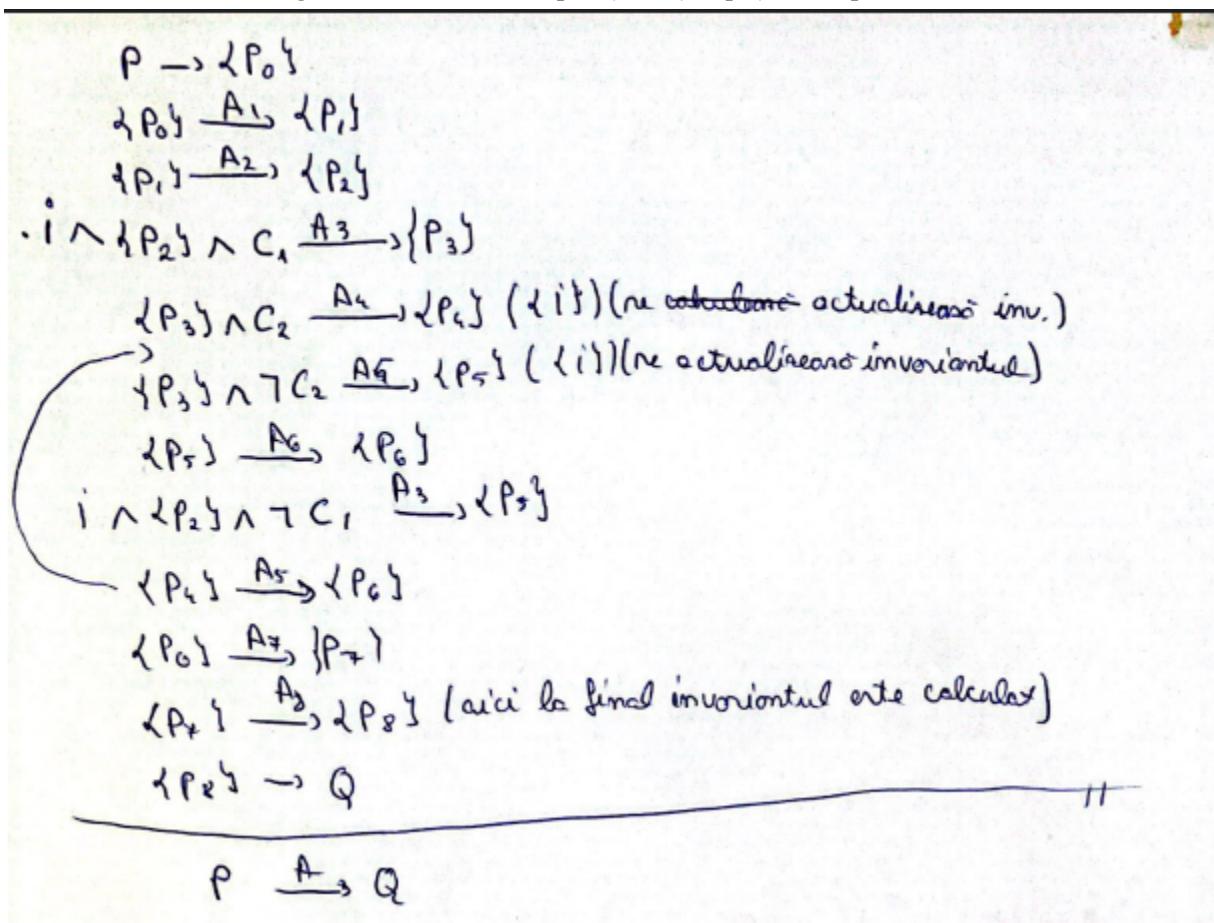


Figura 18. Descrierea formală

- *Invariantul*: în cazul nostru avem o singură buclă iar invariantul pentru aceasta este individul *Potential* care își va păstra semnificația și va fi inițializat pe tot parcursul buclei

Dem. invariант

i = individ potential (vector)

$\mathbb{I} P(0)$: la început este null deci nu î-a făcut deloc nicio vd. \textcircled{A}

$\mathbb{II} P(k) \rightarrow P_{k+1}$ dem $P(k+1)$

$\bullet P(k)$: număr. k număr ^{num} individ potential de ~~pătrat~~ ~~stare~~

individ potential $[k] = \begin{cases} \text{Mutation}(\dots), & k = \text{punct divizion} (1 \text{ vcl random} \in \mathbb{R}) \\ \text{individ}[0][k], & \text{în cas contrac} \end{cases}$

\textcircled{A} (mai dem. practic că la fiecare pas urmărește un individ potential)

$P(k+1)$: individ potential $[k+1] = \begin{cases} \text{Mutation}(\dots), & k+1 = \text{pct. diviz. 11 vcl. random} \in \mathbb{R} \\ \text{individ}[0], & \text{în cas contrac} \end{cases}$

Practic $P(k)$ înnă spune că \mathbb{A} -a găsit pătrat atunci căto un individ potential

pt. individ potential $[i]$, $0 \leq i \leq k$

Obs. că nu la $P(k+1)$ ne alege un individ ~~potential~~

$$\Rightarrow P_k = P_{k+1} \text{ } \textcircled{A}$$

Figura 19. Demonstrația invariантului

Funcția de terminare

$$F(\text{gama}) = m0 \{ \text{item} - \text{gama} + 1 \} \rightarrow \text{unde } 0 \text{ pentru } \text{gama} = m0 \{ \text{item} \}$$

Figura 20. Funcția de terminare

7. Testarea programului

Interfața a fost testată inițial manual. Cu verde sunt trecute testele care au trecut.

7.1. Teste pentru prima fereastră:

Precondiții: Aplicația este lansată în execuție. Există produse adăugate la lista de produse din aplicație.

- Este afișată o listă cu toate produsele
- Oricare dintre produse poate fi selectat



Figura 21. Cazul în care sunt selectate toate produsele

- Nu se poate trece la fereastra următoare decât dacă au fost selectate exact 5 produse, nici mai mult, nici mai puțin. (Butonul Next este activ/inactiv)

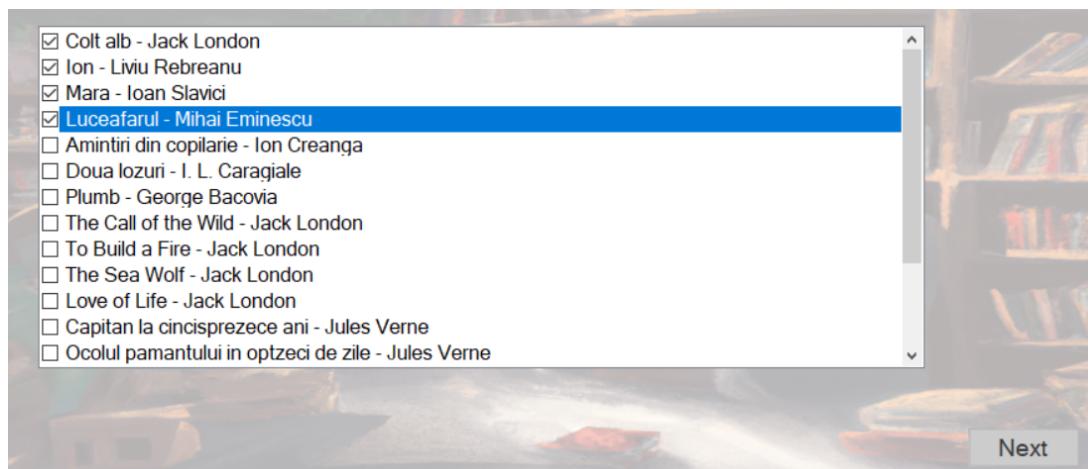


Figura 22. Demonstrația că dacă avem mai puțin de 5 produse, nu se poate apăsa pe Next

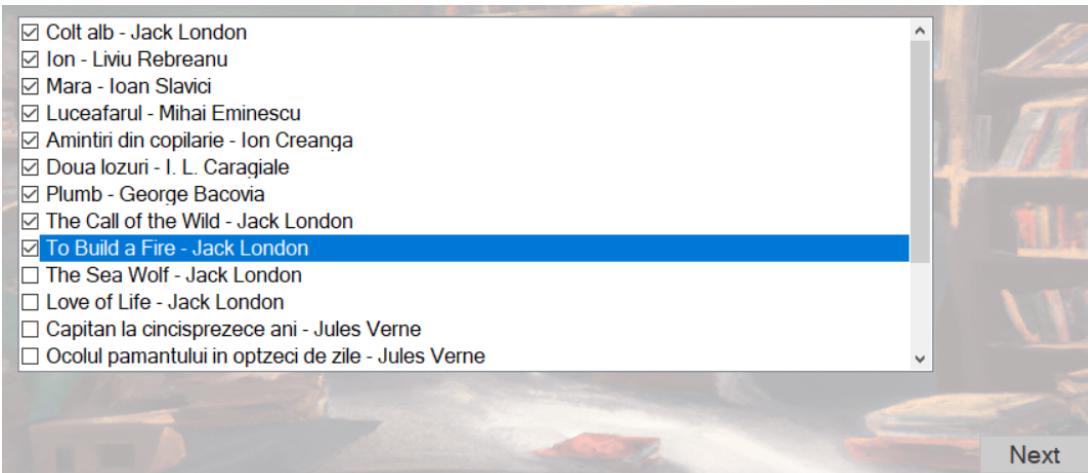


Figura 23. Demonstrația că dacă avem mai mult de 5 produse, nu se poate apăsa pe Next

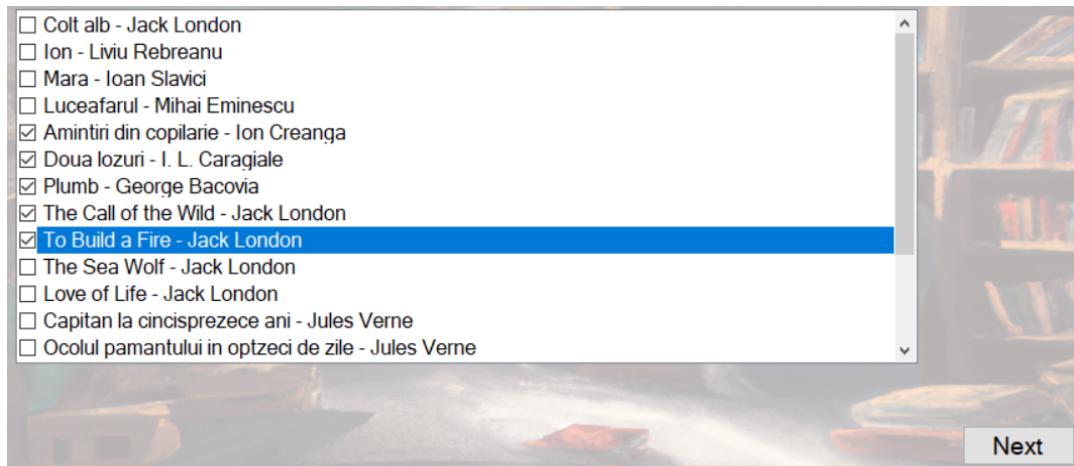


Figura 24. Demonstrația că dacă am selectat 5 produse, se poate apăsa pe Next

7.2. Teste pentru a doua fereastră:

Precondiții: Au fost selectate exact 5 produse din lista de produse de pe prima pagină (Amintiri din copilarie, Doua lozuri, Plumb, The call of the wild, To build a fire). La apăsarea butonului Next se trece la a doua fereastră.

- Cele 5 produse selectate sunt afișate în ordinea selectării și în dreptul lor există un sistem de notare a produselor.



Figura 25. Exemplificare a cum arată fereastra a doua cu produsele selectate din prima fereastră

- Sistemul de notare nu acceptă decât valori de la 1 la 5

S-a concluzionat că nu pot fi introduse decât caractere numerice, respectiv caracterele necesare pentru a scrie numere reale (caracterul punct și caracterul virgulă).

Dacă sunt introduse valori mai mici decât 1, sistemul va reține valoarea 1.

Dacă sunt introduse valori mai mari decât 5, sistemul va reține valoarea 5.

Dacă sunt introduse valori reale, acestea sunt rotunjite la valoarea lor întreagă. Dacă partea fracționară este mai mică decât 5, se face rotunjire prin lipsă. Dacă partea fracționară este mai mare sau egală cu 5, se face rotunjire prin adaos.

Dacă este introdus doar caracterul punct, respectiv doar caracterul virgulă, sistemul va reține valoarea 1.

Dacă este introdus caracterul virgulă alături de caractere numerice, sistemul va reține valoarea 5.

Nu apar excepții neratate.

7.3. Teste pentru a treia fereastră:

Precondiții: La apăsarea butonului Next de pe a doua fereastră se trece la a treia fereastră.

- Sunt afișate produsele alături de imagini reprezentative pentru fiecare și bara de scroll permite vizualizarea tuturor produselor.

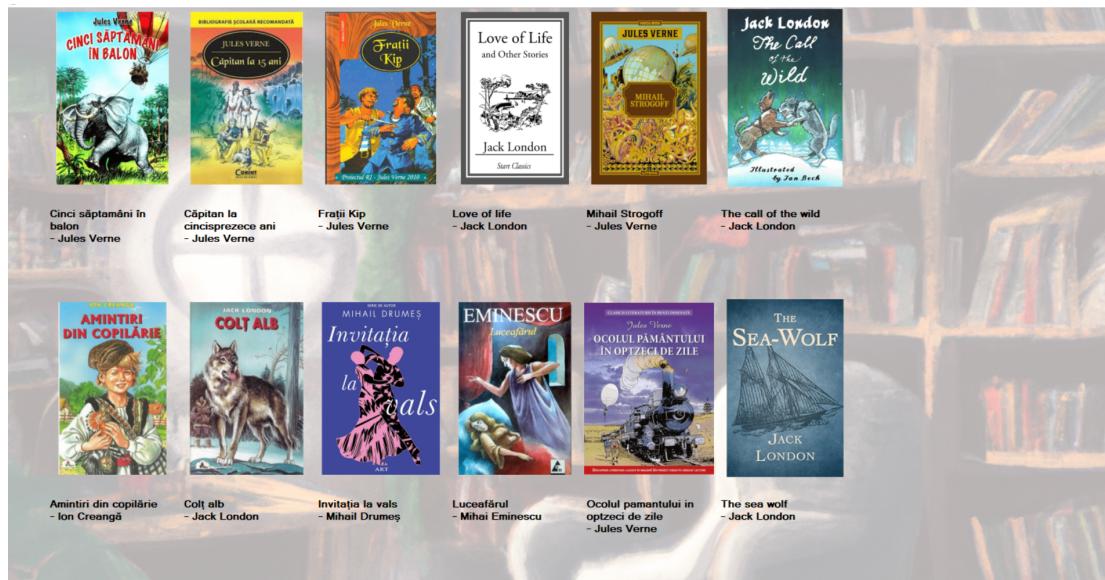


Figura 26. Afisarea produselor pe Form 3

- Butonul “Recomandare de carte” este activ, iar la apăsarea lui se face o recomandare de produs.

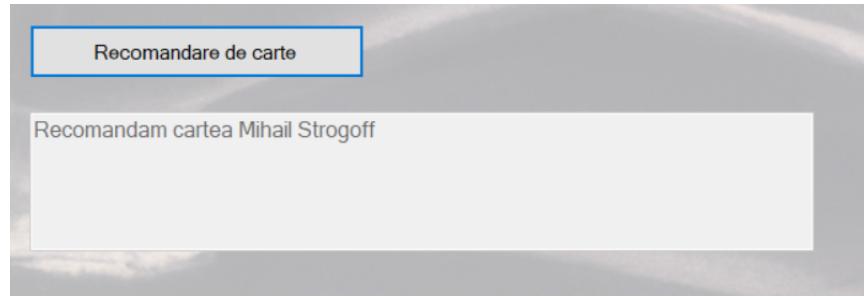


Figura 27. Afişarea butonului de recomandare de carte pentru a demonstra că este activ

Pentru partea de testare automată a aplicației au fost efectuate 18 teste.

1. TestMutation1()

Testează funcția Mutation ce primește ca parametri o matrice de tip double, un int și un double. Pentru valorile setate în funcția de test se va obține:

$\text{indivizi}[3][\text{gena}] + F * (\text{indivizi}[1][\text{gena}] - \text{indivizi}[2][\text{gena}]) = \text{indivizi}[3][1] + 1 * (\text{indivizi}[1][1] - \text{indivizi}[2][1]) = 5,3 + 1,3 - 4,3 = 2,3$.

Deci valoarea așteptată este 2,3.

```
[TestMethod]
public void TestMutation1()
{
    double[] value = { 1.1d, 4.2d, 5.2d, 9.7d };
    double[] value1 = { 9.0d, 1.3d, 7.8d, 2.8d };
    double[] value2 = { 7.6d, 4.3d, 8.8d, 2.9d };
    double[] value3 = { 1.8d, 5.3d, 8.3d, 9.9d };
    double[][] indivizi = { value, value1, value2, value3 };

    double F = 1.0d;
    int gena = 1;

    Assert.AreEqual(2.3d, BookShopGUI.Rezolvare.Mutation(indivizi, gena, F));
}
```

Figura 28. Testul pentru mutation

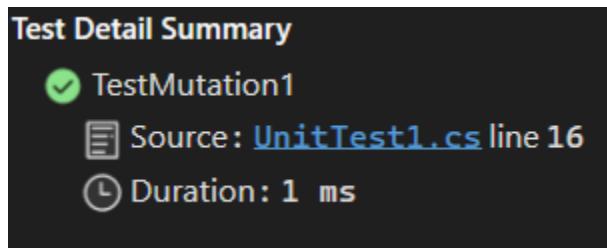


Figura 29. Rezultatul testului

2. TestGenerateUserRatings1()

Testează funcția GenerateUserRatings ce primește ca parametru numele fișierului, de tip String, în care vor fi generate valorile. Funcția de test verifică faptul că fișierul de test există și are conținut.

```
[TestMethod]

✓ | 0 references
public void TestGenerateUserRatings1()
{
    BookShopGUI.Rezolvare.GenerateUserRatings("TrialUsersRatings.txt");
    StreamReader file = new StreamReader("TrialUsersRatings.txt");
    Assert.IsNotNull(file.ReadLine());
}
```

Figura 30. Testul pentru generarea de user ratings

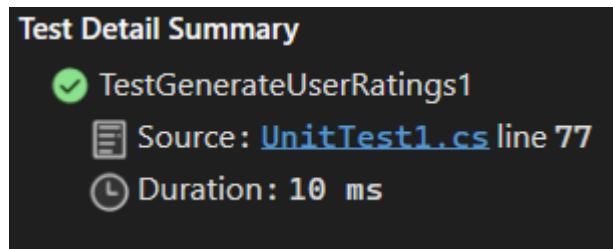


Figura 31. Rezultatul testului

3. TestGenerateUserRatings2()

Testează tot funcția GenerateUserRatings. Însă de această dată se verifică faptul că numărul valorilor generate în interiorul fișierului este corect și anume 180.(noi pornim de la 10

useri și 18 produse, deci $10*18=180$)

```
[TestMethod]
0 | 0 references
public void TestGenerateUserRatings2()
{
    //BookShopGUI.Rezolvare.GenerateUserRatings();
    StreamReader file = new StreamReader("TrialUsersRatings.txt")
    String line = "";
    List<double> lstOfValues = new List<double>();
    while ((line = file.ReadLine()) != null)
    {
        lstOfValues.Add(Double.Parse(line));
    }
    Assert.AreEqual(180, lstOfValues.Count);
}
```

Figura 32. Testul 2 pentru generarea de user ratings

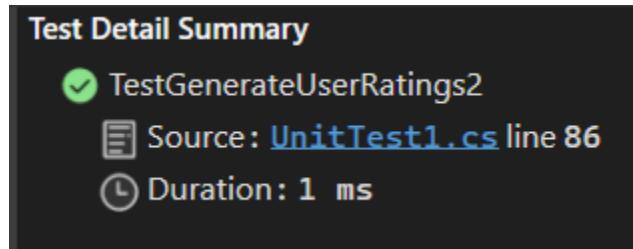


Figura 33. Rezultatul testului

4. TestGenerateUserRatingsValuesBetween1And5()

Testează tot funcția GenerateUserRatings. Însă de această dată se verifică faptul că valorile generate în interiorul fișierului sunt cuprinse între 1 și 5.

```

[TestMethod]
public void TestGenerateUserRatingsValuesBetween1And5()
{
    //BookShopGUI.Rezolvare.GenerateUserRatings();
    StreamReader file = new StreamReader("TrialUsersRatings.txt");
    String line = "";
    List<double> lstOfValues = new List<double>();
    while ((line = file.ReadLine()) != null)
    {
        lstOfValues.Add(Double.Parse(line));
    }
    for(int i = 0; i < lstOfValues.Count; i++)
    {
        Assert.IsTrue(lstOfValues[i]>= 1 && lstOfValues[i] <= 5);
    }
}

```

Figura 34. Testul 3 pentru generarea de user ratings

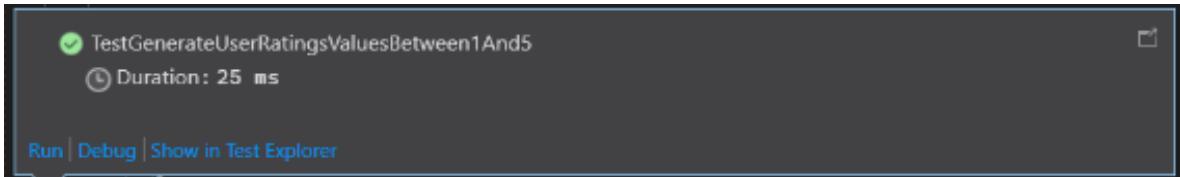


Figura 35. Rezultatul testului

5. TestGenerateInitialPopulation1()

Testează funcția GenerateInitialPopulation. Se verifică faptul că fișierul generat există și nu este gol.

```

[TestMethod]
public void TestGenerateInitialPopulation1()
{
    List<KeyValuePair<int, double>> alreadyGivenRatings = new List<KeyValuePair<int, double>>()
    {
        new KeyValuePair<int, double>(1,1),
        new KeyValuePair<int, double>(2,1),
        new KeyValuePair<int, double>(3,1),
        new KeyValuePair<int, double>(4,1),
        new KeyValuePair<int, double>(5,1),
    };
    int noOfItems = 18;
    int populationCount = 10;
    BookShopGUI.Rezolvare.GenerateInitialPopulation(alreadyGivenRatings, noOfItems, populationCount, "TrialPopulation1.txt");
    StreamReader file = new StreamReader("TrialPopulation1.txt");
    Assert.IsNotNull(file.ReadLine());
}

```

Figura 36. Testul pentru verificarea că fișierul nu este gol

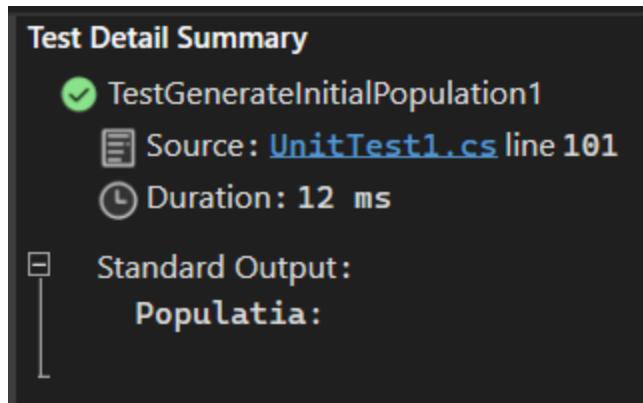


Figura 37. Rezultatul testului

6. TestGenerateInitialPopulation2()

Testează tot funcția GenerateInitialPopulation. Însă aici este verificat faptul că numărul valorilor generate este corect(180).

```
[TestMethod]
● | 0 references
public void TestGenerateInitialPopulation2()
{
    StreamReader file = new StreamReader("TrialPopulation1.txt");
    String line = "";
    List<double> lstOfValues = new List<double>();
    while ((line = file.ReadLine()) != null)
    {
        lstOfValues.Add(Double.Parse(line));
    }
    Assert.AreEqual(180, lstOfValues.Count);
}
```

Figura 38. Testul pentru verificarea că fișierul are câte valori trebuie

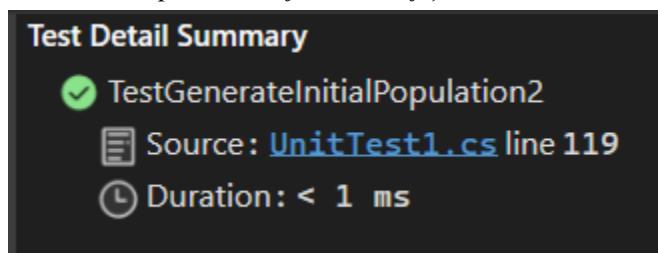


Figura 39. Rezultatul testului

7. TestGenerateInitialPopulationBetween1And5()

Testează tot funcția GenerateInitialPopulation. Însă de data aceasta este verificat faptul că valorile generate în fișier sunt cuprinse între 1 și 5.

```
[TestMethod]
0 | 0 references
public void TestGenerateInitialPopulationBetween1And5()
{
    StreamReader file = new StreamReader("TrialPopulation1.txt");
    String line = "";
    List<double> lstOfValues = new List<double>();
    while ((line = file.ReadLine()) != null)
    {
        lstOfValues.Add(Double.Parse(line));
    }
    for (int i = 0; i < lstOfValues.Count; i++)
    {
        Assert.IsTrue(lstOfValues[i] >= 1 && lstOfValues[i] <= 5);
    }
}
```

Figura 40. Testul 3 pentru generarea populației inițiale

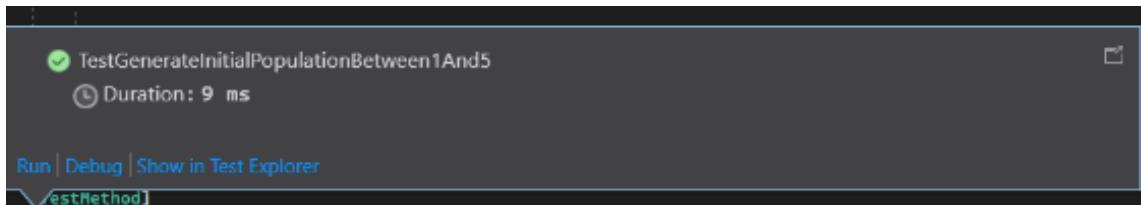


Figura 41. Rezultatul testului

8. TestReadUserRatings()

Testează funcția ReadUsersRatings ce primește ca parametri numărul de useri, numărul de produse și numele fișierului din care se face citirea și returnează o matrice ce conține ratingurile userilor. Se verifică faptul că citirea s-a efectuat corect și matricea nu este nulă.

```
[TestMethod]
0 | 0 references
public void TestReadUserRatings()
{
    int noOfUsers = 10;
    int noOfItems = 18;
    double[][] userRatings = BookShopGUI.Rezolvare.ReadUsersRatings(noOfUsers, noOfItems, "TrialUsersRatings.txt");
    Assert.IsNotNull(userRatings);
}
```

Figura 42. Testul pentru verificarea că citirea din fișier nu returnează null

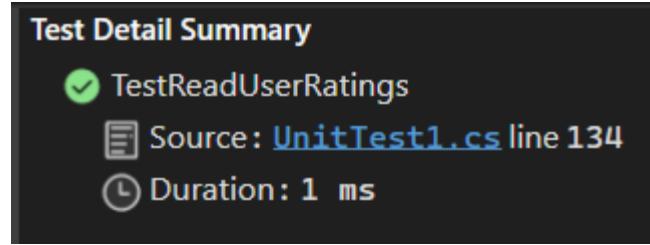


Figura 43. Rezultatul testului

9. TestReadPopulation()

Testează funcția ReadPopulation și verifică faptul că citirea din fișier s-a efectuat corect, iar matricea returnată de funcție nu este nulă.

```
[TestMethod]
● 0 references
public void TestReadPopulation()
{
    int populationCount = 10;
    int noOfItems = 18;
    double[][] userRatings = BookShopGUI.Rezolvare.ReadPopulation(populationCount, noOfItems);
    Assert.IsNotNull(userRatings);

}
```

Figura 44. Testul pentru verificarea că citirea din fișier nu returnează null

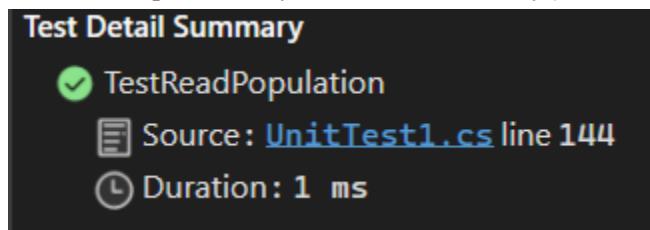


Figura 45. Rezultatul testului

10. TestCosineSimilarityV()

Testează funcția CosineSimilarityV. Pentru valorile oferite în funcția de test calculul s-a efectuat cu ajutorul formulei de mai jos astfel:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| * \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} * \sqrt{\sum_{i=1}^n B_i^2}}$$

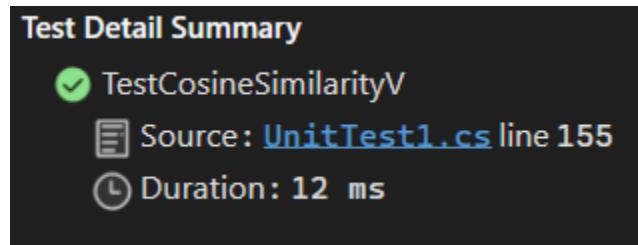
similarity=(4+2*5+3*6)/sqrt(1+2*2+3*3)*sqrt(4*4+5*5+6*6)=32/sqrt(14)*sqrt(77)=32/32.83291=0.97463

```

[TestMethod]
0 | 0 references
public void TestCosineSimilarityV()
{
    double[] v1 = { 1, 2, 3 };
    double[] v2 = { 4, 5, 6 };
    Assert.AreEqual(0.97463, BookShopGUI.Rezolvare.GetCosineSimilarityV(v1, v2), 1e-5);
}

```

Figura 46. Testul pentru verificarea scorului de similitudine



11. TestFitnessScore()

Testează funcția de fitness a algoritmului. Semnifică faptul că pentru o matrice cu valorile din weights, atunci când se calculează similaritatea dintre fiecare linie a acesteia cu vectorul userWeights, cel mai mare scor obținut a fost -0.99922. Calculele s-au făcut calculând similaritatea cosinus dintre fiecare linie și userWeights() și apoi alegând maximul.

Figura 47. Rezultatul testului

```

[TestMethod]
0 | 0 references
public void TestFitnessScore()
{
    double[][] weights = {
        new double[] { 1, 2, 3 },
        new double[] { 4, 5, 6 },
        new double[] { 7, 8, 9 }
    };

    double[] userWeights = { 3, 4, 5 };
    Assert.AreEqual(-0.99922, BookShopGUI.Rezolvare.FitnessScore(weights, userWeights), 1e-5);
}

```

Figura 48. Testul funcției de fitness

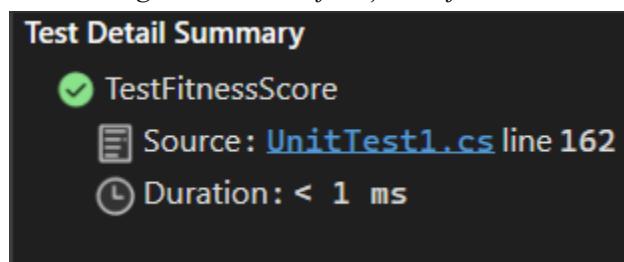


Figura 49. Rezultatul testului

12. TestSelection()

Testează faptul că dintre doi indivizi: cel potențial și cel existent se alege cel cu funcția de fitness cel mai mică (întrucât este o problemă de optimizare). Calculul pentru alegere s-a făcut calculând funcția de fitness pentru fiecare (testată la testul 11) și returnând individul cel mai potrivit.

```
[TestMethod]
● 0 references
public void TestSelection()
{
    double[][] userRatings={new double[] { 1, 2, 3 },
    new double[] { 4, 5, 6 },
    new double[] { 7, 8, 9 }

    double[] individPotential= { 3, 4, 5 };
    double[] individExistant = { 1, 2, 3 };
    double[] val = {1,2,3};
    Assert.AreEqual(val.ToString(), BookShopGUI.Rezolvare.Selection(userRatings, individPotential, individExistant).ToString());
}
```

Figura 50. Testul funcției de selecție

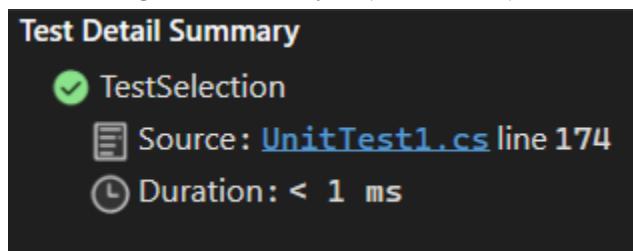


Figura 51. Rezultatul testului

13. TestCrossover()

În acest test verificăm doar că funcția returnează un vector de o lungime validă care trebuie să fie egală cu noOfItems deoarece nu putem verifica valoarea returnată exact, funcția depinzând de valori random.

```
[TestMethod]
● 0 references
public void TestCrossover()
{
    int noOfItems = 18;
    int populationCount = 10;
    int CR = 30;
    double F = 0.9;
    int punctDivizare = 1;
    Random rand = new Random();
    List<int> listOfAlreadyRatedItems = new List<int>();
    double[][] indivizi = BookShopGUI.Rezolvare.ReadPopulation(populationCount, noOfItems);
    double[] sol = BookShopGUI.Rezolvare.Crossover(punctDivizare, noOfItems, CR, indivizi, F, rand, listOfAlreadyRatedItems);
    Assert.AreEqual(noOfItems, sol.Length);
}
```

Figura 52. Testul funcției de Crossover

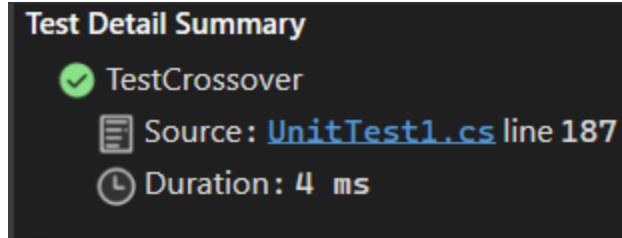


Figura 53. Rezultatul testului

14. TestCrossover2()

În acest test verificăm doar că funcția de crossover nu returnează un vector nul deoarece nu putem verifica valoarea returnată exact, funcția depinzând de valori random..

```
[TestMethod]
● | 0 references
public void TestCrossover2()
{
    int noOfItems = 18;
    int populationCount = 10;
    int CR = 30;
    double F = 0.9;
    int punctDivizare = 1;
    Random rand = new Random();
    List<int> listOfAlreadyRatedItems = new List<int>();
    double[][] indivizi = BookShopGUI.Rezolvare.ReadPopulation(populationCount, noOfItems);
    double[][] sol = BookShopGUI.Rezolvare.Crossover(punctDivizare, noOfItems, CR, indivizi, F, rand, listOfAlreadyRatedItems);
    Assert.IsNotNull(sol);
}
```

Figura 54. Testul functiei de Crossover

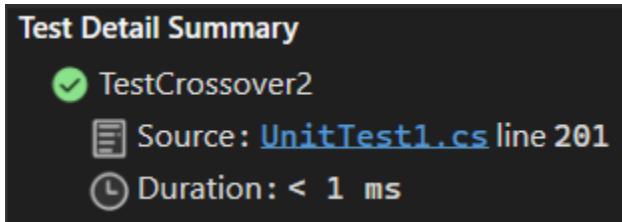


Figura 55. Rezultatul testului

15. TestRecommend()

În acest test am inițializat valorile din fișierul cu recomandările date de alți utilizatori în aşa fel încât să existe un user care are date note de 1 la toate produsele mai puțin la primul produs (Colț alb), unde are dată nota de 5. Toți ceilalți utilizatori au dat note maxime de 5. Astfel, suntem siguri că recomandarea pe care trebuie să o primim este pentru Colț Alb atunci când noul utilizator selectează alte 5 cărți decât Colț Alb și dă rating de 1 la toate.

```

[TestMethod]
0 | 0 references
public void TestRecommend()
{
    int noOfUsers = 10;
    int populationCount = 50;
    int maxNrOfGenerations = 6000;
    int CR = 30;
    double F = 0.9;
    int noOfItems = 18;
    double[][] userRatings = new double[noOfUsers][];
    List<KeyValuePair<int, double>> alreadyGivenRatings = new List<KeyValuePair<int, double>>()
    {
        new KeyValuePair<int, double>(1,1),
        new KeyValuePair<int, double>(2,1),
        new KeyValuePair<int, double>(3,1),
        new KeyValuePair<int, double>(4,1),
        new KeyValuePair<int, double>(5,1),
    };
    List<Carte> listOfProducts = new List<Carte>();
    listOfProducts.Add(new Carte(0, "Colt alb", "Jack London"));
    listOfProducts.Add(new Carte(1, "Ion", "Liviu Rebreanu"));
    listOfProducts.Add(new Carte(2, "Mara", "Ioan Slavici"));
    listOfProducts.Add(new Carte(3, "Luceafarul", "Mihai Eminescu"));
    listOfProducts.Add(new Carte(4, "Amintiri din copilarie", "Ion Creanga"));
    listOfProducts.Add(new Carte(5, "Doua lozuri", "I. L. Caragiale"));
    listOfProducts.Add(new Carte(6, "Plumb", "George Bacovia"));
    listOfProducts.Add(new Carte(7, "The Call of the Wild", "Jack London"));
    listOfProducts.Add(new Carte(8, "To Build a Fire", "Jack London"));
    listOfProducts.Add(new Carte(9, "The Sea Wolf", "Jack London"));
    listOfProducts.Add(new Carte(10, "Love of Life", "Jack London"));
}

```

Figura 56. Testul funcției de recommend partea 1

```

listOfProducts.Add(new Carte(11, "Capitan la cincisprezece ani", "Jules Verne"));
listOfProducts.Add(new Carte(12, "Ocolul pamantului in optzeci de zile", "Jules Verne"));
listOfProducts.Add(new Carte(13, "O calatorie spre centrul pamantului", "Jules Verne"));
listOfProducts.Add(new Carte(14, "Fratii Ki", "Jules Verne"));
listOfProducts.Add(new Carte(15, "Mihail Strogoff", "Jules Verne"));
listOfProducts.Add(new Carte(16, "Cinci saptamani in balon", "Jules Verne"));
listOfProducts.Add(new Carte(17, "Invitatie la vals", "Mihail Drumes"));

Assert.AreEqual("Colt alb", BookShopGUI.Rezolvare.Recommend(noOfItems, noOfUsers, maxNrOfGenerations, CR, F, userRatings, alreadyGivenRatings, populationCount));

```

Figura 57. Testul funcției de recommend partea 2

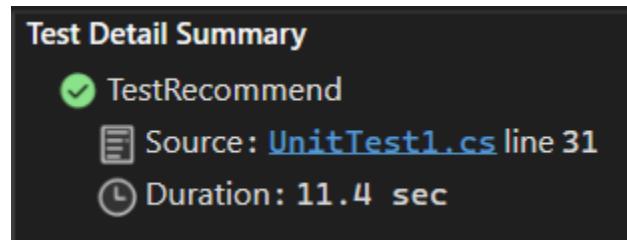
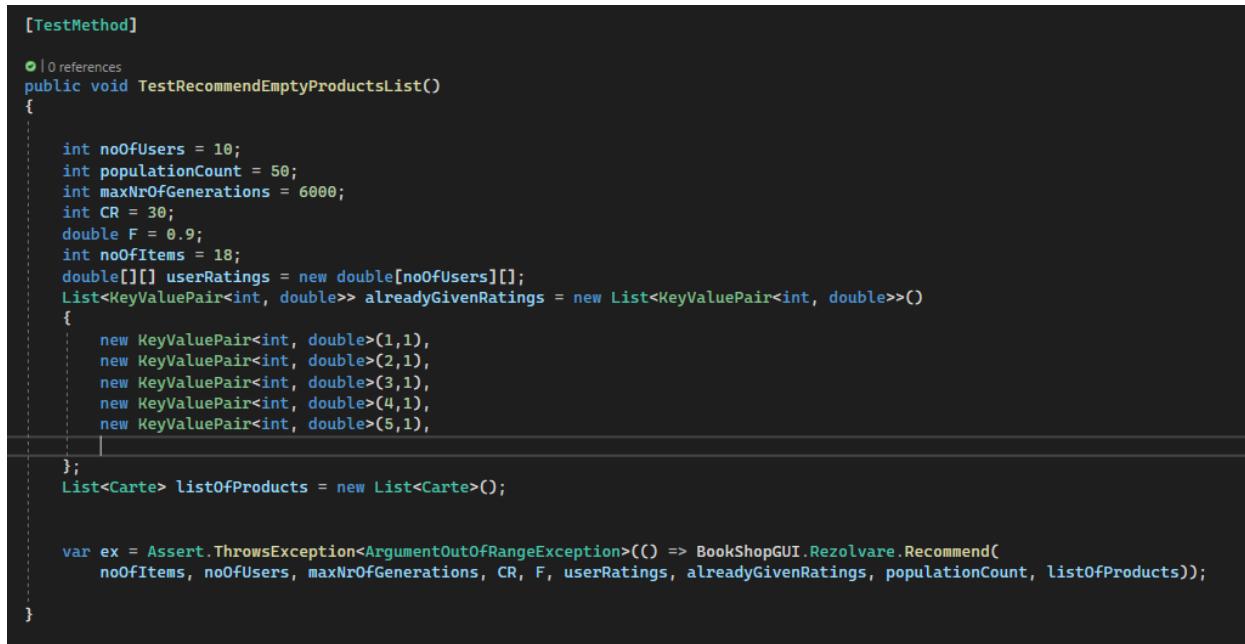


Figura 58. Rezultatul testului

16. TestRecommendEmptyProductList()

În acest test am verificat faptul că în cazul în care nu există produse în lista de produse(deci nu se poate face o recomandare) este aruncată o excepție.

6



```
[TestMethod]
public void TestRecommendEmptyProductsList()
{
    int noOfUsers = 10;
    int populationCount = 50;
    int maxNrOfGenerations = 6000;
    int CR = 30;
    double F = 0.9;
    int noOfItems = 18;
    double[][] userRatings = new double[noOfUsers][];
    List<KeyValuePair<int, double>> alreadyGivenRatings = new List<KeyValuePair<int, double>>()
    {
        new KeyValuePair<int, double>(1,1),
        new KeyValuePair<int, double>(2,1),
        new KeyValuePair<int, double>(3,1),
        new KeyValuePair<int, double>(4,1),
        new KeyValuePair<int, double>(5,1),
    };
    List<Carte> listOfProducts = new List<Carte>();

    var ex = Assert.ThrowsException<ArgumentOutOfRangeException>(() => BookShopGUI.Rezolvare.Recommend(
        noOfItems, noOfUsers, maxNrOfGenerations, CR, F, userRatings, alreadyGivenRatings, populationCount, listOfProducts));
}
```

Figura 59. Testul pentru cazul în care funcția ar trebui să arunce eroare

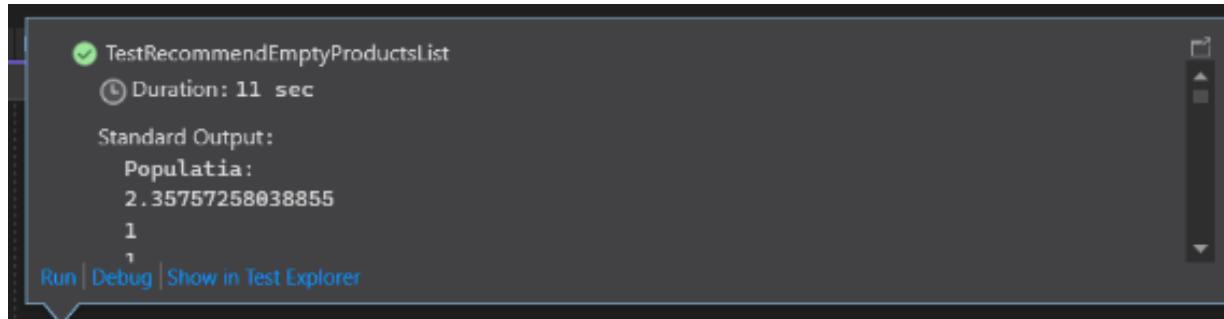


Figura 60. Rezultatul testului

17. TestRecommendDifferentRecommendations()

În acest test verificăm că dacă dăm un set de rating-uri la 5 produse obținem o recomandare diferită decât dacă dăm un alt set de rating-uri la aceleași produse.

```

[TestMethod]
0 | 0 references
public void TestRecommendDifferentRecommendations()
{
    int noOfUsers = 10;
    int populationCount = 50;
    int maxNrOfGenerations = 6000;
    int CR = 30;
    double F = 0.9;
    int noOfItems = 18;
    double[][] userRatings1 = new double[noOfUsers][];
    double[][] userRatings2 = new double[noOfUsers][];

    List<KeyValuePair<int, double>> alreadyGivenRatings1 = new List<KeyValuePair<int, double>>()
    {
        new KeyValuePair<int, double>(1,1),
        new KeyValuePair<int, double>(2,1),
        new KeyValuePair<int, double>(3,1),
        new KeyValuePair<int, double>(4,1),
        new KeyValuePair<int, double>(5,1),
    };
    List<KeyValuePair<int, double>> alreadyGivenRatings2 = new List<KeyValuePair<int, double>>()
    {
        new KeyValuePair<int, double>(1,5),
        new KeyValuePair<int, double>(2,5),
        new KeyValuePair<int, double>(3,5),
        new KeyValuePair<int, double>(4,5),
        new KeyValuePair<int, double>(5,5),
    };
    List<Carte> listOfProducts = new List<Carte>();
}

```

Figura 60. Testul pentru funcția recommend (ar trebui să obținem recomandări diferite) partea 1

```

};

List<Carte> listOfProducts = new List<Carte>();
listOfProducts.Add(new Carte(0, "Colt alb", "Jack London"));
listOfProducts.Add(new Carte(1, "Ion", "Liviu Rebreanu"));
listOfProducts.Add(new Carte(2, "Mara", "Ioan Slavici"));
listOfProducts.Add(new Carte(3, "Luceafărul", "Mihai Eminescu"));
listOfProducts.Add(new Carte(4, "Măintiri din copilarie", "Ion Creanga"));
listOfProducts.Add(new Carte(5, "Două lozuri", "I. L. Caragiale"));
listOfProducts.Add(new Carte(6, "Plumb", "George Bacovia"));
listOfProducts.Add(new Carte(7, "The Call of the Wild", "Jack London"));
listOfProducts.Add(new Carte(8, "To Build a Fire", "Jack London"));
listOfProducts.Add(new Carte(9, "The Sea Wolf", "Jack London"));
listOfProducts.Add(new Carte(10, "Love of Life", "Jack London"));
listOfProducts.Add(new Carte(11, "Capitan la cincisprezece ani", "Jules Verne"));
listOfProducts.Add(new Carte(12, "Ocolul pamantului în optzeci de zile", "Jules Verne"));
listOfProducts.Add(new Carte(13, "O călătorie spre centrul pamantului", "Jules Verne"));
listOfProducts.Add(new Carte(14, "Fratii Kî", "Jules Verne"));
listOfProducts.Add(new Carte(15, "Mihail Strogoff", "Jules Verne"));
listOfProducts.Add(new Carte(16, "Cinci săptămâni în balon", "Jules Verne"));
listOfProducts.Add(new Carte(17, "Invitație la vals", "Mihail Drumes"));

Assert.AreEqual(BookShopGUI.Rezolvare.Recommend(noOfItems, noOfUsers, maxNrOfGenerations, CR, F, userRatings1, alreadyGivenRatings1, populationCount, listOfProducts),
    BookShopGUI.Rezolvare.Recommend(noOfItems, noOfUsers, maxNrOfGenerations, CR, F, userRatings2, alreadyGivenRatings2, populationCount, listOfProducts));
}

```

Figura 61. Testul pentru funcția recommend (ar trebui să obținem recomandări diferite) partea 2

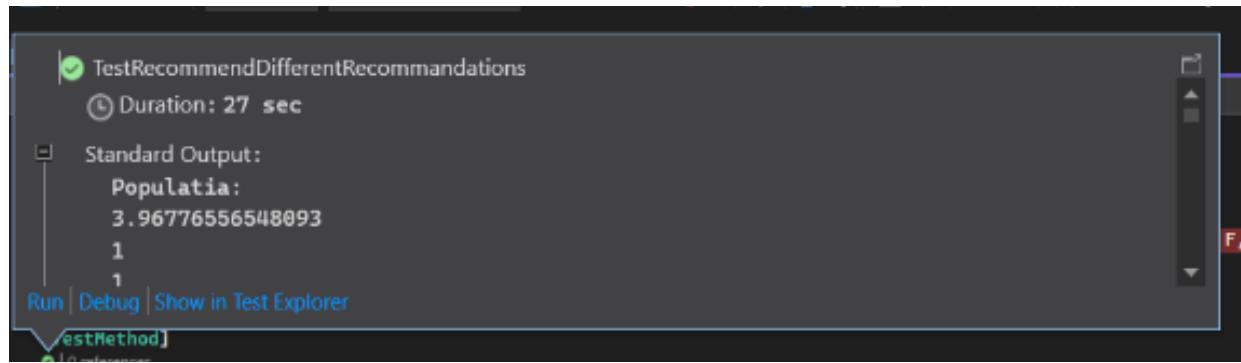


Figura 62. Rezultatul testului