

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Verificarea formală în Dafny a algoritmului  
Greedy pentru Problema Bancnotelor cu  
bancnote puteri ale lui 2**

propusă de

**Alexandra Elena Contur**

**Sesiunea: februarie, 2023**

Coordonator științific

**Conf. Dr. Ciobâcă Ștefan**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**

**Verificarea formală în Dafny a  
algoritmului Greedy pentru Problema  
Bancnotelor cu bancnote puteri ale lui 2**

**Alexandra Elena Contur**

**Sesiunea: februarie, 2023**

Coordonator științific

**Conf. Dr. Ciobâcă Ștefan**

Avizat,  
Îndrumător lucrare de licență,  
Conf. Dr. Ciobâcă Ștefan.

Data: ..... Semnătura: .....

### **Declarație privind originalitatea conținutului lucrării de licență**

Subsemnatul **Contur Alexandra Elena** domiciliat în **România, jud. Iași, com. Holboca, str. Dascălilor, nr. 10**, născut la data de **04 martie 2001**, identificat prin CNP **6010304226743**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2022, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Verificarea formală în Dafny a algoritmului Greedy pentru Problema Bancnotelor cu bancnote puteri ale lui 2** elaborată sub îndrumarea domnului **Conf. Dr. Ciobâcă Ștefan**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: .....

Semnătura: .....

## **Declarație de consimțământ**

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Verificarea formală în Dafny a algoritmului Greedy pentru Problema Bancnotelor cu bancnote puteri ale lui 2**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Alexandra Elena Contur**

Data: .....

Semnătura: .....

# Cuprins

<b>1</b>	<b>Context</b>	<b>2</b>
<b>2</b>	<b>Reprezentarea problemei în Dafny</b>	<b>3</b>
2.1	Reprezentarea datelor de intrare . . . . .	3
2.2	Reprezentarea datelor de ieşire . . . . .	3
2.3	Structuri de date folosite . . . . .	3
2.3.1	Condiţii ca o soluţie finală să fie optimă . . . . .	3
<b>3</b>	<b>Verificarea formală a problemei</b>	<b>5</b>
3.1	Entry point-ul problemei . . . . .	5
3.2	Demonstrarea optimalităţii . . . . .	6
3.2.1	Schema verificării . . . . .	6
3.3	Etapele verificării . . . . .	6
3.3.1	banknoteMinimum . . . . .	6
3.3.2	maxBanknote . . . . .	8
3.3.3	banknoteMaxim . . . . .	8
3.3.4	exchangeArgument . . . . .	9
3.3.5	banknoteMaxim32 . . . . .	11
3.3.6	exchangeArgument32 . . . . .	12
	<b>Concluzii</b>	<b>15</b>
	<b>Bibliografie</b>	<b>16</b>

# Capitolul 1

## Context

Dafny este un limbaj de programare și verificare, capabil să verifice corectitudinea funcțională a unui program.

Verificarea este posibilă datorită caracteristicilor specifice limbajului precum precondiții, postcondiții, invariante, ș.a.m.d. De asemenea, verificatorul Dafny are grijă ca adnotările făcute să se îndeplinească, astfel acesta ne scapă de povara de a scrie cod fără erori, în schimbul scrierii de adnotări fără erori.

Problema bancnotelor are ca scop reprezentarea unei sume într-un număr minim posibil de bancnote.

Metoda greedy face alegerea cea mai bună la fiecare pas, construind soluția finală. Verificarea formală în Dafny a problemei bancnotelor demonstrează faptul că soluția construită este optimă pentru orice sumă dată ca input.

Reprezentarea folosită pentru soluție este :  $banknote_1 := 1 < banknote_2 < \dots < banknote_n$ .

Anterior am menționat forma generală a problemei bancnotelor.

Bancnotele posibile în problema bancnotelor cu bancnote puteri ale lui 2 sunt:  $[1, 2, 4, 8, 16, 32]$ .

Datorită bancnotelor care sunt puteri ale lui 2, dacă avem mai mult de o bancnotă de valoare mai mică decât 32, putem înlocui 2 bancnote de aceea valoare cu o bancnotă de valoarea următoare și am obține o soluție cu cost mai mic.

Astfel, am descoperit proprietatea:  $\text{forall } i :: 0 \leq i \leq 4 \implies s[i] \leq 1$

# Capitolul 2

## Reprezentarea problemei în Dafny

### 2.1 Reprezentarea datelor de intrare

În cazul problemei discutate datele de intrare sunt reprezentate de o variabilă `sum` ce reprezintă suma pentru care se verifică dacă se produce soluția finală.

### 2.2 Reprezentarea datelor de ieșire

Datele de ieșire sunt reprezentate de o secvență de forma :

- soluție =  $b_0, b_1, b_2, b_3, b_4, b_5$  , unde  $\sum_{k=0}^5 b_k \cdot 2^k = suma$

### 2.3 Structuri de date folosite

-soluția reprezentată ca secvență -cele 3 predicate explicate + exemple cod

#### 2.3.1 Condiții ca o soluție finală să fie optimă

- Pentru a avea o soluție optimă trebuie să avem o soluție validă (cu 6 elemente), care produce suma corectă și care are costul cel mai mic.
- Pentru a avea o soluție optimă finală, pe parcursul construirii soluției, în buclă, trebuie menținută proprietatea de a alege soluția optimă locală pentru rest, iar suma soluțiilor optime locale să fie soluție optimă pentru sumă.
- Soluția formată dintr-o bancnotă, cea aleasă în iterația curentă, produce o soluție optimă pentru suma de valoare bancnotă, asigurând faptul că avem o soluție optimă locală.

- O soluție optimă pentru suma  $x$ , adunată cu o soluție optimă pentru suma  $y$  creează o soluție optimă pentru suma  $x + y$ , asigurând faptul că suma soluțiilor locale creează soluția finală optimă.



# Capitolul 3

## Verificarea formală a problemei

### 3.1 Entry point-ul problemei

Un entry point este locația din cod în care se face transferul controlului, acolo se fac apelurile altor funcții ș.a.m.d. Implementarea algoritmului propriu-zis care rezolvă problema a fost primul și cel mai ușor pas.

Am început prin crearea unei bucle care la fiecare pas alegea bancnota optimă, o adăuga în secvența de bancnote considerată soluție și o scădea din sumă, fiind un algoritm tipic metodei greedy.

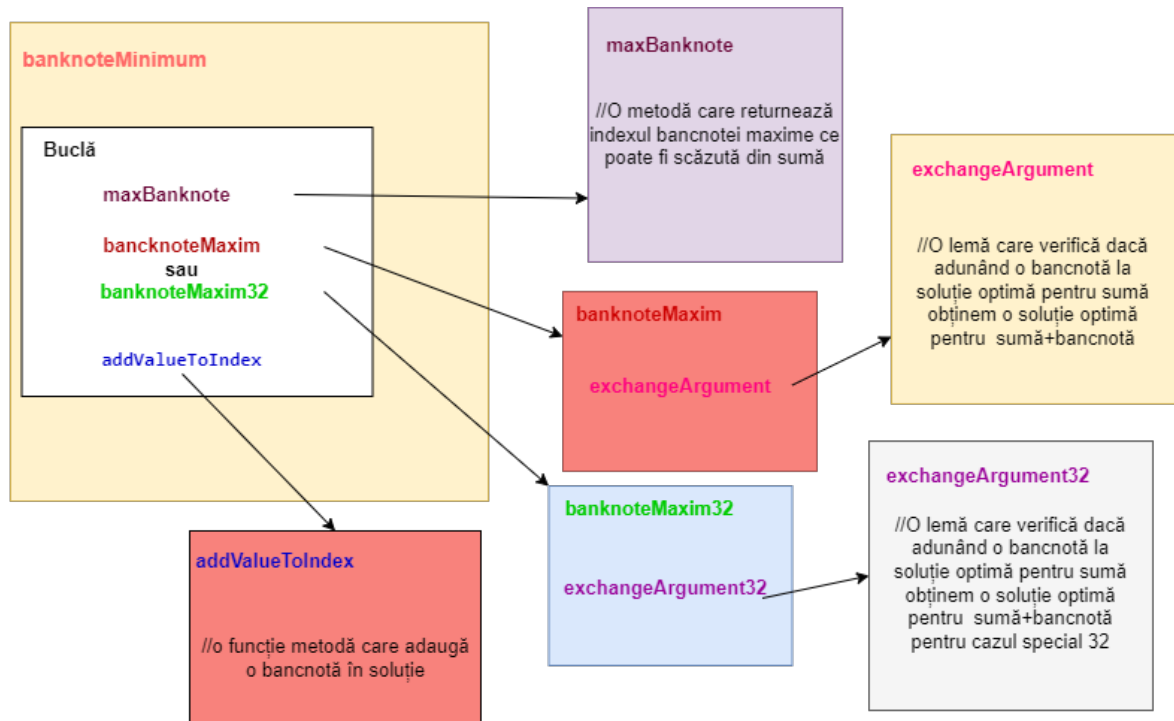
```
1   var rest:= sum;
2   solution:= [0, 0, 0, 0, 0, 0];
3   while (0 < rest)
4       decreases rest
5       {
6           index:= maxBanknote(rest);
7           var banknote:= power(2, index);
8           solution:= addValueToIndex(solution,1,index);
9           rest:= rest - banknote;
10  }
```

Acest algoritm era suficient pentru a rezolva problema, dar nu era suficient pentru a demonstra că soluția produsă este optimă.

Considerăm o soluție optimă soluția de cost minim, costul fiind numărul de bancnote.

## 3.2 Demonstrarea optimalității

### 3.2.1 Schema verificării



În schema de mai sus am exemplificat modul în care funcțiile, lemele și metodele se apelează una pe cealaltă pentru a demonstra faptul că soluția găsită este soluție optimă.

## 3.3 Etapele verificării

Am implementat algoritmul greedy care rezolva problema bancnotelor cu bancnote puteri ale lui 2, apoi am adăugat treptat condițiile care garantează că soluția găsită este soluție optimă.

### 3.3.1 banknoteMinimum

BanknoteMinimum returnează soluția optimă pentru o sumă dată ca input.

Această metodă folosește o buclă, în care, la fiecare iterație, se adaugă o bancnotă în soluție.

În primă fază calculează bancnota cea mai mare ce poate fi dată ca rest, cu ajutorul funcției maxBanknote și funcției power.

Ulterior, folosește lema `banknoteMaxim` sau `banknoteMaxim32`, în funcție de bancnota aleasă, pentru a demonstra că soluția calculată până în prezent, adunată cu bancnota aleasă și cu o soluție optimă pentru  $rest - bancnota$  produce o soluție optimă pentru sumă .

Proprietățile necesare unei soluții optime se păstrează pe parcursul buclei cu ajutorul invariantelor.

```

1  method banknoteMinimum(sum: int) returns(solution: seq <int> )
2      requires sum >= 0
3      ensures isValidSolution(solution)
4      ensures isSolution(solution, sum)
5      ensures isOptimalSolution(solution, sum)
6  {
7      var rest:= sum;
8      solution:= [0, 0, 0, 0, 0, 0];
9      var index:= 0;
10     assert isOptimalSolution(solution, sum - rest);
11     while (0 < rest)
12         invariant 0 <= rest <= sum
13         invariant isValidSolution(solution)
14         invariant addOptimRestEqualsOptimSum(rest, sum, solution)
15         decreases rest
16     {
17         index:= maxBanknote(rest);
18         var banknote:= power(2, index);
19         if (index != 5)
20         {
21             banknoteMaxim(rest, sum, solution, index);
22         }
23         else
24         {
25             banknoteMaxim32(rest, sum, solution);
26         }
27         solution:= addValueToIndex(solution,1,index);
28         rest:= rest - banknote;
29     }
30 }
```

### 3.3.2 maxBanknote

Metoda maxBanknote este folosită pentru a returna un index cu proprietatea:

$bancnota_{index} \leq rest < bancnota_{index+1}$ .

```
1  method maxBanknote(sum: int) returns(index: int)
2      requires sum > 0
3      ensures 0 <= index <= 5
4      ensures 0 <= power(2, index) <= sum
5      ensures (index != 5 && power(2, index + 1) > sum) || index == 5
6  {
7      index := 5;
8      if (power(2, index) > sum)
9      {
10         assert power(2, index + 1) > sum;
11         while (power(2, index) > sum && index > 0)
12             invariant power(2, index + 1) > sum
13             {
14                 index := index - 1;
15                 assert power(2, index + 1) > sum;
16             }
17     }
18     else
19     {
20         assert index == 5;
21     }
22 }
```

### 3.3.3 banknoteMaxim

Lema banknoteMaxim este folosită pentru a demonstra faptul că dacă adăugăm o bancnotă în soluția pe care o construim, în continuare suma acestei soluții cu soluția optimă pentru  $rest - bancnota$  va produce soluția optimă pentru suma întreagă.

Pentru a demonstra acest lucru avem nevoie să știm că dacă în soluția curentă, optimă pentru  $rest - bancnota$  adăugăm bancnota aceasta devine optimă pentru  $rest$ .

Acest lucru este demonstrat cu ajutorul lemei exchangeArgument.

```
1  lemma banknoteMaxim(rest: int, sum: int, finalSolution: seq <int> ,
2      index: int)
3      requires 0 <= index <= 4
```

```

3      requires power(2, index) <= rest < power(2, index + 1)
4      requires isValidSolution(finalSolution)
5      requires addOptimRestEqualsOptimSum(rest, sum, finalSolution)
6      ensures addOptimRestEqualsOptimSum(rest - power(2, index), sum,
      finalSolution[index:= finalSolution[index] + 1])
7
8      {
9      var banknote:= power(2, index);
10     forall currentSolution | isValidSolution(currentSolution) &&
      isOptimalSolution(currentSolution, rest - banknote)
11         ensures isOptimalSolution(solutionsSum(solutionsSum(
      currentSolution, finalSolution), [0, 0, 0, 0, 0, 0][index:=
      1]), sum)
12     {
13         assert isSolution(currentSolution[index:= currentSolution[index
      ] + 1], rest);
14         exchangeArgument(rest, currentSolution, index);
15     }
16     assert forall currentSolution::isValidSolution(currentSolution) &&
      isOptimalSolution(currentSolution, rest - banknote) ==>
      isOptimalSolution(solutionsSum(solutionsSum(currentSolution,
      finalSolution), [0, 0, 0, 0, 0, 0][index:= 1]), sum);
17 }

```

### 3.3.4 exchangeArgument

Lema `exchangeArgument` presupune că nu este optimă soluția dacă adaugăm bancnota aleasă și ajunge la o contradicție. Știm că soluția e optimă pentru  $rest - bancnota$ , dar nu pentru  $rest$ , dacă adaugăm bancnota.

Consideră că există o altă soluție optimă pentru  $rest$ .

Verifică în presupusa soluție pentru  $rest$  că nu avem bancnota în soluție deja, altfel acea presupusă soluție – bancnota are cost mai mic decât soluția noastră care e optimă pentru  $rest - bancnota$ , contradicție. Apoi verificăm proprietatea descrisă la observație.

Nu există două bancnote de aceeași valoare în soluție, mai mici de 32.

În acest mod știm că soluția presupusă nu poate fi optimă, deoarece, dacă respectă proprietatea suma produsă de aceasta nu poate fi egală cu  $rest$ .

Deoarece :

- $\sum_{k=0}^{index-1} 2^k = 2^{index} - 1$
- $rest \geq 2^{index}$

Deci presupusa soluție pentru  $rest$  nu poate fi egală decât cu cel mult  $rest - 1$ , astfel nu poate fi soluție optimă și se ajunge la o contradicție.

```

1 lemma exchangeArgument (rest: int, currentSolution: seq <int> , index: int)
2   requires 0 <= index <= 4
3   requires power(2, index) <= rest < power(2, index + 1)
4   requires isValidSolution(currentSolution)
5   requires isOptimalSolution(currentSolution, rest - power(2, index))
6   ensures isOptimalSolution(currentSolution[index:= currentSolution[index
7     ] + 1], rest)
8 {
9   var banknote:= power(2, index);
10  var solution:= currentSolution[index:= currentSolution[index] + 1];
11  assert isValidSolution(solution);
12  assert isSolution(solution, rest);
13  var i:= index;
14  if (!isOptimalSolution(solution, rest))
15  {
16    var optimalSolution:| isValidSolution(optimalSolution) && isSolution(
17      optimalSolution, rest) &&
18      isOptimalSolution(optimalSolution, rest) && cost(optimalSolution) <
19      cost(solution);
20    if (optimalSolution[index] - 1 >= 0)
21    {
22      var betterSolution:= addValueToIndex(optimalSolution, -1, index);
23      assert isSolution(betterSolution, rest - banknote);
24      assert cost(betterSolution) == cost(optimalSolution) - 1;
25      assert cost(optimalSolution) - 1 < cost(currentSolution);
26      assert false;
27    }
28  else
29  {
30    while (0 < i)
31      invariant 0 <= i <= index
32      invariant forall x::index >= x >= i ==> optimalSolution[x] <= 1
33    {
34      i:= i - 1;
35      assert isOptimalSolution(optimalSolution, rest);
36      if (optimalSolution[i] > 1)

```

```

34     {
35         var optimalSolution' := optimalSolution[i:=optimalSolution[i
           ]-2];
36         optimalSolution' := optimalSolution' [i + 1:= optimalSolution' [
           i+1]+1];
37         assert isSolution(optimalSolution', rest);
38         assert cost(optimalSolution') == cost(optimalSolution) - 1;
39         assert cost(optimalSolution') < cost(optimalSolution);
40         assert false;
41     }
42 }
43 assert solutionElementsSum(optimalSolution) <= banknote - 1;
44 assert rest >= banknote;
45 assert solutionElementsSum(optimalSolution) <= rest - 1;
46 assert isOptimalSolution(optimalSolution, rest);
47 assert false;
48 }
49 }
50 }

```

### 3.3.5 banknoteMaxim32

BanknoteMaxim32 demonstrează proprietatea menționată anterior.

Știm că finalSolution este soluție optimă pentru  $sum - rest$ .

Dacă currentSolution la care adăugăm bancnota 32 este soluție optimă pentru  $rest$ , atunci suma soluției optime pentru  $sum - rest$  adunată cu suma soluției optime pentru  $rest$  este soluție optimă pentru sumă.

BanknoteMaxim32 se asigură că este soluție optimă pentru  $rest$  currentSolution folosind lema exchangeArgument32.

```

1 lemma banknoteMaxim32(rest: int, sum: int, finalSolution: seq <int> )
2     requires rest >= 32
3     requires isValidSolution(finalSolution)
4     requires addOptimRestEqualsOptimSum(rest, sum, finalSolution)
5     ensures addOptimRestEqualsOptimSum(rest - 32, sum, solutionsSum(
           finalSolution, [0, 0, 0, 0, 0, 1]))
6 {
7     forall currentSolution | isValidSolution(currentSolution) && isSolution
           (currentSolution, rest - 32)

```

```

8      ensures isSolution(solutionsSum(solutionsSum(finalSolution,
      currentSolution), [0, 0, 0, 0, 0, 1]), sum)
9    {
10     assert isSolution(solutionsSum(currentSolution, [0, 0, 0, 0, 0, 1]),
      rest);
11   }
12
13   forall currentSolution | isValidSolution(currentSolution) &&
      isOptimalSolution(currentSolution, rest - 32)
14     ensures isOptimalSolution(solutionsSum(solutionsSum(finalSolution,
      currentSolution), [0, 0, 0, 0, 0, 1]), sum)
15   {
16     forall someSolution | isValidSolution(someSolution) && isSolution(
      someSolution, sum)
17       ensures cost(someSolution) >= cost(solutionsSum(solutionsSum(
      finalSolution, currentSolution), [0, 0, 0, 0, 0, 1]))
18     {
19       exchangeArgument32(rest, sum, currentSolution);
20     }
21   }
22 }

```

### 3.3.6 exchangeArgument32

Lema exchangeArgument32 presupune că nu este optimă soluția dacă adăugăm bancnota 32 și ajunge la o contradicție.

Știm că soluția e optimă pentru  $rest - 32$ , dar nu pentru  $rest$ , dacă adăugăm bancnota.

Consideră că există o altă soluție optimă pentru  $rest$ .

Deoarece nu avem o bancnotă mai mare de 32 cu care am putea înlocui 2 apariții ale acestei bancnote, verificăm doar proprietatea descrisă la observație.

Nu există două bancnote de aceeași valoare în soluție, mai mici de 32.

În acest mod știm că soluția presupusă nu poate fi optimă, deoarece, dacă respectă proprietatea suma produsă de aceasta nu poate fi egală cu  $rest$ .

```

1 lemma exchangeArgument32(rest: int, sum: int, optimalSolution: seq <int> )
2   requires 32 <= rest
3   requires isValidSolution(optimalSolution)
4   requires isOptimalSolution(optimalSolution, rest - 32)

```



```

5     ensures isOptimalSolution(optimalSolution[5:= optimalSolution[5] + 1],
6         rest)
7 {
8     var solution:= optimalSolution[5:= optimalSolution[5] + 1];
9     var i:= 4;
10    if (!isOptimalSolution(solution, rest))
11    {
12        if (optimalSolution[i] > 1)
13        {
14            var solution:= optimalSolution[i:= optimalSolution[i] - 2];
15            solution:= solution[i + 1:= solution[i + 1] + 1];
16            assert isSolution(solution, rest - 32);
17            assert cost(solution) == cost(optimalSolution) - 1;
18            assert cost(optimalSolution) - 1 < cost(solution);
19            assert false;
20        }
21        else
22        {
23            while (0 < i)
24                invariant 0 <= i <= 4
25                invariant forall index::4 >= index >= i ==> optimalSolution
26                    [index] <= 1
27            {
28                i:= i - 1;
29                if (optimalSolution[i] > 1)
30                {
31                    var solution:= optimalSolution[i:= optimalSolution[i] -
32                        2];
33                    solution:= solution[i + 1:= solution[i + 1] + 1];
34                    assert isSolution(solution, rest - 32);
35                    assert cost(solution) == cost(optimalSolution) - 1;
36                    assert cost(optimalSolution) - 1 < cost(solution);
37                    assert false;
38                }
39            }
40            assert solutionElementsSum(optimalSolution) <= rest - 1;
41            assert isOptimalSolution(solution, rest);
42            assert false;
43        }
44    }
45 }

```



# Concluzie

În cadrul acestei lucrări am realizat verificarea formală a unei implementări a metodei greedy ce rezolvă problema bancnotelor cu bancnote puteri ale lui 2. Una dintre dificultățile întâlnite în acest proces a fost crearea invariantului care menține proprietatea de soluție optimă, așa a apărut predicatul `addOptimRestEqualsOptimSum`. Acest predicat afirmă că o soluție optimă pentru o sumă  $x$  adunată cu o soluție pentru  $y$  are ca rezultat o soluție optimă pentru suma  $x + y$ . Astfel, secvența ce reprezintă soluția care se construiește pe parcursul buclei are garanția că atât timp cât se adună cu soluții locale optime, în final, vom avea o soluție finală optimă.

Parcurgerea acestor etape m-a făcut să aprofundez noțiunea de soluție și să înțeleg în amănunt formarea acesteia prin metoda Greedy.

Pe viitor îmi propun să găsesc o modalitate de a unifica metodele `banknoteMaxim` pentru a funcționa pentru orice caz. Un alt lucru interesant de făcut în viitor, ar fi o găsirea unei proprietăți care să se aplice pentru orice sistem de bancnote, pentru a verifica faptul că produce o soluție optimă.

# Bibliografie

1. <https://github.com/alexandra21/Licenta2023>
2. <https://core.ac.uk/works/69828860>
3. <https://arxiv.org/pdf/1412.4395.pdf>
4. <https://jarednielsen.com/sum-consecutive-powers-2/>
5. <http://www.cse.unsw.edu.au/se2011/DafnyDocumentation/Dafny>

Verificarea formală în Dafny a problemei bancnotelor Verificarea formală în Dafny a implementării problemei bancnotelor folosind metoda Greedy cu bancnote puteri ale lui 2 Verificarea formală în Dafny a implementării unui algoritm Greedy pentru Problema Bancnotelor cu bancnote puteri ale lui 2