UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DOMENIUL: Calculatoare și tehnologia informației

SPECIALIZAREA: Tehnologia informației

Colorarea unei hărți bazate pe un graf

folosind Algoritmul “Forward Checking”

Profesor coordonator

Prof. dr. ing. Florin Leon

Studenți

Brumă Constantin-Marius – grupa 1410A

Socea Gabriel – grupa 1410B

Matei Denisa-Dimitriana – grupa 1409A

Iași, 2020

Cuprins

[1. Descrierea problemei 3](#_Toc29888521)

[2. Aspecte teoretice privind algoritmul „Forward Checking” 4](#_Toc29888522)

[3. Modalitatea de rezolvare 5](#_Toc29888523)

[3.1. Algoritmul 5](#_Toc29888524)

[3.2. Interfața grafică 5](#_Toc29888525)

[4. Părțile semnificative din codul sursă 5](#_Toc29888526)

[5. Rezultate obținute prin rularea programului 8](#_Toc29888527)

[5.1.Cazul în care aplicația rulează cum trebuie și colorează harta cu numărul maxim de culori (4) 8](#_Toc29888528)

[5.2.Cazul în care aplicația rulează cum trebuie și colorează harta cu un număr mai mic de culori decât maxim (2 sau 3) 8](#_Toc29888529)

[5.3.Cazul în care graful nu reprezintă o hartă 9](#_Toc29888530)

[6. Lista cu ce a lucrat fiecare membru 10](#_Toc29888531)

[7. Concluzii 10](#_Toc29888532)

[8. Bibliografie 10](#_Toc29888533)

# Descrierea problemei

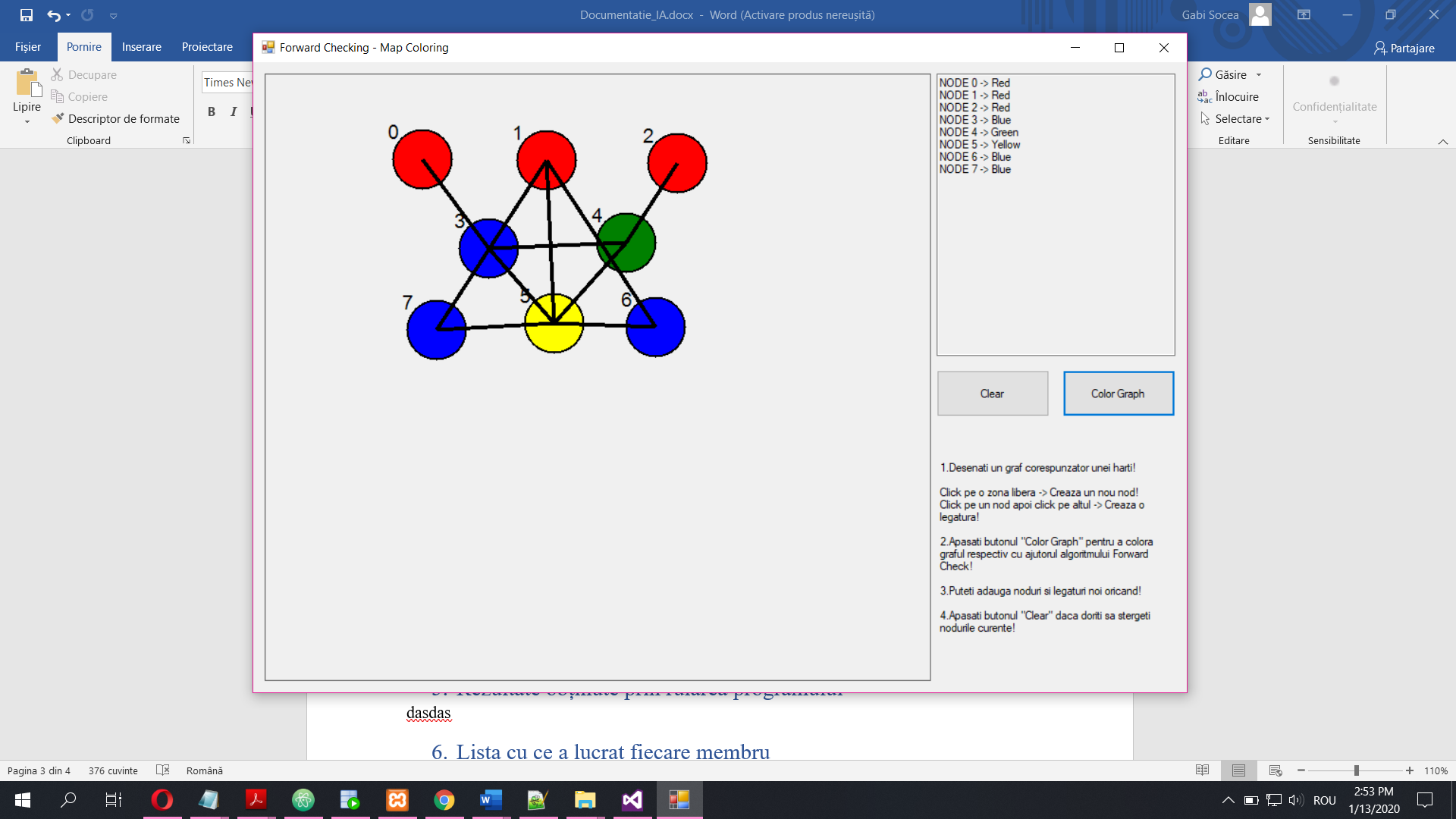
Pentru acest proiect, din lista de 10 proiecte câte sunt în total, noi am avut de implementat algoritmul “Forward Checking”, care este un important algoritm din domeniul Inteligenței Artificiale.

Ca temă am ales “Colorarea unei hărți” folosindu-ne de un graf neorientat ajutător în care județele/țările sunt reprezentate de noduri, iar nodurile care au legături între ele reprezintă județe/țari învecinate. De fapt, în interfața noastră grafică județele/țările sunt chiar nodurile care le reprezintă și colorarea se face asupra nodurilor care sunt niște cercuri numerotate de la 0 la numărul de noduri minus unu.

Pentru ca algoritmul să furnizeze un răspuns trebuie ca graful introdus să corespunda unei harți in care toate regiunile sunt suprafețe continue.

Interfața grafică conține suprafața de desenare a grafului care în final va fi colorat în funcție de ce culori atribuie algoritmul fiecărui nod, o zonă în care se specifică în cuvinte ce culoare are fiecare nod, un buton de Clear care resetează zona de desenare, un buton Color graph care colorează graful pe baza algoritmului.

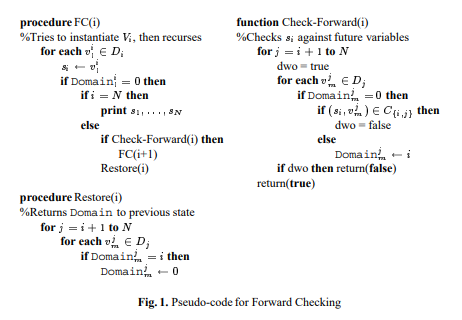
Se poate modifica graful dinamic în sensul în care se pot adăuga noi noduri la graful existent chiar și după ce algoritmul a mai rulat.



# Aspecte teoretice privind algoritmul „Forward Checking”

Algoritmul forward checking este un algoritm de căutare folosit în rezolvarea unor probleme speciale(CSP-Constraint Satisfaction Problems). Acesta folosește un set de variabile fixe, în acest caz fiind vorba de regiunile și culorile puse la dispoziție de utilizator. Ideea de funcționare se bazează pe verificarea repetitivă a culorilor disponibile pentru fiecare nod al grafului, reprezentând una din regiunile hărții, în funcție de vecinii acesteia.

La început, pentru fiecare variabilă se memorează mulțimea curentă de valori permise. Când se atribuie o valoare în procesul de căutare, se actualizează mulțimile de valori permise pentru toate variabilele. Dacă o mulțime devine vidă, se revine pe nivelul anterior (se face backtracking).

 Algoritmul ține evidența valorilor legale rămase pentru variabilele neasignate: când o variabila X este asignată, pentru fiecare variabilă Y conectată cu X printr-o constrângere, din domeniul de valori se șterg acele valori care nu sunt consistente cu valoarea aleasa pentru X. Căutarea se termina când nici o variabila nu mai are valori legale.

Algoritmul este compus din 5 funcții:

* public ForwardChecking(List<Node> nodes) – constructorul clasei
* public void setGraph(List<Node> nodes) – seteaza graful care va fi utilizat de algoritmul “Forward Chechink”;
* public bool StartAlgoritm(int index) – functia care implementeaza algoritmul;
* public void Restore(int index) – functie folosita in cadrul algoritmului pentru a restaura domeniul posibil al fiecarui nod dupa o incercare de colorare nereusita;
* public bool CheckForward(int index) – functie care varifica daca am atribuit o culoare nodului current ce nu afecteaza vecinii;

# Modalitatea de rezolvare

## 3.1. Algoritmul

Algoritmul începe prin colorarea unui nod, după care verifică daca atribuirea acelei culori nu împiedică nodurile vecine din a fi colorate. Daca nu, atunci culoarea aleasa este validă se reține indexul nodului colorat în vectorul “domain” pe poziția ce corespunde culorii alese și se continuă parcurgerea nodurilor în acest fel. Daca verificarea eșuează atunci se restaurează vectorii „domain” ai nodurilor vecine și se încearcă colorarea nodului cu altă culoare.

Algoritmul se termină când se colorează ultimul nod sau când se iese din stivă de recursie, caz în care nu s-a reușit colorarea grafului.

## 3.2. Interfața grafică

Suprafață de desenare a grafului funcționează în următorul mod:

Un click pe suprafața de desenare într-un loc gol creează un nou nod în graf, reprezentat în interfață printr-un cerc.

Click în interiorul unui nod, începe desenarea unei legături, dacă se selectează un alt nod față de cel inițial selectat atunci se creează o legătură intre cele două noduri. Dacă după selectarea primului nod se dă click pe o zona goală din suprafața de desenat atunci se anulează crearea unei legături.

Pentru ștergerea grafului se apasă pe butonul Clear, acțiune ce va șterge suprafața de desenare și graful din memoria aplicației.

# Părțile semnificative din codul sursă

În continuare sunt afișate clasa Node care stochează informațiile aferente fiecărui nod în parte și clasa ForwardChecking în care este implementat algoritmul folosit.

namespace MapColoring

{

//Clasa folosita pentru a memora detalii despre fiecare regiune( culoarea, vecinii, vectorul domeniu)

class Node

{

public String color = "White";

public List<Node> neighbors = new List<Node>();

//Domeniul de culori posibile pentru un nod

// Daca o valoare este -1 inseamna ca se poate atribui culoarea respectiva

// Altfel, valoarea salvata reprezinta nodul care impiedica nodul curent din a lua culoarea respectiva

public int[] domain = { -1, -1, -1, -1 };

//Poziitiile de pe axele oX si oY pe suprafata de desenare din interfata

public int x;

public int y;

}

}

namespace MapColoring

{

class ForwardChecking

{

//Lista de noduri folosita de metoda care implementeaza algoritmul

List<Node> nodes;

String[] colors = { "Red", "Blue", "Green", "Yellow" };

public ForwardChecking(List<Node> nodes)

{

//Se reseteaza culorile si domeniul nodurilor

for (int i = 0; i < nodes.Count(); i++)

{

nodes[i].color = "White";

nodes[i].domain[0] = -1;

nodes[i].domain[1] = -1;

nodes[i].domain[2] = -1;

nodes[i].domain[3] = -1;

}

this.nodes = new List<Node>(nodes);

}

public ForwardChecking() { }

public void setGraph(List<Node> nodes)

{

//Se reseteaza culorile si domeniul grafului

for (int i = 0; i < nodes.Count(); i++)

{

nodes[i].color = "White";

nodes[i].domain[0] = -1;

nodes[i].domain[1] = -1;

nodes[i].domain[2] = -1;

nodes[i].domain[3] = -1;

}

this.nodes = new List<Node>(nodes);

}

public bool StartAlgoritm(int index)

{

// Se incarca fiecare culoare

for (int c = 0; c < colors.Length; c++)

{

if (nodes[index].domain[c] == -1)

{

// Daca o valoarea corespunzatoare culorii din vectorul domeniu este -1 se poate colora nodul cu acea culoare

// Se coloreaza nodul

nodes[index].color = colors[c];

// Daca s-a ajuns la ultimul nod functia returneaza true;

if (index == nodes.Count() - 1)

return true;

// Se verifica daca culoarea setata nu impiedica nodurile vecine din a fi colorate

else if (CheckForward(index))

{

// Daca algoritmul returneaza true, se continua returnarea valorii pana se iese din stiva de recursivitate

if (StartAlgoritm(index + 1))

return true; // Daca nu se continua algoritmul

}

// Daca s-a ajuns aici insemna ca ce culoare am atribuit nodului impiedica colorarea corecta a grafului

//Functia Restore aduce domeniul vecinilor nodului la stare de dinainte de atribuire a culorii

Restore(index);

}

}

return false;

}

public void Restore(int index)

{

nodes[index].color = "White";

for(int j = 0; j<nodes[index].neighbors.Count(); j++)

{

for (int c = 0; c < colors.Length; c++)

if (nodes[index].neighbors[j].domain[c] == index)

nodes[index].neighbors[j].domain[c] = -1;

}

}

public bool CheckForward(int index)

{

// Se trece prin fiecare vecin al nodului cu indexul "index"

for (int j = 0; j < nodes[index].neighbors.Count(); j++)

{

bool DWO = true;

for (int c = 0; c < colors.Length; c++)

{

// Daca o culoare este disponibilia

if (nodes[index].neighbors[j].domain[c] == -1)

{

// Se verifica daca acea culoare este culoarea cu care s-a colorat nodul "index"

if (!nodes[index].color.Equals(colors[c]))

{

// Daca nu, DWO se seteaza pe fals, ceea ce inseamna ca nodul "index" nu

// elimina ultima valoare posibila pentru nodul sau vecin

DWO = false;

}

else

{

// Daca culoare verificata este aceiasi ca cea a nodului "index"

//se setaza valoarea domeniului cu valoarea "index":

nodes[index].neighbors[j].domain[c] = index;

}

}

}

// Daca s-a trecut prin toate nodurile si DWO este true inseamna ca valoarea atribuita culorii nodului "index"

//incurca macar un vecin si functia returneaza false;

if (DWO)

return false;

}

return true;

}

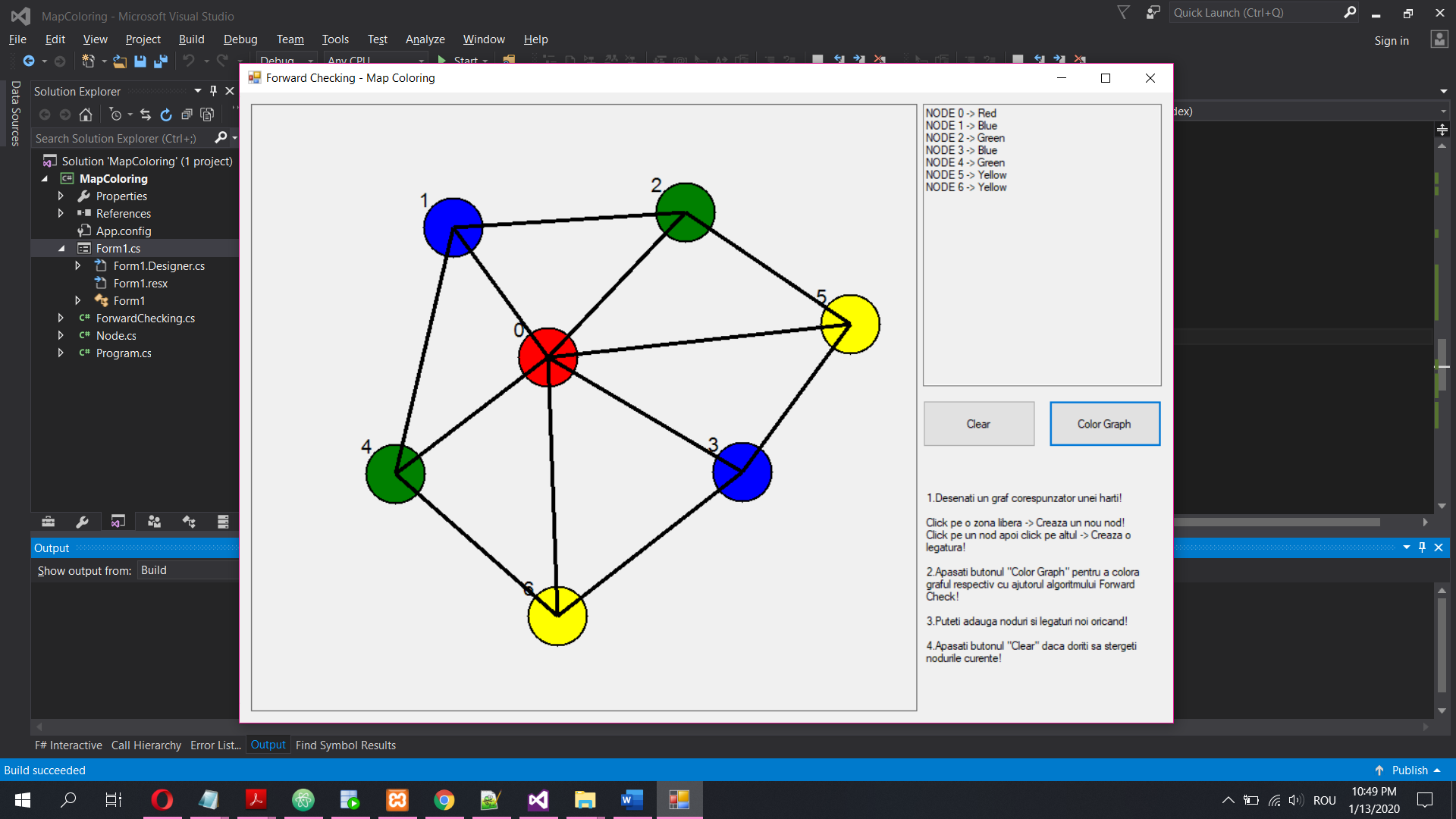
}

}

# Rezultate obținute prin rularea programului

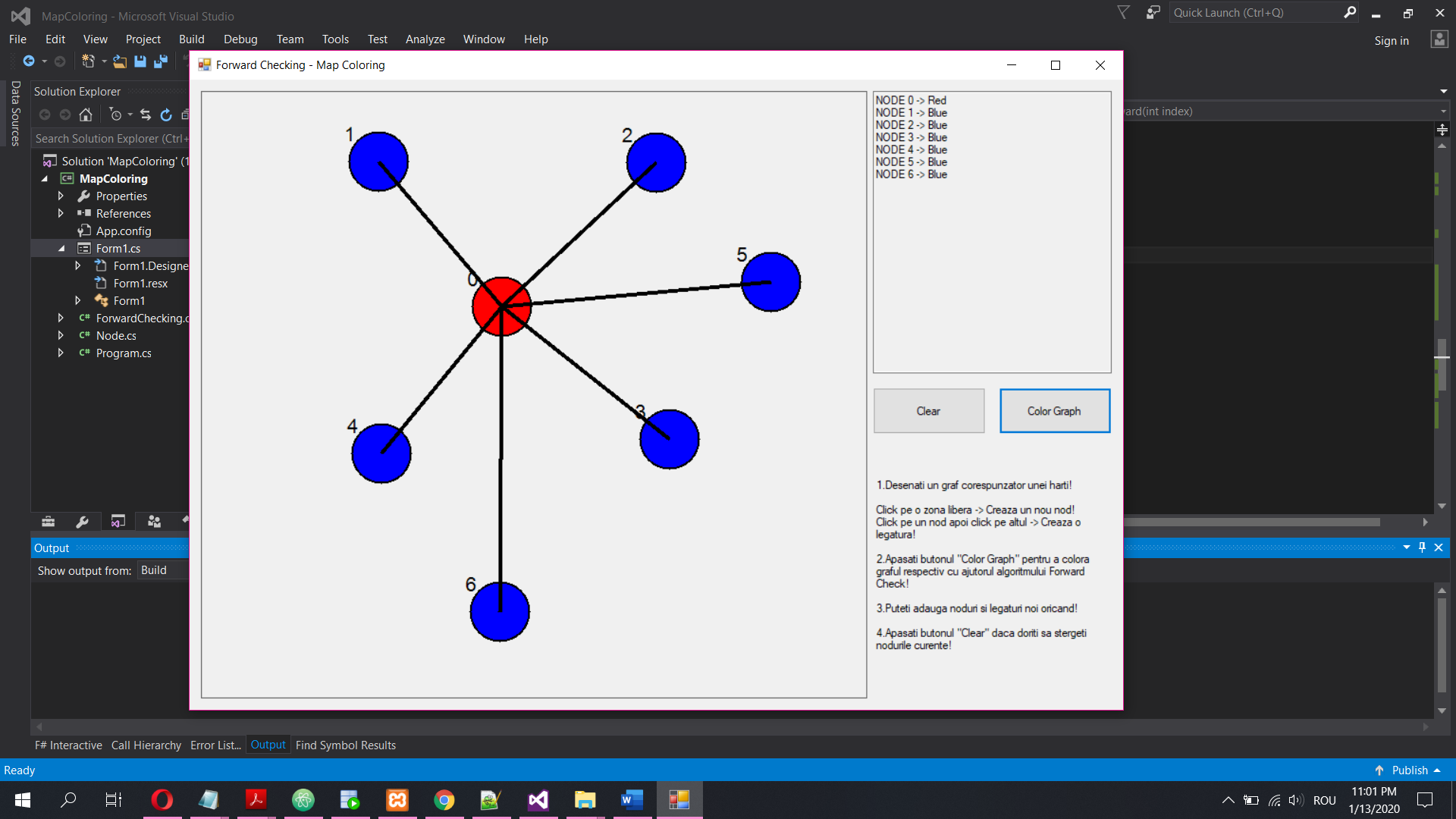
## 5.1.Cazul în care aplicația rulează cum trebuie și colorează harta cu numărul maxim de culori (4)

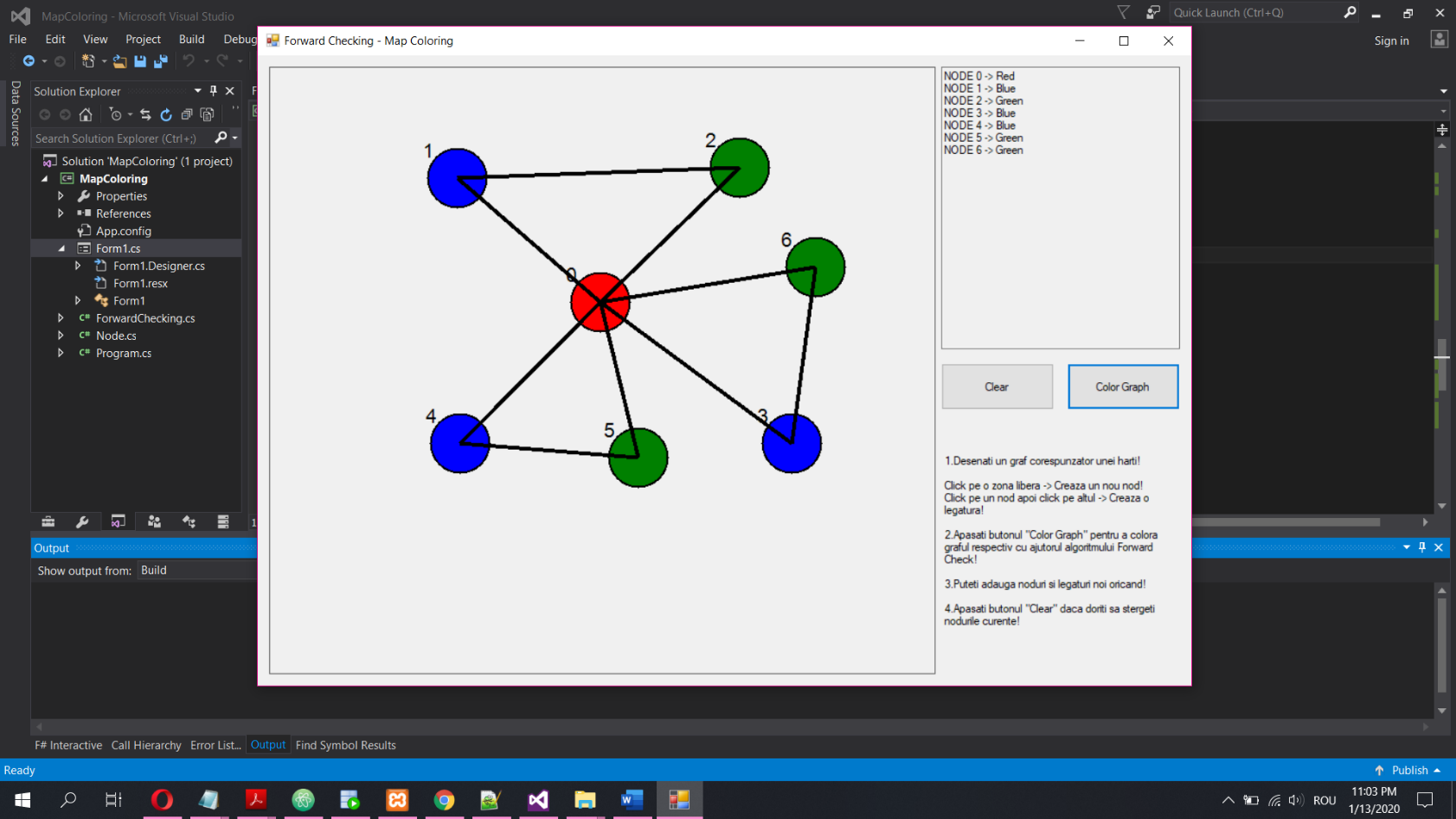
În acest caz harta reprezentată de graf are noduri cu suficiente conexiuni cu celelalte noduri astfel încât să se folosească toate culorile pentru desenare.



## 5.2.Cazul în care aplicația rulează cum trebuie și colorează harta cu un număr mai mic de culori decât maxim (2 sau 3)

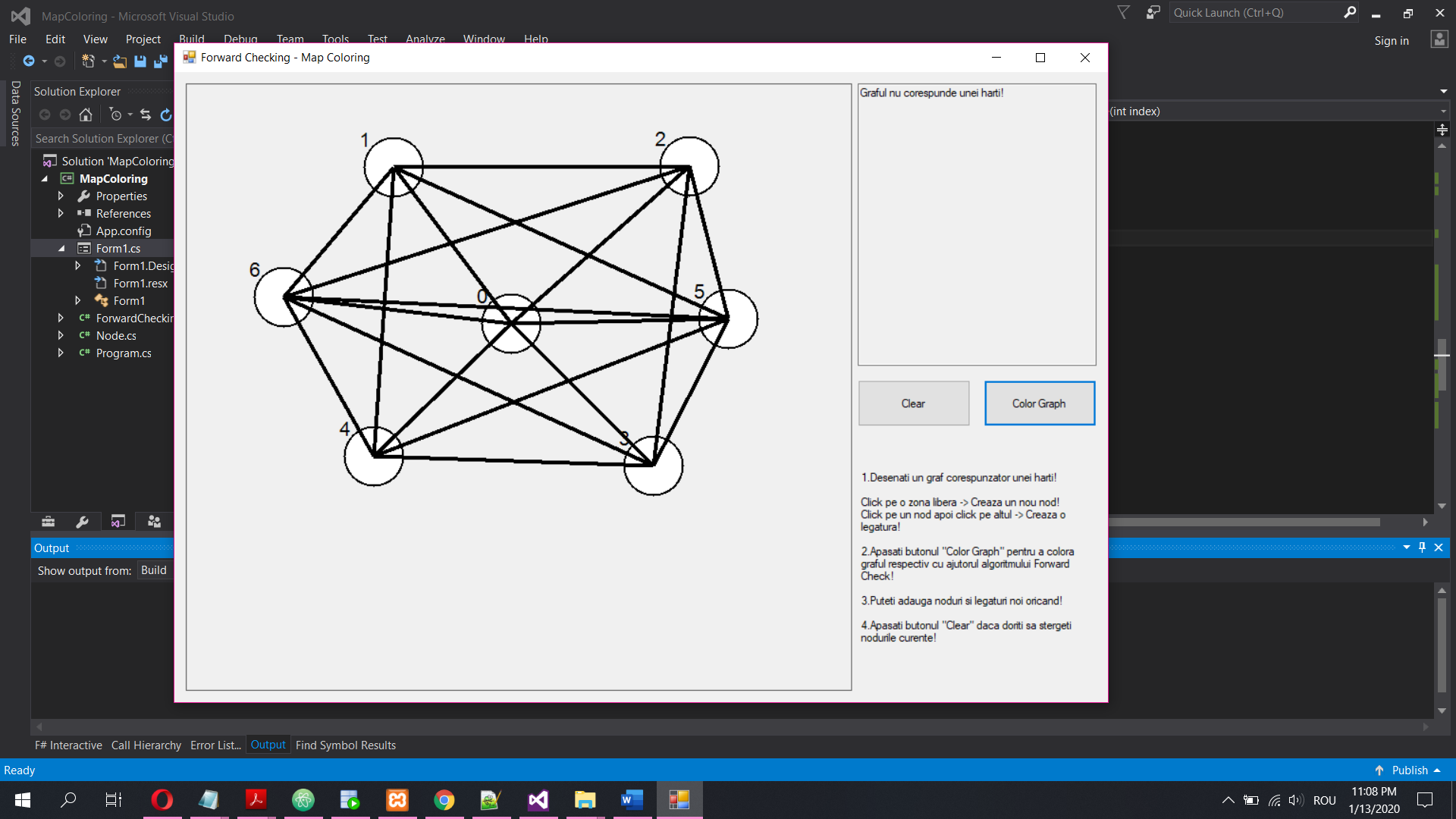
În acest caz harta reprezentată de graf are suficiente noduri, dar nu are foarte multe conexiuni astfel încât să se folosească toate culorile pentru desenare ci se folosesc 2 sau 3, după caz.





## 5.3.Cazul în care graful nu reprezintă o hartă

În cazul acesta se poate introduce, de exemplu, un graf în care toate nodurile sunt conectate cu toate celelalte noduri.



# Lista cu ce a lucrat fiecare membru

* La partea de implementare a aplicației am lucrat toți cei 3 membri: Denisa, Marius și Gabriel
* Documentarea despre algoritmul folosit a fost realizată de Marius
* Partea de interfață grafică a fost gândită de Denisa
* Documentația a fost realizată de Gabriel

# Concluzii

În concluzie, am reușit să rezolvam problema colorării unei hărți folosind algoritmul cerut în problemă. Aplicația funcționează pentru orice hartă atât timp cât se introduce graful corespunzător.

Am observat că acest algoritm este folosit pe scară largă în aplicațiile din domeniul Inteligenței Artificiale și este unul mai eficient decât alți algoritmi de genul acesta.

# Bibliografie

1. <http://cs-gw.utcluj.ro/~anca/iia/slide07.pdf>
2. <http://www.cs.ubbcluj.ro/~gabis/mas/Lectures/8_CSP&DSCP/CSP_DCSP%20(Romanian).pdf>
3. <http://florinleon.byethost24.com/Curs_IA/IA03_Jocuri_CSP.pdf>
4. <http://www.cs.toronto.edu/~fbacchus/Papers/BGCP95.pdf>