

Numărul Cromatic

Cristina Bivolan

June 5, 2018

Profesor: Costin Bădică

Profesor laborator: Alex Becheru

Abstract

Acest document introduce metodologia de dezvoltare a unei librării pentru colorarea unui graf neorientat și determinarea numărului cromatic (**Graph coloring**). Documentul conține, de asemenea, și descrierea tuturor variabilelor și funcțiilor folosite pentru a duce la bun sfârșit cerința problemei. Tinta acestui proiect: studenții din anul I , specializarea Calculatoare.

Precizări:

- Specializarea : Calculatoare cu predare în limba Română
- Anul : I
- Grupa : 1.1 A

1 Cerinta problemei

Scopul acestui proiect este de a implementa doi algoritmi diferiti pentru aflarea numarului cromatic al unui graf neorientat. Marimea grafului cat si a matricei de adiacenta va fi alocata dinamic pentru a folosi economic memoria. De asemenea, vor fi create functii pentru rezolvarea problemei folosind algoritmul Greedy, si pentru algoritmul Backtracking.

2 Pseudocodul

FUNCTIA-Greedy(*graf*)

1. i, j
2. $VectorCulori \leftarrow colors$
3. $NrMinCulori \leftarrow minColors$
4. **pentru** $i \leftarrow 1, nrNoduri, i \leftarrow i \rightarrow next$
 - 4.1. $colors[i] \leftarrow 1$
 - 4.2. **pentru** $i \leftarrow 1, nrNoduri, i \leftarrow i \rightarrow next$
 - 4.2.1. **daca** $grafAdjMatr[i][j] = 1$ **si** $colors[i] = colors[j]$ **atunci**
 - 4.2.1.1 $colors[i] \leftarrow colors[j] + 1$
 - 4.2.1.2 $minColors \leftarrow colors[j] + 1$

FUNCTIA-Printeaza(*colors, graf*)

1. i
2. $NrMinCulori \leftarrow minColors$
3. **pentru** $i \leftarrow 1, nrNoduri, i \leftarrow i \rightarrow next$
 - 3.1. **scrie** nodul i are culoarea $colors[i]$
 - 3.2. **daca** $colors[i] > minColors$ **atunci**
 - 3.2.1. $minColors \leftarrow colors[i]$

FUNCTIA-Verificare(*nod, graf, colors, culoare*)

1. i, j
2. **pentru** $i \leftarrow 1, nrNoduri, i \leftarrow i \rightarrow next$
 - 2.1. **daca** $grafAdjMatr[nod][i] = 1$ **si** $culoare = colors[i]$ **atunci**
 - 2.1.1. **returneaza** 0
 3. **returneaza** 1

FUNCTIA-ColorareRecursiva(*graf*, *max*, *colors*, *nod*)

1. *i*
2. **pentru** *i* $\leftarrow 1$, nrNoduri, *i* $\leftarrow i \rightarrow \text{next}$
- 2.1. *colors*[*i*] $\leftarrow 1$
- 2.2. **pentru** *i* $\leftarrow 1$, maxi, *i* $\leftarrow i \rightarrow \text{next}$
- 2.2.1. **daca** *Verificare*(*nod*, *graf*, *colors*, *i*) = 1 **atunci**
- 2.2.1.1. *colors*[*nod*] $\leftarrow i$
- 2.2.2. **daca** *ColorareRecursiva*(*graf*, *max*, *colors*, *nod* + 1) = 1 **atunci**
- 2.2.2.1. **returneaza** 1
- 2.2.3. *colors*[*nod*] $\leftarrow 0$
3. **returneaza** 0

FUNCTIA-Backtracking(*graf*, *max*)

1. *VectorCulori* $\leftarrow colors$
2. **daca** *ColorareRecursiva*(*graf*, *max*, *colors*, 0) = 0 **atunci**
- 2.1. **returneaza** 0
3. *Printeaza*(*colors*, *graf*)
4. **returneaza** 1

3 Schema aplicatiei

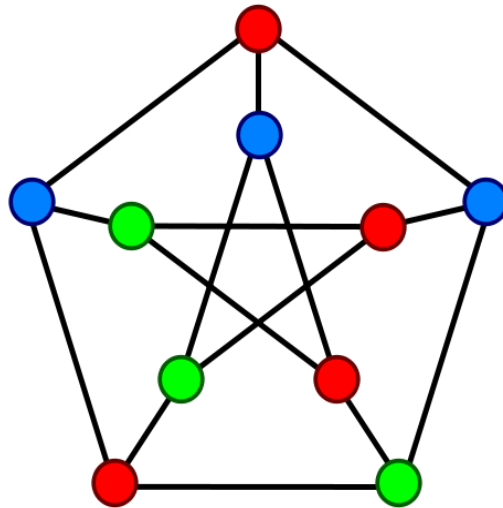


Figure 1: A proper vertex coloring of the Petersen graph with 3 colors, the minimum number possible.

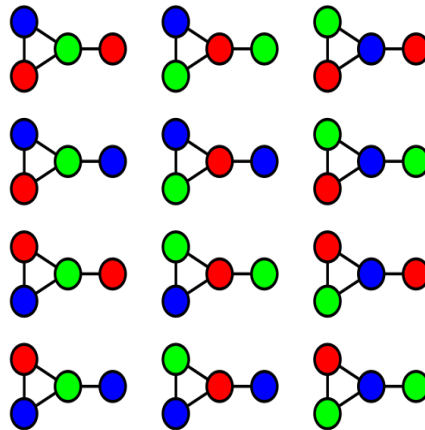


Figure 2: This graph can be 3-colored in 12 different ways.

4 Descrierea aplicatiei

Pentru a reprezenta un graf, într-un program, am folosit structuri de date ce reprezintă grafurile prin matricea de adiacență, o matrice pătratică în care adiacența dintre 2 noduri este notată cu 1, iar în caz contrar cu 0. Colorarea grafurilor și aflarea numărului cromatic se face folosind 2 algoritmi (Greedy și Backtracking). Am creat funcții pentru rezolvarea fiecărui algoritm. Pentru realizarea grafurilor, am folosit structura:

- i) **struct a-graph**, cu ajutorul căreia am definit numărul nodurilor, inițializarea grafurilor și matricea de adiacență. Structura conține: numărul de noduri (int no-nodes), inițializarea (int init) și un pointer către matricea de adiacență (int *adj-matrix).

În continuare, am definit funcții pentru citirea matricei de adiacență atât de la tastatură cât și din fișier astfel :

- i) Funcția **set-adj-matrix-value()**, ce setează pe poziția curentă din matrice valoarea 0 sau 1.
- ii) Funcția **init-graph()**, ce inițializează grafurile citind matricea de adiacență de la tastatură și o setează cu ajutorul funcției set-adj-matrix-value.
- iii) Funcția **init-graph-file()**, ce inițializează grafurile citind matricea de adiacență din fișier obținută prin generare random a valorilor (la care vom avea nevoie de biblioteca time.h) și scrierea valorilor în fișier, și o setează cu ajutorul funcției set-adj-matrix-value.
- iiii) Funcția **get-adj-matrix-value()**, ce returnează valoarea elementului matricei de adiacență pe poziția curentă, dacă grafurile sunt inițializate. În caz contrar, returnează -1.
- iiiii) Funcția **print-adj-matrix()**, funcție ce afișează matricea de adiacență cu ajutorul funcției get-adj-matrix-value, dacă grafurile sunt inițializate. În caz contrar, afișează "Te rog citește grafurile mai întâi".
- iiiii) Funcția **adj-check()**, care verifică dacă 2 noduri sunt adiacente cu ajutorul funcției get-adj-matrix-value, returnând valoarea 1 dacă verificarea este adevărată și 0 dacă e falsă.

Odată inițializat grafurile și cunoscând matricea de adiacență, putem construi algoritmi ajutatori rezolvării problemei:

Descrierea functiilor principale ale acestui proiect:

1. Functia **void chromatic-number-greedy ()** primeste ca parametru structura **struct a-graph *graph**. Aceasta functie rezolva problema prin algoritmul Greedy , care face la nivel local alegerea optim pentru fiecare etap n sperana de a gsi un optim global. Pentru inceput se defineste vectorul de marime alocata dinamic in care vor fi stocate culorile necesare si fisierul in care vor fi stocate datele de iesire. Initializam minimul numarului de culori necesare care este 1 deoarece avem nevoie de cel putin o culoare pt a colora un graf, apoi parcurgem nodurile grafului cu ajutorul iteratorului *iterator₁*. Initial toate nodurile vor avea culoarea 1, apoi parcurgem toate nodurile de dinaintea iteratorului *iterator₁* cu ajutorul iteratorului *iterator₂* si verificam daca cele doua noduri sunt adiacente si au aceeasi culoare. Daca verificarea este adevarata atunci culoarea nodului *iterator₂* va creste cu 1 , adica nodului ii vom atribui o alta culoare, iar minimul numarului de culori va fi valoarea culorii *iteratorului₂* crescuta cu 1. Se repeta pana cand gasim un nod *iterator₂* ce nu are culoarea unui nod adiacent *iterator₁*. La final parcurgem din nou nodurile si afisam ce culoare i-a fost atribuita fiecarui nod si de asemenea afisam numarul minim de culori necesare ,adica numarul cromatic.
2. Functia **void print-chromatic-number(int *colors, struct a-graph *graph)**, functie ajutatoare rezolvarii cu metoda Backtracking. Aceasta afiseaza in fisierul unde sunt stocate datele de iesire ce culoare i-a fost atribuita fiecarui nod si numarul minim de culori necesare ,adica numarul cromatic.
3. Functia **int check-same-color(int node, struct a-graph *graph, int *colors, int color)**, verifica daca 2 noduri sunt adiacente si au aceeasi culoare. Este de asemenea o functie ajutatoare ce are ca parametrii un nod *node* si o culoare *color* si parcurgand nodurile grafului cu ajutorul iteratorului *iterator₁* , verifica daca nodul *node* si nodul *iterator₁* sunt adiacente si culoarea nodului *iterator₁* este aceeasi cu culoarea *color*. Daca este adevarat returneaza 0. Altfel ,returneaza 1.
4. Functia **int recursive-backtracking-coloring(struct a-graph *graph, int max-no-colors, int *colors, int node)** este o functie recursiva ajutatoare colorarii grafului. Verificam daca nodul *node* a ajuns la final si nu ne-a mai ramas niciun nod, in acest caz returnam 1 ca conditie de oprire. Parcurgand culorile necesare cu ajutorul iteratorului *color_iterator* pana la valoarea maxima a numarului de culori necesare data ca parametru *max – no –*

colors ,verificam daca nodul *node* este adiacent cu alt nod din graf si daca culoarea *color_ierator* este aceeaasi , cu alte cuvinte verificam daca ii putem atribui in siguranta nodului *node* culoarea *color_ierator*. Autoapelam functia pentru a atribui culori pentru restul nodurilor. Cand au fost atribuite culori tuturor nodurilor functia returneaza 1. Daca culoarea *color_ierator* nu poate duce la nicio solutie atunci o eliminam atribuindu-i valoarea 0. Ultima conditie de oprire : returneaza 0 cand niciuna dintre conditiile de mai sus nu s-au rulat;

5. Functia **int chromatic-number-backtracking(struct a-graph *graph, int max-no-colors)** rezolva problema prin aloritmul Backtracking bazat pe functiile anterioare la care vom face apel. Verificam daca culorile au fost atribuite cu ajutorul functiei recursive-backtracking-coloring(graph, max-no-colors, colors, 0) pornind de la nodul 0. Daca verificarea este falsa atunci nu exista nicio solutie cu conditia de oprire return 0 , altfel, afisam numarul cromatic cu ajutorul functiei print-chromatic-number(colors, graph) avand conditie de oprire return 1.
6. Functia **int main()**,cuprinde declararea variabilelor: variabila *graph de tip struct a-graph *graph si variabila pentru a alege ce metoda sa folosim, **int choose-algorithm**;variabilei *graph i se aloca dinamic memorie de marimea struct a-graph , variabila prin intermediul careia putem ajunge la nodurile grafului sau la matricea de adiacenta; Urmeaza citirea datelor,care se realizeaza dupa dorinta user-ului, fie de la tastatura fie random din fisierul **in**,afisarea realizandu-se in fisierul **out**.
Dupa citire afisam matricea de adiacenta , apoi se realizeaza alegerea algoritmului de rezolvare tot cu ajutorul variabilei *choose – algorithm*.
In final , eliberam memoria utilizata de variabila *graph si de matricea de adiacenta folosind functia predenita void free (void*ptr) din libraria stdlib.h.

5 Concluzii

La prima vedere,sau mai bine spus,atunci cand citesti prima data cerinta problemei,nu pare deloc complicat,insa din punctul meu de vedere,cel mai greu a fost sa definesc functiile principale si cele ajutatoare ce rezolva problema prin 2 algoritmi.

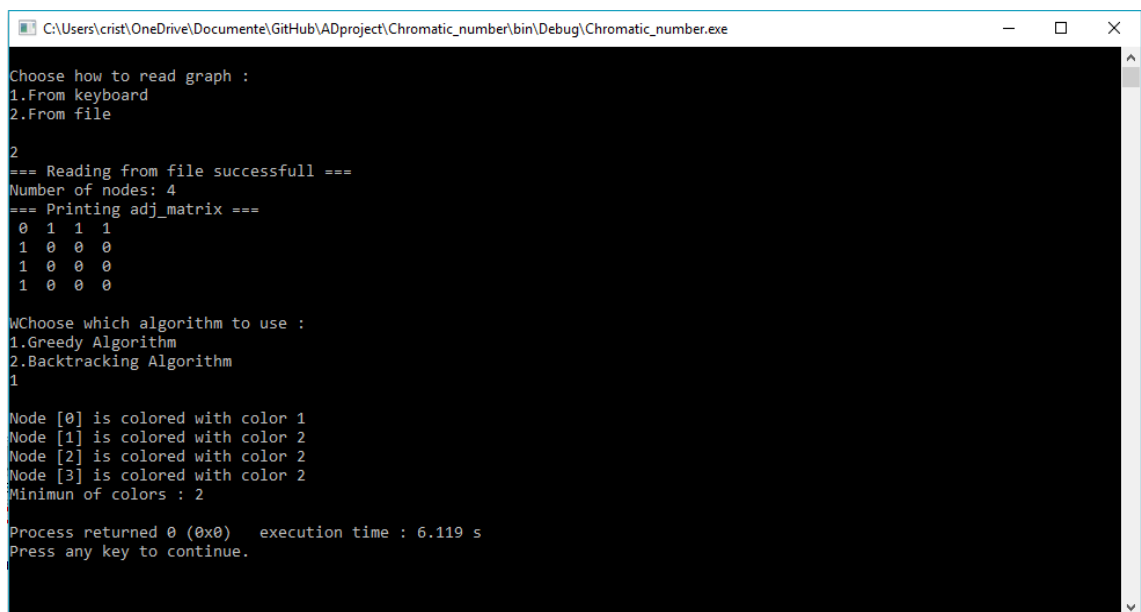
6 Bibliografie

- i) *The C Programming language, Second Edition*, Brian W. Kernighan, Dennis M. Ritchie
- ii) *Programming Techniques course*
- iii) *Wikipedia*
- iiii) *Geeks for Geeks*

7 Experimente si rezultate

Generarea automata non-trivial input test data a fost facuta initializand functia `rand((unsigned) time(t))`, atribuind random numarul de noduri de la 1 la $5 \leftarrow 1 + \text{rand() modulo } 5$, (pentru facilitatea citirii de catre user) si matricea de adiacenta random luand valoarea 0 sau 1 $\leftarrow (\text{rand() modulo } 2)$.

Compararea celor 2 algoritmi:



```
C:\Users\crist\OneDrive\Documente\GitHub\ADproject\Chromatic_number\bin\Debug\Chromatic_number.exe

Choose how to read graph :
1.From keyboard
2.From file

2
=== Reading from file successfull ===
Number of nodes: 4
=== Printing adj_matrix ===
0 1 1 1
1 0 0 0
1 0 0 0
1 0 0 0

WChoose which algorithm to use :
1.Greedy Algorithm
2.Backtracking Algorithm
1

Node [0] is colored with color 1
Node [1] is colored with color 2
Node [2] is colored with color 2
Node [3] is colored with color 2
Minimun of colors : 2

Process returned 0 (0x0)   execution time : 6.119 s
Press any key to continue.
```

Figure 3: Test random generator Algoritm 1.

