Analiza Algoritmilor

Problema colorării nodurilor unui graf

Arina Emanuela Turcu

Universitatea Politehnica Bucureşti
Facultatea de Automatica si Calculatoare
Grupa 323CA
arina.turcu@stud.acs.upb.ro

18 Decembrie 2020

Rezumat Proiecul are ca scop analizarea unor algoritmi care rezolvă problema colorării nodurilor unui graf din punct de vedere al complexităţii lor. Se va propune cate o implementare in C++ pentru fiecare algoritm, urmând să fie testate, şi se vor prezenta concluzii privind performanţa şi resursele folosite de fiecare.

Cuvinte cheie: Graf, Număr cromatic, Complexitate, Adiacenţă

1 Introducere

1.1 Descrierea problemei rezolvate

Problema colorării nodurilor unui graf este o problemă NP-completă care işi propune să gasească o colorare a nodurilor unui graf cu numărul minim de culori necesare în aşa fel încât oricare două noduri adiacente să nu aibă aceeaşi culoare. Numărul minim de culori de care este nevoie pentru a colora graful în acest fel este numit *număr cromatic* şi este notat χ(G).

1.2 Exemple de aplicaţii practice pentru problema aleasă

Problema colorării nodurilor unui graf poate reprezenta programarea examenelor într-o facultate astfel: fiecare materie este reprezentată de un nod, iar o muchie între două noduri(materii) arată că există cel puţin un student care va da examen la ambele materii. Se incearcă găsirea unor intervale orare în care se vor da examenele astfel încât un student să nu aibă două examene în acelaşi timp.

Rezolvarea unui joc Sudoku poate fi asemănată cu problema colorării nodurilor unui graf în care nodurile reprezintă celulele. Există o muchie între două noduri dacă două celule sunt pe aceeaşi linie, acelaşi rând sau se află în acelaşi bloc. În acest caz, numărul cromatic este 9.

1.3 Specificarea soluţiilor alese

Dându-se un graf neorientat se poate gasi numarul cromatic prin mai multe metode, după cum urmează.

1.3.1 Backtracking

Această soluţie foloseşte metoda backtracking ca să genereze toate posibilele colorari ale grafului până ajunge la o colorare validă cu un număr minim de culori. Se încearcă colorarea grafului cu k culori, unde k ia valori intre 1 şi numărul total de noduri ale grafului, până când se găseşte o valoare a lui k pentru care graful poate fi colorat. Această metoda devine ineficientă atunci cand graful are un număr mare de noduri si de muchii.

1.3.2 Greedy algorithm

Algoritmul greedy incearcă găsirea numarului cromatic printr-o parcurgere liniara a nodurilor. Asignând mereu prima culoare validă fiecarui nod, ajunge să realizeze o colorare validă a grafului cu un numar de culori apropriat sau chiar egal cu numărul cromatic. Problema acestei metode este că nu va găsi întotdeauna o colorare optimă a grafului.

1.4 Specificarea criteriilor de evaluare alese pentru validarea soluţiilor

Pentru evaluarea soluţiilor, va fi realizat un set de teste care cuprinde grafuri neorientate cu un număr variabil de noduri şi muchii. Grafurile vor avea între 1 şi 50 de noduri şi vor fi rare sau dese, cu mai puţine sau mai multe componente conexe pentru o testare mai variată. Rezultatul soluţiei care foloseşte metoda backtracking va fi validat sau respins dupa compararea cu rezultatul corect. Rezultatul soluţiei care foloseşte algoritmul greedy va fi comparat cu rezultatul corect, urmand să se calculeze o eroare relativă pentru a stabili precizia acesteia.

2 Prezentarea soluţiilor

2.1 Descrierea modului în care funcţioneaza algoritmii aleşi

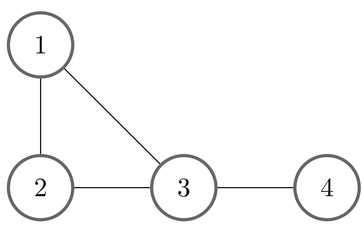


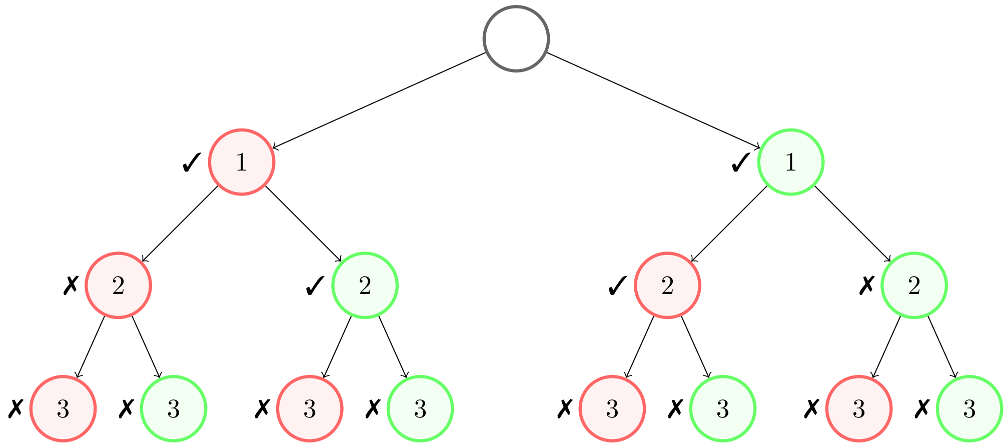
Figura 1: Un graf simplu

2.1.1 Backtracking

Soluţia aceasta verifică dacă nodurile grafului pot fi colorate cu un numar k de culori cuprins intre 1 si numărul total de noduri. Pentru fiecare k, se încearcă fiecare posibilitate de colorare astfel: primului nod i se asignează prima culoare, urmatorului nod i se asignează urmatoarea culoare care respectă regula şi aşa mai departe. Dacă culorile sunt epuizate, programul se întoarce recursiv şi încearcă altă ordine a culorilor pană cand se realizează o colorare validă(caz in care s-a găsit numarul cromatic) sau se epuizează metodele de colorare cu respectivul numarăr de culori şi se încearcă colorarea cu următorul număr.

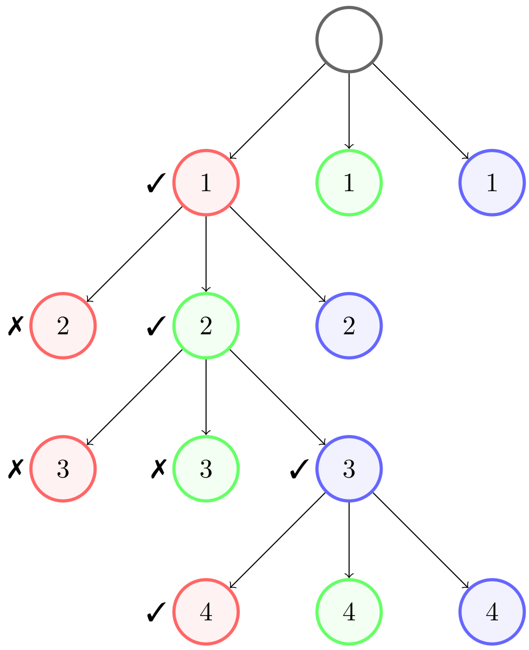
Pentru graful din figura 1, algoritmul va funcţiona astfel atunci când $k = 2$ şi $k = 3$:

$k = 2$:



Primului nod i se dă prima culoare (roşu), iar urmatorului nod i se dă urmatoarea culoare disponibilă (verde). Celui de al treilea nod nu i se poate asigna niciuna dintre cele 2 culori deoarece ambele sunt folosite de vecinii săi aşa că programul se întoarce recursiv la ultimul nod căruia îi poate schimba culoarea, în acest caz nodul 1, şi îi dă urmatoarea culoare, verde. În continuare, nodului 2 i se dă prima culoare disponibila, roşu, iar nodului 3 nu i se poate asigna nicio culoare. Fiind epuizate toate posibilităţile de colorare, se află că graful din figura 1 nu poate fi colorat cu 2 culori.

$k = 3$:

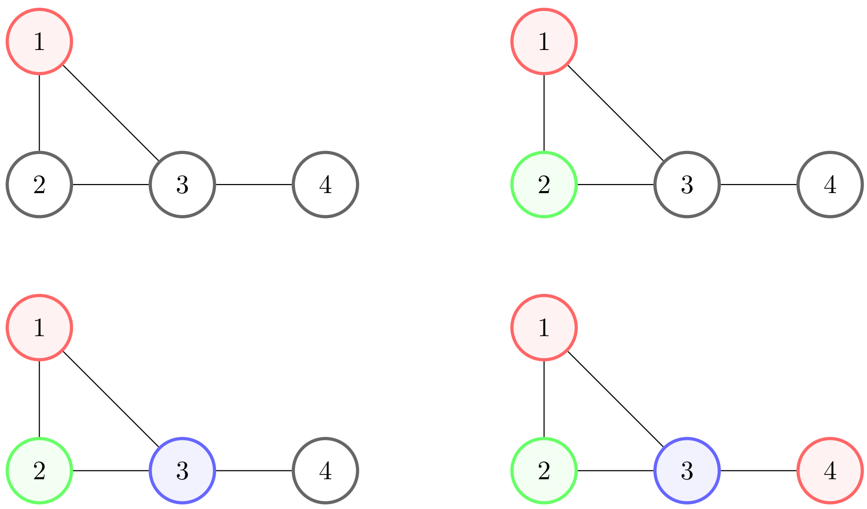


Nodului 1 i se dă prima culoare, roşu, iar nodului 2 i se dă prima culoare disponibila, verde.

Nodul 3, fiind adiacent şi cu nodul 1 şi cu nodul 2, poate fi colorat doar cu albastru. Nodul 4 este adiacent doar cu nodul 3, deci poate fi colorat cu roşu sau verde. Fiind colorate toate nodurile, programul a ajuns la o colorare valida a grafului cu 3 culori şi se opreşte.

2.1.2 Greedy algorithm

Acest algoritm va aproxima o soluţie astfel: primului nod din graf i se va asigna prima culoare, iar urmatorului nod i se va asigna urmatoarea culoare nefolosită de niciunul dintre nodurile adiacente colorate anterior. Dacă niciuna din culoarile folosite anterior nu poate fi folosită, i se va da o culoare nouă. Când se ajunge la ultimul nod, algoritmul returnează numarul de culori folosite. O exemplificare a algoritmului se găseşte mai jos:



După ce s-a colorat primul nod cu o culoare, a fost nevoie de adăugarea unei noi culori pentru nodul 2. Nodul 3, fiind adiacent cu nodurile 1 si 2, a avut, de asemenea, nevoie de o nouă culoare. Pentru nodul 4 s-a putut alege o culoare deja existenta, astfel colorând graful cu 3 culori.

2.2 Analiza complexitaţii soluţiilor

2.2.1 Backtracking

Pentru un număr de culori k, fiecare nod se poate colora cu una dintre cele k culori, generânduse k^n posibilităţi de colorare(unde n este numarul de noduri ale grafului), deci pogramul va face k^n operaţii pentru a verifica dacă graful poate fi colorat cu k culori. Ştiind că k ia valori de la 1 până la n, numarul total de operaţii devine:

$$1^n + 2^n + 3^n + \ldots + n^n$$

Rezultatul acestei sume este o funcţie de n^{n+1}, deci algoritmul are o complexitate temporala $\mathrm{O}(n^n)$. Luând in considerare că k va ajunge la valoarea n doar dacă graful este complet şi că nodurile vor ajunge toate colorate doar in cazul unei colorări valide, ne putem asigura că complexitatea temporală a algoritmului va fi mereu mai bună sau la fel ca $\mathrm{O}(n^n)$.

Complexitatea spaţiala a algorirtmului este $\mathrm{O}(n)$, întrucât este folosită memorie suplimentară pentru a păstra un vector de lungime n care reţine pe poziţia i culoarea nodului i(reţine colorarea grafului).

2.2.2 Greedy algorithm

Algoritmul greedy colorează graful prin parcurgerea nodurilor şi colorarea lor cu prima culoare disponibilă din cele folosite la nodurile anterioare sau o culoare nouă, în cazul in care niciuna din cele deja folosite nu este disponibilă.

Găsirea culorii potrivite pentru un nod necesită o parcurgere a nodurilor anterioare, în total realizânduse următorul număr de paşi:

$$0 + 1 + 2 + 3 + \ldots + n = \frac{(n-1) \cdot n}{2}$$

Deci, complexitatea temporală a algoritmului greedy este O(n^2).

Complexitatea spaţială este aceeaşi cu cea a algoritmului care foloseşte metoda backtraking, fiind nevoie de o cantitate suplimentară de memorie pentru a păstra culoarea fiecărui nod, deci O(n).

2.3 Prezentarea principalelor avantaje şi dezavantaje pentru soluţiile luate în considerare

2.3.1 Backtracking

Avantajul algoritmului care foloseşte metoda backtracking este că va colora întotdeauna graful cu un număr minim de culori, deci este exact. Dezavantajul este ca devine foarte costisitor foarte repete. Pentru grafuri cu un număr de noduri apropiat de 100, algoritmul va deveni extrem de ineficient.

Deci, algoritmul este practic doar pentru grafuri mici.

2.3.2 Greedy algorithm

Avantajul algoritmului greedy este că are o complexitate polinomială, iar dezavantajul este că nu colorează mereu graful cu numărul minim de culori. Precizia algoritmului depinde de ordinea în care sunt date nodurile. Dacă, având o colorare optima a grafului, se aranjează nodurile după culori si apoi se aplica algoritmul greedy, acesta va genera mereu tot o colorare optima a grafului. Însă aceasta ordonare a grafului este tot o problema NP-completă.

Deşi are acest dezavantaj, algoritmul greedy este mult mai practic pentru că coloreaza corect graful intr-un timp rezonabil si cu un numar de culori nu foarte diferit de numarul minim, in majoritatea cazurilor.

3 Evaluare

3.1 Descrierea modalităţii de construire a setului de teste folosite pentru validare

Pentru validare, s-au generat 8 teste cu grafuri diverse. Pe prima linie din fişierele de intrare se găsesc două numere, n şi m, care reprezintă numărul de noduri, respectiv numărul de muchii. Apoi, pe următoarele m linii, se găsesc m perechi de noduri de forma *src dest* cu semnificaţia că exista o muchie între nodul *src* şi nodul *dest*.

Testele au fost construite în aşa fel încât sa pună în evidenţă timpul si precizia fiecărui algoritm. Primul test are scopul de a putea fi verficat pe hârtie, de aceea este foarte mic. Urmatoarele teste au fost generate random de un program scris in Java şi alese pentru a putea testa algoritmii pe cazuri cât mai diverse.

3.2 Specificaţiile sistemului de calcul pe care au fost rulate testele

Testele au fost rulate pe un sistem de calcul pe 64 de biti cu procesor Intel Core i5-7200U şi memorie RAM 8GB.

3.3 Ilustrarea rezultatelor evaluării soluţiilor pe setul de teste şi observaţii

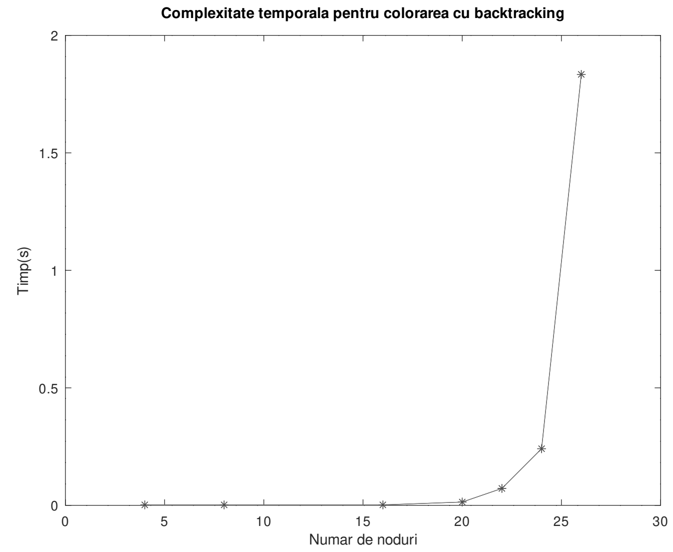


Figura 2: Complexitate temporală backtracking

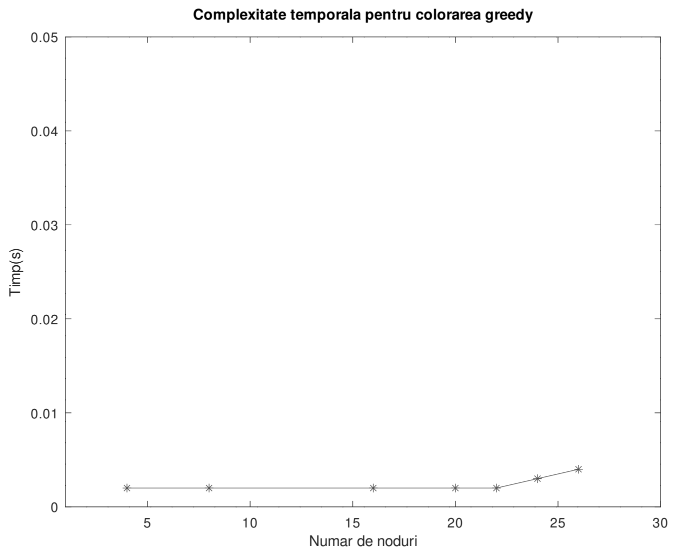


Figura 3: Complexitate temporală greedy

Se observă că pentru un număr de noduri apropiat de 30, colorarea cu backtracking devine costisitoare, în timp ce algoritmul greedy funcţionaează aproape la fel de repede pentru fiecare test. Creşterea in complexitate pentru algoritmul greedy nu este foarte vizibilă pentru că aceasta ar necesita teste foarte mari care ar fi prea costisitoare pentru algoritmul care foloseşte backtracking.

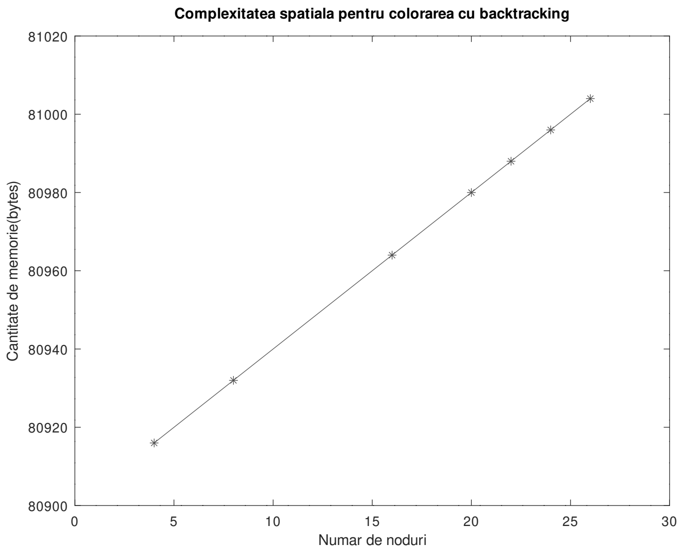


Figura 4: Complexitate spaţială backtracking

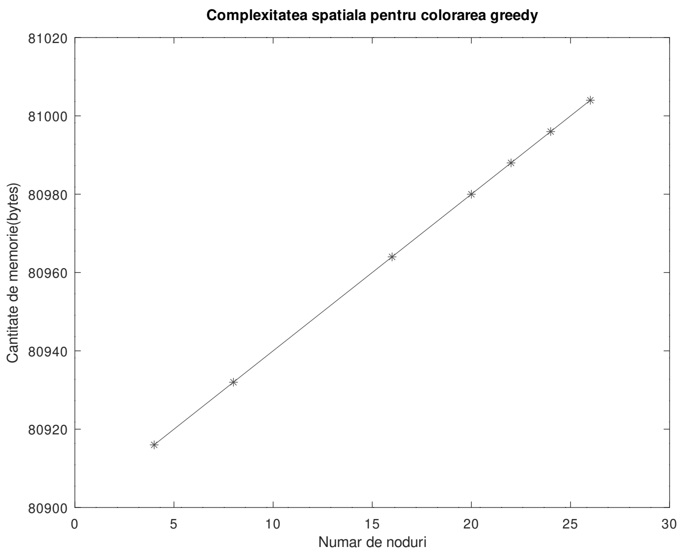


Figura 5: Complexitate spaţială greedy

Cum ambii algoritmi folosesc aceeaşi cantitate de memorie suplimentara pentru a stoca vectorul de culori pentru fiecare nod, complexitatea lor spaţiala va fi identică.

4 Concluzii

În urma testelor efectuate, se poate observa că colorarea cu algoritmul greedy este mult mai rapidă decât colorarea cu backtracking, însă nu oferă mereu o colorare optimă. Pentru grafuri de dimensiuni mici (mai putin de 25 de noduri), colorarea cu backtracking este o soluţie bună pentru că oferă o colorare optimă întru-un timp rezonabil. Dacă însă graful are dimensiuni mari sau foarte mari, colorarea cu backtracking poate dura foarte mult, deci algoritmul greedy este o variantă mai bună de rezolvare a problemei. Având în vedere că algoritmul greedy nu va duce niciodată la o colorare incorectă a grafului, acesta reprezintă o soluţie care poate rezolva multe dintre aplicaţiile practice ale acestei probleme.

Bibliografie

[1] Techie Delight. Graph coloring problem. https://www.techiedelight.com/greedy-coloring-graph/, ultima accesare: 10 Decembrie 2020.

[2] GeeksForGeeks. Graph coloring (greedy algorithm). https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/?ref=lbp, ultima accesare: 18 Decembrie 2020.

[3] GeeksForGeeks. m coloring problem — backtracking. https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/?ref=lbp, ultima accesare: 18 Decembrie 2020.