**Colegiul Național „Octavian Goga” Sibiu**

Examenul de atestare a competențelor profesionale

a absolvenților de matematică-informatică,

intensiv informatică

Aplicație analiză grafuri

orientate și neorientate

Aprilie 2021

Cuprins

[1. Scopul lucrării 3](#_Toc83986969)

[2. Tehnologii utilizate 3](#_Toc83986970)

[3. Rezumat 4](#_Toc83986971)

[4. Algoritmul 9](#_Toc83986972)

[5. Bibliografie 12](#_Toc83986973)

# Scopul lucrării

Am decis să creez acest program pentru a facilita reprezentarea și înțelegerea grafurilor și a operațiilor ce se pot efectua pe acestea.

Programul automatizează procesul de reprezentare grafică a grafurilor pornind de la matricea sa de adiacență (sau de la o listă de muchii). Programul permite crearea în mod facil a schemei unui graf și exportarea acestui graf sub formă de matrice de adiacență.

Cu ajutorul acestei aplicații, utilizatorul poate să:

* genereze, în mod automat, o reprezentare grafică a unui graf
* introducă date despre graf într-un fișier text, în consolă sau grafic (cu ajutorul mouse-ului)
* obțină informații despre un lanț/drum din graf;
* afișeze matricea de adiacență a grafului, componentele sale conexe/tari conexe etc. ;
* afișeze modul în care se realizează parcurgeri în interiorul grafului

# Tehnologii utilizate

Limbajul de programare folosit este C++, mediul de programare este Code::Blocks, iar compilatorul folosit este GNU GCC Compiler.

Code::Blocks este un IDE cross-platform gratuit, open-source, care acceptă mai multe compilatoare, inclusiv GCC, Clang și Visual C++. Code::Blocks este un IDE des folosit în mediul școlar.

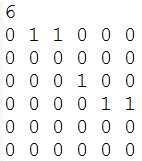
GNU Compiler Collection (Colecția de compilatoare GNU, numită de obicei GCC) este un set de compilatoare pentru diverse limbaje de programare produs de Proiectul GNU.

C++ este un limbaj de programare general, compilat. Este un limbaj multi-paradigmă, cu verificarea statică a tipului variabilelor ce suportă programare procedurală, abstractizare a datelor, programare orientată pe obiecte.

Ca bibliotecă grafică, am ales WinBGIm, deoarece este ușor de utilizat și deoarece permite input cu ajutorul mouse-ului. WinBGIm este o bibliotecă grafică Windows C++ bazată pe interfața clasică Borland Graphics Interface. Pe lângă interfața originală BGI, WinBGIm oferă, de asemenea, suport pentru accesarea mouse-ului și pentru modificarea ferestrei grafice.

# Rezumat

Datele despre graf pot fi introduse printr-o matrice de adiacență sau prin listă de muchii. Primul număr este întotdeauna numărul de noduri.

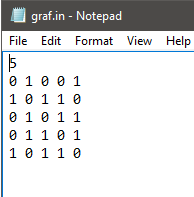
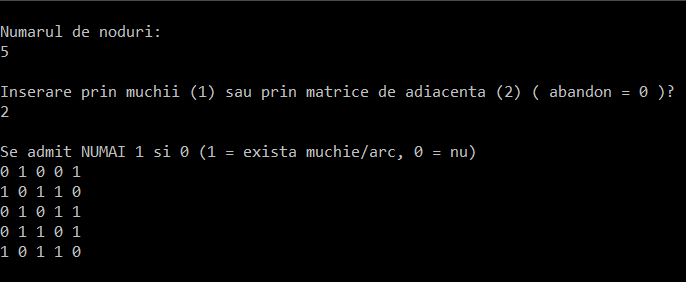




*Matrice de adiacență Listă de muchii*

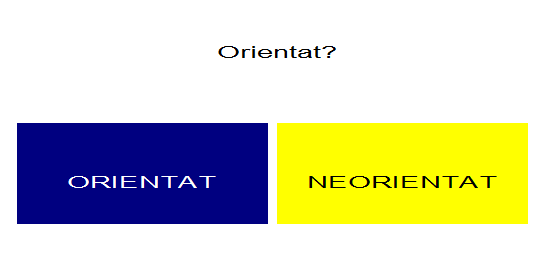
Aceste date pot fi introduse prin:

Fișierul de intrare ”graf.in” care trebuie să fie în același folder cu executabilul,

1. De la tastatură, prin intermediul consolei ce se deschide alături de aplicație.

*Fișierul graf.in poate fi editat în notepad*

*Datele introduse în consolă, conform indicațiilor*



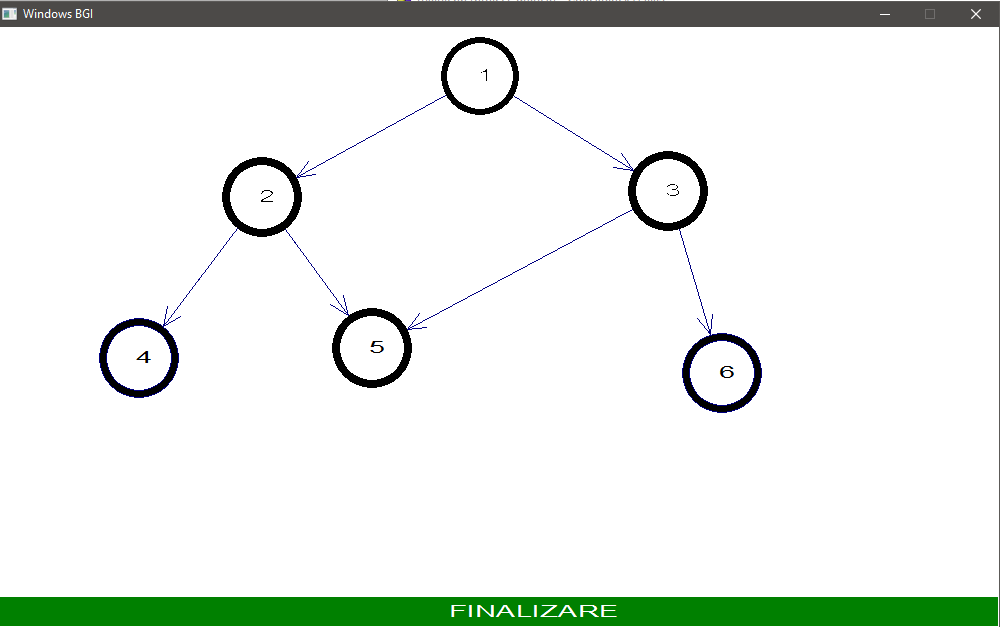
*Această întrebare va apărea numai în cazul în*

*care nu este clar dacă graful introdus este sau*

*nu orientat (dacă este dat prin listă de muchii*

*sau dacă s-a ales opțiunea de a desena graful)*

În afară de aceste două opțiuni, utilizatorul poate opta să introducă graful în mod interactiv, cu ajutorul mouse-ului. Pentru a adăuga un nod, se dă click în locul în care vrem să apară nodul. Pentru a crea o muchie între două noduri, se apasă pe primul nod, apoi pe al doilea nod (pentru grafuri orientate, ordinea contează astfel: dacă se selectează întâi nodul A, apoi nodul B, muchia va fi A→B). Dacă s-a terminat inserarea, se dă click pe „FINALIZARE”.

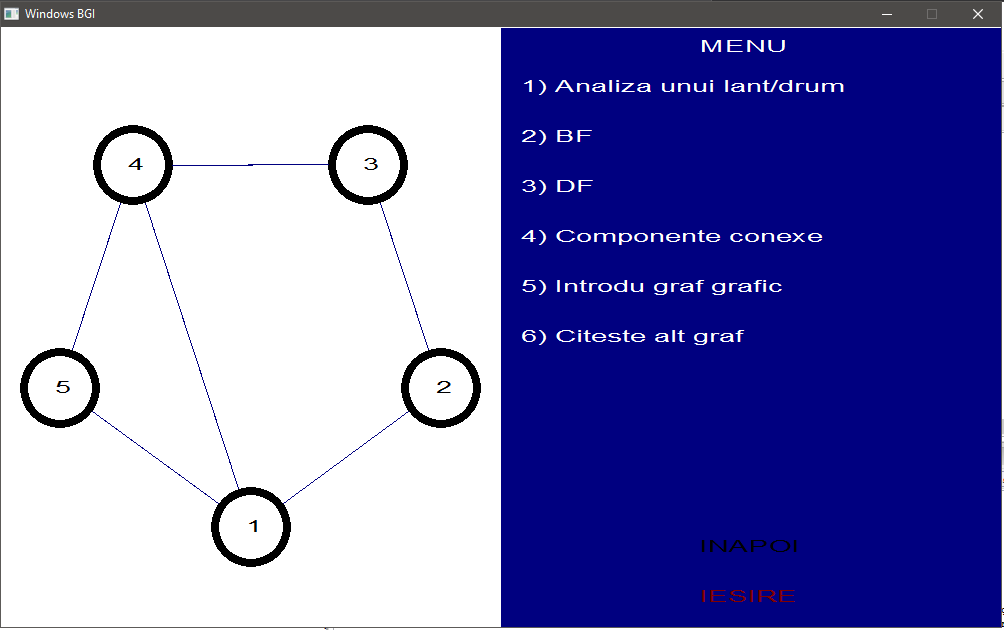


După încărcarea grafului, va apărea meniul principal, iar în consolă va apărea matricea de adiacență a grafului, care poate fi copiată.

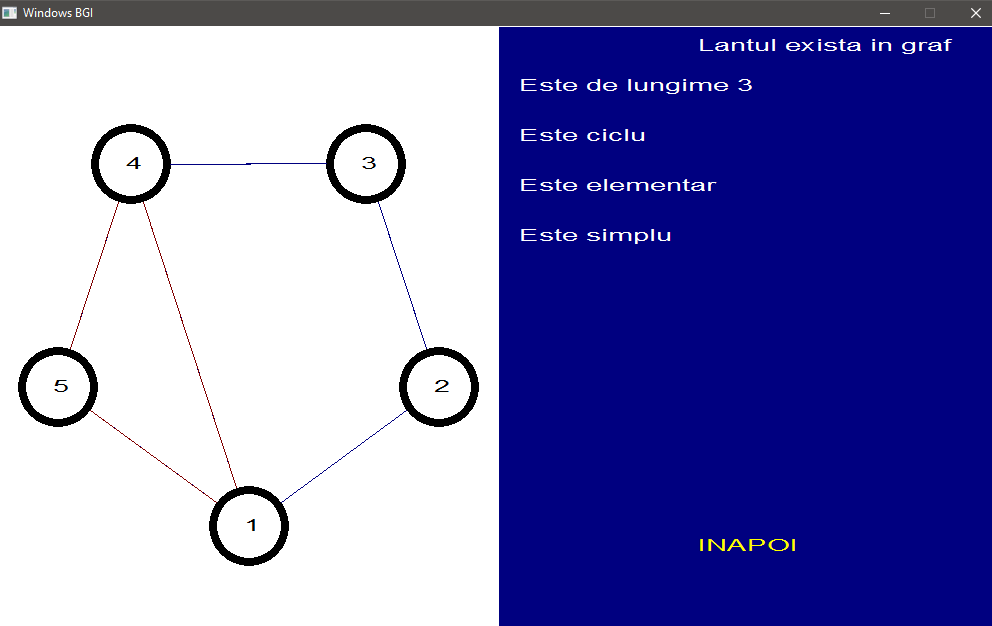
Meniul principal este împărțit în două:

1. În stânga, reprezentarea grafică a grafului, unde toate nodurile sunt puse în ordine, în cerc;
2. În dreapta, unde se află opțiunile de vizualizare și prelucrare a datelor despre graf (și butonul de ieșire din aplicație).

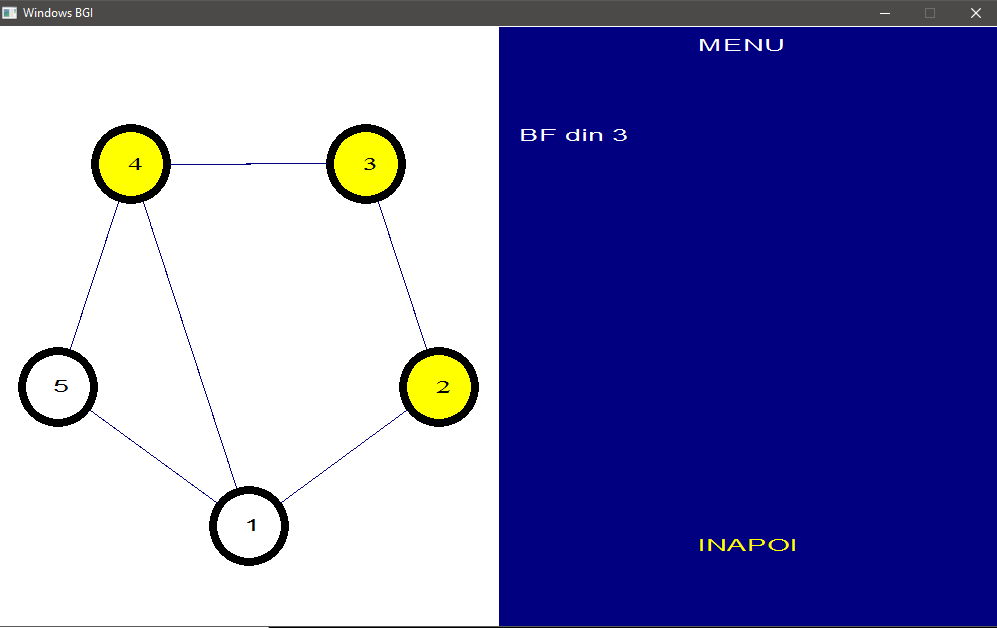
Pentru a selecta oricare opțiune din panoul din dreapta, se dă click pe opțiunea dorită.

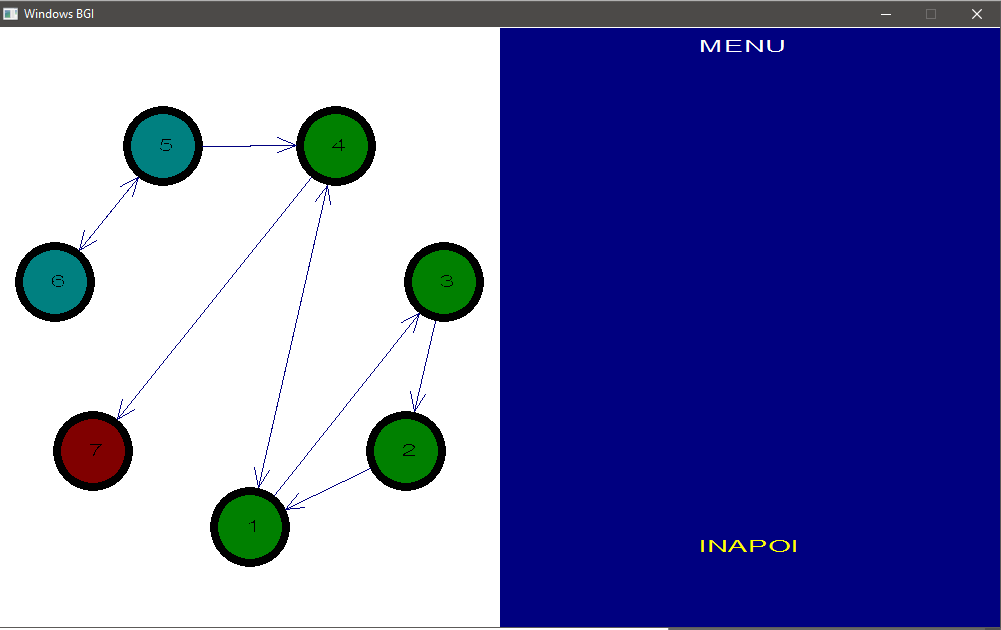


1. „Analiza unui lanț/drum” oferă informații legate de un lanț/drum din graf. Introducerea elementelor lanțului se face în consolă, conform instrucțiunilor care apar în consolă. Muchiile din lanțul selectat vor deveni roșii.



2) și 3) arată animații pentru parcurgerea în lățime (BF = breadth first) și respectiv pentru parcurgerea în adâncime (DF = Depth first) dintr-un nod al grafului. Pentru oricare dintre cele două, se alege nodul din care se începe parcurgerea. Animația se sfârșește când toate nodurile sunt parcurse (toate sunt galbene)

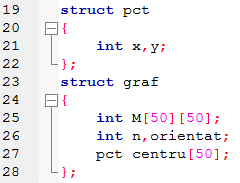


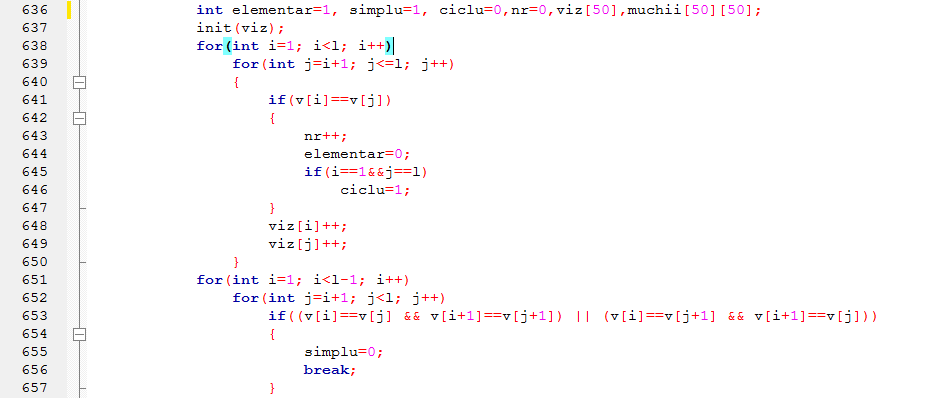
Opțiunea 4) permite afișarea componentelor conexe sau tare conexe (pentru grafuri orientate) ale grafului. Nodurile cu aceeași culoare corespund aceleiași componentă (tare) conexă.

Opțiunea 5) permite introducerea unui alt graf cu ajutorul mouse-ului, iar opțiunea 6) reia citirea din fișier. Dacă fișierul e gol, utilizatorul poate opta pentru citirea în consolă sau cu ajutorul mouse-ului.

„IESIRE” – închide programul.

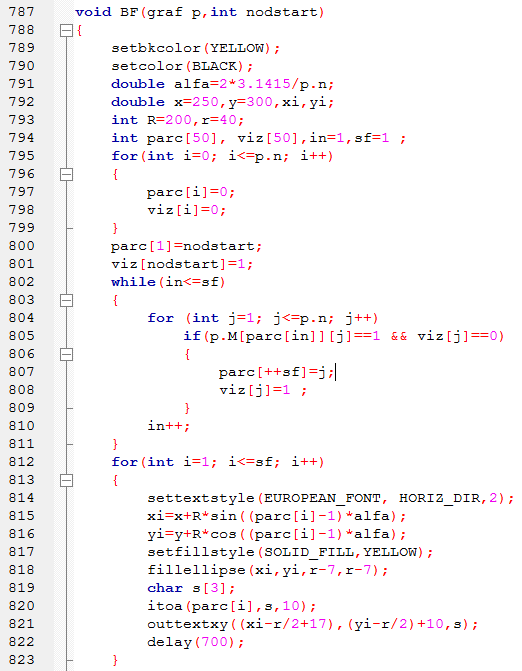
# Algoritmul

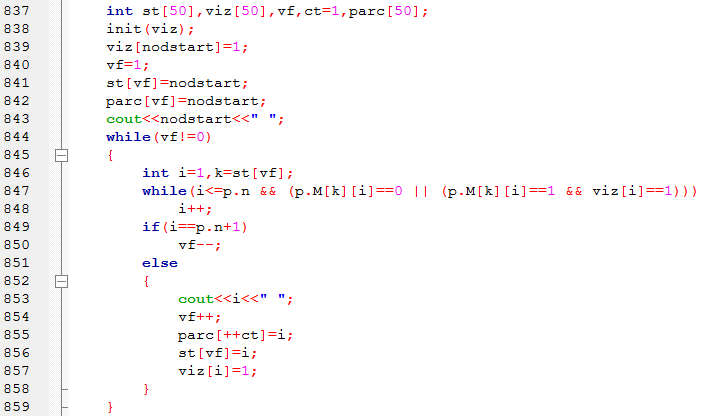
Graful este reținut în structura graf, unde matricea *M* reține matricea de adiacență, *n* reține numărul de noduri, *orientat* reține dacă graful este orientat, iar vectorul de coordonate *centru* reține coordonatele centrelor cercurilor corespunzătoare nodurilor grafului.



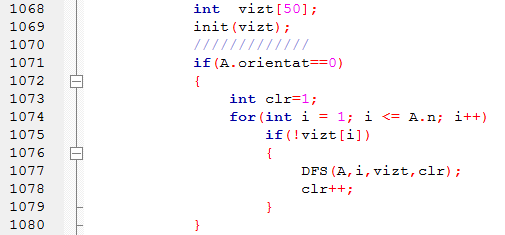
Pentru analiza lanțului, variabilele *elementar, simplu, ciclu* rețin caracteristicile lanțului, iar vectorul *v* reține elementele lanțului. Pentru fiecare muchie dintre fiecare 2 elemente consecutive ale vectorului v se verifică dacă apare în graf. Dacă există vreo muchie din lanț care nu apare în matricea de adiacență atunci lanțul nu apare în graf. Dacă primul element al vectorului v este egal cu ultimul, atunci lanțul este ciclu (*ciclu=1*). Inițial, lanțul se consideră a fi elementar. Se parcurge din nou vectorul v și dacă se găsesc două elemente egale atunci vectorul nu mai este elementar. Dacă se parcurge aceeași muchie de mai multe ori, lanțul nu mai este simplu.

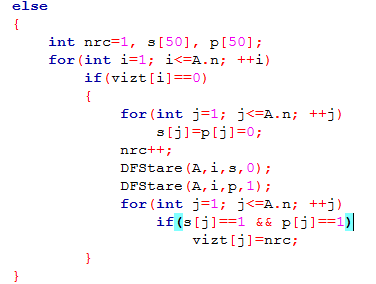
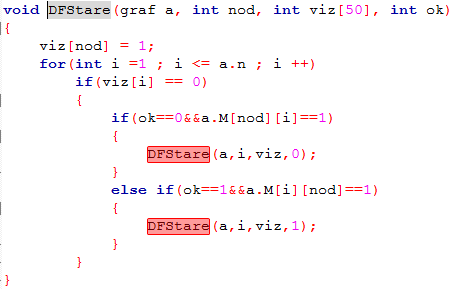
La final se afișează grafic informațiile în funcție de variabilele *elementar, simplu, ciclu,* și de lungimea *l* a vectorului *v*.

Parcurgerea în lățime se realizează folosind un vector *viz* care reține dacă un nod *i* a fost vizitat sau nu, și un vector *parc* care reține nodurile grafului în ordinea în care au fost descoperite. Prima poziție în vectorul *parc* conține primul nod, ales de utilizator. Variabila *in* reține poziția din *parc* care conține nodul din care se caută noduri nevizitate, iar *st* reține ultima poziție din *parc*. Pentru fiecare nod din vectorul *parc* se caută nodurile nevizitate vecine cu nodul curent șise adaugă la sfârșitul vectorului *parc*, apoi se iterează *in*. La final, vectorul *parc* conține elementele găsite din parcurgerea BF din nodul *nodstart* în ordinea în care au fost găsite.



Algoritmul de parcurgere DF implică utilizarea și a unei stive, pe lângă vectorii utilizați pentru BF. Pentru fiecare nod din stivă, se caută primul nod nevizitat vecin cu acesta. Dacă s-a găsit, se adaugă în stivă și se continuă căutarea de la el. Dacă nu există, se elimină elementul curent din stivă. La fel ca la BF, vectorul *parc* va reține la final ordinea nodurilor parcurse prin DF.

Componentele conexe ale grafurilor neorientate se găsesc cu ajutorul parcurgerii DF. Pentru fiecare nod nevizitat, se realizează parcurgerea DF, iar în interiorul funcției de parcurgere DF se notează cu *clr* poziția în *vizt* a nodurilor găsite, iar după ce s-au parcurs toate nodurile posibile, se incrementează *clr*. La final, vectorul *vizt* va conține ID-ul culorii corespunzătoare unei anumite componente conexe pentru fiecare din nodurile grafului.

Pentru tare conexitate se folosește algoritmul plus-minus. Pentru fiecare nod al grafului care încă nu a fost plasat într-o componentă tare conexă determinăm toate nodurile în care se poate ajunge din acel nod (folosind parcurgerea DF) și le marcăm în vectorul s[], apoi determinăm toate nodurile din care se poate ajunge în nodul curent (tot printr-o parcurgere DF) și le marcăm în vectorul p[]. Nodurile marcate atât în s[] cât și în p[] formează o componentă tare conexă.

Funcția DFStare realizează o parcurgere DF în funcție de argumentul *ok*. Dacă *ok* este 0, se parcurg numai nodurile în care se poate ajunge din nodul curent, iar dacă *ok* este 1, se parcurg numai nodurila din care se poate ajunge în nodul curent.

# Bibliografie

<https://en.cppreference.com/w/>

<https://www.pbinfo.ro/articole/6036/tare-conexitate>

<https://www.pbinfo.ro/articole/8/parcurgerea-grafurilor-neorientate>

<https://winbgim.codecutter.org/>

<https://home.cs.colorado.edu/~main/bgi/doc/>