

Logică și structuri discrete

Grafuri

Casandra Holotescu

casandra@cs.upt.ro

<https://tinyurl.com/lecturesLSD>

Teoria grafurilor și știința rețelelor

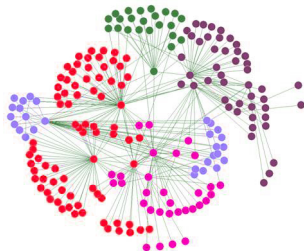
Teoria grafurilor:

studiul matematic al grafurilor
(reprezentând relații între obiecte)

De aici a evoluat

știința rețelelor (network science):

studiul rețelelor complexe:
de calculatoare, telecomunicații,
energie, biologice, sociale...



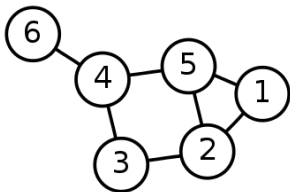
“studiul reprezentărilor ca rețele a fenomenelor fizice, biologice și sociale, ducând la *modele predictive* ale acestor fenomene”.

[US National Research Council]

Grafuri: definiție. Grafuri și relații

Definiția grafurilor

Informal, un graf reprezintă o mulțime de *obiecte* (*noduri*) între care există anumite *legături* (*muchii* sau *arce*).

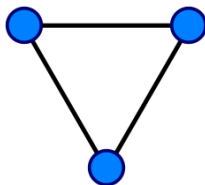
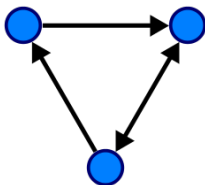


Formal, un graf e o pereche ordonată $G = (V, E)$, unde V e mulțimea nodurilor și E (mulțimea muchiilor) e o mulțime de perechi $(u, v) \in V \times V$

Grafuri orientate și neorientate

Un graf e *orientat* dacă muchiile sale sunt perechi *ordonate*

Un graf e *neorientat* dacă muchiile sale sunt perechi *neordonate*
(nu contează sensul parcurgerii)



Imagini: <http://en.wikipedia.org/wiki/File:Directed.svg>

<http://en.wikipedia.org/wiki/File:Undirected.svg>

Grafuri și relații

Mulțimea muchiilor unui graf formează o *relație* $E \subseteq V \times V$ pe mulțimea nodurilor.

Un graf *neorientat* poate fi reprezentat printr-o relație *simetrică*

$$\forall u, v \in V . (u, v) \in E \rightarrow (v, u) \in E$$

Într-un graf *orientat*, E e o relație oarecare
(nu trebuie să fie simetrică, dar poate fi)

Reciproc, *orice relație binară* poate fi văzută ca un *graf orientat*
pentru $(u, v) \in E$ introducem o muchie $u \rightarrow v$

Drumuri în graf

Drumuri în graf

Un *drum* (o cale) într-un graf e o *secvență de muchii* care leagă o *secvență de noduri* x_0, \dots, x_n cu $n \geq 0$ astfel ca $(x_i, x_{i+1}) \in E$ pentru orice $i < n$.

$$x_0 \longrightarrow x_1 \longrightarrow \dots \longrightarrow x_{n-1} \longrightarrow x_n$$

Putem defini un drum atât în grafuri orientate cât și neorientate

Un drum are un *nod inițial* x_0 și un *nod final* x_n .

Lungimea unui drum e numărul de muchii.

în particular, poate fi zero (un nod x_0 , fără niciun fel de muchii)

Drumuri și închiderea tranzitivă

Mulțimea tuturor drumurilor de *lungime nenulă* este *închiderea tranzitivă* a relației E :

$$E^+ = \bigcup_{k=1}^{\infty} E^k = E \cup E^2 \cup \dots$$

relația E^k ($k \geq 1$) corespunde drumurilor de lungime k

$$E^2 = E \circ E = \{(u, v) \mid \exists w. (u, w) \in E \wedge (w, v) \in E\}$$

$u \rightarrow w \rightarrow v$

$$E^3 = E^2 \circ E = \{(u, v) \mid \exists w. (u, w) \in E \wedge (w, v) \in E^2\} \quad \text{etc.}$$

$u \rightarrow w \xrightarrow{2\text{pasi}} v$ adică $u \rightarrow w \rightarrow w' \rightarrow v$

Cicluri în graf

Un *ciclu* e un drum de *lungime nenulă* în care nodurile de început și sfârșit sunt identice (aceleași).

Adeseori, lucrăm cu *cicluri simple*:

cicluri în care muchiile și nodurile *nu apar de mai multe ori* (cu excepția nodului inițial care e și cel final).

Grafuri & componente conexe

Grafuri și componente conexe

Un graf e *conex* dacă are un drum *de la orice nod la orice nod*.
(definiție generală, depinde de noțiunea de *drum*
– în graf orientat sau neorientat)

Pentru grafuri *neorientate*:

O *componentă conexă* e un subgraf conex maximal.
deci are un drum între oricare două noduri
nu s-ar mai putea adăuga alte noduri păstrând-o conexă

Un graf cu n noduri și e muchii are $\geq n - e$ componente conexe.

Demonstrăm prin inducție după e .

$e = 0 \Rightarrow$ fiecare nod e o componentă conexă.

$e > 1$: ștergem o muchie \Rightarrow obținem cel mult o componentă în plus

Grafuri orientate: slab conexe și tare conexe

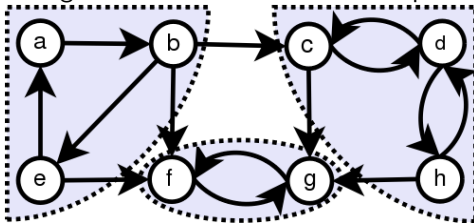
Un graf *orientat* e *slab conex* dacă are un drum *neorientat* de la orice nod la orice nod, și *tare conex* dacă are un drum *orientat* de la orice nod la orice nod.

O *componentă tare conexă* e un *subgraf* tare conex maximal.

Componentele tare conexe sunt *disjuncte*:

$R(u, v) : \text{drum}(u, v) \wedge \text{drum}(v, u)$ e o *relație de echivalență*, și componentele tare conexe sunt *clase de echivalență*.

Graful orientat din figură e *slab conex*. Are trei componente *tare conexe*.



Determinarea componentelor conexe (graf neorientat)

Componentele conexe sunt *clase de echivalență*

orice nod e în componenta proprie

un drum de la u la v e și drum de la v la u

$drum(u, v) \wedge drum(v, w) \rightarrow drum(u, w)$

reflexivitate

simetrie

tranzitivitate

Determinăm componentele conexe parcurgând muchiile grafului:

inițial, fiecare nod e în propria componentă

pentru o muchie (u, v) *unim* componentele lui u și v

Drumuri Euleriene (în grafuri neorientate)

Gradul unui nod (într-un graf neorientat) e numărul de muchii care ating nodul.

Un *drum eulerian* e un *drum* care conține *toate* muchiile unui graf exact o dată.

Un *ciclu eulerian* e un *ciclu* care conține *toate* muchiile unui graf exact o dată.

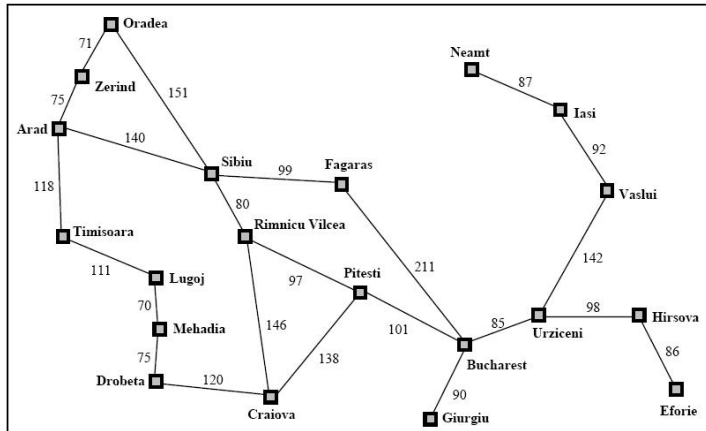
Un graf conex neorientat are un *ciclu* eulerian dacă și numai dacă *toate nodurile au grad par*.

Un graf conex neorientat are un *drum* (dar nu și un ciclu) eulerian dacă și numai dacă *exact două noduri au grad impar*.

(primul și ultimul nod din drum)

Exemple: hărțile ca și grafuri ponderate

Graf ponderat: fiecare muchie are asociată o valoare numerică numită *cost* (poate reprezenta lungime, capacitate, etc.)



Exemple: Graful fluxului de control (control flow graph)

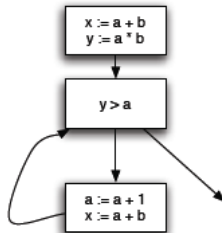
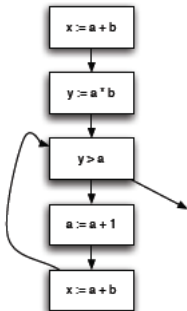
reprezentarea programelor în compilatoare, analizare de cod, etc.

nodurile: *instrucțiuni*

sau secvențe liniare de instrucțiuni (*basic blocks*)

muchiile: descriu secvențierea instrucțiunilor (*fluxul de control*)

```
x := a + b;  
y := a * b;  
while (y > a) {  
  a := a + 1;  
  x := a + b  
}
```

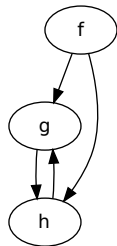


Exemple: Graful de apel al funcțiilor (call graph)

Introducem o muchie $f \rightarrow g$ dacă funcția f apelează pe g

\Rightarrow graful de apel e ciclic dacă există funcții (direct sau indirect) recursive

```
let rec g n =  
  if n = 0 then 0 else 1 + h (n-1)  
and h n =  
  if n = 0 then 1 else 2 * g (n-1)  
let f n = g n + h n
```



Reprezentare și parcurgeri

Reprezentarea grafurilor

Dacă identificăm nodurile prin numere (consecutive), putem reprezenta graful ca *matrice de adiacență* pătratică

$M[i,j] = 1$ dacă *există* muchie de la i la j

$M[i,j] = 0$ dacă *nu există* muchie de la i la j

sau $M[i,j]$ poate conține lungimea/costul muchiei (graf ponderat)

Reprezentarea prin *liste de adiacență*

pentru fiecare nod u : lista/mulțimea nodurilor v cu muchii (u, v)

putem păstra informația într-un *dicționar*:

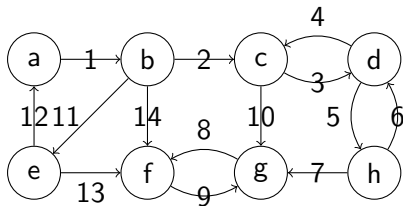
nod = cheie

valoare = lista/mulțimea nodurilor adiacente

Parcurgerea în adâncime (depth-first)

e o traversare în *preordine*
după vizitarea nodului se parcurg (recursiv)
toți vecinii (dacă nu au fost vizitați încă)
ca și cum vecinii ar fi introduși într-o *stivă*

Fie graful de mai jos, cu listele de adiacență ordonate după litere
Ordinea muchiilor parcurse de la *a* în adâncime e cea indicată:

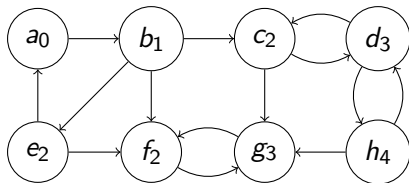


O implementare: funcție recursivă, acumulând mulțimea nodurilor vizitate

Parcurgerea prin cuprindere (breadth-first)

vizitează nodurile în ordinea distanței minime de nodul de plecare
(în “valuri” care se depărtează de la nodul de pornire)
nodurile încă nevizitate se pun într-o *coadă*

În figura de mai jos, se indică distanța minimă de la nodul a
(noduri cu distanță mai mare sunt parcurse mai târziu)



O implementare: funcție recursivă, acumulând:
mulțimea tuturor nodurilor vizitate

frontiera: mulțimea nodurilor noi atinse în runda curentă