Logică și structuri discrete

Grafuri. Calculabilitate

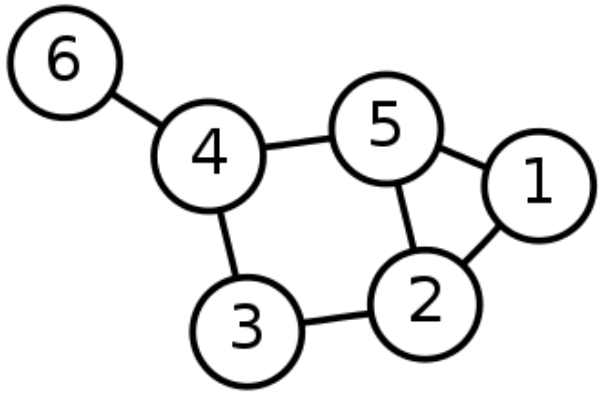
Marius Minea
marius@cs.upt.ro

http://www.cs.upt.ro/~marius/curs/lsd/

5 ianuarie 2015

Definiția grafurilor

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



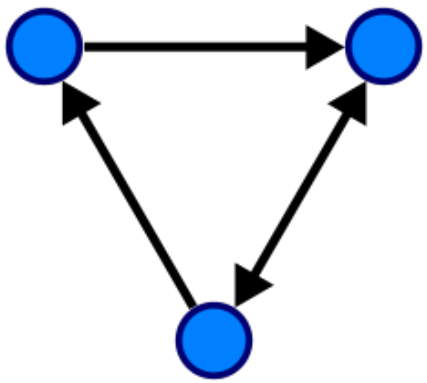
Formal, un graf e o pereche ordonată $G = (V, E)$, unde
V e mulțimea nodurilor și
E (mulțimea muchiilor) e o mulțime de perechi $(u, v) \in V \times V$

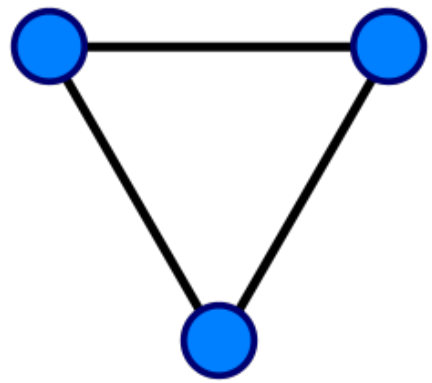
Imagine: `http://en.wikipedia.org/wiki/File:6n_graf.svg`

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

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

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

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

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

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

Drumuri în graf

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

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

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

Drumurile de lungime nenulă sunt date de *închiderea tranzitivă* a relației E:

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

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

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

Putem deasemenea defini un predicat *drum* cu proprietățile
$\forall (u, v) \in E \,.\, drum(u, v)$
$\forall (u, v) \in E \,.\, (\exists w \in W \,.\, (u, w) \in E \wedge drum(w, v)) \rightarrow drum(u, v)$

Cicluri în graf

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

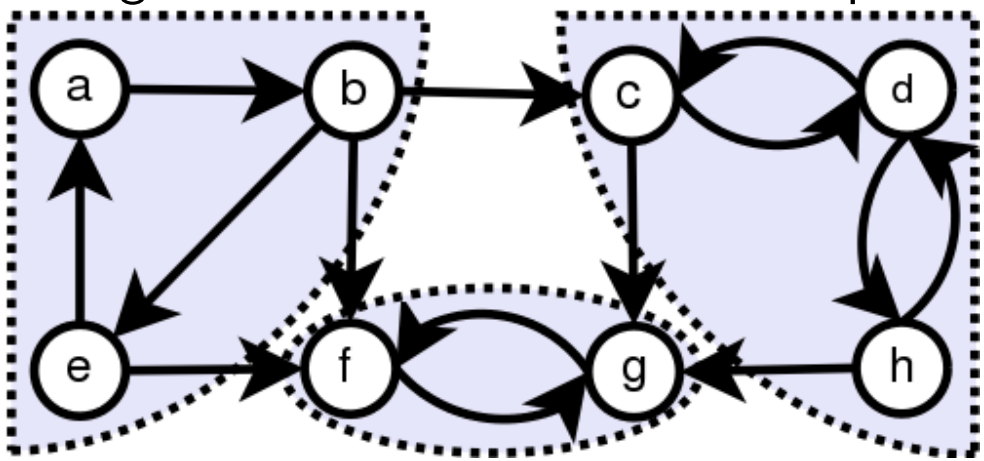
Grafuri conexe și tare conexe

Un graf e *conex* dacă are un drum de la orice nod la orice nod.

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 al unui graf.
Componentele tare conexe sunt *disjuncte*:
relația $R(u, v) : drum(u, v) \wedge drum(v, u)$ e o *relație de echivalență*,
și componentele tare conexe sunt *clase de echivalență*.

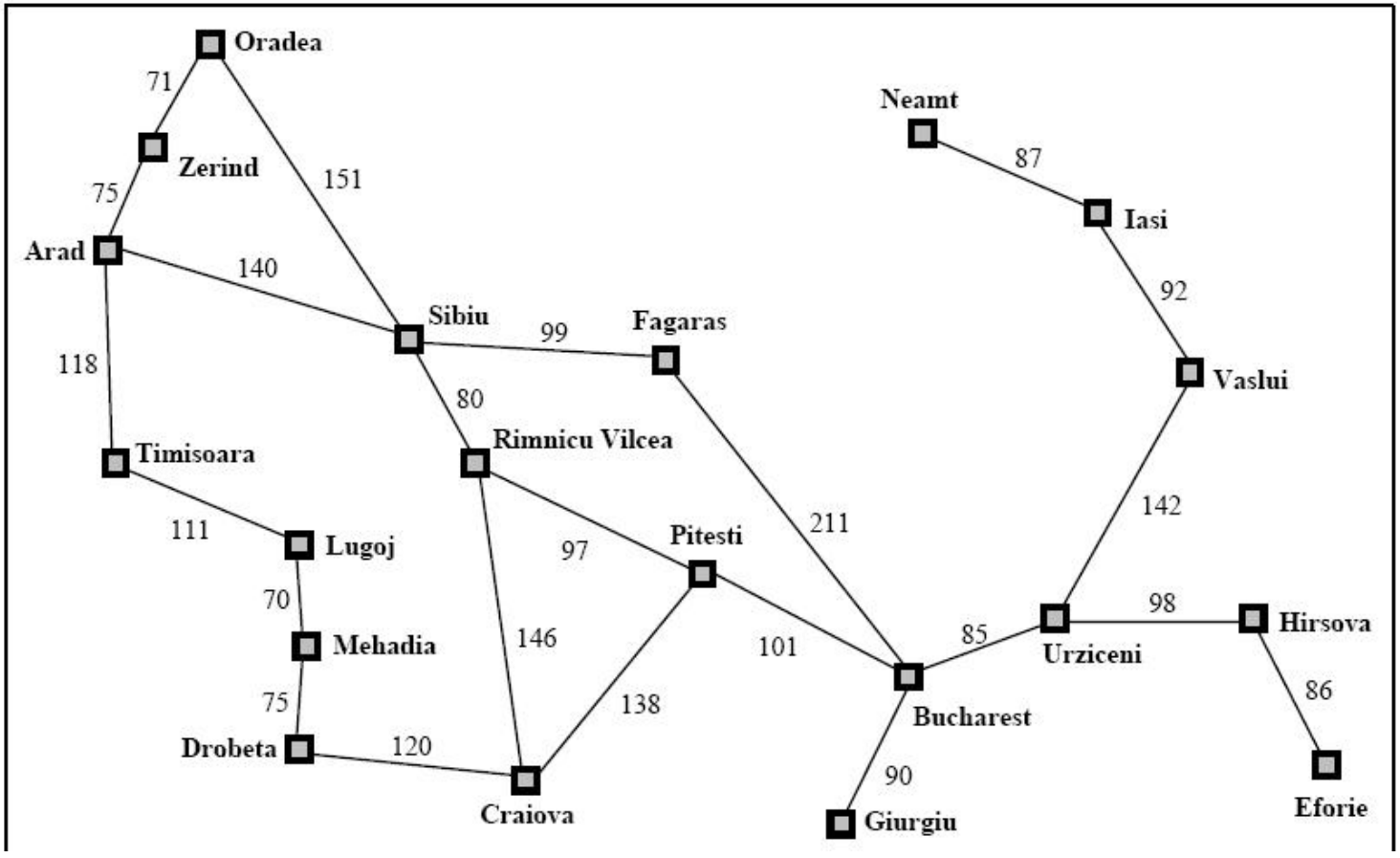
Graful orientat din figură e slab conex. Are trei componente tare conexe.



Imagine: `http://en.wikipedia.org/wiki/File:Scc.png`

Exemple: hărțile ca și grafuri ponderate

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



Hartă (inexactă) din Russell & Norvig, Introduction to AI

Drumuri Euleriene (în grafuri neorientate)

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

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

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.

Exemple: Graful fluxului de control (control flow graph)

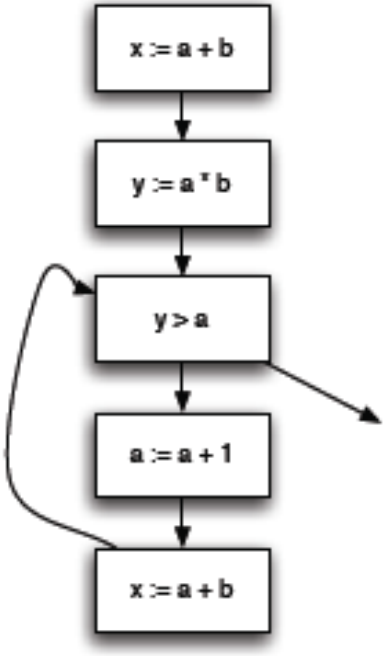
reprezentarea programelor în compilatoare, analizoare de cod, etc.

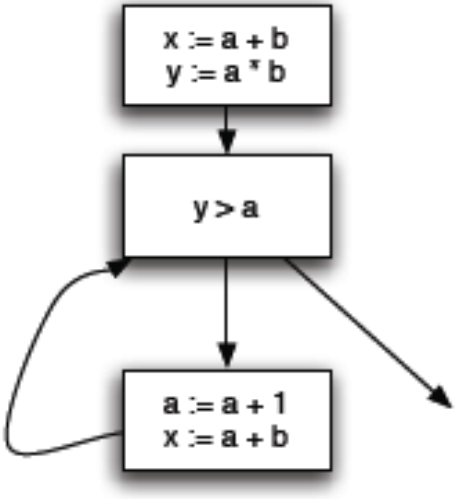
nodurile: *instrucțiuni*

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
}
```





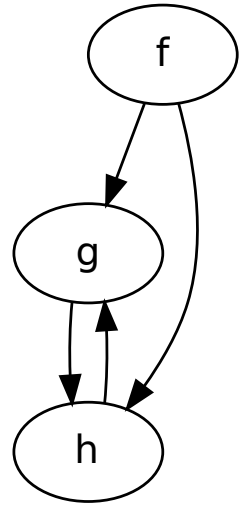
http://vinaytech.wordpress.com/2008/10/04/abstract-syntax-tree/

Exemple: Graful de apel al funcțiilor (call graph)

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

$\Rightarrow$ graful de apel e ciclic dacă există funcții (mutual) 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
```



Traversarea grafurilor

Traversarea în adâncime (depth-first)

e o traversare în *preordine*

nodurile încă nevizitate se pun într-o *stivă*

Traversarea prin cuprindere (breadth-first)

vizitează nodurile în ordinea distanței minime de nodul de plecare

nodurile încă nevizitate se pun într-o *coadă*

În încheiere: Calculabilitate

Ce se poate *calcula*, și cum putem defini această noțiune ?

Teza Church-Turing

o afirmație despre noțiunea de *calculabilitate*:
următoarele modele de calcul sunt echivalente:

- lambda-calculul
- mașina Turing
- funcțiile recursive

Lambda-calcul

Definit de Alonzo Church (1932); poate fi privit ca fiind cel mai simplu limbaj de programare

O expresie în lambda-calcul e fie:

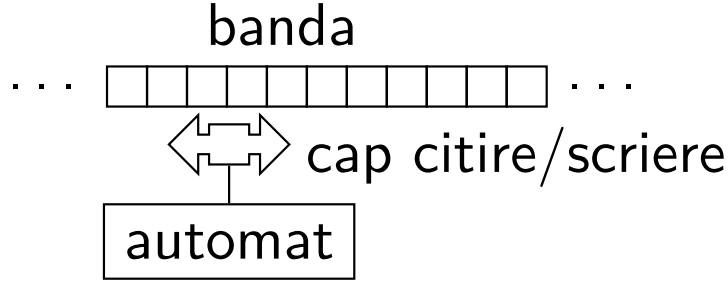
- o *variabilă* — x
- o *funcție* — $\lambda x\,.\,e$ (funcție de variabilă x)
  în ML: `fun x -> e`
- o *evaluare* de functie — $e_1\ e_2$ (funcția e_1 aplicată argumentului e_2)
  la fel în ML: `f 3` fără paranteze
  asociativă la stânga: `f x y` = `(f x) y`

Toate noțiunile fundamentale (numere naturale, booleni, perechi, etc.) pot fi exprimate în lambda-calcul.

Mașina Turing

Mașina Turing e compusă din:

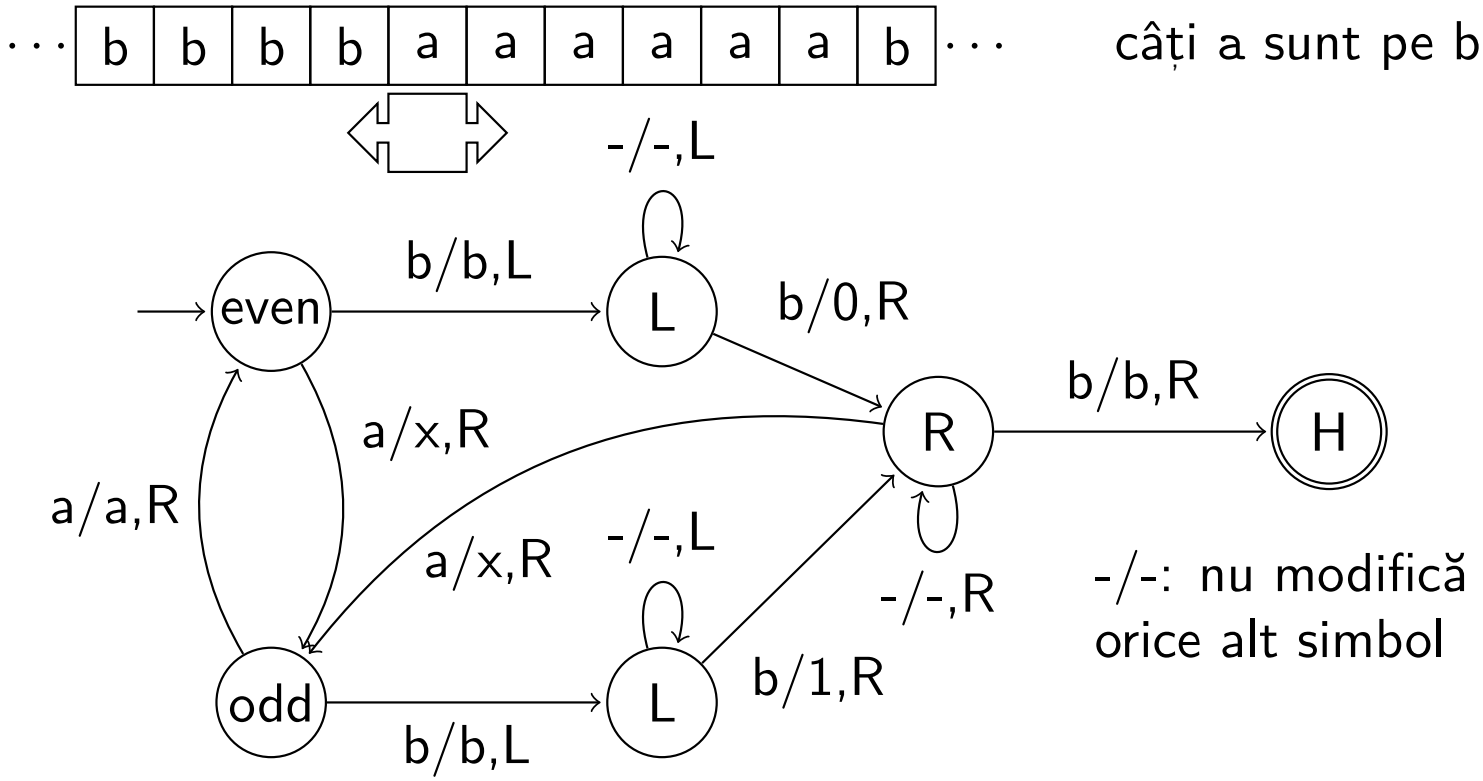
- o *bandă* cu un număr infinit de *celule*; fiecare conține un *simbol*
  (banda poate fi infinită la unul/ambele capete, e echivalent)
- un *cap* de citire/scriere, controlat de un *automat cu stări finite*



Automatul și conținutul benzii determină comportarea.
După 1) starea curentă și 2) simbolul aflat sub cap, mașina:
1) trece în starea următoare, 2) scrie un (alt) simbol sub cap
3) mută capul la stânga sau la dreapta

Inițial, banda are un șir finit de simboluri, capul e pe cel din stânga;
restul celulelor conțin un simbol special (numit vid sau blanc).

Exemplu: numără simboluri și scrie în binar



obține fiecare bit din numărul de a
$\rightarrow$: schimbă tot al doilea a cu x
$\leftarrow$: scrie 0 sau 1 după paritate
repetă până nu mai sunt a: Halt

```
   bbbbaaaaaab → bbbbxaxaxab
← bbb0xaxaxab → bbb0xxxaxxb
← bb10xxxaxxb → bb10xxxxxxb
← b110xxxxxxb → b110xxxxxxb
Halt
```

Mașina Turing – descriere formală

Formal, mașina Turing se descrie printr-un tuplu cu 7 elemente:

Q: mulțimea stărilor automatului finit (de control)
Σ: mulțimea finită a *simbolurilor de intrare* (din șirul inițial)
Γ: mulțimea simbolurilor de pe bandă; $\Sigma \subset \Gamma$
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{l, r\}$: funcția de tranziție:
dă starea următoare, simbolul cu care e înlocuit cel curent, și mutarea la stânga sau dreapta
(în unele versiuni, echivalente, capul poate și rămâne pe loc)
$q_0 \in Q$: starea inițială a automatului de control
$b \in \Gamma \setminus \Sigma$: simbolul vid (blanc): toate celulele cu excepția unui număr finit sunt inițial vide
$F \subseteq Q$: mulțimea stărilor finale, automatul se oprește (halt)

Poate descrie *orice calcul* (implementabil prin program)
Nu există algoritm care să decidă pentru orice automat și intrare dacă se oprește (*halting problem*) – la fel pentru programe