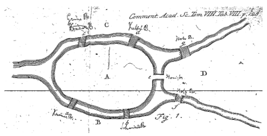
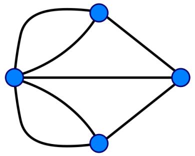




Algoritmica grafurilor

II. Reprezentări, parcurgeri în grafuri, drumuri

Mihai Suciu

Facultatea de Matematică și Informatică (UBB)
Departamentul de Informatică

Martie, 9, 2023

Conținut

Reprezentarea / memorarea grafurilor

- în general se alege una din două variante pentru a reprezenta un graf G=(V,E):
  - liste de adiacență
  - matrice de adiacență
- ambele variante pot fi folosite pentru grafuri orientate sau neorientate
- deoarece reprezentarea prin listă de adiacență este mult mai compactă este de preferat în cazul grafurilor rare

Graf rar

un graf este rar dacă $|E| << |V|^2$

Reprezentări (II)

- reprezentarea prin matrice de adiacență este preferată în cazul grafurilor dense

Graf dens

un graf este dens dacă $|E| \approx |V|^2$

- sau când trebuie să stabilim rapid dacă o muchie (sau un arc) leagă două vârfuri

Listă de adiacență

- pentru un graf $G = (V, E)$ lista de adiacență reprezintă o matrice de $|V|$ liste

Lista de adiacență pentru un nod

fie $x \in V(G)$, lista de adiacență pentru nodul x, $Adj[x]$, conține toate vârfurile j astfel încât $\{x, j\} \in E(G)$

- $Adj[x]$ constă din toate nodurile adiacente lui x din G
- suma lungimilor fiecărei liste într-un
  - graf orientat este $|E|$
  - graf neorientat este $2|E|$
- pentru un graf ponderat se salvează ponderea împreună cu vârful în listă
- reprezentarea sub formă de lista de adiacență necesită $\Theta(V + E)$ memorie

Matrice de adiacență

- un dezavantaj al listei de adiacență este: nu putem determina rapid dacă muchia $\{x, y\}$ aparține grafului G
- trebuie cautat vârful y în $Adj[x]$
- pentru a elimina acest dezavantaj se folosește matricea de adiacență

Matrice de adiacență

fie un graf $G = (V, E)$, reprezentarea sub formă de matrice de adiacență $A = (a_{ij})$ pentru G este o mtrice de dimensiune $|V|x|V|$ unde

$$a_{ij} = \begin{cases} 1, & (i,j) \in E \\ 0, & (i,j) \notin E \end{cases}$$

Matrice de adiacență (II)

- matricea de adiacență necesită $\Theta(V^2)$ memorie
- pentru un graf neorientat A este simetrică de-a lungul diagonalei principale
- pentru grafuri ponderate a_{ij} este ponderea muchiei
- avantaje:
  - reprezentare mai simplă
  - pentru un graf neorientat și neponderat este nevoie de un singur bit pentru un element din matrice

Matrice de incidență

matrice de incidență

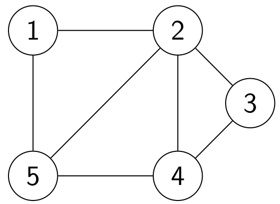
matricea de incidență pentru un graf simplu orientat $G = (V, E)$ este o matrice, $B = (b_{ij})$, de dimensiunea $|V|x|E|$ unde

$$b_{ij} = \begin{cases} 1, & \exists j \in V | e = \{i, j\}, \\ -1, & \exists j \in V | e = \{j, i\}, i \in V, e \in E \\ 0, & altfel \end{cases}$$

E fiind sortată.

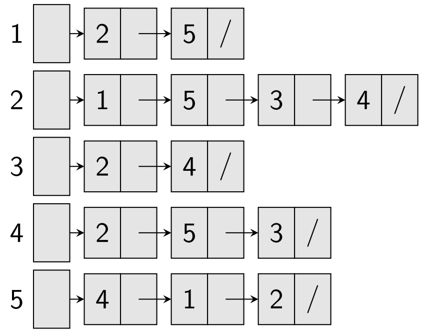
Exemplu - graf neorientat

graf → listă adiacență → matrice de adiacență



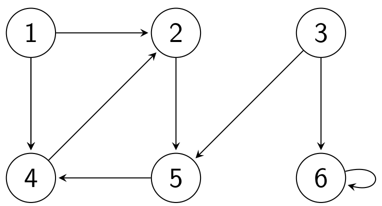
Lista de adiacență și matricea de adiacență

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

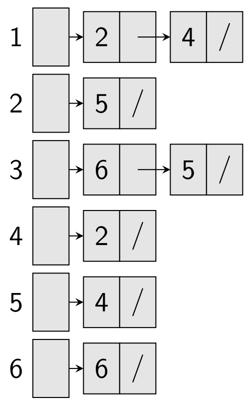


Exemplu - graf orientat

graf → listă adiacență → matrice de adiacență

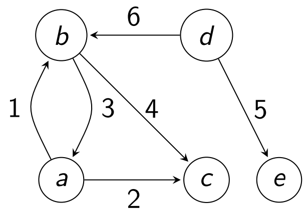


Lista de adiacență și matricea de adiacență



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Exemplu - graf orientat, matricea de incidență



$$B = \begin{pmatrix} 1 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Parcurgere în lățime (Breadth-first search BFS)

- un algoritm simplu de căutare în grafuri
- mai mulți algoritmi folosesc idei similare BFS (Prim's minimum-spanning-tree, Dijkstra's single-source shortest-path)

algoritmul BFS

dându-se un graf $G = (V, E)$ și un vârf **sursă** s, algoritmul de parcurgere în lățime explorează sistematic muchiile lui G pentru a descoperi fiecare vârf **accesibil din s**

- algoritm pentru grafuri orientate / neorientate
- algoritmul construiește un arbore cu rădăcina în s, arbore ce conține toate vârfurile accesibile
- pentru fiecare vârf v accesibil din s, lanțul simplu din arbore reprezintă lanțul minim dintre s și v

BFS (II)

- se numește căutare în lățime deoarece algoritmul BFS descoperă toate vârfurile accesibile la distanță k de vârful sursă după care trece la vârfurile de distanță $k + 1$

Exemplu:

- pentru a urmări progresul sunt trei tipuri de vârfuri: albe, gri și negre:
  - alb - nu a fost vizitat
  - negru - dacă $\{u, v\} \in E$, vârful u este negru, vârful v este negru sau gri
  - gri - poate avea adiacent vârfuri albe (vârfurile gri reprezintă frontiera între vârfurile descoperite și cele nedescoperite)
- BFS construiește inițial un arbore ce conține doar vârful sursă s, sunt adăugate vârfuri noi pe măsura ce sunt descoperite

BFS (III)

- procedura presupune graful reprezentat ca și listă de adiacență
- atributul π ține vârful predecesor, atributul d ține distanța de la sursă la nodul curent

BFS (IV) - procedura

```
BFS(G, s)
  for fiecare vârf u ∈ G, V − {s} do
      u.color = alb
      u.d = ∞
      u.π = NIL
  end for
  s.color = gri
  s.d = 0
  s.π = NIL
  Q = ∅
  Enqueue(Q,s)
  while Q ≠ ∅ do
      u = Dequeue(Q)
      for fiecare v ∈ G.Adj[u] do
          if v.color == alb then
              v.color = gri
              v.d = u.d + 1
              v.π = u
              Enqueue(Q,v)
          end if
      end for
      u.color = negru
  end while
```


BFS

- durata în timp a algoritmului este $O(V + E)$

Exemplu

BFS - drumuri / lanțuri elementare minime

- BFS găsește distanța de la nodul sursă s la nodurile accesibile din G

Lanț elementar de distanță minimă

se definește lanțul elementar de distanță minimiă $\delta(s, v)$ de la vârful s la vârful v ca și lanțul elementar între s și v ce conține numărul minim de muchii. Dacă nu există un lanț elementar între vârfurile s și v atunci $\delta(s, v) = \infty$

Lema

fie $G = (V, E)$ un graf orientat sau neorientat și $s \in V$ un vârf ales arbitrar. Pentru oricare arc / muchie $\{u, v\} \in E$

$$\delta(s, v) \leq \delta(s, u) + 1.$$

BFS - drumuri / lanțuri elementare minime (II)

Lema

fie $G = (V, E)$ un graf orientat sau neorientat și BFS e rulat pe G din nodul sursă $s \in V$. După ce a terminat BFS, pentru fiecare $v \in V$, valoarea $v.d$ calculată de BFS satisface

$$v.d \geq \delta(s, v).$$

Lema

dacă în timpul execuției BFS pe un graf $G = (V, E)$ coada Q conține vârfurile $\{v_1, v_2, ..., v_r\}$, unde v_1 este în vârful cozii și v_r este vârful din coada. $v_r.d \leq v_1.d + 1$ și $v_i.d \leq v_{i+1}.d$ pentru $i = 1, 2, ..., r - 1$.

BFS - drumuri / lanțuri elementare minime (III)

Corolar

fie vârfurile v_i și v_j introduse în coadă pe parcursul execuției BFS, vârful v_i este prelucrat înaintea lui v_j. Atunci $v_i.d \leq v_j.d$ în momentul în care v_j este prelucrat.

Teorema: corectitudine BFS

fie $G = (V, E)$ un graf orientat sau neorientat și BFS e rulat pe G din nodul sursă $s \in V$. Pe parcursul execuției BFS descoperă fiecare vârf $v \in V$ accesibil din s și la final $v.d = \delta(s, v), \forall v \in V$. Pentru orice vârf $v \neq s$ care e accesibil din s, unul din lanțurile elementare de dimensiune minimă din s în v este un lanț elementar de dimensiune minimă din s în $v.\pi$ urmat de muchia $\{v.\pi, v\}$.

Parcurgere în adâncime (Depth-first search DFS)

- algoritm de parcurgere care exploreaza muchiile vârfurilor nou descoperite
- dupa ce au fost explorate toate muchiile dintr-un vârf v, algoritmul se întoarce la vârful muchiei care a dus în v și continuă explorarea
- procesul se repetă până au fost explorate toate vârfurile accesibile din sursă
- dacă rămân vârfuri neexplorate, DFS alege unul dintre ele ca și sursă si continua execuția

DFS (II)

Exemplu

- algoritmul colorează vârfurile pe parcursul căutării similar cu BFS, prin culoare se indică starea nodului
- pe lângă stare DFS marchează și timpul când a fost descoperit vârful și timpul când a fost explorat complet arborele din vârful descoperit
  - pentru a masura performanța algoritmului
  - pentru a descoperi structura grafului
- $u.d$ marchează timpul când a fost descoperit vârful u
- $u.f$ marchează timpul când a fost explorat vârful u
- starea unui vârf: alb - $u.d$ - gri - $u.f$ - negru

DFS - procedura

```
DFS(G)
  for fiecare vârf u ∈ G.V do
     u.color = alb
     u.π = NIL
  end for
  time = 0
  for fiecare u ∈ G.V  do
     if u.color == alb then
        DFS_VISIT(G,u)
     end if
  end for
```

DFS - procedura (II)

DFS_VISIT(G, u)
    $time = time + 1$
    $u.d = time$
    u.color = gri
    for fiecare $v \in G.Adj[u]$ **do**
        if v.color == alb **then**
            $v.\pi = u$
            DFS_VISIT(G,v)
        end if
    end for
    u.color = negru
    $time = time + 1$
    $u.f = time$

DFS

- durata în timp a algoritmului este:
  - în timpul execuției bucla din DFS_VISIT se execută de $|Adj[v]|$ ori, deoarece
    $$\sum_{v \in V} |Adj[v]| = \Theta(E)$$
    costul buclei este $\Theta(E)$
  - durata de execuție a algoritmului este $\Theta(V + E)$

DFS - proprietăți

Teoremă

fie $G = (V, E)$ un graf orientat sau neorientat, în DFS pentru oricare noduri u și v una din următoarele condiții este adevărată:

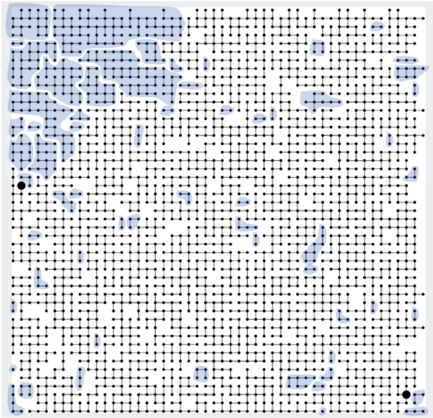
- intervalele $[u.d, u.f]$ și $[v.d, v.f]$ sunt disjuncte, u și v nu sunt descendenți unul altuia
- intervalul $[u.d, u.f]$ este inclus $[v.d, v.f]$, u este un descendent al lui v
- intervalul $[v.d, v.f]$ este inclus in $[u.d, u.f]$ și v este un descendent al lui u

Exemple

- Relația: Din orice punct puteți ajunge la orice punct
- Câte componente conexe are următorul graf?

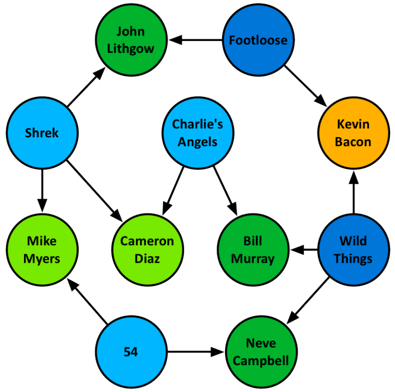
Exemple

- Relația: Din orice punct puteți ajunge la orice punct
- Câte componente conexe are următorul graf?



Exemple (II)

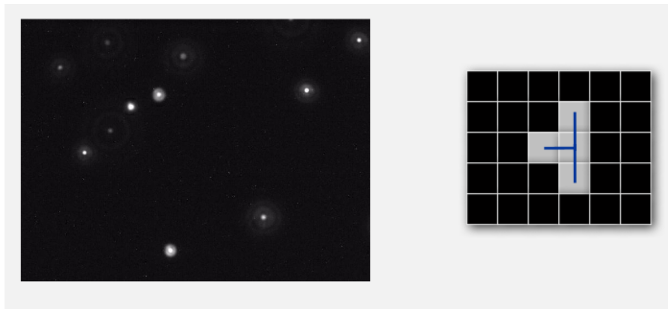
- facebook - sugestie de noi prieteni pe baza BFS
- numărul Kevin Bacon / Erdős Pál



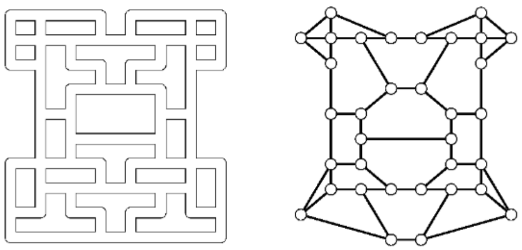
Exemple (II)

Exemple (III) - prelucrare de imagini

- să se caute stelele mai mari din imagine



Exemple (IV) - parcurgerea unui labirint



Algortmul lui Thremaux - secolul 19, bazat pe DFS

Graf tare conex, slab conex

Fie un graf orientat $G = (V, E)$

Graf tare conex

un graf orientat este **tare conex** dacă între oricare două vârfuri ale grafului există un drum.

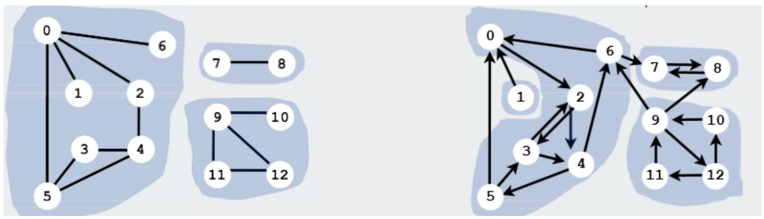
- graf tare conex - prin oricare două vârfuri trece cel puțin un circuit

Graf slab conex

între oricare două vârfuri u și v ale grafului exista un drum de la u la v sau de la v la u, nu există ambele drumuri.

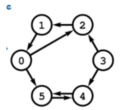
Exemplu

- componente conexe pe grafuri orientate / neorientate (DFS)

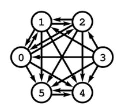


Exemplu DFS

Închiderea tranzitivă a unui graf



	0	1	2	3	4	5
0	1	0	1	0	0	1
1	1	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	1	1	1	0
4	0	0	0	0	1	1
5	0	0	0	0	1	1



	0	1	2	3	4	5
0	1	1	1	0	1	1
1	1	1	1	0	1	1
2	1	1	1	0	1	1
3	1	1	1	1	1	1
4	0	0	0	0	1	1
5	0	0	0	0	1	1

Algoritmul Kosaraju - Sharir

- algoritm pentru determinarea componentelor tare conex dintr-un graf orientat
- pași
  - DFS cu vârfurile puse pe o stiva
  - DFS pe complementul grafului

Exemplu - click

Cel mai scurt drum / lanț

- pentru un graf neponderat, orientat sau neorientat, putem folosi algoritmul lui Moore pentru a găsi cel mai scurt drum / lanț
- notații
  - u - nodul sursă
  - $l(v)$ - lungimea drumului
  - $p(v)$ - părintele vârfului v
  - Q - o coadă

Algoritmul lui Moore

$\text{MOORE}(G, u)$

$l(u) := 0$
for toate vârfurile $v \in V(G),\ v \neq u$ **do**
    $l(v) := \infty$
$Q = \varnothing$
$u \rightarrow Q$
while $Q \neq \emptyset$ **do**
    $Q \rightarrow x$
    for toți vecinii $y \in N(x)$ **do**
        if $l(y) = \infty$ **then**
            $p(y) := x$
            $l(y) := l(x) + 1$
            $y \rightarrow Q$
return l, p

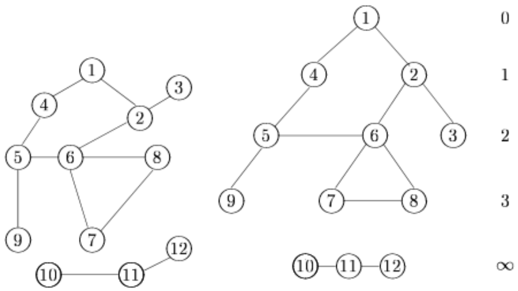
Algoritmul lui Moore (II)

- știind l, p, v cum putem afla drumul

$\text{MOORE\_DRUM}(l, p, v)$

$k := l(v)$
$u_k := v$
while $k \neq 0$ **do**
    $u_{k-1} := p(u_k)$
    $k := k - 1$
return u

Exemplu



	1	2	3	4	5	6	7	8	9	10	11	12
l	0	1	2	1	2	2	3	3	3	∞	∞	∞
p		1	2	1	4	2	6	6	5			