

# Algoritmica grafurilor

## III. Drumuri în grafuri

Mihai Suci

Facultatea de Matematică și Informatică (UBB)  
Departamentul de Informatică

Martie, 11, 2021

1 / 43

## Continut



- 1 Sortare topologica
- 2 Componente tare conexe
- 3 Drum de lungime minima
  - Sursa unica
    - Bellman-Ford
    - Grafuri orientate aciclice
    - Dijkstra
  - versiuni Floyd-Warshall

2 / 43

## DFS



DFS(G)

```
for fiecare vârf  $u \in G.V$  do
   $u.color = alb$ 
   $u.\pi = NIL$ 
   $time = 0$ 
  for fiecare  $u \in G.V$  do
    if  $u.color == alb$  then
      DFS_VISIT(G, u)
```

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)
 $u.color = negru$ 
 $time = time + 1$ 
 $u.f = time$ 
```

3 / 43

## DFS (II)



**Teorema (Teorema parantezelor)**

în orice căutare în adâncime a unui graf  $G = (V, E)$  (orientat sau neorientat), pentru orice două vârfuri  $u$  și  $v$ , exact una din următoarele trei afirmații este adevărată:

- intervalele  $[u.d, u.f]$  și  $[v.d, v.f]$  sunt total disjuncte
- intervalul  $[u.d, u.f]$  este conținut în întregime în intervalul  $[v.d, v.f]$  iar  $u$  este descendent al lui  $v$  în arborele de adâncime
- intervalul  $[v.d, v.f]$  este conținut în întregime în intervalul  $[u.d, u.f]$  iar  $v$  este descendent al lui  $u$  în arborele de adâncime

4 / 43

## DFS (III)



**Theorem (Teorema drumului alb)**

într-o pădure de adâncime a unui graf  $G = (V, E)$  (orientat sau neorientat), vârful  $v$  este descendent al vârfului  $u$  dacă și numai dacă la momentul  $u.d$ , când căutarea descoperă vârful  $u$ , vârful  $v$  este accesibil din  $u$  printr-un drum format în întregime din vârfuri albe.

5 / 43

## DFS - clasificarea muchiilor



- pentru un graf  $G = (V, E)$ , fie  $(u, v) \in E$ , în funcție de timp tipul arcelor pentru DFS:

tip arc	d	f
t (tree)	$u.d < v.d$	$u.f > v.f$
b (back)	$u.d > v.d$	$u.f < v.f$
f (forward)	$u.d < v.d$	$u.f > v.f$
c (cross)	$u.d > v.d$	$u.f > v.f$

- $u.d$  marchează timpul când a fost descoperit vârful  $u$
- $u.f$  marchează timpul când a fost explorat vârful  $u$

6 / 43



## Sortare topologica

- folosind algoritmul DFS se poate sorta topologic un graf orientat fără circuite (DAG - directed acyclic graph)
- realizează o aranjare liniară a vârfurilor unui graf în funcție de arcele grafului

### Sortare topologică

fie un graf orientat aciclic  $G = (V, E)$ , sortarea topologică reprezintă ordonarea vârfurilor astfel încât dacă  $G$  conține arcul  $(u, v)$  atunci  $u$  apare înaintea lui  $v$  în înșiruire.

- multe aplicații folosesc grafuri orientate fără circuite pentru a indica precedența între evenimente

9 / 43



## Sortare topologica (II)

- un set de acțiuni ce trebuie îndeplinite într-o anumită ordine
- unele sarcini trebuie executate înaintea ca alte acțiuni să înceapă
- *în ce ordine trebuie executate sarcinile?*
- problema poate fi rezolvată reprezentând sarcinile ca vârfuri într-un graf
- un arc  $(u, v)$  indică precedență între activități, activitatea  $u$  înaintea activității  $v$
- sortând topologic graful se arată ordinea efectuării acțiunilor

8 / 43



## Sortare topologica (III)

### sortare\_topologică(G)

- 1: apel DFS(G) pentru a determina timpii  $v.f, v \in V$
  - 2: sortare descrescătoare în funcție de timpul de finalizare (când fiecare vârf e terminat e inserat într-o listă înlănțuită)
  - 3: **return** lista înlănțuită de vârfuri
- un graf se poate sorta topologic în timpul  $\Theta(V + E)$ 
    - DFS durează  $\Theta(V + E)$
    - pentru a insera un vârf  $v \in V$  în listă e nevoie de  $O(1)$  timp

9 / 43



## Sortare topologică (IV)

### Lema 3.1

un graf orientat  $G$  este aciclic dacă și numai dacă DFS aplicat pe el nu găsește arce pentru care  $u.d > v.d$  și  $u.f < v.f$ .

### Teorema 3.1

procedura sortare\_topologică(G) produce o sortare topologică a unui graf orientat aciclic primit ca și parametru.

10 / 43



## Componente tare conex

### componente\_tare\_conexe(G)

- 1: apel DFS(G) pentru a determina timpii  $v.f, v \in V$
  - 2: determină  $G^T$
  - 3: apel DFS( $G^T$ ) dar în bucla principală a DFS nodurile sunt sortate descrescător după  $v.f$
  - 4: fiecare arbore din pădurea găsită de DFS în pasul 3 este o componentă tare conexă
- pentru  $G = (V, E)$ ,  $G^T = (V, E^T)$  unde  $E^T = \{(u, v) : (v, u) \in E\}$
  - pentru reprezentarea sub formă de listă de adiacență, pentru a determina  $G^T$  e nevoie de  $O(V + E)$  timp
  - fie  $G$  reprezentat ca listă de adiacență, pentru procedura componente\_tare\_conexe(G) complexitatea în timp este
    - $\Theta(V + E)$

11 / 43



## Componente tare conex (II)

### Lema 3.2

fie  $C$  și  $C'$  două componente tare conex din graful  $G = (V, E)$ .  $u, v \in C$ ,  $u', v' \in C'$  și  $G$  conține un drum  $u \rightsquigarrow u'$ . Atunci  $G$  nu poate avea un drum  $v' \rightsquigarrow v$ .

- graful format din componente tare conex este un graf orientat aciclic

12 / 43

## Componente tare conexe (III)



- fie  $U \subseteq V$ , putem defini  $d(U) = \min_{u \in U} \{u.d\}$  și  $f(U) = \max_{u \in U} \{u.f\}$ 
  - $d(U)$  reprezintă timpul pentru primul vârf descoperit de DFS din  $U$
  - $f(U)$  reprezintă timpul pentru ultimul vârf prelucrat de DFS din  $U$

## Lema 3.3

fie  $C$  și  $C'$  două componente tare conexe distincte din graful orientat  $G = (V, E)$ . Dacă există un arc  $(u, v) \in E$ , unde  $u \in C$  și  $v \in C'$  atunci  $f(C) > f(C')$ .

## Corolar 3.1

fie  $C$  și  $C'$  două componente tare conexe distincte din graful orientat  $G = (V, E)$ . Dacă există un arc  $(u, v) \in E^T$ , unde  $u \in C$  și  $v \in C'$  atunci  $f(C) < f(C')$ .

13 / 43

## Componente tare conexe (IV)



## Teorema 3.2

procedura **componente\_tare\_conexe(G)** găsește corect componentele tare conexe din graful orientat  $G$ .

14 / 43

## Probleme de drum de lungime minimă



- o problema de drum minim de la un nod sursă  $s$  este mapată pe un graf orientat  $G = (V, E)$  ponderat, funcția  $w : E \rightarrow \mathbb{R}$  mapează arcele la ponderi
- ponderea  $w(p)$  a drumului  $p = \{v_1, v_2, \dots, v_k\}$  este suma ponderilor arcelor ce compun drumul

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

- se poate defini drumul cu pondere minimă  $\delta(u, v)$  pentru un drum de la  $u$  la  $v$ :

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \rightsquigarrow v\} & \text{dacă există un drum de la } u \text{ la } v, \\ \infty & \text{în rest.} \end{cases}$$

15 / 43

## Probleme de drum de lungime minimă (



- un drum de lungime minimă de la vârful  $u$  la vârful  $v$  se definește ca un drum  $p$  cu ponderea  $w(p) = \delta(u, v)$
- un drum minim nu poate conține un circuit cu pondere negativă
- un drum minim nu poate conține un circuit
- reprezentarea unui drum de lungime minimă
  - pentru fiecare vârf  $v \in V$  se menține predecesorul lui în drum,  $v.\pi$

16 / 43

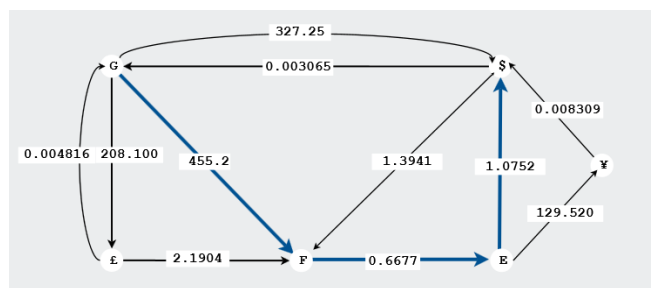
## Exemplu



Currency	£	Euro	¥	Franc	\$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3941	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
Gold (oz.)	0.004816	0.003295	0.0000255	0.002201	0.003065	1.0000

17 / 43

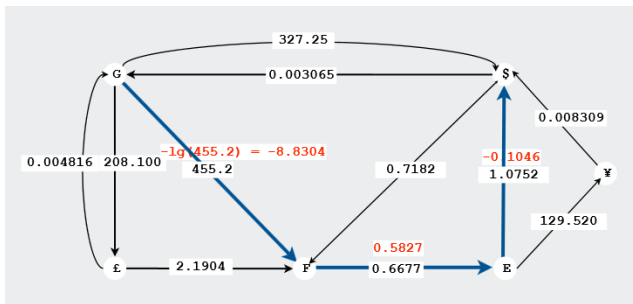
## Exemplu (II)



18 / 43



## Exemplu (III)



19 / 43



## Algoritmul Bellman-Ford

- algoritmul Bellman-Ford rezolvă problema drumului minim de la un nod sursă  $s$  pentru cazul general când avem și ponderi negative

**Bellman\_Ford**( $G, w, s$ )

```

1: INITIALIZARE_S( $G, s$ )
2: for  $i = 1$  la  $|V| - 1$  do
3:   for fiecare arc  $\{u, v\} \in E$  do
4:     RELAX( $u, v, w$ )
5: for fiecare arc  $\{u, v\} \in E$  do
6:   if  $v.d > u.d + w(u, v)$  then
7:     return FALSE
8: return TRUE

```

20 / 43



## Bellman-Ford (II)

**INITIALIZARE\_S**( $G, s$ )

```

1: for  $v \in V$  do
2:    $v.d = \infty$ 
3:    $v.\pi = NIL$ 
4:  $s.d = 0$ 

```

**RELAX**( $u, v, w$ )

```

1: if  $v.d > u.d + w(u, v)$  then
2:    $v.d = u.d + w(u, v)$ 
3:    $v.\pi = u$ 

```

- Exemplu - click

21 / 43



## Bellman-Ford (III)

- algoritmul rulează în  $O(VE)$  timp
- pasul de inițializare (linia 1) durează  $\Theta(V)$
- parcurea din liniile 2-4 durează  $O(VE)$
- bucla for din liniile 5-7 durează  $O(E)$

22 / 43



## Bellman-Ford (IV)

### Lema 3.4

fie  $G = (V, E)$  un graf ponderat orientat cu nodul sursă  $s$  și funcția de pondere  $w : E \rightarrow \mathbb{R}$ , presupunem că  $G$  nu conține circuite de pondere negativă accesibile din vârful  $s$ . După  $|V| - 1$  iterații ale buclei for din liniile 2-4 a procedurii  $Bellman\_Ford(G)$  avem  $v.d = \delta(s, v)$  pentru toate vârfurile  $v$  accesibile din  $s$ .

### Corolar 3.2

fie  $G = (V, E)$  un graf ponderat orientat cu nodul sursă  $s$  și funcția de pondere  $w : E \rightarrow \mathbb{R}$ . Pentru fiecare vârf  $v \in V$  există un drum de la  $s$  la  $v$  dacă și numai dacă procedura  $Bellman\_Ford(G)$  se termină cu  $v.d < \infty$ .

23 / 43



## Bellman-Ford (V)

### Teorema 3.2 (corectitudine Bellman-Ford)

fie procedura  $Bellman\_Ford(G)$  care este rulată pe un graf orientat și ponderat  $G = (V, E)$  din nodul sursă  $s$  și funcția de pondere  $w : E \rightarrow \mathbb{R}$ . Dacă  $G$  nu conține circuite de pondere negativă accesibile din  $s$  algoritmul va întoarce **TRUE**,  $v.d = \delta(s, v) \forall v \in V$  iar grafal predecesorilor  $G_\pi$  este un arbore minim cu rădăcina în  $s$ . Dacă  $G$  conține un circuit de pondere negativă accesibil din  $s$ , algoritmul întoarce **FALSE**.

24 / 43

## Bellman-Ford - corectitudine



Operația de relaxare este sigură

## Lema

algoritmul de relaxare menține  $v.d \geq \delta(s, v) \forall v \in V$ .

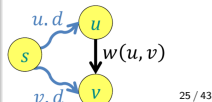
## Demonstrație.

- prin inducție
- considerăm  $relax(u, v)$
- prin inducție  $u.d \geq \delta(s, u)$
- din inegalitatea triunghiului

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

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

- $v.d = u.d + w(u, v)$  este "sigur"



25 / 43

## Bellman-Ford - corectitudine (II)

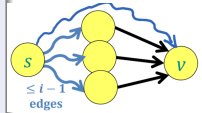


## Afirmație

după iterația  $i$  a algoritmului  $v.d$  are cel mult valoarea ponderilor drumurilor de la  $s$  la  $v$  de cel mult  $i$  arce  $\forall v \in V$ .

## Demonstrație.

- prin inducție pe  $i$
- înainte de iterația  $i$ ,  $v.d \leq \min\{w(p) : |p| \leq i-1\}$
- $relax()$  doar scade  $v.d$
- iterația  $i$  consideră toate drumurile de  $\leq i$  arce când se aplică  $relax()$  pe arcul lui  $v$ .



26 / 43

## Bellman-Ford - corectitudine (III)



## Teorema

dacă  $G = (V, E, w)$  nu are circuite negative, la sfârșitul algoritmului,  $v.d = \delta(s, v) \forall v \in V$ .

## Demonstrație.

- fără circuite negative, drumurile de cost minim sunt simple
- fiecare drum minim are  $\leq |V|$  arce, deci  $\leq |V| - 1$  vârfuri
- afirmație  $\Rightarrow |V| - 1$  iterații fac  $v.d \leq \delta(s, v)$

27 / 43

## Bellman-Ford - corectitudine (IV)



## Teorema

Bellman-Ford raportează corect circuitele negative accesibile din  $s$ .

## Demonstrație.

- dacă nu există circuite negative  $v.d = \delta(s, v)$  și din inegalitatea triunghiului  $\delta(s, v) \leq \delta(s, u) + w(u, v)$ , Bellman-Ford nu va raporta greșit existența unor circuite negative
- dacă există un circuit negativ atunci unul din arcele sale poate fi relaxat, Bellman-Ford va raporta acest lucru.

28 / 43

## Grafuri orientate aciclice



## drum\_minim\_dag(G)

- 1: sortare\_topologica(G)
  - 2: INITIALIZARE\_S(G, s)
  - 3: **for** fiecare vârf  $v$  sortat topologic **do**
  - 4:     **for**  $v \in G.Adj[u]$  **do**
  - 5:         RELAX( $u, v, w$ )
- timp de rulare
    - sortare topologică:  $\Theta(V + E)$
    - INITIALIZARE\_S:  $\Theta(V)$
    - bucla for (liniile 4-5) relaxează fiecare arc o singură dată
    - timpul total de rulare  $\Theta(V + E)$

29 / 43

## Grafuri orientate aciclice (II)



## Teorema 3.3 (corectitudine drum\_minim\_dag(G))

dacă un graf orientat, ponderat și aciclic  $G = (V, E)$  are ca și sursă vârf  $s$ , la terminarea procedurii  $drum\_minim\_dag(G)$   $v.d = \delta(s, v) \forall v \in V$  iar grafal predecesorilor  $G_\pi$  este un arbore minim.

30 / 43

## Dijkstra



- algoritmul lui Dijkstra rezolvă problema drumului minim pentru un graf orientat ponderat  $G = (V, E)$  în care  $w(u, v) \geq 0, \{u, v\} \in E$
- algoritmul menține un set  $S$  de vârfuri pentru care drumul minim de la sursa  $s$  a fost determinat
- în implementarea prezentată se folosește o coadă cu priorități pentru vârfuri, cheia fiind  $v.d$

31 / 43

## Algoritmul Dijkstra



## Dijkstra\_queue(G)

```

1: INITIALIZARE_S(G,s)
2:  $S = \emptyset$ 
3:  $Q = V$ 
4: while  $Q \neq \emptyset$  do
5:    $u = \text{EXTRACT\_MIN}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for  $v \in G.Adj[u]$  do
8:     RELAX( $u, v, w$ )

```

32 / 43

## Dijkstra (II)



## Teorema 3.4 (corectitudine Dijkstra)

fie procedura  $Dijkstra\_queue(G)$  rulată pe un graf orientat, ponderat  $G = (V, E)$  ce nu conține ponderi negative și  $s$  vârful sursă. La terminare  $u.d = \delta(s, u) \forall u \in V$ .

33 / 43

## Dijkstra - analiză



Cât de rapid este algoritmul  $Dijkstra\_queue$ ?

- menține o coadă cu priorități  $Q$  prin apelul operațiilor: INSERT (implicit în linia 3), EXTRACT\_MIN (linia 5) și DECREASE\_KEY (implicit în RELAX, linia 8)
- algoritmul apelează INSERT și EXTRACT\_MIN pentru fiecare vârf
- fiecare vârf este adăugat în setul  $S$  o singură dată, fiecare arc din  $Adj[u]$  este examinat o singură dată pe liniile 7-8
- numărul total de arce din lista de adiacență este  $|E|$ , bucla for iterează de  $|E|$  ori (liniile 7-8), algoritmul<sup>34/43</sup> apelează DECREASE\_KEY de cel mult  $|E|$  ori
- timpul total de rulare depinde de implementarea cozii cu priorități

## Dijkstra - analiză (II)



- dacă vârfurile sunt numerotate de la 1 la  $|V|$ 
  - $v.d$  e stocat pe poziția  $v$
  - fiecare operație INSERT și DECREASE\_KEY necesită  $O(1)$  timp iar operația EXTRACT\_MIN necesită  $O(V)$  timp
  - pentru un timp total  $O(V^2 + E) = O(V^2)$
- dacă graful este suficient de rar, coada se poate implementa ca și *binary min-heap*
  - fiecare operație EXTRACT\_MIN durează  $O(\lg V)$ , există  $|V|$  astfel de operații
  - timpul necesar construirii *binary min-heap* este  $O(V)$
  - fiecare operație DECREASE\_KEY necesită  $O(\lg V)$  timp, există  $|E|$  astfel de operații
  - timpul total de rulare este  $O((V + E) \lg V)$ , dacă toate vârfurile sunt accesibile din sursă timpul este  $O(E \lg V)$ <sup>35/43</sup>

## Dijkstra - analiză (III)



- se poate obține un timp de rulare  $O(V \lg V + E)$  dacă coada cu priorități este implementată cu un *heap Fibonacci*

36 / 43



## Floyd-Warshall

```

FLOYDWARSHALL( $D_0$ )
 $D := D_0$ 
for  $k := 1$  to  $n$  do
  for  $i := 1$  to  $n$  do
    for  $j := 1$  to  $n$  do
      if  $d_{ij} > d_{ik} + d_{kj}$  then
         $d_{ij} := d_{ik} + d_{kj}$ 
         $p_{ij} := p_{kj}$ 
return  $D, p$ 

```

37 / 43



## Floyd-Warshall pentru a determina nr de drumuri

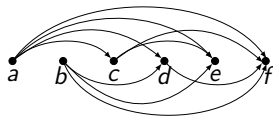
$FW(A, n)$

1.  $W \leftarrow A$
2. for  $k \leftarrow 1$  to  $n$
3. for  $i \leftarrow 1$  to  $n$
4. for  $j \leftarrow 1$  to  $n$
5. do  $w_{ij} \leftarrow w_{ij} + w_{ik} w_{kj}$
6. return  $W$

38 / 43



## Exemplu



39 / 43



## Exemplu (II)

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

Rezultat FW:

$$W = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

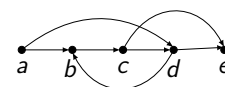
40 / 43



## Floyd-Warshall-Latin

$Floyd-Warshall-Latin(\mathcal{A}, n)$

1.  $\mathcal{W} \leftarrow \mathcal{A}$
2. for  $k \leftarrow 1$  to  $n$
3. for  $i \leftarrow 1$  to  $n$
4. for  $j \leftarrow 1$  to  $n$
5. if  $W_{ik} \neq \emptyset$  and  $W_{kj} \neq \emptyset$
6.  $W_{ij} \leftarrow W_{ij} \cup W_{ik} \cdot W_{kj}$
7. return  $\mathcal{W}$



41 / 43



## Exemplu

42 / 43



## Exemplu (II)

$$\begin{pmatrix} \emptyset & \{adb, ab\} & \{adbc, abc\} & \{abcd, ad\} & \{ade, adbce, abcde, abce\} \\ \emptyset & \emptyset & \{bc\} & \{bcd\} & \{bcde, bce\} \\ \emptyset & \{cdb\} & \emptyset & \{cd\} & \{cde, ce\} \\ \emptyset & \{db\} & \{dbc\} & \emptyset & \{dbce, de\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$