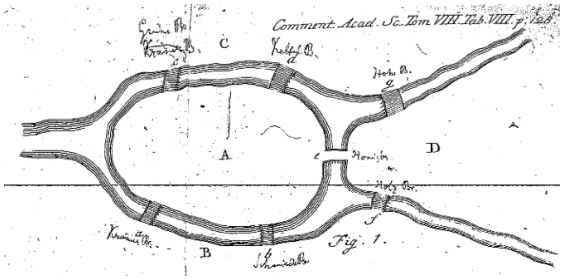
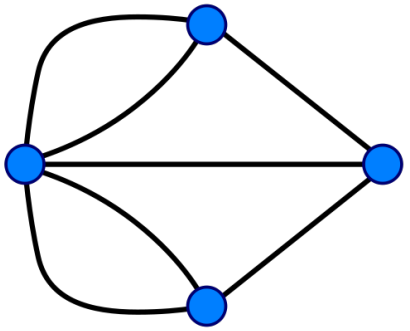




Algoritmica grafurilor
III. Drumuri în grafuri

Mihai Suciu

Facultatea de Matematică și Informatică (UBB)
Departamentul de Informatică

Martie, 14, 2019

Continut

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

DFS

DFS(G)
- **for** fiecare vârf $u \in G.V$ **do**
  - u.color $=$ alb
  - u.π $=$ 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
- $time = time + 1$
- $u.f = time$

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*

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.

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

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 contine arcul (u, v) atunci u apare înaintea lui v în înșiruire.

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

Sortare topologica (II)

- un set de acțiuni ce trebuie îndeplinite într-o anumită ordine
- unele sarcini trebuie executate înaninte 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

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

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

Componente tare conexe

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 principala 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 determină 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
  - $\Theta(V + E)$

Componente tare conexe (II)

Lema 3.2

fie C și C' două componente tare conexe 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 conexe este un graf orientat aciclic

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 $f(C) < f(C')$.

Componente tare conexe (IV)

Teorema 3.2

procedura **componente_tare_conexe(G)** găsește corect componentele tare conexe din graful orientat G.

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 $w : E \to \mathbb{R}$ mapează arcele la ponderi
- ponderea $w(p)$ a drumului $p = \{v_1, v_2, ..., v_k\}$ 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:

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

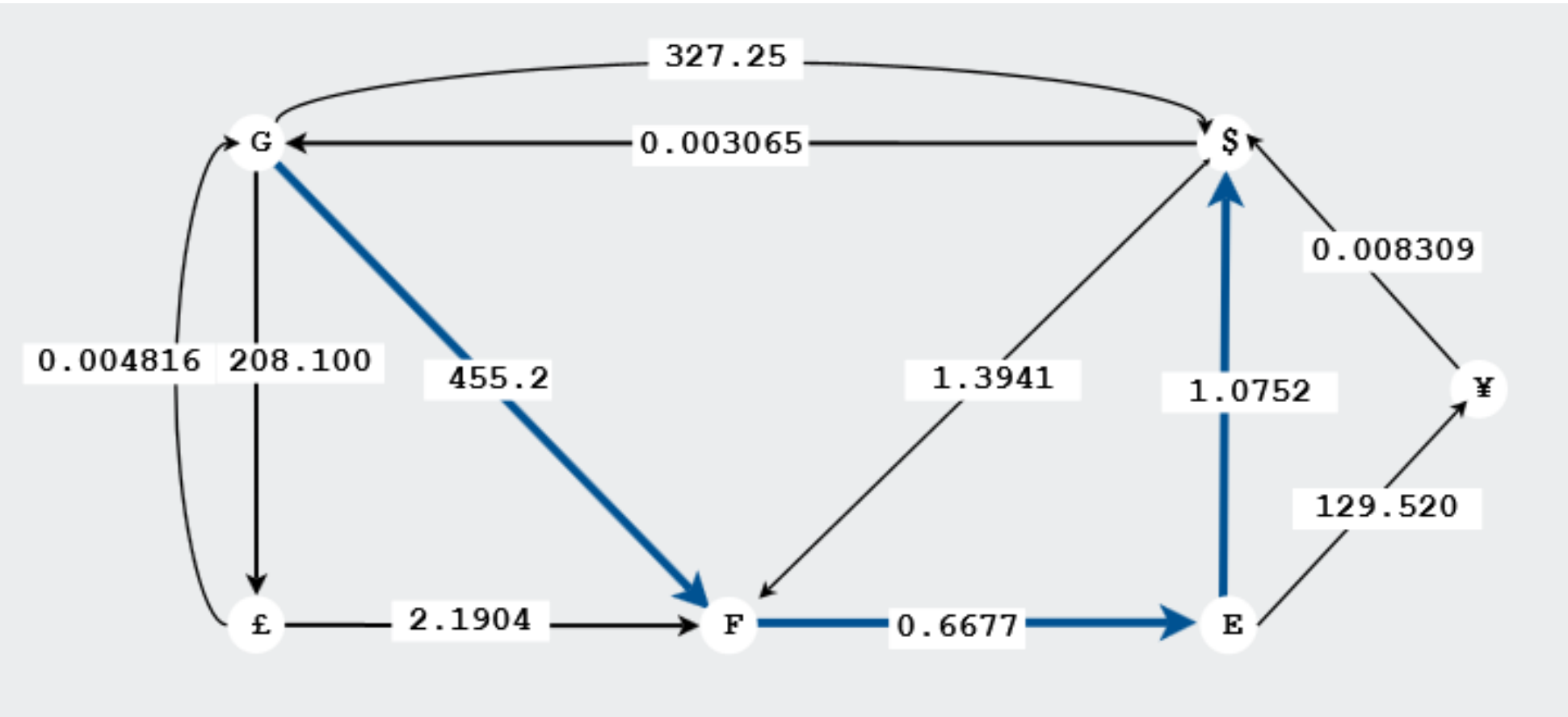
Probleme de drum de lungime minimă (II)

- 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, $v.\pi$

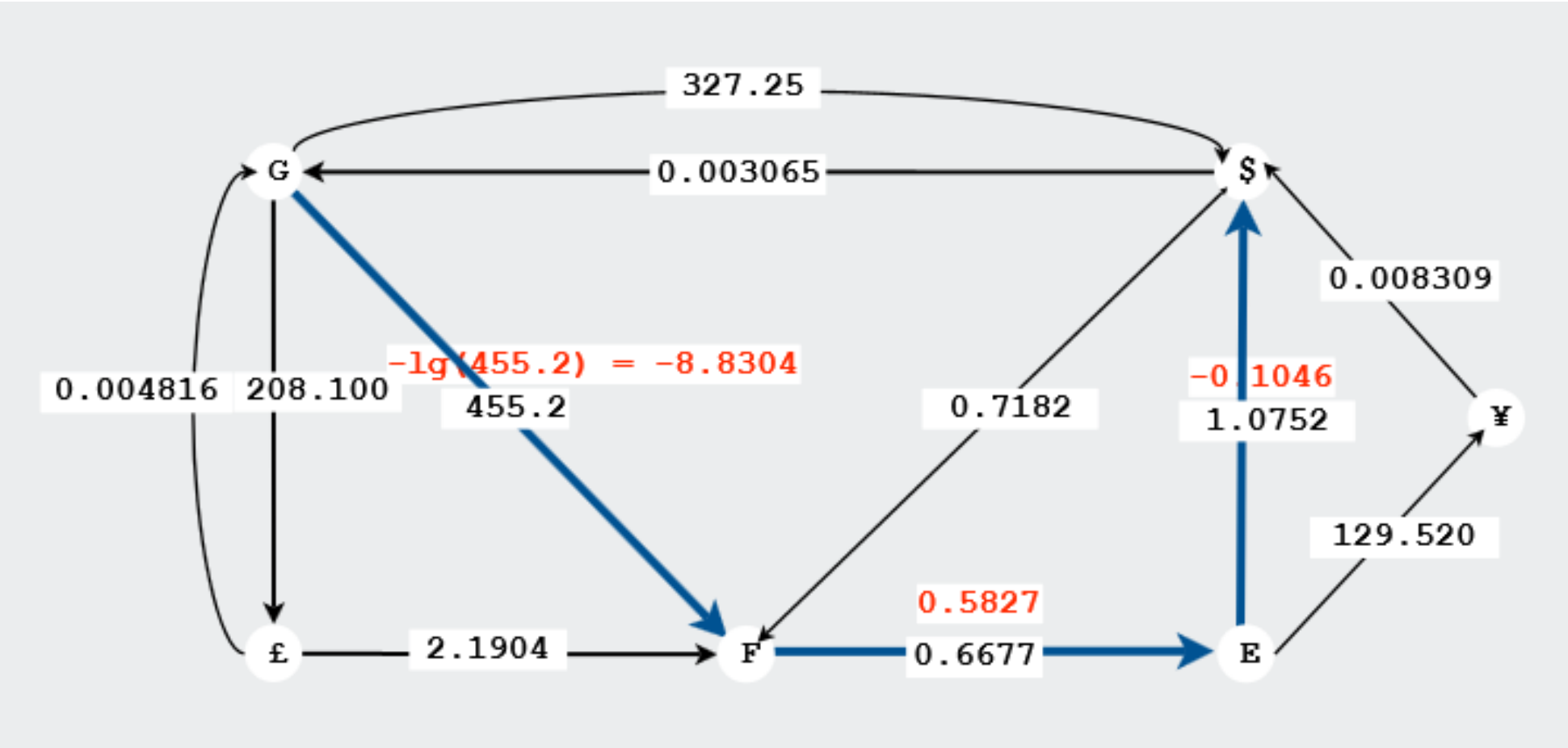
Exemplu

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

Exemplu (II)


327.25
G
0.003065
$
0.008309
0.004816
208.100
455.2
1.3941
1.0752
¥
129.520
£
2.1904
F
0.6677
E

Exemplu (III)


327.25
G
0.003065
$
0.008309
-lg(455.2) = -8.8304
0.004816
208.100
455.2
0.7182
-0.1046
1.0752
¥
129.520
0.5827
£
2.1904
F
0.6677
E

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)

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

Bellman-Ford (II)

INITIALIZARE_S(G,s)

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

RELAX(u, v, w)

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

- Exemplu - click

Bellman-Ford (III)

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

Bellman-Ford (IV)

Lema 3.4

fie $G = (V, E)$ un graf ponderat orientat cu nodul sursă s și funcția de pondere $w : E \to \mathbb{R}$, 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 $w : E \to \mathbb{R}$. 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 $v.d < \infty$.

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 $w : E \to \mathbb{R}$. 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 graful predecesorilor G_π este un arbore minim cu rădăcina în s. Dacă G conține un circuit de pondere negativă accesibil din s, algoritmul întoarce *FALSE*.

Bellman-Ford - corectitudine

Operația de relaxare este sigură

Lema

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

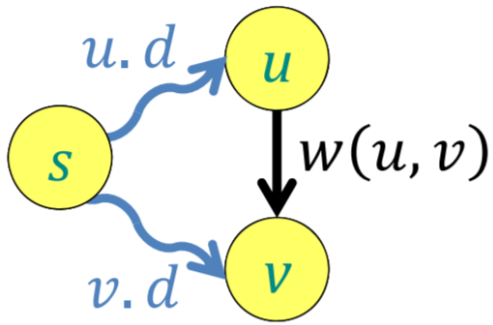
Demonstrație.

- 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"



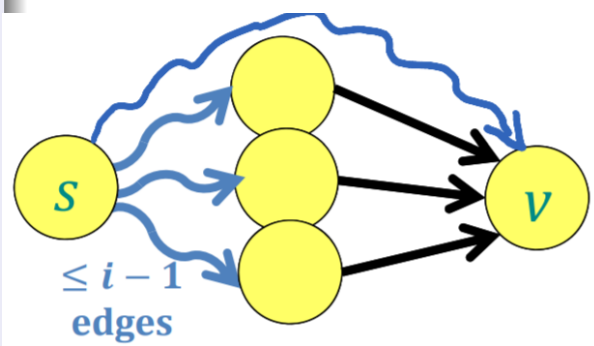
Bellman-Ford - corectitudine (II)

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.



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

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.

Grafuri orientate aciclice

drum_minim_dag(G)

1: sortate_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
  - sortate topologică: $\Theta(V + E)$
  - INITIALIZARE_S: $\Theta(V)$
  - bucla for (liniile 4-5) relaxează fiecae arc o singură dată
  - timpul total de rulare $\Theta(V + E)$

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 vârful s, la terminarea procedurii *drum_minim_dag(G)* $v.d = \delta(s, v) \forall v \in V$ iar graful predecesorilor G_π este un arbore minim.

Dijkstra

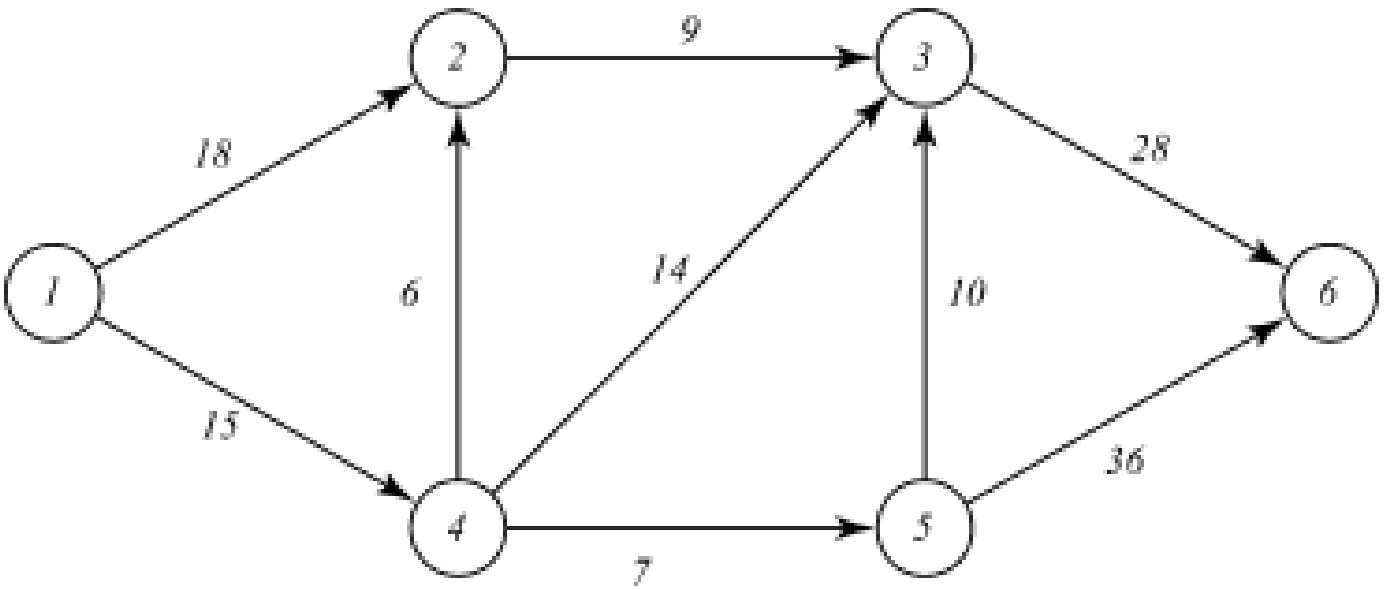
- algoritmul lui Dijkstra rezolvă problema drumului minim pentru un graf orientat ponderat $G = (V, E)$ în care $w(u, v) \geq 0, \{u, v\} \in E$
- algoritmul menține un set S de vârfuri pentru care drumul minim de la sursa s a fost determinat
- în implementarea prezentata se folosește o coadă cu priorități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

Algoritmul Dijkstra

Dijkstra_queue(G)

```
1: INITIALIZARE_S(G,s)
2: S = ∅
3: Q = V
4: while Q ≠ ∅ do
5:     u = EXTRACT_MIN(Q)
6:     S = S ∪ {u}
7:     for v ∈ G.Adj[u] do
8:         RELAX(u,v,w)
```

Exemplu



Exemplu (II)

Vertex (v)	**1**	2	3	4	5	6
Label (v)	**0**	∞	∞	∞	∞	∞
Status (v)	***P***	*T*	*T*	*T*	*T*	*T*
Predecessor (v)	–	–	–	–	–	–

Vertex (v)	1	**2**	3	**4**	5	6
Label (v)	0	18	∞	15	∞	∞
Status (v)	*P*	*T*	*T*	*T*	*T*	*T*
Predecessor (v)	–	1	–	1	–	–

Exemplu (III)

3

Vertex (v)	1	2	3	**4**	5	6
Label (v)	0	18	∞	**15**	∞	∞
Status (v)	*P*	*T*	*T*	***P***	*T*	*T*
Predecessor (v)	–	1	–	**1**	–	–

4

Vertex (v)	1	**2**	**3**	4	**5**	6
Label (v)	0	18	29	15	22	∞
Status (v)	*P*	*T*	*T*	***P***	*T*	*T*
Predecessor (v)	–	1	4	1	4	–

5

Vertex (v)	1	**2**	3	4	5	6
Label (v)	0	**18**	29	15	22	∞
Status (v)	*P*	***P***	*T*	*P*	*T*	*T*
Predecessor (v)	–	**1**	4	1	4	–

Exemplu (IV)

6

Vertex (v)	1	2	**3**	4	5	6
Label (v)	0	18	27	15	22	∞
Status (v)	*P*	*P*	*T*	*P*	*T*	*T*
Predecessor (v)	–	1	2	1	4	–

7

Vertex (v)	1	2	3	4	**5**	6
Label (v)	0	18	27	15	**22**	∞
Status (v)	*P*	*P*	*T*	*P*	***P***	*T*
Predecessor (v)	–	1	2	1	**4**	–

8

Vertex (v)	1	2	**3**	4	5	**6**
Label (v)	0	18	27	15	22	58
Status (v)	*P*	*P*	*T*	*P*	*P*	*T*
Predecessor (v)	–	1	2	1	4	5

Exemplu (V)

9

Vertex (v)	1	2	**3**	4	5	6
Label (v)	0	18	**27**	15	22	58
Status (v)	*P*	*P*	***P***	*P*	*P*	*T*
Predecessor (v)	–	1	**2**	1	4	5

10

Vertex (v)	1	2	3	4	5	**6**
Label (v)	0	18	27	15	22	55
Status (v)	*P*	*P*	*P*	*P*	*P*	*T*
Predecessor (v)	–	1	2	1	4	3

11

Vertex (v)	1	2	3	4	5	**6**
Label (v)	0	18	27	15	22	**55**
Status (v)	*P*	*P*	*P*	*P*	*P*	***P***
Predecessor (v)	–	1	2	1	4	**3**

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 $u.d = \delta(s, u) \forall u \in V$.

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 data, 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 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 $O(E \lg V)$

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*

Floyd-Warhsall

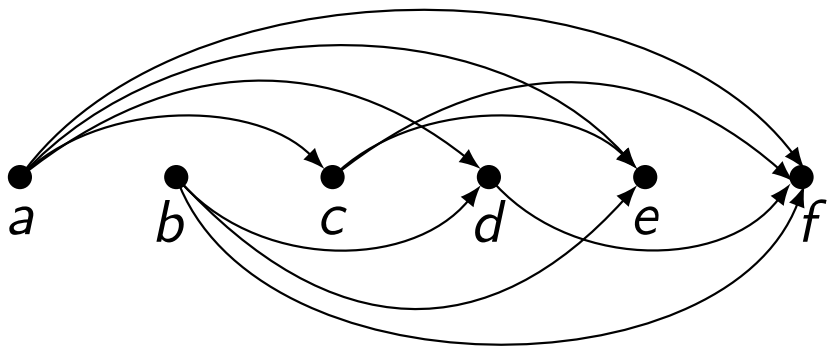
```
FloydWarshall(D_0)
    D := D_0
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                if d_ij > d_ik + d_kj then
                    d_ij := d_ik + d_kj
                    p_ij := p_kj
    return D, p
```

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** $j \leftarrow 1$ n
5. **do** $w_{ij} \leftarrow w_{ij} + w_{ik} w_{kj}$
6. **return** W

Exemplu



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:

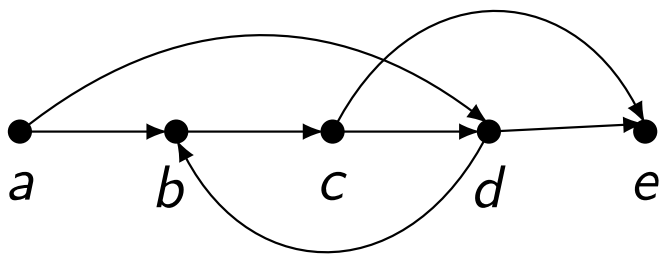
$$W = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Floyd-Warshal-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** $\mathcal{W}$

Exemplu



Exemplu (II)

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