Legrövidebb utak nem irányított gráfokban

Moore algoritmusa

Ez az algoritmus nem súlyozott gráfokban meghatározza a legrövidebb utakat egy adott csúcsból az összes többi csúcsba. Az algoritmus szélességi bejáráson alapul.

Legyen $G = (V, E)$ Egy adott gráf. A következő jelöléseket használjuk:
– u a kezdőcsúcs,
– $l(v)$ a v-nek az u-tól való távolsága,
– $p(v)$ a legrövidebb 7 a v-t megelőző csúcs,
– Q egy sor (elemet hozzáadni egyik végén lehet, elemet kivenni a másikon).

Az algoritmusban használjuk még a következő műveleteket:
$v \to Q$ jelentése: beírja a v elemet a Q sorba,
$Q \to v$ jelentése: kiveszi a sor egy elemét (és ki is törli onnan), amelyet v-vel jelöl.

a) Algoritmus legrövidebb távolságok keresésére az u csúcsból kiindulva

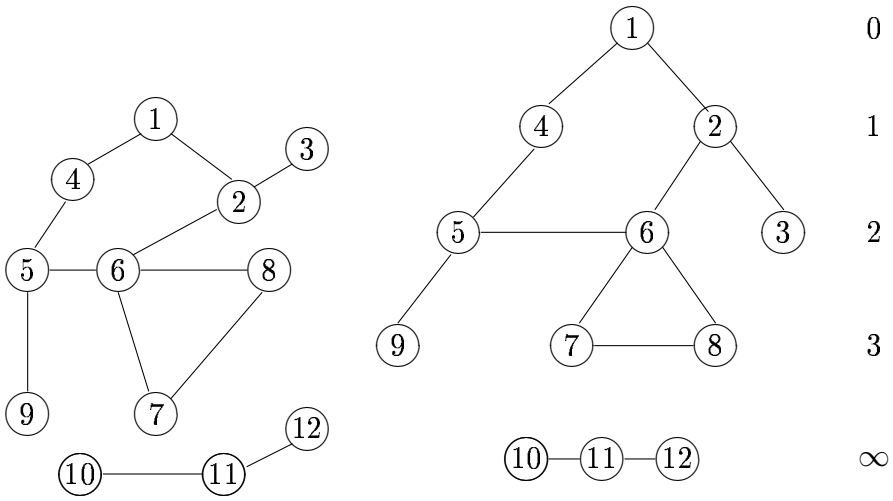
MOORETÁVOLSÁG(G, u)
1. $l(u) := 0$
2. **for** minden $v \in V(G),\ v \neq u$ csúcsra **do**
3. $\quad l(v) := \infty$
4. legyen Q egy üres sor
5. $u \to Q$
6. **while** $Q \neq \emptyset$ **do**
7. $\quad Q \to x$
8. $\quad$ **for** minden $y \in N(x)$ **do**
9. $\quad\quad$ **if** $l(y) = \infty$ **then**
10. $\quad\quad\quad p(y) := x$
11. $\quad\quad\quad l(y) := l(x) + 1$
12. $\quad\quad\quad y \to Q$
13. **return** l, p

b) Algoritmus a legrövidebb u–v utak keresésére

MOOREÚT(l, p, v)
1. $k := l(v)$
2. $u_k := v$
3. **while** $k \neq 0$ **do**
4. $\quad u_{k-1} := p(u_k)$
5. $\quad k := k - 1$
6. **return** u

Az eredményt az $u_0, u_1, \ldots, u_k$ vektor tartalmazza.

Irányított gráfok esetében: $N(x)$ helyett $N^{\text{ki}}(x)$



	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			

Példa.
Ha $u = 1$ és $v = 8$, akkor $k := 3$, és az algoritmus lépései:
$u_3 := 8$, $u_2 := 6$, $u_1 := 2$, $u_0 := 1$.
Tehát a legrövidebb út 1 és 8 között: 1, 2, 6, 8.

Legrövidebb utak súlyozott gráfokban

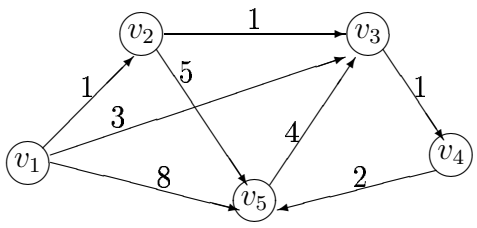
Legrövidebb utak egy csúcsból minden csúcsba

Dijkstra algoritmusa

DIJKSTRA(G, u)
1. $S := \{u\},\ T := V \setminus S,\ l(u) := 0$
2. **for** minden $v \in V,\ v \neq u$ **do**
3. $\quad l(v) := \infty$
4. $x := u$
5. **while** $T \neq \emptyset$ **do**
6. $\quad$ **for** minden $v \in N(x) \cap T$ **do**
7. $\quad\quad$ **if** $l(v) > l(x) + \mathcal{W}(x, v)$ **then**
8. $\quad\quad\quad l(v) := l(x) + \mathcal{W}(x, v)$
9. $\quad\quad\quad p(v) := x$
10. $\quad$ Legyen $x \in T$: $l(x) = \min\limits_{y \in T} l(y)$
11. $\quad S := S \cup \{x\},\ T := T \setminus \{x\}$
12. **return** l, p

Irányított gráfok esetében a 6. sorban $N(x)$ helyett $N^{\text{ki}}(x)$-et írunk.

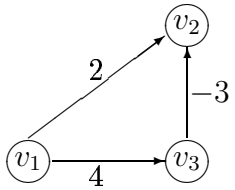
A legrövidebb utat ugyanaz az algoritmus adja meg, mint a Moore-algoritmus esetében.



A u kezdőcsúcs különböző értékeire, a következőket kapjuk.

	v_i	v_1	v_2	v_3	v_4	v_5
$u = v_1$	l_i	0	1	2	3	5
	p_i	$-$	v_1	v_2	v_3	v_4
$u = v_2$	l_i	∞	0	1	2	4
	p_i	$-$	$-$	v_2	v_3	v_4
$u = v_3$	l_i	∞	∞	0	1	3
	p_i	$-$	$-$	$-$	v_3	v_4
$u = v_4$	l_i	∞	∞	6	0	2
	p_i	$-$	$-$	v_5	$-$	v_4
$u = v_5$	l_i	∞	∞	4	5	0
	p_i	$-$	$-$	v_5	v_3	$-$

A Dijsktra-algoritmus nem működik negatív súlyok esetén. Erre példa lehet a következő gráf:



Ford algoritmusa

Legyenek $V = \{v_1, v_2, \ldots, v_n\}$ a súlyozott gráf csúcsai.

FORD(G)
1. $i := 1$
2. $l(v_1) := 0$
3. $l(v_i) := \infty$, minden $i = 2, 3, \ldots, n$.
4. **while** $i \leq n$ **do**
5. minden $v \in N(v_i)$:
6. $l(v) := \min\big(l(v), l(v_i) + \mathcal{W}(v_i, v)\big)$
7. **if** $l(v)$ módosult és ($v = v_j$, ahol $j < i$)
8. **then** $i := j - 1$
9. exit minden
10. $i := i + 1$
11. **return** l

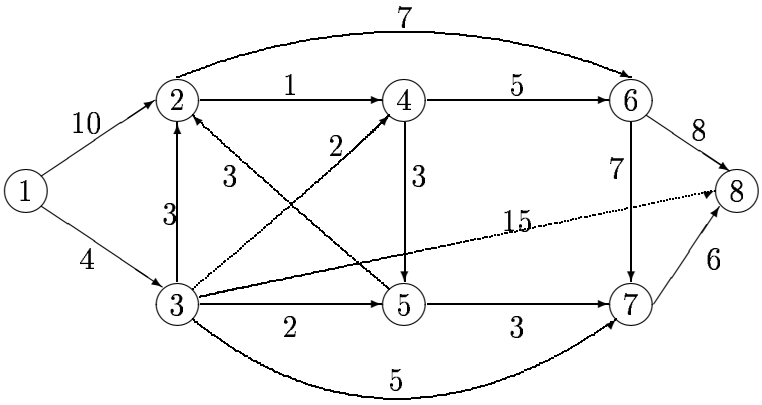
Részletesebben:

A p vektor egy eleme itt is mindig az előző csúcsra mutat. Ha x és y között nincs él, akkor $\mathcal{W}(x, y) = \infty$ értéket veszünk.

FORD(G)
1. $l(v_1) := 0$
2. $l(v_i) := \infty$, minden $i = 2, 3, \ldots, n$.
3. $i := 1$
4. **while** $i \leq n$ **do**
5. $j := 1$
6. **while** $j \leq n$ **do**
7. **if** $l(v_j) - l(v_i) > \mathcal{W}(v_i, v_j)$ **then**
8. $l(v_j) := l(v_i) + \mathcal{W}(v_i, v_j)$
9. $p(v_j) := v_i$
10. **if** $j < i$ **then**
11. $i := j - 1$
12. $j := n$
13. $j := j + 1$
14. $i := i + 1$
15. **return** l, p

Irányított gráfokra $N(v_i)$ helyett $N^{\text{ki}}(v_i)$ szerepel.

Példa. Adott a következő súlyozott gráf:



amelynek szomszédsági mátrixa:

$$\begin{pmatrix} 0 & 10 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 7 & 0 & 0 \\ 0 & 3 & 0 & 2 & 2 & 0 & 5 & 15 \\ 0 & 0 & 0 & 0 & 3 & 5 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$u = 1$-re a Ford-algoritmus a következőt adja:

i:	1	2	3	4	5	6	7	8
l_i:	0	7	4	6	6	11	9	15
p_i:	0	3	1	3	3	4	3	7

Ez az algoritmus akkor is 0, ha bizonyos élek negatív súlyúak, amennyiben a gráfban nincs negatív hosszúságú kör.

Legrövidebb utak minden csúcsból egy csúcsba

A Bellman–Kalaba-algoritmus

A módszer a szomszédsági mátrixot használja. Kiválasztjuk azt a csúcsot, amelybe a legrövidebb utakat keressük. Ennek a csúcsnak megfelel a mátrix egy oszlopa. Jelöljük ezt a oszlopot a következőképpen: $V^{(1)} =$

$\left(V_i^{(1)}\right)_{i=\overline{1,n}}$. A szomszédsági mátrix: $A = \left(a_{ij}\right)_{i,j=\overline{1,n}}$, ahol $a_{ij} = d_{ij}^{(0)}$, azaz:

$$a_{ij} = \begin{cases} \mathcal{W}(v_i, v_j) & \text{ha } \{v_i, v_j\} \in E(G) \ \ (\text{vagy } (v_i, v_j) \in E(\vec{G})) \\ 0 & \text{ha } i = j \\ \infty & \text{ha } \{v_i, v_j\} \notin E(G) \ \ (\text{vagy } (v_i, v_j) \notin E(\vec{G})) \end{cases}$$

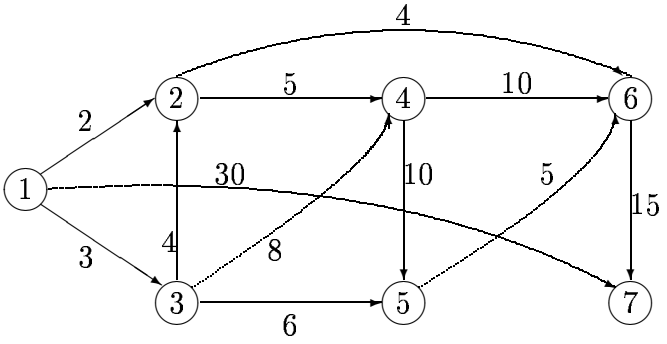
Kiszámítjuk a következő vektorokat ($k = 1, 2, \ldots$):

$$V_i^{(k)} = \min_{j=\overline{1,n}} \left(a_{ij} + V_j^{(k-1)}\right) \text{ for } i = 1, 2, \ldots, n$$

ameddig $V^{(l)} = V^{(l-1)}$ egy bizonyos l értékre.

Példa.

Keressük meg a következő gráfban a 7-es csúcsba befutó legrövidebb utakat minden csúcsból. Tehát a $V^{(1)}$ vektor a mátrix 7-es oszlopa.



	1	2	3	4	5	6	7	$V^{(1)}$	$V^{(2)}$	$V^{(3)}$	$V^{(4)}$
1	0	2	3	∞	∞	∞	30	30	30	21	21
2	∞	0	∞	5	∞	4	∞	∞	19	19	19
3	∞	4	0	8	6	∞	∞	∞	∞	23	23
4	∞	∞	∞	0	10	10	∞	∞	25	25	25
5	∞	∞	∞	∞	0	5	∞	∞	20	20	20
6	∞	∞	∞	∞	∞	0	15	15	15	15	15
7	∞	∞	∞	∞	∞	∞	0	0	0	0	0

Egy u–v utat, az előző algoritmusokhoz hasonlóan, egy p vektor segítségével határozhatunk meg. Kezdetben $p(i) := i$ minden $i = 1, 2, \ldots, n$ értékre. Ha a $\min_{j=\overline{1,n}} \left(a_{ij} + V_j^{(k-1)}\right)$ minimumot a $j = \ell$ értékre kapjuk meg, akkor, ha $i <> \ell$ a $p_i := \ell$ lesz.

BellmanKalaba$(A, V^{(1)})$
1. **for** $i = 1, 2, \ldots, n$ **do**
2. **if** $V_i^{(1)} = 1$ **then** $p_i :=$ végpont
3. **else** $p_i := -1$
4. $i := 1$
5. **repeat**
6. $i := i + 1$
7. **for** $k := 1, 2, \ldots n$ **do**
8. $min := a_{k1} + V_1^{(i-1)}$
9. **for** $j := 2, 3, \ldots n$ **do**
10. **if** $min > a_{kj} + V_j^{(i-1)}$ **then**
11. $min := a_{kj} + V_j^{(i-1)}$
12. **if** $k \neq j$ **then** $p_k := j$
13. $V_k^{(i)} := min$
14. **until** $V^{(i)} = V^{(i-1)}$
15. **return** $V^{(i)}, p$

Egy v_i–v_j utat (a fenti algoritmusban az első vektor a j oszlop) a következő algoritmussal határozzuk meg:

$k := 1$
$u_k := i$
while $p_{u_k} \neq j$ **do** $u_{k+1} := p_{u_k}$
$k := k + 1$

Előbbi példánkban $p = (2, 6, 2, 6, 6, 7, 7)$. Az 1 és 7 csúcsok között a legrövidebb út a következő csúcsokból áll:
$u_1 := 1,\ u_2 := p_1 = 2,\ u_3 := p_2 = 6,\ u_4 := p_6 = 7$. Vagyis a megfelelő út: 1,2,6,7.

Legrövidebb utak minden csúcsból minden csúcsba

A távolsági mátrix meghatározására egy Warshall-típusú algoritmust használunk. Hogy meghatározhassuk nemcsak a távolságokat, hanem az utakat is, egy P mátrixot használunk az előző csúcsok megőrzésére.

A Floyd–Warshall-algoritmus

Kezdetben $p_{ij} := i$ ha $d_{ij} \neq \infty$ és $i \neq j$; más esetekben $p_{ij} := 0$.

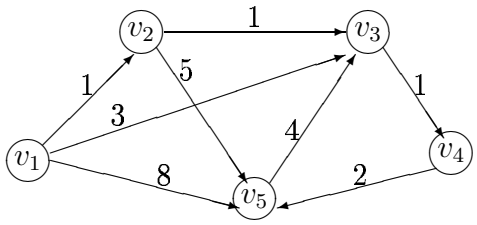
FLOYDWARSHALL(D_0)
1. $D := D_0$
2. **for** $k := 1$ **to** n **do**
3. **for** $i := 1$ **to** n **do**
4. **for** $j := 1$ **to** n **do**
5. **if** $d_{ij} > d_{ik} + d_{kj}$ **then**
6. $d_{ij} := d_{ik} + d_{kj}$
7. $p_{ij} := p_{kj}$
8. **return** D, p

Egy u_x–u_y utat a következő algoritmussal határozzuk meg:

1. $k := n$:
2. $u_k := y$
3. **while** $u_k \neq x$ **do**
4. $u_{k-1} := p_{xu_k}$
5. $k := k - 1$

A keresett út: $u_k, u_{k+1}, \ldots, u_n$.

Példa.



Az előbbi gráf szomszédsági mátrixa és a megfelelő P mátrix kezdeti értéke:

$$D_0 = \begin{pmatrix} 0 & 1 & 3 & \infty & 8 \\ \infty & 0 & 1 & \infty & 5 \\ \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & 4 & \infty & 0 \end{pmatrix} \qquad P_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$

Az algoritmus eredménye a D és P mátrixok:

$$D = \begin{pmatrix} 0 & 1 & 2 & 3 & 5 \\ \infty & 0 & 1 & 2 & 4 \\ \infty & \infty & 0 & 1 & 3 \\ \infty & \infty & 6 & 0 & 2 \\ \infty & \infty & 4 & 5 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 2 & 3 & 4 \\ 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 5 & 0 & 4 \\ 0 & 0 & 5 & 3 & 0 \end{pmatrix}$$

Az algoritmusok bonyolultsága

Felmerülhet a kérdés, hogy melyik jobb a bemutatott algoritmusok közül? Hogyan határozhatjuk meg a bonyolultságukat?

Logikus lenne, hogy az algoritmus bonyolultságán a feladat megoldásához szükséges időt értsük. Mivel azonban az algoritmus különféle nyelveken és gépeken valósítható meg, a megfelelő program percekben, órákban mért futási ideje nem lehet alapja az összehasonlításnak. Ehelyett az algoritmus jól megválasztott műveleteinek vagy lépéseinek a számát vizsgáljuk. Gyakorlatilag azonban kevés olyan eset van, amikor pontosan meg lehet határozni egy algoritmus lépésszámát adott nagyságú bemenetre. Ezért a lépésszámot vagy az algoritmus szempontjából fontos műveletek számát a *legrosszabb esetben* vizsgáljuk minden azonos nagyságú bemenet esetében. Gráfok esetében azonos csúcsszámú vagy azonos élszámú gráfokat vizsgálunk. A következőkben bonyolultságon mindig ilyen értelemben vett bonyolultságot értünk. A bonyolultság a bemeneti adatok nagyságán értelmezett függvény.

Az $O\big(f(n)\big)$ jelölés

Legyenek adottak az $f, g : \mathbf{N} \to \mathbf{R}^+$ függvények. Ha létezik egy C pozitív valós szám és egy n_0 természetes szám úgy, hogy

$$f(n) \leq Cg(n) \quad \text{ha} \quad n \geq n_0,$$

akkor az f függvény *rendje* kisebb vagy egyenlő, mint a g rendje, és ezt így írjuk:

$$f(n) = O\big(g(n)\big).$$

Ha $f(n) = O\big(g(n)\big)$ és $g(n) = O\big(f(n)\big)$, akkor az f és g függvények azonos rendűek, és ennek jelölése: $f(n) = \Theta\big(g(n)\big)$.

Példák. A közismert mátrixszorzás bonyolultsága, ha mindkettő $n \times n$ típusú, $O(n^3)$. Itt alapműveletnek két elem szorzatát vesszük.

n-elemű lista esetében a szekvenciális keresés bonyolultsága $O(n)$,

a buborékos rendezés esetében pedig $O(n^2)$ (mindkét esetben alapműveletként két elem összehasonlítást vesszük).

A legrövidebb utak algoritmusainak bonyolultsága

A Floyd–Warshall-algoritmus kivételével a bemutatott algoritmusok vagy egy csúcsból minden csúcsba vagy minden csúcsból egybe keresik a legrövidebb utakat. Ezért, hogy egységesen kezelhessük őket, beleértve a Floyd–Warshall-algoritmust is, átszámítjuk a bonyolultságot a „minden csúcsból minden csúcsba esetre", ami azt jelenti, hogy szorzunk n-nel.

A Moore- és Dijkstra-algoritmusoknál egy-egy **while** és **for** ciklus van egymásba ágyazva. Mivel a gráf n-csúcsú, az elsőt legfennebb n-szer, a másodikat pedig legfennebb $(n-1)$-szer végezzük el. Ezért a bonyolultság $O(n^2)$, mivel $n(n-1)$ másodfokú.

algoritmus	bonyolultság	bonyolultság „minden csúcsból minden csúcsba"
Moore	$O(n^2)$	$O(n^3)$
Dijkstra	$O(n^2)$	$O(n^3)$
Ford	$O(n^3)$	$O(n^4)$
Bellman–Kalaba	$O(n^3)$	$O(n^4)$
Floyd–Warshall	$O(n^3)$	$O(n^3)$

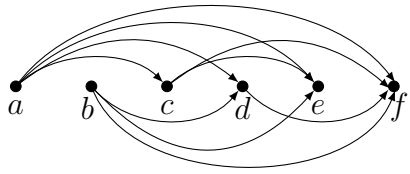
Utak száma gráfokban

$G = (V, E)$, ahol
$V = \{a_1, a_2, \ldots, a_n\}$,
E élek 0.

Floyd–Warshall-szerű algoritmus irányított kör nélküli gráfban megadja az irányított utak számát.

Szomszédsági mátrix: $A = \left(a_{ij}\right)_{\substack{i=\overline{1,n} \\ j=\overline{1,n}}}$

```
FW(A, n)
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4              do for j ← 1 to n
5                      do w_ij ← w_ij + w_ik w_kj
6  return W
```



1. ábra.

Tekintsük az 1. ábra gráfját. Ennek szomszédsági mátrixa:

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

Az FW-algoritmus alkalmazása után:

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

A fenti algoritmust a latin-négyzetes módszerrel kombinálva, meghatározhatjuk az utakat is. Sőt, ez az algoritmus képes arra is, hogy olyan gráfokban is meghatározza az utakat, amelyekben van irányított kör. Ezért csupán annyit kell tennünk, hogy csak olyan szavakat konkatenáljunk, amelyek nem tartalmaznak közös betűket.

A szomszédsági mátrix mintájára használjuk az $\mathcal{A}$ mátrixot, amelynek A_{ij} elemei a gráf csúcsaiból (mint 0) képzett szavak. Kezdetben ezek a 2 az éleket jelölik, később az utakat. Kezdetben

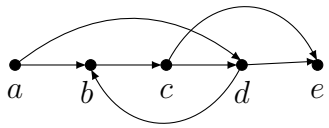
$$A_{ij} = \begin{cases} \{a_i a_j\}, & \text{ha } (a_i a_j) \text{ irányított él,} \\ \emptyset, & \text{különben,} \end{cases} \qquad i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n.$$

Ha $\mathcal{A}$ és $\mathcal{B}$ szóhalmazok, legyen $\mathcal{A} \cdot \mathcal{B}$ az a halmaz, amelyet úgy képezünk, hogy $\mathcal{A}$ elemeit megszorozzuk (konkatenáljuk) $\mathcal{B}$ elemeivel, de csak akkor, ha a két szorzandó szó nem tartalmaz közös betűket.

$$\mathcal{A} \cdot \mathcal{B} = \big\{ ab \,\big|\, a \in \mathcal{A}, b \in \mathcal{B}, a \text{ és } b \text{ különböző betűkből áll} \big\}.$$

Az $s = s_1 s_2 \cdots s_p$ szóból képezzük az $'s$ szót úgy, hogy elhagyjuk s első betűjét: $'s = s_2 s_3 \cdots s_p$. Képezzük az $'A_{ij}$ halmazt az A_{ij} halmazból, annak minden eleméből elhagyván az első betűt. Ekkor az $'\mathcal{A}$ mátrix elemei $'A_{ij}$.

FLOYD–WARSHALL–LATIN$(\mathcal{A}, n)$
1 $\mathcal{W} \leftarrow \mathcal{A}$
2 **for** $k \leftarrow 1$ **to** n
3 **do for** $i \leftarrow 1$ **to** n
4 **do for** $j \leftarrow 1$ **to** n
5 **do if** $W_{ik} \neq \emptyset$ and $W_{kj} \neq \emptyset$
6 **then** $W_{ij} \leftarrow W_{ij} \cup W_{ik} \cdot' W_{kj}$
8 **return** $\mathcal{W}$



2. ábra.

A 2. ábrán lévő gráfra az eredmény:

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