Programare dinamica II

În acest curs vom trata problema găsirii celor mai scurte drumuri dintre toate perechile de noduri din graf. Aplicația acestei probleme poate fi în a crea o situație a distanțelor dintre toate perechile de orașe de pe o hartă. Fie un graf ponderat orientat, $G(V, E)$ cu o funcție pentru ponderile muchiilor $w: E \to R$. Dorim să găsim, pentru fiecare pereche de noduri $u, v \in V$, un drum de la u la v, unde ponderea unui drum este suma tuturor ponderilor muchiilor ce alcătuiesc drumul.

Putem rezolva problema tuturor drumurilor celor mai scurte rulând de $|V|$ ori, un algoritm de tipul Dijkstra ce rezolvă cel mai scurt drum dintre un nod sursă și un nod destinație (formând perechea de noduri de mai sus). Dacă toate muchiile sunt pozitive, putem folosi algoritmul lui Dijkstra, altfel va trebui să folosim un algoritm de tipul Bellman-Ford. Dacă folosim algoritmul lui Dijkstra obținem un ordin de timp $O(V^3)$.

Dacă graful conține și muchii negative, va trebui să aplicăm un algoritm de tipul Bellman Ford ce are ca ordin de timp total $O(V^2E)$. Dacă avem un graf dens putem ajunge și la un ordin de $O(V^4)$. Vom vedea în continuare cum putem să realizăm un algoritm mai rapid.

Spre deosebire de algoritmii cu o singură sursă, ce se folosesc mai mult de lista de adiacență pentru reprezentarea unui graf (din motive evidente), algoritmii prezentați în continuare vor folosi matrice de adiacență ca reprezentare a grafurilor. Intrarea este o matrice de $n \times n$ și anume W. Adică $W = (w_{ij})$ unde

$$w_{ij} = \begin{cases} 0 & dacă\ i = j \\ costul\ muchiei\ (i,j) & dacă\ i \neq j\ și\ (i,j) \in E \\ \infty & dacă\ i \neq j\ \ și\ (i,j) \notin E \end{cases}$$

Sunt permise muchii de cost negativ, dar graful nu are voie să conțină cicluri de cost negativ.

Soluția algoritmilor prezentați în acest capitol, este o matrice $n \times n$ și anume $D = (d_{ij})$, unde d_{ij} conține costul celui mai scurt drum de la nodul i la nodul j. Fie notația $\delta(i,j)$ pentru cel mai scurt drum de la nodul i la nodul j, ce duce la egalitatea $d_{ij} = \delta(i,j)$, egalitate ce are loc la sfârșitul rulării algoritmului.

Pentru a rezolva problema celor mai scurte drumuri folosind o matrice de adiacență, trebuie să calculăm și matricea de predecesori $\Pi = (\pi_{ij})$ unde π_{ij} este $NULL$ dacă fie $i = j$, fie nu există drum de la i la j.

Pentru fiecare nod $i \in V$, vom defini un subgraf predecesor al lui G pentru i astfel ca

$G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$ unde

$V_{\pi,i} = \{j \in V : \pi_{ij} \neq NULL\} \cup \{i\}$ și

$E_{\pi,i} = \{(\pi_{ij}, i) : j \in V_{\pi,i} \text{ și } \pi_{ij} \neq NULL\}$

Dacă $G_{\pi,i}$ este un arbore al celor mai scurte drumuri, atunci următoarea procedură, va afișa cel mai scurt drum de la un nod i la un nod j.

$PRINT - ALL - PAIRS - SHORTEST - PATH(\Pi, i, j)$

1. ***if*** $i = j$
2. ***then*** $print\ i$
3. ***else if*** $\pi_{ij} = NULL$
4. ***then*** $print$ "$no\ path\ from\ i\ to\ j\ exists$"
5. ***else*** $PRINT - ALL - PAIRS - SHORTEST - PATH((\Pi, i, \pi_{ij})$
6. $print\ j$

În continuare vom studia o prima metodă ce implică și programarea dinamică pentru a rezolva problema tuturor celor mai scurte drumuri.

Cele mai scurte drumuri și multiplicarea matricelor

Secțiunea prezintă un algoritm în care fiecare buclă mare va apela o operație similar multiplicării matricelor. Vom dezvolta un algoritm de ordin $O(V^4)$ ce poate fi îmbunătățit până la $O(V^3 \log V)$. Să ne reamintim care sunt pașii unu algoritm de programare dinamică:

1. Caracterizarea structurii unei soluții optime
2. Definirea recursivă a unei valori a soluției optime
3. Calculul valorii soluției optime totale de jos în sus.

Structura unui drum minim

Pentru a caracteriza această structură se poate demonstra că toate subdrumurile unui drum minim sunt la rândul lor minime. Să presupunem că graful este reprezentat de matricea de adiacență $W = (w_{ij})$. Fire un drum minim p de la nodul i la nodul j, să presupunem că p conține maxim m muchii.

Presupunând că nu există cicluri de cost negativ, m este finit. Dacă $i = j$ atunci $p = 0$ și nu are nici o muchie. Dacă i, j sunt distincte atunci separăm p în drumul $i \xrightarrow{p'} k \to j$ unde p' conține maxim $m - 1$ muchii. Știm că p' este un drum minim de la i la k. Atunci $\delta(i, j) = \delta(i, k) + w_{kj}$.

Solutia recursivă a problemei celor mai scurte drumuri

Fie $d_{ij}^{(m)}$ ponderea minimă a oricărui drum de la nodul i la nodul j ce conține maxim m muchii. Atunci când $m = 0$, există un drum minim de la i la j fără muchii doar dacă $i = j$. De aceea

$$d_{ij}^{0} = \begin{cases} 0 & dacă\ i = j \\ \infty & dacă\ i \neq j \end{cases}$$

Pentru $m \geq 1$ calculăm $d_{ij}^{(m)}$ ca minimum de $d_{ij}^{(m-1)}$ (ponderea celui mai scurt drum de la i la j format din $m - 1$ muchii) și ponderea minimă a oricărui drum de la i la j ce constă din maxim m muchii ce a fost obținut uitându-ne la toți predecesorii k ai lui j. De aceea definim recursiv:

$$d_{ij}^{m} = \min\left(d_{ij}^{(m-1)}, \min_{1 \leq k \leq n} d_{ik}^{(m-1)} + w_{kj}\right) = \min_{1 \leq k \leq n} d_{ik}^{(m-1)} + w_{kj}$$

A doua egalitate vine din faptul că $w_{jj} = 0$.

În ce constau ponderile celor mai scurte drumuri $\delta(i, j)$? Dacă graful nu conține cicluri de cost negativ, atunci toate drumurile minime sunt simple deci conțin maxim $n - 1$ muchii. Un drum de la nodul i la nodul j cu mai mult de $n - 1$ noduri nu poate avea o pondere mai mică decât un drum minim de la nodul i la nodul j.

Cel mai scurt drum dintre două noduri are un cost ce îndeplinește relația:

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \cdots$$

Calculul celui mai scurt drum de sus în jos

Fie matricea de intrare $W = (w_{ij})$, va trebui să calculăm seriile de matrice $D^{(1)}, D^{(2)}, \dots, D^{(n-1)}$ unde pentru $m = 1,2, \dots, n-1$ avem $D^{(m)} = \left(d_{ij}^{(m)}\right)$. Matricea finală: $D^{(n-1)}$ conține cele mai scurte drumuri. Se poate observa că $D^{(1)} = W$.

Principala funcție a algoritmului este cea de mai jos, care, dat fiind matricele $D^{(m-1)}$ și W , va calcula matricea $D^{(m)}$. Adică extinde cele mai scurte drumuri (de până acum) cu o muchie în plus.

$EXTEND - SHORTEST - PATHS(D, W)$

1. $n \leftarrow rows[D]$
2. $fie\ D' = \left(d'_{ij}\right)\ o\ matrice\ n \times n$
3. $\boldsymbol{for}\ i \leftarrow 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
4. $\quad \boldsymbol{for}\ j \leftarrow 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
5. $\quad\quad d'_{ij} \leftarrow \infty$
6. $\quad\quad \boldsymbol{for}\ k \leftarrow 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
7. $\quad\quad\quad d'_{ij} \leftarrow \min(d'_{ij}, d_{ik} + w_{kj})$
8. $\boldsymbol{return}\ D'$

Procedura calculează o matrice $D' = \left(d'_{ij}\right)$ ce va fi returnată la sfârșit. Practic se aplică ecuația $d_{ij}^{m} = \min_{1 \le k \le n} d_{ik}^{(m-1)} + w_{kj}$ de mai sus. Ordinul este $O(n^3)$ din cauza celor 3 for-uri imbricate. Se poate vedea asemănarea cu multiplicarea matricelor. De exemplu dacă dorim să calculăm produsul $C = A \cdot B$ a două matrice $n \times n$, pentru $i, j = 1,2, \dots, n$ vom calcula:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Dacă facem înlocuirile :

$d^{(m-1)} \rightarrow a$

$w \rightarrow b$

$d^{(m)} \rightarrow c$

$min \rightarrow +$

$+ \rightarrow \cdot$

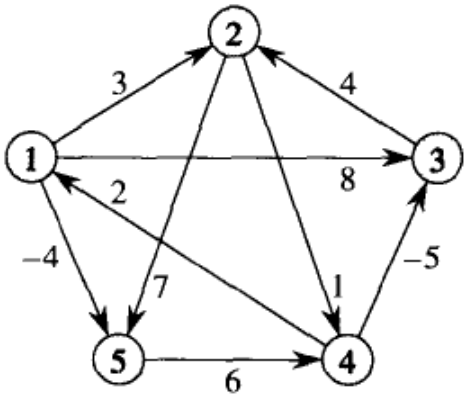
obținem exact algoritmul de înmulțire a două matrice.

Aceasta este funcția care calculează drumul minim pentru $D^{(n-1)}$. Pentru a calcula toate drumurile minime avem nevoie să apelăm această funcție în cadrul următoarei proceduri.

$SLOW - ALL - PAIRS - SHORTEST - PATHS(W)$

1. $n \leftarrow rows[W]$
2. $D^{(1)} \leftarrow W$
3. $\boldsymbol{for}\ m \leftarrow 2\ \boldsymbol{to}\ n-1\ \boldsymbol{do}$
4. $\quad D^{(m)} \leftarrow EXTEND - SHORTEST - PATHS(D^{(m-1)}, W)$
5. $\boldsymbol{return}\ D^{(n-1)}$

În figura 1 avem un exemplu de funcționare al acestui algoritm, pentru graful dat. Avem 5 noduri, astfel că metoda $EXTEND - SHORTEST - PATHS(D^{(m-1)}, W)$ va fi apelată de 3 ori.



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Figura 1. Un digraf și secvența de matrice D^m calculate cu funcția $SLOW{-}ALL{-}PAIRS{-}SHORTEST{-}PATHS(W)$. Se poate verifica $D^{(5)} = D^{(4)}$.

Problema este că ordinul de timp al acestui algoritm este de $O(V^4)$, de acolo vine și numele.

Îmbunătățirea timpului

Scopul este de a calcula toate matricele $D^{(m)}$, dar ne interesează doar matricea $D^{(n-1)}$. Absența ciclurilor negative implică $D^{(m)} = D^{(n-1)}$ pentru orice $m \geq n-1$. De aceea putem calcula $D^{(n-1)}$ cu $\lceil \log(n-1) \rceil$ produse de matrice astfel:

$D^{(1)} = W$

$D^{(2)} = W^2 = W \cdot W$

$D^{(4)} = W^4 = W^2 \cdot W^2$

$D^{(8)} = W^8 = W^4 \cdot W^4$

Generalizând obținem

$D^{2^{\lceil \log(n-1) \rceil}} = W^{2^{\lceil \log(n-1) \rceil}} = W^{2^{\lceil \log(n-1) \rceil - 1}} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}}$

Următoarea procedură permite calcularea secvenței de mai sus folosind tehnica numită ridicare la pătrat repetată.

$FASTER - ALL - PAIRS - SHORTEST - PATHS(W)$

1. $n \leftarrow rows[W]$
2. $D^{(1)} \leftarrow W$
3. $m \leftarrow 1$
4. $\boldsymbol{while}\ n-1 > m\ \boldsymbol{do}$
5. $\quad D^{(2m)} \leftarrow EXTEND - SHORTEST - PATHS(D^{(m)}, D^{(m)})$
6. $\quad m \leftarrow 2m$
7. $\boldsymbol{return}\ D^{(m)}$

În fiecare iterație a buclei while vom calcula $D^{(2m)} = (D^{(m)})^2$ începând cu $m = 1$. Ultima iterație calculează $D^{(n-1)}$ calculând $D^{(2m)}$ pentru $n-1 < 2m < 2n-2$. Timpul acestui algoritm este $O(n^3 \log n)$ deci este mai bun decât cel prezentat anterior.

Urmează să prezentăm un alt algoritm ce rezolvă problema și mai rapid.

Algoritmul Floyd-Warshall

Vom folosi o altă formulare, pentru a rezolva problema tuturor drumurilor minime dintr-un digraf $G = (V, E)$. Algoritmul denumit Floyd-Warshall rulează în $O(V^3)$. Digraful poate conține muchii negative, însă nu poate avea cicluri de cost negativ. Vom redefini problema folosind programarea dinamică, iar apoi vom prezenta o metodă similară pentru a găsi închiderea tranzitivă dintr-un graf orientat.

Structura drumului minim

În acest algoritm vom folosi o altă caracterizare a drumului minim și anume algoritmul va considera noduri *intermediare* din drumul minim, unde un nod *intermediar* este un nod din $p = \langle v_1, v_2, \dots, v_l \rangle$, altul decât v_1 sau v_l. p este un drum simplu ce conține acele noduri.

Algoritmul lui Floyd-Warshall se bazează pe următoarea observație. Fie nodurile grafului: $V = \{1,2, \dots, n\}$ și fie o submulțime $\{1,2, \dots, k\}$ de noduri. Pentru orice pereche de noduri $i, j \in V$, să considerăm toate drumurile de la i la j ce au ca noduri intermediare pe cele din mulțimea $\{1,2, \dots, k\}$. Fie p drumul de cost minim dintre ele (este simplu pentru ca G nu conține cicluri negative). Algoritmul va exploata relația dintre drumul p și cel mai scurt drum dintre i și j ce are ca noduri intermediare pe cele din mulțimea $\{1,2, \dots, k-1\}$. Relația depinde de apartenența lui k la p.

- Dacă k nu este nod intermediar pentru p atunci toate nodurile intermediare ale lui p sunt în $\{1,2, \dots, k-1\}$. De aceea drumul de cost minim de la i la j cu noduri intermediare din mulțimea $\{1,2, \dots, k-1\}$ este și drumul de cost minim cu nodurile intermediare din $\{1,2, \dots, k\}$.
- Dacă k este nod intermediar al lui p, împărțim p în două subdrumuri $i \overset{p_1}{\to} k \overset{p_2}{\to} j$. După cum apare în figura 2. p_1 este drumul minim de la i la k și are nodurile intermediare în $\{1,2, \dots, k\}$. Similar, p_2 este drumul minim de la k la j și are nodurile intermediare în $\{1,2, \dots, k-1\}$. Reunind cele două mulțimi, probleme se poate constata că relația este îndeplinită.

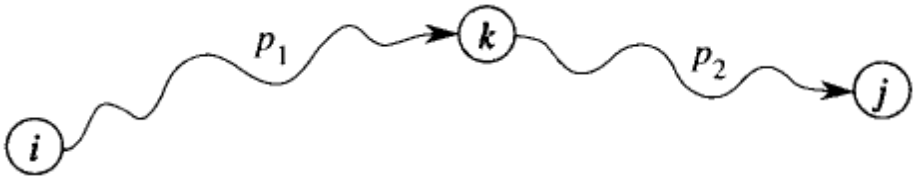


Figura 2. Drumul p este cel de cost minim dintre i și j

O soluție recursivă la problema tuturor drumurilor de cost minim

Pe baza observației de mai sus vom defini o formulă recursivă pentru estimarea drumurilor de cost mini. Fie $d_{ij}^{(k)}$, costul drumului de cost minim de la nodul i la nodul j ce are nodurile intermediare în mulțimea $\{1,2,\dots,k\}$. Atunci când $k = 0$, un drum de la i la j nu are noduri intermediare, ceea ce înseamnă că are cel mult o muchi deci $d_{ij}^{0} = w_{ij}$. O soluție recursivă ar fi:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & dacă\ k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & dacă\ k \geq 1 \end{cases}$$

Matricea $D^{(n)} = (d_{ij}^{(n)})$ dă la final $d_{ij}^{(n)} = \delta(i,j)$ pentru toate $i,j \in V$.

Calculul drumurilor celor mai scurte de jos în sus

Pe baza acestei recurențe putem folosi o procedură care să calculeze valorile $d_{ij}^{(k)}$ incrementând valoarea lui k. Intrarea este matricea W de $n \times n$. Procedura returnează matricea $D^{(n)}$ ce conține cele mai scurte drumuri între oricare două noduri.

$FLOYD - WARSHALL(W)$

1. $n \leftarrow rows[W]$
2. $D^{(0)} \leftarrow W$
3. $\boldsymbol{for}\ k \leftarrow 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
4. $\quad \boldsymbol{for}\ i \leftarrow 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
5. $\quad\quad \boldsymbol{for}\ j \leftarrow 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
6. $\quad\quad\quad d_{ij}^{(k)} \leftarrow min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
7. $\boldsymbol{return}\ D^{(n)}$

Ordinul de timp al algoritmului Floyd-Warshall este determinat de cele trei bucle *for* imbricate. De aceea algoritmul are un ordin de timp $O(n^3)$.

Construirea celui mai scurt drum

Există mai multe moduri de a construi cele mai scurte drumuri, folosind algoritmul lui Floyd-Warshall. Un mod este de a calcula matricea D din cele drumurile de pondere minimă, și apoi a construi matricea de predecesori Π. Odată ce avem matricea Π, putem apela metoda $PRINT - ALL - PAIRS - SHORTEST - PATH(\Pi, i, j)$ pentru a lista nodurile dintr-un anumit drum de cost minim.

Putem calcula matricea Π "on line" pe măsură ce algoritmul Floyd-Warshall rulează. Vom calcula secvența de matrice $\Pi^{(0)}$, $\Pi^{(1)}$, …, $\Pi^{(n)}$ unde $\Pi = \Pi^{(n)}$ și π_{ij}^{k} este definit ca predecesorul nodului j într-un drum minim de la nodul i cu toate nodurile intermediare din mulțimea $\{1,2, \dots, k\}$.

Putem formula recursiv $\pi_{ij}^{(k)}$. Atunci când $k = 0$, un drum minim dintre i și j nu conține nici o muchie. De aceea pute formula

$$\pi_{ij}^{(0)} = \begin{cases} NULL & dacă\ i = j\ sau\ w_{ij} = \infty \\ i & dacă\ i \neq j\ si\ w_{ij} < \infty \end{cases}$$

Pentru $k \geq 1$ dacă luăm drumul $i \to k \to j$, atunci predecesorul lui j va fi același cu predecesorul lui j pe care îl alegem din drumul de cost minim ce are ca noduri intermediare $\{1,2, \dots, k-1\}$, sau cel care are i ca nod intermediar.

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k)} & dacă\ d_{ij}^{(k)} \leq d_{ij}^{(k-1)} \\ \pi_{kj}^{(k-1)} & dacă\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Figura 3 prezintă secvențele de matrice $\Pi^{(k)}$ ce se formează în timpul rulării algoritmului Floyd-Warshall pentru graful din figura 1.

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Figura 3. Secvențele de matrice ce sunt obținute în urma execuției algoritmului Floyd-Warshall

Închiderea tranzitivă a unui digraf

Dat fiind un graf orientat $G = (V, E)$, dorim să aflăm dacă există drum între i și j pentru orice noduri $i, j \in V$. *Închiderea tranzitivă* a lui G este definită ca graful $G^* = (V, E^*)$ unde $E^* = \{(i, j): \text{există un drum între } i \text{ și } j\}$.

Un mod de a calcula închiderea tranzitivă dintr-un graf în $O(n^3)$ este să atribuim o valoare 1 fiecărei muchii și să rulăm algoritmul Floyd-Warshall. Dacă există drum între nodul i și j obținem $d_{ij} < n$ altfel $d_{ij} = \infty$.

Mai există un mod asemănător de a calcula închiderea tranzitivă a lui G în timp $O(n^3)$. Această metodă implică substituția operațiunilor logice $\vee$ și $\wedge$ cu operațiunile aritmetice de min și + din algoritmul Floyd-Warshall. Pentru $i, j, k = 1,2, \dots, n$ vom defini $t_{ij}^{(k)}$ ca 1 dacă există drum între i și j ce are noduri intermediare în mulțimea $\{1,2, \dots, k\}$ sau 0 altfel. Astfel, definiția recursivă este:

$$t_{ij}^{(0)} = \begin{cases} 0 & dacă\ i \neq j\ si\ (i,j) \notin E \\ 1 & dacă\ i = j\ sau\ (i,j) \in E \end{cases}$$

Pentru $k \geq 1$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

Ca și în algoritmul Floyd-Warshall vom calcula matricele după k astfel:

$TRANZITIVE - CLOSURE(G)$

1. $n \leftarrow |V(G)|$
2. **for** $i \leftarrow 1$ **to** n **do**
3. **for** $j \leftarrow 1$ **to** n **do**
4. **if** $i = j$ **or** $(i,j) \in E[G]$
5. **then** $t_{ij}^{(0)} \leftarrow 1$
6. **else** $t_{ij}^{(0)} \leftarrow 0$
7. **for** $k \leftarrow 1$ **to** n **do**
8. **for** $i \leftarrow 1$ **to** n **do**
9. **for** $j \leftarrow 1$ **to** n **do**
10. $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$
11. **return** $T^{(n)}$