Suport curs algoritmica grafurilor
V. Drumuri minime între toate perechile de vârfuri

Problema găsirii derumului minim între toate perechile de vârfuri ale unui graf:

- se dă: un graf orientat $G = (V, E)$ cu funcția de pondere $\omega : E \to \mathbb{R}$,
- se vrea: să se găsescă pentru fiecare pereche de vârfuri $u, v \in E$ un drum minim de la u la v (suma ponderilor arcelor din drum să fie minimă),
- afișarea rezultatului se face sub forma unei matrici $A_{n,n}$, $n = |V|$, unde elementul $a_{i,j} = \delta(i, j)$ arată lungimea drumului de la vârful i la vârful j.

Idee: se poate rezolva problema dacă se rulează un algoritm de drum minim între un vârf sursă s și toate vârfurile din graf $V \setminus \{s\}$ de $|V|$ ori (pentru fiecare vârf din graf ca și sursă). De exemplu:

- dacă există ponderi negative se rulează BELLMAN_FORD pentru fiecare vârf din graf, complexitatea $O(V^2E)$ iar dacă graful este dens $O(V^4)$;
- dacă toate ponderile nu sunt negative se poate rula Dijkstra pentru fiecare vârf din graf, complexitatea $O(VElgV)$ dacă se folosește un binary heap pentru coada de priorități ($O(V^3lgV)$ pentru un graf dens) și $O(V^2\,lg\,V + VE)$ dacă se folosește un Fibonacci heap pentru coada de priorități ($O(V^3)$ dacă graful este dens).

Vom vedea cum putem determina drumul de cost minim între oricare două vârfuri din graf în $O(V^3)$ în toate cazurile, fără a folosi structuri de date speciale.

5.1 Drumuri minime și înmulțirea unor matrici

Majoritatea algoritmilor își reprezintă un graf folosind ca și reprezentare matricea de adiacență. Fie un graf $G = (V, E)$, vârfurile sunt numerotate de la 1 la n, reprezentat de o matrice de adiacență de ponderi $A = (a_{i,j})_{i,j=\overline{1,n}}$ unde:

$$a_{i,j} = \begin{cases} 0 & \text{dacă } i = j, \\ \text{ponderea arcului } (i,j) & \text{dacă } \ i \neq j, (i,j) \in E, \\ \infty & \text{dacă } i \neq j, (i,j) \notin E. \end{cases}$$

Rezultatul va fi matricea $D = (d_{i,j})$, unde $d_{i,j} = \delta(i, j)$.

Se prezintă o soluție bazată pe programare dinamică pentru a rezolva problema drumului minim între oricare două vârfuri din graf.

Pentru aceasta trebuie:

1. caracterizată structura unei soluții optimale
2. definită recursiv valoarea soluției optimale
3. calculată valoarea soluției optimale

5.1.1 Structura unui drum minim

Pe baza lemei drumului minim (vezi cursurile anterioare): oricare subdrum dintr-un drum minim este drum minim. Graful $G = (V, E)$ este reprezentat de matricea de adiacență $A = (a_{i,j})$.

- fie p drumul minim de la vârful i la vârful j format din m arcuri;
- dacă nu există un circuit negativ m este finit;
- dacă $i = j$ ponderea lui p este 0;
- dacă $i \neq j$ atunci $i \overset{p'}{\rightsquigarrow} k \to j$, unde drumul p' conține $m - 1$ arce;
- din lema drumului minim: p' este drum minim de la i la k și $\delta(i, j) = \delta(i, k) + a_{k,j}$.

5.1.2 O soluție recursivă

Fie $l_{ij}^{(m)}$ ponderea minimă a unui drum $i \rightsquigarrow j$ de m arce

$$l_{ij}^{(0)} = \begin{cases} 0, & \text{dacă } i = j, \\ \infty, & \text{dacă } i \neq j. \end{cases}$$

De la $m \geq 1$ se calculează $l_{ij}^{(m)}$ ca minimul lui $l_{ij}^{(m-1)}$ (ponderea drumului minim de la i la j format din cel mult $m-1$ arce) și ponderea minimă a oricărui drum de la i la j format din m arce, obținută dacă ne uităm la toți predecesorii k ai lui j.

Se pot defini recursiv:

$$l_{ij}^{(m)} = min(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \{l_{ij}^{(m-1)} + a_{ij}\}) = \min_{1 \leq k \leq n} \{l_{ij}^{(m-1)} + a_{jk}\} \tag{1}$$

deoarece $a_{jj} = 0 \forall j$.

Care este drumul minim $\delta(i, j)$?

Dacă G nu are circuite negative, pentru orice pereche i si j pentru care $\delta(i, j) < \infty$ există un drum minim de la i la j simplu ce conține cel mult $n - 1$ arce. Deci:

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

5.1.3 Calculul drumului minim

Ca și date de intrare se primește $A = (a_{ij})$ și se calculează o serie de matrici $L^{(1)}, L^{(2)}, ..., L^{(n-1)}$ unde pentru $m = 1, 2, ..., n - 1$ avem $L^{(m)} = (l_{ij}^{(m)})$. $L^{(n-1)}$ conține drumul minim (ponderile drumului minim)

$$l_{ij}^{(1)} = a_{ij} \forall i, j \in \mathbb{N} \Rightarrow L^{(1)} = A.$$

Următoarea procedură primește ca și parametrii $L^{(n-1)}$, A și întoarce $L^{(n)}$.

EXTINDE_DRUM_MINIM(L,A)

$n = L$.rânduri

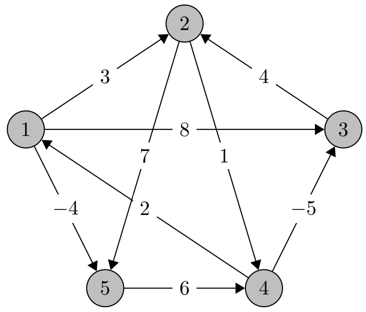

Figura 1: Exemplu determinare drum minim

fie $L' = (l'_{ij})$ o matrice $n \times x$
for $1 \leq i \leq n$ **do**
  for $1 \leq j \leq n$ **do**
    $l'_{ij} = \infty$
    for $1 \leq k \leq n$ **do**
      $l'_{ij} = min(l'_{ij}, l_{ik} + a_{kj})$
return L'

Procedura determină matricea L' folosind relația (1), timpul de execuție este $\Theta(V^3)$. Se poate face o paralelă cu procedura de inmulțire a două matrici.

Practic drumul minim se determină prin extinderea drumului minim arc cu arc. Trebuie să se determine:

$$\begin{aligned}
&L^{(1)} = L^{(0)} \cdot A = A \\
&L^{(2)} = L^{(1)} \cdot A = A^2 \\
&L^{(3)} = L^{(2)} \cdot A = A^3 \\
&... \\
&L^{(n-1)} = L^{(n-2)} \cdot A = A^{n-1}
\end{aligned}$$

Matricea $L^{(n-1)} = A^{n-1}$ conține drumul minim. Următoarea procedură determină secvența în $\Theta(V^4)$.

DETERMINA_TOATE_DRUMURILE_MIN(A)

$n = A$.rânduri
$L^{(1)} = A$
for $2 \leq m \leq n-1$ **do**
  fie $L^{(m)}$ o matrice $n \times x$
  $L^m = EXTINDE\_DRUM\_MINIM(L^{m-1}, A))$
return $L^{(n-1)}$

De exemplu, fie graful din figura 1 pentru care se determină matricea $L^{(n-1)}$:

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

...

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

5.1.4 Îmbunătățirea timpului de rulare

Nu trebuie determiate m matrici, ne interesează doar $(n-1)$ matrici. $L^{(n-1)}$ se poate calcula din $\lceil \lg(n-1) \rceil$ pași deoarece $2^{\lceil \lg(n-1) \rceil} \geq n-1$ produsul final $L^{(2^{\lceil \lg(n-1) \rceil})}$ este $L^{(n-1)}$.

$$\begin{aligned} &L^{(1)} = A \\ &L^{(2)} = A^2 = A \cdot A \\ &L^{(4)} = A^4 = A^2 \cdot A^2 \\ &L^{(8)} = A^8 = A^4 \cdot A^4 \\ &\ldots \\ &L^{(2^{\lceil \lg(n-1) \rceil})} = A^{2^{\lceil \lg(n-1) \rceil}} = A^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot A^{2^{\lceil \lg(n-1) \rceil - 1}} \end{aligned}$$

Procedura este:

MAI_RAPID_TOATE_DRUMURILE_MIN(A)

```
n = A.rânduri
L^(1) = A
m = 1
while m ≤ n − 1 do
    fie L^(2m) o matrice n × x
    L^2m = EXTINDE_DRUM_MINIM(L^m, L^m))
    m = 2m
return L^(m)
```

Timpul de execuție este $\Theta(V^3 \lg V)$, fiecare produs de matrice ia $\Theta(V^3)$ datorită $\lceil \lg(n-1) \rceil$.

5.2 Floyd-Warshall

Algoritmul a fost discutat în cursurile anterioare.

Recursiv, fie $d_{ij}^{(k)}$ ponderea drumului minim de la i la j, vârfurile intermediare drumului sunt în setul $\{1, 2, ..., k\}$.

$$d_{i,j}^{(k)} = \begin{cases} a_{ij} & \text{, dacă } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{, dacă } k \geq 1. \end{cases}$$

5.3 Închiderea tranzitivă a unui graf orientat

Fie $G = (V, E)$ un graf orientat cu setul $V = \{1, 2, ..., n\}$, trebuie să se determine dacă există un drum în graful G de la vârful i la vârful j, unde $i, j \in V$.

Închiderea tranzitivă a lui G este definită ca și graful $G^* = (V, E^*)$, unde

$$E^* = \{(i, j) | \text{ dacă există } i \rightsquigarrow j \text{ în } G\}.$$

O modalitate de a determina închiderea tranzitivă în $\Theta(V^3)$ e de a rula algoritmul *Floyd-Warshall* pe G cu ponderea 1 pe fiecare arc. Dacă există un drum $i \rightsquigarrow j$ atunci $d_{ij} < n$ altfel $d_{ij} = \infty$.

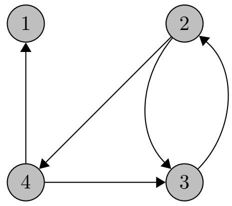

Figura 2: Exemplu determinare închidere tranzitivă

Există o altă metodă pentru a determina închiderea tranzitivă în $\Theta(V^3)$ care poate salva timp și spațiu în practică: se substituie oeprațiile aritmetice *min* și + din *Floyd-Warshall* cu operațiile logice $\vee$ (*SAU* logic) și $\wedge$ (*SI* logic).

Pentru $i, j, k = 1, ..., n$ se definește $t_{ij}^{(k)} = 1$ dacă exista un drum $i \rightsquigarrow j$ cu vârfurile intermediare în setul $\{1, ..., k\}$ și $t_{ij}^{(k)} = 0$ în rest.

Se construiește închiderea tranzitivă $G^* = (V < E^*)$ prin adăugarea arcului (i, j) în E^* dacă și numai dacă $t_{ij}^{(k)} = 1$.

Pentru a calcula recursiv închiderea tranzitivă, putem defini:

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

și pentru $k \geq 1$

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

Se determină matricile $T^k = (t_{ij}^{(k)})$ pentru un $k \nearrow\nearrow$. Procedura pentru determinarea închiderii tranzitive este:

INCHIDERE_TRANZITIVA(G)

$n = |V|$
fie $T^{(0)} = (t_{ij}^{(0)})$ o matrice $n \times x$
for $1 \leq i \leq n$ **do**
    for $1 \leq j \leq n$ **do**
        if $i == j$ sau $(i, j) \in E$ **then**
            $t_{ij}^{(0)} = 1$
        else
            $t_{ij}^{(0)} = 0$
for $1 \leq k \leq n$ **do**
    fie $T^{(k)} = (t_{ij}^{(k)})$ o matrice $n \times x$
    for $1 \leq i \leq n$ **do**
        for $1 \leq j \leq n$ **do**
            $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$
return $T^{(m)}$

De exemplu, fie graful din figura 2 pentru care se determină matricile $T^{(k)}$:

5.4 Algoritmul lui Johnson pentru grafuri rare

Algoritmul găsește drumul minim în $O(V^2 \lg V + VE)$. Pentru grafuri rare e asimtotic mai rapid decât înmulțirea de matrici și Floyd-Warshall. Algoritmul găsește o solutie pentru grafuri fără circuite negative. Folosește ca și subrutina *Dijkstra* și *Bellman-Ford*.

$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Algoritmul folosește tehnica de reponderare:

- dacă toate ponderile sunt strict pozitive pot rula *Dijkstra* din fiecare vârf și pot găsi drumul minim în $O(V^2 \lg V + VE)$ (Fibonacci heap)
- dacă G are ponderi negative dar nu are circuit negativ trebuie recalculate ponderile astfel încât să fie pozitive, noul set de ponderi $\widehat{w}$ trebuie să satisfacă următoarele:
  1. pentru toate perechile $u, v \in V$, un drum p este minim de la u la v folosind ponderile w dacă p e drum minim de la u la v și pentru ponderile $\widehat{w}$;
  2. pentru toate arcele (u, v), $\widehat{w}(u, v) \geq 0$;
- $\widehat{w}$ se poate determina în $O(VE)$.

Prin reponderare drumul minim trebuie păstrat. Notăm cu:

- δ drumul minim din ponderile w,
- $\widehat{\delta}$ drumul minim din ponderile $\widehat{w}$.

Lema 5.1 (Reponderarea nu schimbă drumurile minime) *fie un graf orientat și ponderat $G = (V, E)$ cu funcția de pondere $w : E \to \mathbb{R}$, fie $h : V \to \mathbb{R}$ o funcție ce mapează vârfurile la numere reale. Pentru fiecare arc $(u, v) \in E$ se definește*

$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v).$$

Fie $p = \langle v_0, ..., v_k \rangle$ un drum de la v_0 la v_k, p este un drum minim de la v_0 la v_k cu w dacă e drum minim și pentru $\widehat{w}$, $w(p) = \delta(v_0, v_k) \Leftrightarrow \widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.

Reponderare pozitivă se vrea ca $\widehat{w}(u, v) \geq 0$ pentru $(u, v) \in E$. Fie $G = (V, E)$ cu $w : E \to \mathbb{R}$, se construiește un nou graf $G' = (V', E')$ unde:

- $V' = V \cup \{s\}$, $s \in V$
- $E' = E \cup \{(s, v) | v \in V\}$, $w(s, v) = 0, \forall v \in V$.

G' nu are circuite negative daca G nu are circuite negative. Se definește $h(v) = \delta(s, v), \forall v \in V'$

$$h(v) \leq h(u) + w(u, v), \forall (u, v) \in E'$$

deci

$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0.$$

Algoritmul lui Johnson este:

JOHNSON(G)

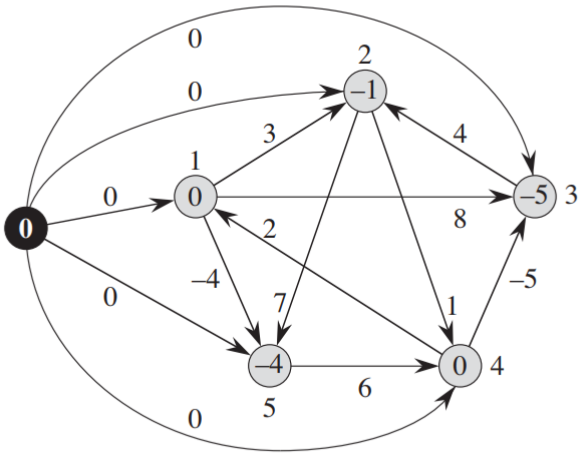

Figura 4: Graful G' cu funcţia de pondere w.

determină $G', V' = V \cup \{s\}, E' = E \cup \{(s,v)|v \in V\}$ și $w(s,v) = 0 \forall v \in V$
if *BELLMAN_FORD*(G', w, s)==*FALSE* **then**
  exit
else
  for $v \in V'$ **do**
    pune $h(v) = \delta(s,v)$ determinată de *BELLMAN_FORD*
  for $(u,v) \in E'$ **do**
    $\widehat{w}(u,v) = w(u,v) + h(u) - h(v)$
  fie $D = (d_{uv})$ o matrice $n \times x$
  for $u \in V$ **do**
    rulează *DIJKSTRA*($G, \widehat{w}, u$) pentru a determina $\widehat{\delta}(u,v) \forall v \in V$
    for $v \in V$ **do**
      $d_{uv} = \widehat{\delta}(u,v) + h(u) - h(v)$
  return D

Ca și exemplu fie graful G' cu funcția de pondere w din figura 4. Figura 5 prezintă graful după reponderare. Figurile 6a-6e prezintă rezultatul rulării algoritmului *Dijkstra* pentru fiecare vârf din G ca și sursă, vârful sursă este negru iar drumjul minim este reprezentat de arcele marcate cu gri. Fiecare vârf conține valorile $\widehat{\delta}(u,v)/\delta(u,v)$. Valoarea $d_{uv} = \delta(u,v)$ este $\widehat{\delta}(u,v) + h(u) - h(v)$.

5.5 Referințe

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.

2. Geir Agnarsson and Raymond Greenlaw. 2006. Graph Theory: Modeling, Applications, and Algorithms. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

3. Mark Newman. 2010. Networks: An Introduction. Oxford University Press, Inc., New York, NY, USA.

4. Cristian A. Giumale. 2004. Introducere în analiza algoritmilor, teorie și aplicație. Polirom.

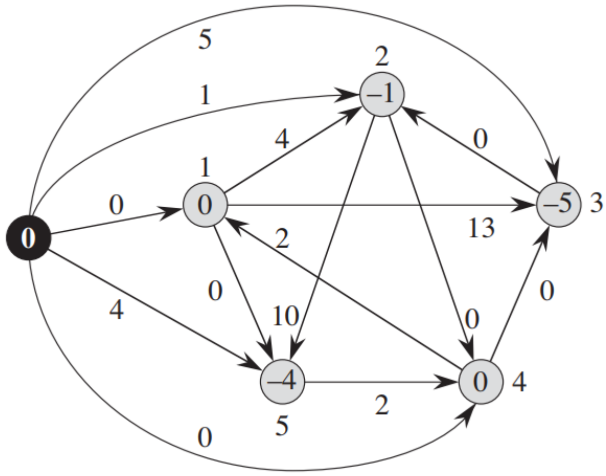

Figura 5: Graful G' cu funcţia de pondere $\widehat{w}$.

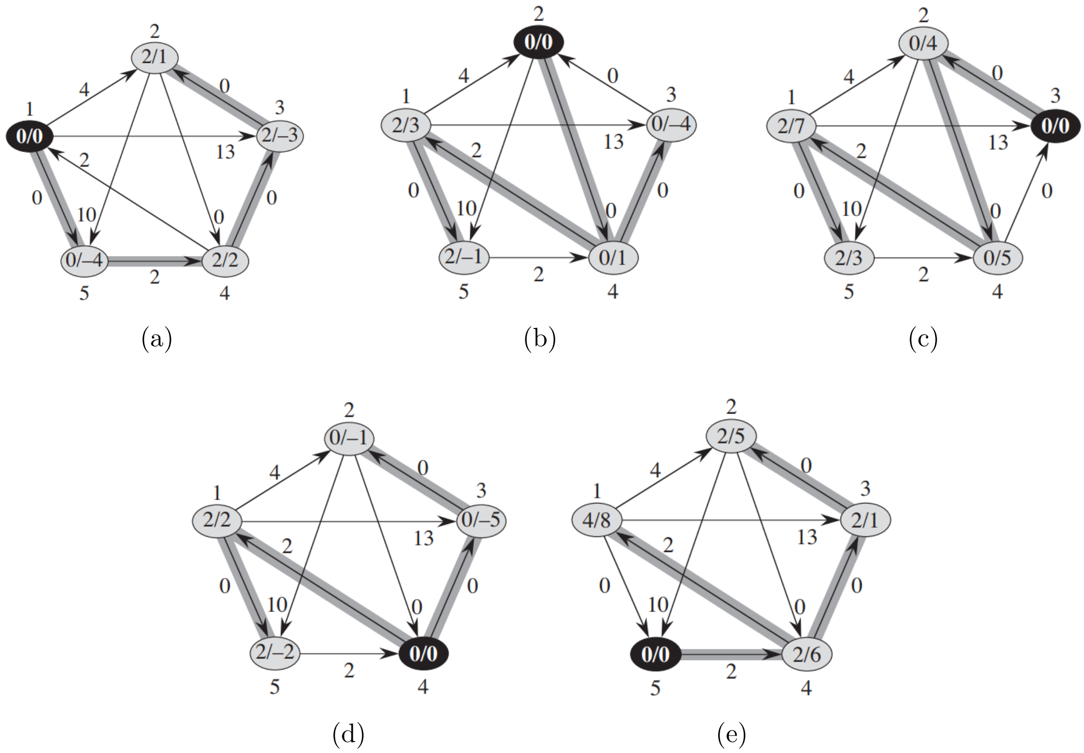

Figura 6: Rezultatul rulării lui *Dijkstra* pentru fiecare vârf din G folosind funcţia pondere $\widehat{w}$.