Floyd-Warshallov algoritam

Mauro Raguzin

10. siječnja 2023.

Opis algoritma

- Ulaz: Matrica susjednosti W težinskog digrafa $G = (V, E)$ koji ne sadrži negativne cikluse
- Izlaz: Matrica P najkraćih puteva između svih parova vrhova iz V

Za riješiti problem pronalaska **duljine** najkraćih puteva između svih parova vrhova, možemo primijeniti i neke druge poznate algoritme, npr. Dijkstrin algoritam ili Bellman-Fordov algoritam, tako da ih primijenimo na svaki vrh grafa.

Opis algoritma

- Ulaz: Matrica susjednosti W težinskog digrafa $G = (V, E)$ koji ne sadrži negativne cikluse
- Izlaz: Matrica P najkraćih puteva između svih parova vrhova iz V

Za riješiti problem pronalaska **duljine** najkraćih puteva između svih parova vrhova, možemo primijeniti i neke druge poznate algoritme, npr. Dijkstrin algoritam ili Bellman-Fordov algoritam, tako da ih primijenimo na svaki vrh grafa.
No, to često nameće dodatne restrikcije (za Dijkstrin algoritam, sve težine moraju biti nenegativne) ili dovodi do složenijih implementacija čija vremenska složenost ovisi o upotrebljenim strukturama podataka.

Opis algoritma

- Ulaz: Matrica susjednosti W težinskog digrafa $G = (V, E)$ koji ne sadrži negativne cikluse
- Izlaz: Matrica P najkraćih puteva između svih parova vrhova iz V

Za riješiti problem pronalaska **duljine** najkraćih puteva između svih parova vrhova, možemo primijeniti i neke druge poznate algoritme, npr. Dijkstrin algoritam ili Bellman-Fordov algoritam, tako da ih primijenimo na svaki vrh grafa.
No, to često nameće dodatne restrikcije (za Dijkstrin algoritam, sve težine moraju biti nenegativne) ili dovodi do složenijih implementacija čija vremenska složenost ovisi o upotrebljenim strukturama podataka.
Cilj Floyd-Warshallovog algoritma [1] je relativno **jednostavno** rješenje ovog problema, bez ikakvih dodatnih struktura, vremenske složenosti koja ne ovisi o prirodi ulaznog grafa. To postižemo primjenom dinamičkog programiranja.

Format grafa

- Svi ulazni grafovi su težinski digrafovi $G = (V, E)$, sa $|V| = n$.
- Težinska funkcija grafa G se predstavlja $n \times n$ matricom $W = (w_{ij})$ t.d.

$$w_{ij} = \begin{cases} 0 & i = j, \\ \text{težina brida } (i,j) & i \neq j \wedge (i,j) \in E, \\ \infty & i \neq j \wedge (i,j) \notin E. \end{cases}$$

Podjela na potprobleme

- Neka je zadan graf $G = (V, E)$ sa $V = \{1, 2, \ldots, n\}$ te podskup $V_k = \{1, 2, \ldots, k\} \subseteq V$ takav da je neki put $p = \langle v_1, v_2, \ldots, v_l \rangle$ najkraći put između dva vrha $i, j \in V$, sa svim svojim vrhovima u V_k.
- Vrh v koji je dio puta p, a nije jednak v_1 niti v_l, zovemo **međuvrhom** puta p.

Podjela na potprobleme

Uočimo da, za dani put p i podskup vrhova $V_k \subseteq V$ za neki k, mora vrijediti jedna od sljedećih tvrdnji:

- Ako k nije međuvrh puta p, tada su svi međuvrhovi puta p iz skupa $\{1, 2, \ldots, k-1\}$. To očito povlači da je najkraći put između istih vrhova i i j, koji se sastoji samo od vrhova iz $V_{k-1} = \{1, 2, \ldots, k-1\}$, jednak putu p (kada bi postojao neki kraći put u V_{k-1}, tada bi on bio kraći i u V_k).

Podjela na potprobleme

Uočimo da, za dani put p i podskup vrhova $V_k \subseteq V$ za neki k, mora vrijediti jedna od sljedećih tvrdnji:

- Ako k nije međuvrh puta p, tada su svi međuvrhovi puta p iz skupa $\{1, 2, \ldots, k-1\}$. To očito povlači da je najkraći put između istih vrhova i i j, koji se sastoji samo od vrhova iz $V_{k-1} = \{1, 2, \ldots, k-1\}$, jednak putu p (kada bi postojao neki kraći put u V_{k-1}, tada bi on bio kraći i u V_k).
- Ako pak k jest međuvrh puta p, tada možemo podijeliti put p na dva puta q i r tako da vrijedi $q = \langle i, \ldots, k \rangle$ i $r = \langle k, \ldots, j \rangle$. Sada je očito $p = qr$ te najkraći put q između i i k jest upravo najkraći put između ta dva vrha s vrhovima u V_{k-1}. To slijedi iz prošle točke primijenjene na put q, jer vrh k nije međuvrh puta q. Slično dobivamo i da je najkraći put r između k i j jednak najkraćem putu između ta dva vrha s vrhovima u V_{k-1}.

Rekurzivno rješenje

Na ovaj smo način dobili prirodnu i optimalnu strategiju podjele početnog problema na potprobleme: za sve vrhove k grafa, promatramo za koje sve puteve koji počinju vrhom i a završavaju u j bi taj vrh mogao biti međuvrh. Dakle, idemo redom kroz sve moguće međuvrhove k i računamo duljinu najkraćeg puta od i do k čiji vrhovi leže u V_{k-1} te zatim duljinu najkraćeg puta od j do k čiji vrhovi leže u V_{k-1}. Uzimanjem minimuma od sume ovih dviju duljina te duljine najkraćeg puta između i i j koji prolazi samo kroz vrhove u V_{k-1} dobivamo duljinu najkraćeg puta između i i j kroz V_k. To vodi do sljedećeg rekurzivnog rješenja našeg problema:

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

Iterativno rješenje

- Ako bi koristili ovu rekurziju za račun najkraćih puteva, trebalo bi nam trodiomenzionalno polje za smještaj $d_{ij}^{(k)}$ vrijednosti te bi minimalnu udaljenost za par (i, j) na kraju pronašli u $d_{ij}^{(n)}$.

Iterativno rješenje

- Ako bi koristili ovu rekurziju za račun najkraćih puteva, trebalo bi nam trodiomenzionalno polje za smještaj $d_{ij}^{(k)}$ vrijednosti te bi minimalnu udaljenost za par (i, j) na kraju pronašli u $d_{ij}^{(n)}$.
- Primjenom memoizacije, možemo postupno graditi rješenja počevši od $k = 0$ prema većim k-ovima — iterativno rješenje.

Iterativno rješenje

- Ako bi koristili ovu rekurziju za račun najkraćih puteva, trebalo bi nam trodiomenzionalno polje za smještaj $d_{ij}^{(k)}$ vrijednosti te bi minimalnu udaljenost za par (i, j) na kraju pronašli u $d_{ij}^{(n)}$.
- Primjenom memoizacije, možemo postupno graditi rješenja počevši od $k = 0$ prema većim k-ovima — iterativno rješenje.
- Ipak, takvo rješenje će i dalje imati prostornu složenost kao (1): $\Theta(n^3)$.

Iterativno rješenje

- Ako bi koristili ovu rekurziju za račun najkraćih puteva, trebalo bi nam trodiomenzionalno polje za smještaj $d_{ij}^{(k)}$ vrijednosti te bi minimalnu udaljenost za par (i, j) na kraju pronašli u $d_{ij}^{(n)}$.
- Primjenom memoizacije, možemo postupno graditi rješenja počevši od $k = 0$ prema većim k-ovima — iterativno rješenje.
- Ipak, takvo rješenje će i dalje imati prostornu složenost kao (1): $\Theta(n^3)$.
- Možemo bolje: samo $\Theta(n^2)$ korištene memorije tj. dovoljna nam je samo jedna (po mogućnosti ulazna) matrica.
- Tako dobivamo izvorni Floyd-Warshallov algoritam (radi jednostavnosti, pseudokod koristi indekse koji počinju sa 1).

Iterativno rješenje — pseudokod

Algoritam 1: Floyd-Warshall

ulaz: Realna $n \times n$ matrica $m = (m_{ij})$

izlaz: Realna $n \times n$ matrica (d_{ij}) duljine najkraćih puteva od i do j; zauzima isti prostor kao m

for $k \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
        if $m_{ik} < \infty$ **then**
            for $j \leftarrow 1$ **to** n **do**
                if $m_{kj} < \infty$ **then**
                    $\mathsf{s} \leftarrow m_{ik} + m_{kj}$
                    if $\mathsf{s} < m_{ij}$ **then**
                        $m_{ij} \leftarrow \mathsf{s}$
                    end
                end
            end
        end
    end
end

Dokaz da nam je dovoljna samo jedna matrica

- Moramo dokazati da je $d_{ij}^{(k)} = m_{ij}$, za sve (i, j) i m_{ij} je vrijednost pri završetku izvođenja k-te iteracije petlje u liniji 1.
- Dokaz ide indukcijom po k: baza za $k = 1$ je jasna, jer tada je jedino moguće da su i i j povezani bridom.
- Pretpostavimo da tvrdnja vrijedi sve do nekog fiksnog k. Po prije napravljenoj analizi problema, najkraći put između i i j u V_k ne sadrži vrh k ili ga sadrži.
- Ako ga ne sadrži, onda je $m_{ij} = d_{ij}^{(k-1)} = d_{ij}^{(k)}$, pa imamo tvrdnju po pretpostavci indukcije.
- Ako taj put pak sadrži vrh k, tada je m_{ij} opet jednak onom u prošlom slučaju ili, u slučaju ažuriranja, novoj (manjoj) sumi koja je izračunata za manje vrijednosti k.

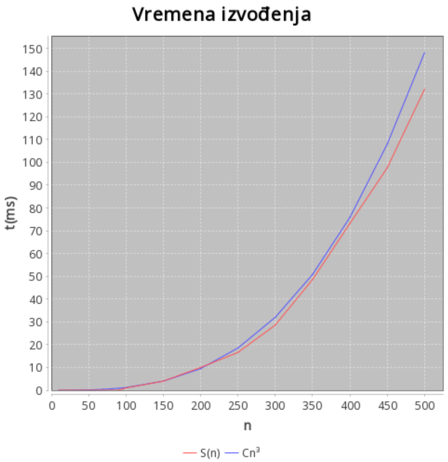
Dokaz (nastavak)

- Uočimo da je ažuriranje m_{ij} moguće samo kada $k \neq i$ i $k \neq j$, no onda nije moguće da m_{ik} ili m_{kj} u liniji 6 poprime nove vrijednosti unutar iteracija za isti k tj. nema opasnosti da će se neki ažurirani m_{ij} u kasnijoj iteraciji za isti k čitati u liniji 6, pa opet primjenjujemo pretpostavku indukcije i imamo tvrdnju.

Analiza vremenske složenosti

- Algoritam 1 se sastoji od tri ugniježđene for-petlje, pa mu je ukupna vremenska složenost očito određena vremenskom složenosti tijela tih petlji.
- Kako se na najdubljoj razini zadnje for-petlje ne radi ništa drugo nego usporedba i, u najgorem slučaju, pridruživanje vrijednosti matrici, to zaključujemo da je količina posla unutar svake iteracije odozgo omeđena konstantom. Također, svaka for-petlja izvede točno n iteracija, pa je vremenska složenost algoritma $O(n^3)$.
- Štoviše, ona je i $\Omega(n^3)$, jer iako je moguće da se for-petlja u liniji 4 u potpunosti preskoči, to malo znači kod gustih grafova, gdje je $m_{ik} = \infty$ jako rijedak događaj.
- Dakle, u najgorem slučaju, imamo vremensku složenost $\Theta(n^3)$ pri čemu je skrivena konstanta malena jer je kôd vrlo jednostavan i ne koristi dodatne strukture podataka[2].

Empirijski izmjerena vremenska složenost



Slika: Izmjerena ovisnost vremena izvođenja Floyd-Warshallovog algoritma o veličini grafa n

Primjene — pronalaženje najkraćih puteva

- Ako je bitno pronaći i odgovarajuće najkraće puteve (ne samo njihove duljine), algoritam 1 se najčešće nadopunjuje dijelom koji, za svaki novi pronađeni međuvrh k preko kojeg dobivamo kraći put od i do j, za par (i, j) pamti idući skok kao onaj preko k.
- Tako dobivamo stablo optimalnih puteva grafa, uz dodatnu prostornu složenost od $\Theta(n^2)$.
- Rekonstrukcija puta između bilo koja dva vrha iz takvog stabla: vidi Umjetnu inteligenciju

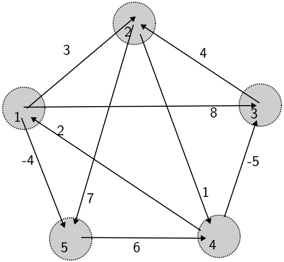
Primjene — račun tranzitivnog zatvorenja grafa

- Još jedna jednostavna primjena ovog algoritma je za račun tranzitivnog zatvorenja grafa $G = (V, E)$, $|V| = n$, koje se definira kao graf $G^* = (V, E^*)$, gdje je

$$E^* = \{(i, j) | \text{postoji put od } i \text{ do } j \text{ u } G\}.$$

- Dodijelimo težinu 1 svakom bridu u E.
- Primijenimo algoritam 1 na G radi dobivanja informacije o tome postoji li put između neka dva vrha i i j ili ne; ako postoji, bit će $d_{ij} < n$ na izlazu iz 1, a inače će biti ∞.

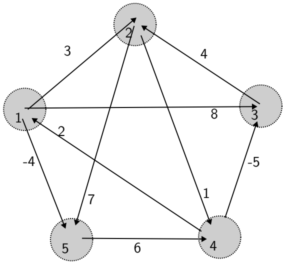
Primjer



Slika: Graf iz primjera

Za kraj, pokažimo rad algoritma na primjeru iz [2]. Automatsko izvođenje i ispisivanje rezultata za ovaj primjer je dio Java programskog rješenja[3], implementirano u datoteci `Primjer1.java`. Tekstualno se prikazuje niz matrica međurezultata m_k, za $k = 1, 2, \ldots, n$ gdje m_k odgovara matrici m u algoritmu 1 u trenutku završetka iteracije k prve for-petlje.

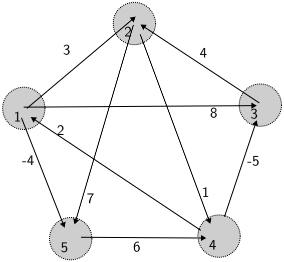
Primjer



k=1:

[0.0, 3.0, 8.0, Infinity, -4.0]
[Infinity, 0.0, Infinity, 1.0, 7.0]
[Infinity, 4.0, 0.0, Infinity, Infinity]
[2.0, 5.0, -5.0, 0.0, -2.0]
[Infinity, Infinity, Infinity, 6.0, 0.0]

Primjer



k=2:

[0.0, 3.0, 8.0, 4.0, -4.0]

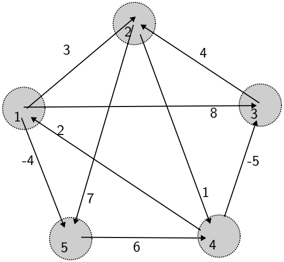
[Infinity, 0.0, Infinity, 1.0, 7.0]

[Infinity, 4.0, 0.0, 5.0, 11.0]

[2.0, 5.0, -5.0, 0.0, -2.0]

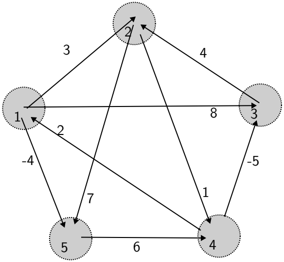
[Infinity, Infinity, Infinity, 6.0, 0.0]

Primjer



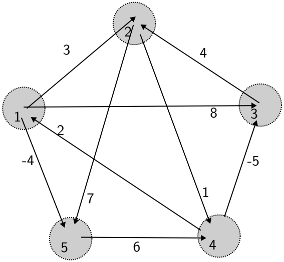

```
k=3:
[0.0, 3.0, 8.0, 4.0, -4.0]
[Infinity, 0.0, Infinity, 1.0, 7.0]
[Infinity, 4.0, 0.0, 5.0, 11.0]
[2.0, -1.0, -5.0, 0.0, -2.0]
[Infinity, Infinity, Infinity, 6.0, 0.0]
```

Primjer




```
k=4:
[0.0, 3.0, -1.0, 4.0, -4.0]
[3.0, 0.0, -4.0, 1.0, -1.0]
[7.0, 4.0, 0.0, 5.0, 3.0]
[2.0, -1.0, -5.0, 0.0, -2.0]
[8.0, 5.0, 1.0, 6.0, 0.0]
```

Primjer




```
k=5:
[0.0, 1.0, -3.0, 2.0, -4.0]
[3.0, 0.0, -4.0, 1.0, -1.0]
[7.0, 4.0, 0.0, 5.0, 3.0]
[2.0, -1.0, -5.0, 0.0, -2.0]
[8.0, 5.0, 1.0, 6.0, 0.0]
```

Korištena literatura

[1] Robert W. Floyd. „Algorithm 97: Shortest Path”. *Commun. ACM* 5.6 (lipanj 1962.), str. 345. ISSN: 0001-0782. DOI: 10.1145/367766.368168. URL: https://doi.org/10.1145/367766.368168.

[2] Thomas H Cormen i dr. *Introduction to Algorithms*. 3. izdanje. London, England: MIT Press, 2009., str. 684, 693, 694, 695.

[3] *Repozitorij s programskim rješenjem*. 2022. URL: https://github.com/mraguzin/floyd-warshall-oaa.