Floyd-Warshallov algoritam

Mauro Raguzin

30. prosinca 2022.

Sadržaj

1 Sažetak

U ovom radu razmatramo Floyd-Warshallov algoritam kroz njegovu konstrukciju, opis jedne implementacije i diskusiju o primjenama i složenosti algoritma. Popratno programsko rješenje je napisano u programskom jeziku Java.

2 Opis algoritma

Floyd-Warshallov algoritam je izvorno objavljen kao vrlo kratak algoritam za pronalazak najkraćih puteva između parova *svih* vrhova danog usmjerenog težinskog grafa, *bez* restrikcija na negativnost težine bridova (iako i dalje pretpostavljamo da graf ne sadrži negativne cikluse), u radu [1]. U tome se odmah bitno razlikuje od npr. Dijkstrinog algoritma, koji računa samo najmanje udaljenosti od samo jednog izvornog vrha grafa do svih ostalih, ali i to

može raditi tek ako su svi bridovi grafa nenegativne težine. Dakako, problem računanja najkraćih udaljenosti između svih parova vrhova grafa bismo mogli riješiti i Dijkstrinim algoritmom, i to njegovom pojedinačnom primjenom na sve vrhove grafa. Vremenska složenost bi se u tom slučaju, za guste grafove čiji je broj bridova blizu n^2 za n vrhova, približavala $O(n^3 + n^2)$ ako se koristi implementacija pomoću prioritetnog reda, što je u $O(n^3)$[2]. Za rijetke grafove, boljim implementacijama Dijkstrinog algoritma preko prioritetnog reda temeljenog na Fibonaccijevoj hrpi, možemo dobiti bolju složenost od $O(|V||E| \lg V)$. Primjenom dinamičkog programiranja, možemo ipak dobiti rješenje ovog problema koje nije osjetljivo na prirodu ulaznog grafa, uvijek u $O(n^3)$ vremena — upravo to je cilj Floyd-Warshallovog algoritma.

Svi grafovi koje spominjemo u ovom radu će biti težinski usmjereni grafovi (digrafovi) $G = (V, E)$, sa $|V| = n$, s tim da ćemo težinsku funkciju grafa G predstavljati $n \times n$ matricom $W = (w_{ij})$ tako da vrijedi

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

Kao što je uobičajeno kod dizajna algoritama pomoću dinamičkog programiranja, za rješenje ovog problema moramo pronaći dobar način podijele početnog problema na manje potprobleme tako da vrijedi princip optimalnosti. Kod Floyd-Warshallovog algoritma to činimo na temelju par činjenica koje ćemo sada predstaviti.

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. U idućim razmatranjima, moguće je i da postoji više najkraćih puteva tj. puteva najmanje duljine između neka dva vrha, no nas samo zanima jedan (bilo koji) od njih. 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 (takvih može biti i više) između ta dva vrha sa (svim svojim) 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}.

Na ovaj smo način dobili prirodnu i optimalnu strategiju podjele početnog problema u potprobleme: za sve vrhove k grafa, promatramo za koje sve puteve koji počinju vrhom i a završavaju 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 minimalnog puta između i i j kroz V_k. To vodi do sljedećeg rekurzivnog rješenja našeg problema:

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

Dakako, osnovni slučaj rekurzije, kada nema međuvrha k (tj. $k = 0$), odgovara situaciji gdje su i i j susjedi, pa je minimalna udaljenost između njih ujedno i težina odgovarajućeg (usmjerenog) brida. 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) pronašli u $d_{ij}^{(n)}$.

3 Komentari o implementaciji

Floyd-Warshallov algoritam se sastoji u efikasnijoj, iterativnoj implementaciji gornje rekurzije koja postupno gradi veća rješenja počevši od manjih. Pritom moramo koristiti memoriju, za što može direktno poslužiti matrica W ako je smijemo „pregaziti" našim algoritmom, pa dodatne memorijske alokacije nisu potrebne[1].

Zaista, implementacija algoritma ne mora pamtiti najkraće udaljenosti vrhova za svaki od najviše n mogućih međuvrhova, kao što nam to (1) daje naslutiti, već može koristiti samo jednu (po mogućnosti ulaznu) matricu. Da bismo se u to uvjerili, dokažimo indukcijom po k 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. Baza indukcije je jasna za $k = 1$, jer tada je jedino moguće da su i i j povezani bridom. Ako pretpostavimo da tvrdnja vrijedi sve do nekog

fiksnog k, nalazimo da, po prošloj 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 (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 7 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 7), pa opet primjenjujemo pretpostavku indukcije i imamo tvrdnju.

Ipak, Java implementacija u ovom radu inicijalno prekopira W u pomoćnu matricu koju tada modificira prema donjem algoritmu, što asimptotski ne mijenja vremensku složenost ovog algoritma. Glavni kôd se nalazi u datoteci `Graf.java`, unutar metode `najkraćiPut`. Ispod se nalazi pseudokod nastao prema [1] koji koristi indekse koji počinju od 1. Slično, glavni program koji očekuje korisnički unos grafa također, radi jednostavnosti, prima indekse koji počinju od 1.

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

```
1  for k ← 1 to n do
2      for i ← 1 to n do
3          if m_ik < ∞ then
4              for j ← 1 to n do
5                  if m_kj < ∞ then
6                      s ← m_ik + m_kj
7                      if s < m_ij then
8                          m_ij ← s
9                      end
10                 end
11             end
12         end
13     end
14 end
```

4 Analiza složenosti i primjene

Algoritam 1 se sastoji od tri ugniježđene for-petlje, pa mu je ukupna vremenska složenost očito određena vremenskom složenosti tijela tih petlji. Kako se i 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 ukupna vremenska složenost algoritma $O(n^3)$. Štoviše, ona je i $\Omega(n^3)$, jer iako je moguće da se (kao možebitna optimizacija) 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 u $\Theta(n^3)$ pri čemu je skrivena konstanta malena jer je kôd vrlo jednostavan i ne koristi dodatne strukture podataka[2]. Također, uz spomenuti trik pri implementaciji algoritma, prostorna mu složenost može biti $\Theta(n^2)$, pa je prikladan i za primjene na većim grafovima.

Ako je bitno pronaći i odgovarajuće najkraće puteve, 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 iz takvog stabla je onda uobičajen linearan algoritam koji iterativno ili rekurzivno sastavlja povezanu listu vrhova između traženih krajeva, što se često rabi u Umjetnoj inteligenciji.

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

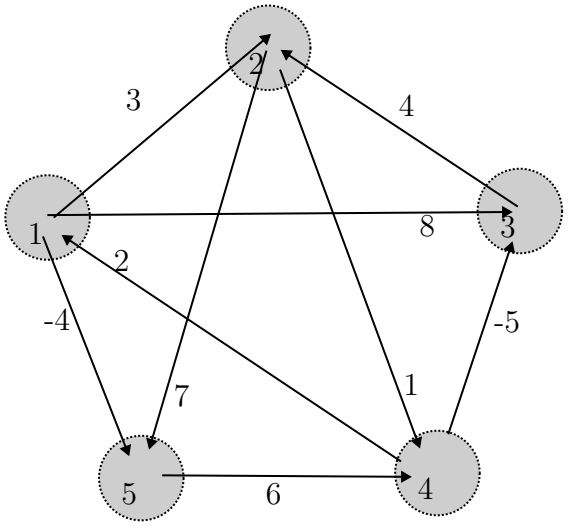
$$E^* = \{(i, j) | \text{postoji put od } i \text{ do } j \text{ u } G\}.$$

Računanje tranzitivnog zatvorenja grafa u $\Theta(n^3)$ vremena je moguće dodjeljivanjem težine 1 svakom bridu u E i korištenjem Floyd-Warshallovog algoritma 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 ∞.

Konačno, spomenimo i da je ovaj algoritam poseban slučaj Kleenijevog algoritma za pronalaženje regularnog izraza kojeg prihvaća zadan konačni automat.

5 Primjer

Zatvaramo ovu diskusiju Floyd-Warshallovog algoritma s jednim primjerom preuzetim iz [2]. Automatsko izvođenje i ispisivanje rezultata za ovaj primjer



Slika 1: Graf iz primjera

je implementirano u datoteci `Primjer1.java`. Tekstualno se prikazuje niz matrica međurezultata W_k, za $k = 1, 2, \ldots, n$ gdje W_k odgovara matrici m u algoritmu 1 u trenutku završetka iteracije k prve for-petlje.

```
    k=0:
[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]
    k=1:
[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]
    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, -1.0, -5.0, 0.0, -2.0]
[Infinity, Infinity, Infinity, 6.0, 0.0]
    k=3:
[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]
    k=4:
[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]
    Konačno rješenje:
[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]
```

Konačno, u datoteci `Primjer2.java` je implementiran nasumični generator grafova raznih veličina na kojem se testira Floyd-Warshallov algoritam. Slika 2 prikazuje tako izmjerenu empirijsku ovisnost vremena o veličini uzorka n (broj vrhova grafa) te možemo uočiti da je poklapanje s kubičnim polinomom očekivano veliko.

6 Zaključak

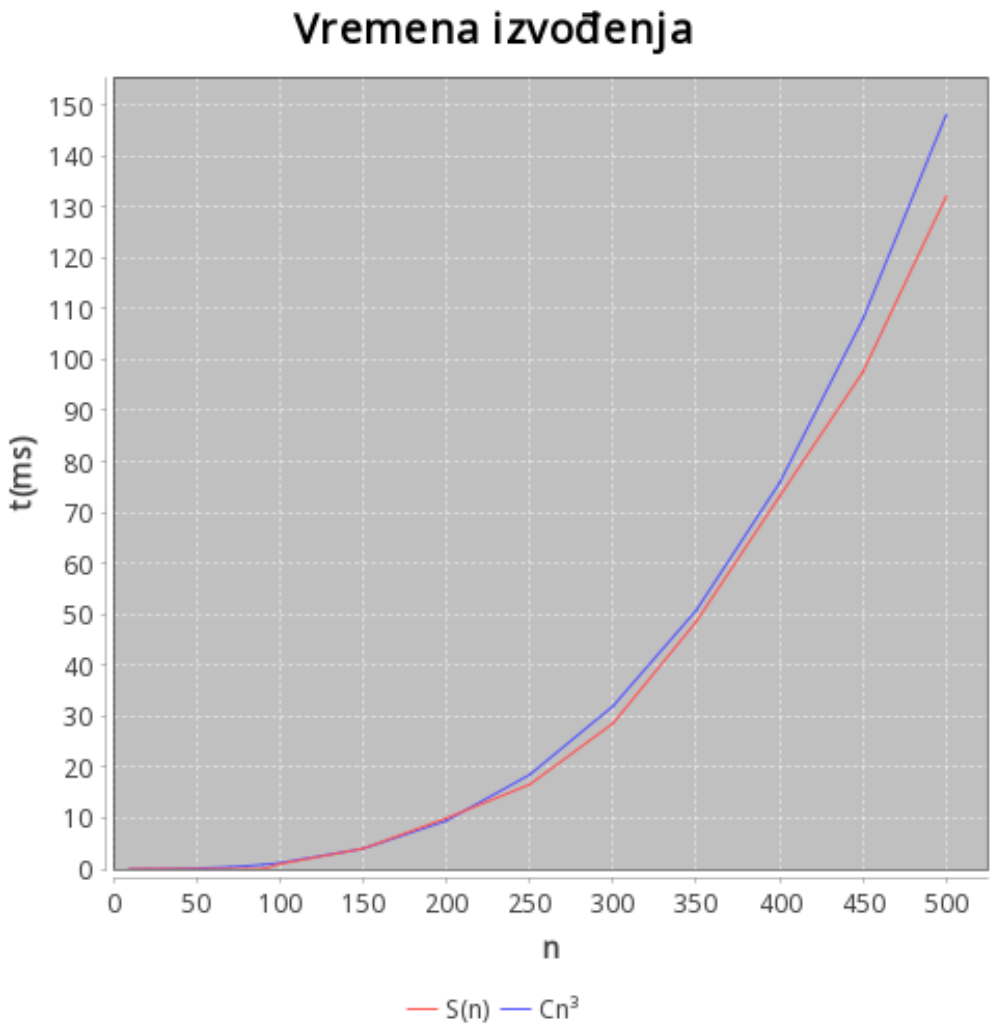
U ovom smo radu razmotrili Floyd-Warshallov algoritam, kao važan primjer dinamičkog programiranja pomoću kojeg se efikasno može riješiti problem određivanja duljina najkraćih puteva između svih parova vrhova danog grafa, kao i rekonstrukcija samih najkraćih puteva između vrhova. Možemo zaključiti da je ovaj algoritam posebno koristan za guste grafove, kada je asimptotski jednake vremenske složenosti kao i najbolje implementacije Dijkstrinog algoritma primijenjenog na svaki vrh, ali zbog jednostavnosti, kratkoće kôda i korištenja samo jedne matrice, u praksi može dati bitno bolje performanse u usporedbi s Dijkstrom. Kada graf sadrži i negativne bridove, onda nije moguća opetovana primjena Dijsktrinog algoritma, ali može poslužiti Bellman-Fordov algoritam koji bi za guste grafove tada imao složenost od $O(|V|^4)$, što očito znači da je tada najbolje odabrati upravo Floyd-Warshallov algoritam.

Kada je pak graf rijedak tj. nema puno bridova, postoje drugi algoritmi za rješenje ovog problema koji se mogu pokazati boljima u praksi, poput Johnsonovog algoritma[2].

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.



Slika 2: Izmjerena vremenska ovisnost vremena izvođenja Floyd-Warshallovog algoritma o veličini grafa n